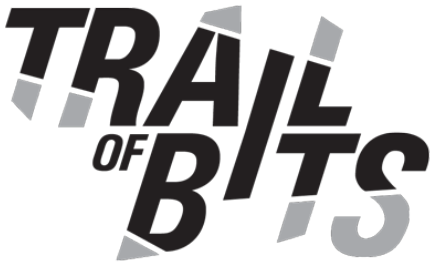

Argent Wallet Smart Contracts

Security Assessment

August 14, 2020

Prepared For:
Julien Niset | *Argent*
julien@argent.xyz

Itamar Lesuisse | *Argent*
itamar@argent.xyz

Prepared By:
Josselin Feist | *Trail of Bits*
josselin@trailofbits.com

Sam Sun | *Trail of Bits*
sam.sun@trailofbits.com

Changelog:

August 14, 2020:	Initial report delivered
September 4, 2020:	Added Appendix F with retest results and TOB-ARGENT-026
October 6, 2020:	Updated Appendix F with retest results on TOB-ARGENT-005

Executive Summary

Argent engaged Trail of Bits August 3–4, 2020, to review the security of the Argent smart wallet. Trail of Bits conducted this assessment over four person-weeks with two engineers working from commit 29ff50da.

In week one, we gained an understanding of the codebase and looked for common Solidity flaws in the wallet and the modules. We also reviewed the contracts for vulnerabilities that would allow an attacker to steal funds, bypass the daily limit, or execute unauthorized operations. In week two, we did a more in-depth review of all the modules to learn how their interactions with the rest of the ecosystem might affect the security of user funds.

Our review resulted in 26 findings, ranging from high to informational severity. All the high-severity findings involve either a compromised owner key for a wallet, or use of the wallet in a specific context to be exploited. Several high-severity issues allow an attacker to bypass the daily limit restriction. Additionally, three issues occurred because the return data is not checked when wallets are called.

Appendix B contains additional code quality issues. Appendix C contains our recommendations to interact with arbitrary tokens. Appendix E describes how to use Slither, a static analysis framework, to detect modules that share the same entry points.

Overall, the codebase is very organized and adheres to Solidity best practices. The contracts are well written and meet most security expectations. However, further work could be done to prevent users from bypassing the daily limit safeguard and ensure that documentation is kept up to date with code.

We recommend that Argent address our findings and document a process for adding new modules. The documentation should highlight factors such as how the new module may interact with existing modules and the wider DeFi landscape. We also recommend moving to a stricter access control model where permission for two modules to interact must be explicitly permitted instead of implicitly allowed. Finally, we recommend creating a guideline for relayers that details how to prevent abuse by users.

Project Dashboard

Application Summary

Name	Argent smart contracts wallet
Version	29ff50da
Type	Solidity
Platforms	Ethereum

Engagement Summary

Dates	August 3–August 14, 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	4 person-weeks

Vulnerability Summary

Total High-Severity Issues	11	■■■■■■■■■■
Total Medium-Severity Issues	1	■
Total Low-Severity Issues	2	■■
Total Informational-Severity Issues	12	■■■■■■■■■■■■
Total	26	

Category Breakdown

Access Controls	7	■■■■■■■
Auditing and Logging	1	■
Cryptography	1	■
Data Validation	16	■■■■■■■■■■■■■■■■
Undefined Behavior	1	■
Total	26	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Satisfactory.** Access control model of the system is too permissive, but caused no direct vulnerabilities at the time of review.
Arithmetic	**Strong.** No issues with arithmetic were found.
Assembly Use	**Strong.** Assembly is used sparingly.
Centralization	**Satisfactory.** An attacker with privileged access can tamper with new wallets, but existing wallets are safe unless wallet owners explicitly opt in to new behavior.
Upgradeability	**Not Applicable.**
Function Composition	**Strong.** Functionality is separated into modules and there is no significant amount of code duplication.
Front-Running	**Strong.** The system is not vulnerable to front-running attacks.
Key Management	**Not Applicable.**
Monitoring	**Satisfactory.** Some functions were lacking events.
Specification	**Satisfactory.** In a few cases the documentation did not reflect the behavior of the code.
Testing & Verification	**Satisfactory.** There are numerous unit tests but no property tests.

Engagement Goals

The engagement was scoped to provide a security assessment of the Argent wallet smart contracts located in the `contracts` directory, as well as the two prior implementations of `BaseWallet` located in `contracts-legacy/v1.6.0/contracts/wallet` and `contracts-legacy/v1.3.0`.

Specifically, we sought to answer the following questions:

- Can an attacker drain funds from an arbitrary user's wallet without knowing the user's private keys?
- Can an attacker send an arbitrary transaction from an arbitrary user's wallet without knowing the user's private keys?
- Can an attacker escalate their privileges and gain access to a user's wallet or the Argent core contracts?
- Can an attacker bypass the daily limit if they have knowledge of the user's private key?
- Can an attacker with privileged access to the Argent core contracts negatively impact an arbitrary user's wallet?
- Can malicious guardian(s) or relayer(s) negatively impact a user's wallet?

Coverage

- **ENS Integration:** This allows an Argent wallet to be associated with a specific ENS subdomain on the root `argent.xyz`. We reviewed the registration behavior to ensure that users cannot override a subdomain owned by another user. We also reviewed the ENS manager to ensure that subdomains cannot be stolen by an attacker.
- **Storage:** These contracts provide a stable storage location for modules to read and write from. We looked for flaws that might allow an attacker to modify the storage for arbitrary Argent wallets.
- **Registries**: These contracts allow the Argent team to update certain system parameters, such as the list of tokens supported on Compound or Maker, or the list of whitelisted modules. We reviewed the registries to ensure that a non-privileged user cannot add new entries or remove existing entries.
- **Wallets:** These contracts are the Argent wallets themselves. We reviewed them to ensure that only the wallet owner is able to make the wallet perform arbitrary calls or transfer tokens.
- **Modules:** Wallet functionality—e.g., token transfer, guardian recovery, and integrations with other protocols such as Maker or Compound—is implemented using modules. We looked for flaws that may accidentally affect a user's Argent wallet negatively. We also looked for flaws that may allow an attacker to affect another user's Argent wallet.

Contracts in the `contracts-legacy` directory were out of scope for this assessment, with the exception of the two `BaseWallet` implementations located in `contracts-legacy/v1.6.0/contracts/wallet` and `contracts-legacy/v1.3.0`.

Tests, helper scripts, and off-chain components such as the Argent relayer were out of scope for this assessment.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

☐ **Consider whitelisting tradeable tokens or enforcing the daily limit in `TokenExchanger`.** An attacker can create a malicious token tradable in `TokenExchanger` to bypass the daily limit (TOB-ARGENT-001).

☐ **To prevent slippage in `TokenExchanger`, either**:

- Decrease the daily limit when assets are swapped in `TokenExchanger`, and document that swapping might decrease the daily limit, or
- Use Argent's Oracle to ensure that the swapped amounts are within an expected range (TOB-ARGENT-002).

☐ **To prevent slippage in `MakerV2Loan`, either:**

- Decrease the daily limit of the ETH swapped in Uniswap by `MakerV2Loan`, and document that repaying a debt might decrease the daily limit, or
- Remove the functionality that automatically performs a swap and require the user to have enough DAI on hand before closing a CDP (TOB-ARGENT-003).

☐ **Use the `chainID` opcode in the signature schema** to prevent signatures from being reused in case of forks (TOB-ARGENT-004).

☐ **If the wallet has one guardian, clarify the recovery process by either:**

- Requiring `(number_of_guardian/ 2) + 1` signatures for `cancelRecovery`, or discard the owner's signature, or
- Updating the documentation and making it explicit that users need two guardians to be protected from theft.

An attacker compromising a wallet with one guardian will be able to prevent any recovery requests (TOB-ARGENT-005).

☐ **Split the Oracle price update into two sets of functions:**

- Create `setPriceLimited` and `setPriceForTokenListLimited` to allow a limited update per day. These functions will be called by the Oracle's bot.

- Allow only a safety stored account or multisig wallet to call `setPrice` and `setPriceForTokenList`.

Splitting the right to update the price into two entities will decrease the overall risks associated with the Oracle (TOB-ARGENT-006).

☐ **Use a two-step procedure for `Owned.changeOwner` and all irrecoverable critical operations.** Using a one-step procedure for critical operations is error-prone (TOB-ARGENT-007).

☐ **Check the contract's existence before every low-level call in `baseWallet.invoke` if `_data` is not empty.** Since there's no existence check for low-level calls, it may be incorrectly assumed that operations have been executed TOB-ARGENT-008.

☐ **Allow owners to remove addresses that are pending but not yet whitelisted.** This will prevent an attacker from compromising a key to add malicious addresses to the whitelisted accounts (TOB-ARGENT-009).

☐ **Use a `uint256` for the loop iterator in `RelayerModule.validateSignatures`.** Using a shorter type can lead to an infinite loop and prevent the function from being executed (TOB-ARGENT-010).

☐ **Update the documentation** in `infrastructure/storage/TokenPriceStorage.sol#L26` to reflect the correct token price scale. Also, either implement a delay for `RecoveryManager.transferOwnership`, or update its documentation. Out-of-date or incorrect documentation impedes code review and may allow issues to escape detection. (TOB-ARGENT-011).

☐ **Revert in `getRequiredSignatures` if the wallet has no guardians.** The concept of a majority does not apply in this case (TOB-ARGENT-012).

☐ **Enforce that the target contract is not a spendable token even if the spender is whitelisted.** With no check on the target contract, the daily limit can be bypassed (TOB-ARGENT-013).

☐ **Replace all low-level calls in `isERC721` with high-level calls.** This will ensure that the function will return success only for `isERC721` contracts (TOB-ARGENT-014).

☐ **Update the daily limit in `TransferManager.executePendingTransfer`.** Since there's no limit on pending transactions, an attacker can bypass the daily limit (TOB-ARGENT-015).

☐ **Check the return value of `transfer` in `RelayerModule.refund`**. If the return value is not checked, relayers may not be paid for their work (TOB-ARGENT-016).

☐ **Revert in `Utils.recoverSigner` if `ecrecover` returns zero.** Additionally, prevent signature malleability by reverting if `s` is greater than `0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0` to block an attacker from abusing the `ecrecover` edge cases (TOB-ARGENT-017).

☐ **Check the return value of `transfer` and `transferFrom in BaseTransfer`.** This will prevent failed transactions from decreasing the daily limit (TOB-ARGENT-018).

☐ **Create `BaseModule.init` to prevent a wallet from calling `init` multiple times, and instead allow it to call the functions in the derived classes.** Since there's no built-in check to prevent the `init` functions from being called multiple times, unexpected behavior can occur (TOB-ARGENT-019).

☐ **Document that a transaction can be discarded due to the 1/64 gas rule** to help ensure that users will increase the transaction's gas limit if they are targeted by an attack (TOB-ARGENT-020).

☐ **Check that the module is (or was) registered in `ModuleRegistry`.** This will prevent an attacker from abusing `RelayerModule` to execute arbitrary contracts (TOB-ARGENT-021).

☐ **Consider restricting module calls to authorized modules, perhaps by using DSAuth.** For example, `TransferManager` should be callable by the wallet's owner or the `RelayerModule` (TOB-ARGENT-022).

☐ **Use `WalletFactory` to track which contracts were generated by the factory**. **Check that the wallet is present in `onlyWallet`, `onlyWalletOwner`, and `onlyWalletModule`.** Begin with the list of known legitimate wallets. This will prevent attackers from calling the modules with malicious wallets (TOB-ARGENT-023).

☐ **Add events for all critical operations listed in TOB-ARGENT-024** to monitor the contracts and detect suspicious behavior.

☐ **Check for `tx.gasprice >= _gasPrice` in `RelayerModule.execute`** to prevent relayers from reducing the gas price (TOB-ARGENT-025).

☐ **Prevent `RecoveryManager.transferOwnership` from being called if the owner is a guardian** so the wallet's owner is not a guardian invariant to be broken (TOB-ARGENT-026).

☐ **Document that `RecoveryManager.finalizeRecovery` can allow a guardian to be the owner.** The wallet's owner is not a guardian invariant to be broken using `RecoveryManager.finalizeRecovery` (TOB-ARGENT-026).

Long term

☐ **Thoroughly evaluate and document all possible asset transfers of every module.** Ensure each transfer is tested and its associated risks on the daily limit considered (TOB-ARGENT-001, TOB-ARGENT-002, TOB-ARGENT-003).

☐ **Document and carefully review any signature schema, including their robustness to reuse on different wallets, contracts, and blockchains.** Make sure the users are aware of signing best practices and the danger of signing messages from untrusted sources (TOB-ARGENT-004).

☐ **Document all the modules' requirements and evaluate their robustness in case of wallet without guardian, with one guardian, or with multiple guardians.** Each module's behavior in case of missing guardian or low number must be carefully considered (TOB-ARGENT-005).

☐ **Carefully review the Oracle's bot setup** and consider conducting a bot infrastructure security review (TOB-ARGENT-006).

☐ **Identify and document all possible actions and their associated risks for privileged accounts.** Document and prepare adequate mitigations for everything that can happen when a wallet is compromised. Risks associated with privileged accounts must be understood and carefully reviewed (TOB-ARGENT-007, TOB-ARGENT-009).

☐ **Avoid low-level calls** since they lack the mitigations present in high-level calls. If they're not avoidable, carefully review the Solidity documentation, especially the Warnings section (TOB-ARGENT-008).

☐ **Identify all loops iterating over a user-controlled value, and thoroughly test their iterations with large values.** Do not use small type integers for loop iterators (TOB-ARGENT-010).

☐ **Ensure the code is thoroughly tested to reflect the properties stated in the documentation.** Any mismatches can lead to undefined behavior and impede code review (TOB-ARGENT-011).

☐ **Evaluate which modules need at least one guardian, and prevent their execution if this condition is not met.** This will prevent unexpected behavior from modules if a wallet has no guardians (TOB-ARGENT-012).

☐ **Long term, consider conducting property testing with Echidna to ensure invariants are properly enforced.** Property testing can detect access controls and state-related issues (TOB-ARGENT-013).

☐ **Consider checking if the potential NFT contract is an ERC20 token for which the price is tracked.** This will prevent the daily limit on ERC20 tokens from being abused by the NFT module (TOB-ARGENT-014).

☐ **Carefully review the Solidity documentation, especially the Warnings section regarding edge cases of Solidity and EVM.** (TOB-ARGENT-014, TOB-ARGENT-017).

☐ **Thoroughly document the impact of each module on the daily limit. Carefully review the external calls when adding new modules, and look for flaws that would allow the daily limit to be bypassed.** We found many issues in modules that allowed the daily limit to be bypassed, so this aspect must be carefully reviewed when adding new modules (TOB-ARGENT-015).

☐ **Write a guideline for transaction relayer integration** that includes recommendations on what to check before accepting a transaction relay: gas limit, price, likelihood of transaction success, etc. (TOB-ARGENT-016).

☐ **Review any call to `invokeWallet` and evaluate whether the return value should be checked.** Without a return value check, unexpected behavior can occur (TOB-ARGENT-016, TOB-ARGENT-018).

☐ **Carefully review any standard you interact with (including the ERC20 standard).** Edge cases of standards must be properly understood and taken into account (TOB-ARGENT-018).

☐ **When adding a module, create a checklist of review questions, including:**

- What functions can the module call? This includes checking for function ID collision (see Slither's `function-id` printer and Appendix E) and calls to modules and wallets.
- Does the module handle assets?
- Does the module hold assets? If so `BaseModule.recoverToken` must be disabled.
- What are the functions that anyone can call? What is the impact of these functions?
- Are the operations at initialization thoroughly reviewed (including for re-entrancies)?
- Are the new static calls legitimate and their purposes documented?

(TOB-ARGENT-019, TOB-ARGENT-021, TOB-ARGENT-022).

☐ **Carefully review the gas rules of Ethereum**. These rules are frequently changing and might impact the behavior of the transaction relayer (TOB-ARGENT-020).

☐ **Properly design mitigations to prevent malicious wallets or modules from calling the contracts.** Access controls should distinguish legitimate calls from malicious ones (TOB-ARGENT-023).

☐ **Consider using a blockchain monitoring system to track any suspicious behavior in the contracts.** The system relies on the correct behavior of several contracts, so a monitoring system that tracks critical events would quickly detect compromised system components (TOB-ARGENT-024).

☐ **Change the refund algorithm to better manage an environment with adversarial relayers** and consider creating economic incentives that encourage relayers to behave well (TOB-ARGENT-025).

☐ **Thoroughly document the system's state transition, and highlight the owner and guardian update.** The system relies on the correct state transition, so clear documentation will aid code review and prevent new issues (TOB-ARGENT-026).

Findings Summary

#	Title	Type	Severity
1	Absence of validation on exchange pairs allows users to bypass daily limit	Data Validation	High
2	Uncapped slippage when swapping tokens allows users to bypass daily limit	Data Validation	High
3	Uncapped slippage when buying DAI allows users to bypass daily limit	Data Validation	High
4	Absence of `chainID` validation allows signatures to be re-used across forks	Access Controls	High
5	Attacks can prevent the recovery of wallets that have a single guardian	Access Controls	High
6	Absence of Oracle restriction is dangerous	Access Controls	High
7	Absence of two-step procedure leaves critical operations error-prone	Data Validation	High
8	Absence of contract existence check in `baseWallet.invoke` may cause errors	Data Validation	High
9	A pending address whitelist request cannot be canceled	Access Controls	Low
10	Wallet with more than 510 guardians cannot relay transactions	Data Validation	Low
11	Mismatches between documentation and code	Undefined Behavior	Informational
12	`ApprovedTransfer` module is usable with no guardians	Data Validation	Informational
13	Whitelisting any spender allows the daily limit to be bypassed	Data Validation	High
14	`isERC721` can be abused to bypass the daily limit	Data Validation	High
15	Pending transactions do not take the daily limit into account	Data Validation	High

16	Users can avoid paying relayers	Data Validation	Medium
17	With no zero comparison for `ecrecover` anyone can perform privileged operations for wallet without owner	Cryptography	Informational
18	Failed transfers can decrease the daily limit	Data Validation	Informational
19	`init` functions can be called multiple times, leading to errors	Data Validation	Informational
20	Relayers can discard transactions with low gas limit	Data Validation	Informational
21	`RelayerModule` can call non-module contracts	Data Validation	Informational
22	Access controls are too permissive	Access Controls	Informational
23	Absence of legitimate wallet list may cause errors	Access Controls	Informational
24	No events for critical operations	Auditing and Logging	Informational
25	`RelayerModule` does not enforce the expected gas price	Data Validation	Informational
26	Wallet's owner can be a guardian	Access Controls	Informational

1. Absence of validation on exchange pairs allows users to bypass daily limit

Severity: High
Difficulty: High
Type: Data Validation
Finding ID: TOB-ARGENT-001
Target: `modules/TokenExchanger.sol`

Description
The `TokenExchanger` module assumes that the token being bought is legitimate and holds value. If an attacker creates a token with the sole intent of selling it to the wallet, this assumption is no longer valid.

`TokenExchanger` allows a user to swap one token for another using ParaSwap. No daily limit is enforced because it is assumed that only rational swaps will be made, i.e., the value of the outgoing token is roughly equal to the value of the incoming token. However, if an attacker creates a new token and lists it on a decentralized exchange such as Uniswap, purchasing that token will result in a net loss of value held by the wallet.

Exploit Scenario
Eve gains access to the owner key for Bob's Argent wallet. She wishes to steal all of the funds in the wallet, but there is a daily limit of 10 ETH. Eve deploys a new token and lists it on Uniswap, then uses the `TokenExchanger` module to exchange all of the wallet's tokens for her new malicious token. After the attack, the wallet only holds Eve's worthless token, and Eve recovers all of the wallet's previous funds by removing liquidity from the Uniswap exchange.

Recommendation
Short term, consider whitelisting tradeable tokens or enforcing the daily limit in `TokenExchanger`. An attacker can create a malicious token tradable in `TokenExchanger` to bypass the daily limit.

Long term, thoroughly evaluate and document all possible asset transfers of every module. Ensure each transfer is tested and its associated risks on the daily limit considered.

2. Uncapped slippage when swapping tokens allows users to bypass daily limit

Severity: High
Type: Data Validation
Target: `modules/TokenExchanger.sol`

Difficulty: High
Finding ID: TOB-ARGENT-002

Description
The `TokenExchanger` module assumes that the net change in value held by a wallet when performing an exchange is zero. However, slippage may cause the incoming value to be lower than the outgoing value. An attacker can intentionally exacerbate conditions and extract a profit by reclaiming the slippage.

When performing a large swap in one direction followed by the same swap in the opposite direction, only a small amount of fees will be paid. If done atomically, an attacker can significantly disrupt the instantaneous state of a reserve. If the attacker can trigger a wallet to purchase from the reserve between the two large swaps, the wallet will transfer significantly more funds due to slippage than under normal circumstances.

The additional transferred funds can make the wallet exceed its daily limit.

Exploit Scenario
Eve gains access to the owner key for Bob's Argent wallet. She wishes to steal all of the funds in the wallet, but there is a daily limit of 10 ETH. Eve deploys a contract that:

1. Uses flash loans to purchase a large amount of USDC with ETH so the price of USDC goes up.
2. Sends a meta-transaction to exchange all of the wallet's ETH for USDC.
3. Sells all of the USDC for ETH.

In step 2, the amount of USDC received by the wallet will be significantly lower due to the poor market conditions caused by step 1. In step 3, Eve recoups most of the ETH sent by the wallet.

Recommendation
Short term, to prevent slippage in `TokenExchanger`, either:

- Decrease the daily limit when assets are swapped in `TokenExchanger`, and document that swapping might decrease the daily limit, or
- Use Argent's Oracle to ensure that the swapped amounts are within an expected range.

Long term, thoroughly evaluate and document all possible asset transfers of every module. Ensure each transfer is tested and its associated risks on the daily limit considered.

3. Uncapped slippage when buying DAI allows users to bypass daily limit

Severity: High
Difficulty: High
Type: Data Validation
Finding ID: TOB-ARGENT-003
Target: `modules/maker/MakerV2Loan.sol`

Description
Repaying a debt with `MakerV2Loan` can lead to a swap that is abused to bypass the wallet daily limit.

When debt is removed from a Maker loan, the `MakerV2Loan` module will buy DAI from Uniswap if necessary. This is done by calculating the amount of ETH required to purchase the requisite amount of DAI, and performing the swap:

```
            // Not enough tokens => Convert some ETH into tokens with Uniswap
             uint256 etherValueOfTokens =
_uniswapExchange.getEthToTokenOutputPrice(_tokenAmountRequired - tokenBalance);
             invokeWallet(
                 _wallet,
                 address(_uniswapExchange),
                 etherValueOfTokens,
                 abi.encodeWithSignature("ethToTokenSwapOutput(uint256,uint256)",
_tokenAmountRequired - tokenBalance, block.timestamp)
             );
```

Figure 3.1: `modules/maker/MakerV2Loan.sol#L391-L398`.

This swap is vulnerable to a slippage-based attack, in which an attacker temporarily increases the purchase price of DAI before closing a loan, causing the wallet to spend more ETH than expected.

Exploit Scenario
Eve has gained access to the owner key for Bob's Argent wallet. She wishes to steal all of the funds in the wallet, but there is a daily limit of 10 ETH. Eve deploys a contract which :

1. Creates a CDP and sells the DAI.
2. Uses flash loans to purchase a large amount of DAI with ETH, increasing the price of DAI.
3. Sends a meta-transaction to close the CDP, triggering a swap from ETH to DAI.s
4. Sells all of the DAI for ETH.

In step 3, the amount of DAI received by the wallet will be significantly lower due to the poor market conditions caused by step 2. In step 4, Eve recoups most of the ETH sent by the wallet.

Recommendation

Short term, to prevent slippage in `MakerV2Loan`, either:

- Decrease the daily limit of the ETH swapped in Uniswap by `MakerV2Loan`, and document that repaying a debt might decrease the daily limit, or
- Remove the functionality that automatically performs a swap and require the user to have enough DAI on hand before closing a CDP

Long term, thoroughly evaluate and document all possible asset transfers of every module. Ensure each transfer is tested and its associated risks on the daily limit considered.

4. Absence of chainID validation allows signatures to be re-used across forks

Severity: High | Difficulty: High
Type: Access Controls | Finding ID: TOB-ARGENT-004
Target: `modules/RelayerModule.sol, infrastructure_0.5/MultiSigWallet.sol`

Description
The signatures used in `MultiSigWallet` and `RelayerModule` do not account for the chain's fork, so a valid signature can be reused in another fork.

Both `MultiSigWallet` and `RelayerModule` allow operations to be performed if they are signed by enough privileged users. Both use a signature schema that does not protect against signature reuse in multiple forks, so if there is a chain fork, the same signature will be usable on both forks.

```
bytes32 txHash = keccak256(abi.encodePacked(byte(0x19), byte(0), address(this), _to, _value,
_data, nonce));
```

Figure 4.1: `infrastructure_0.5/MultiSigWallet.sol#L83`.

```
        return keccak256(
             abi.encodePacked(
                 "\x19Ethereum Signed Message:\n32",
                 keccak256(abi.encodePacked(
                     byte(0x19),
                     byte(0),
                     _from,
                     _to,
                     _value,
                     _data,
                     _nonce,
                     _gasPrice,
                     _gasLimit,
                     _refundToken,
                     _refundAddress))
         ));
```

Figure 4.2: `modules/RelayerModule.sol#L190-L205`.

Exploit Scenario
Bob has a wallet with $1,000,000 worth of assets. A fork of mainet is performed with ETH 2.0, and the tokens on the old chains lose their value. Eve proposes that Bob buy all the tokens on the old chain. Bob calls `ApprovedTransfer` on the old chain with all the required signatures. Eve replays the signature on the new chain and steals all of Bob's funds.

Recommendation
Short term, use the `chainID` opcode in the signature schema to prevent signatures from being reused in case of forks.

Long term, document and carefully review any signature schema, including their robustness to reuse on different wallets, contracts, and blockchains. Make sure the users are aware of signing best practices and the danger of signing messages from untrusted sources.

5. Attacks can prevent the recovery of wallets that have a single guardian

Severity: High
Type: Access Controls
Target: `modules/RecoveryManager.sol`

Difficulty: High
Finding ID: TOB-ARGENT-005

Description
If a compromised wallet has only a single guardian, an attacker can prevent its recovery by canceling all ownership transfer requests.

If a wallet is compromised, the guardian can initiate a recovery by calling `executeRecovery`. Once the recovery request deadline has passed, it can be finalized with `finalizeRecovery`. A request can be canceled with `cancelRecovery`:

```
function cancelRecovery(address _wallet) external onlyWalletModule(_wallet)
onlyWhenRecovery(_wallet) {
     RecoveryConfig storage config = recoveryConfigs[address(_wallet)];
     address recoveryOwner = config.recovery;
     delete recoveryConfigs[_wallet];
     guardianStorage.setLock(_wallet, 0);

     emit RecoveryCanceled(_wallet, recoveryOwner);
 }
```

Figure 5.1: `modules/RecoveryManager.sol#L139-L146`

`cancelRecovery` requires `(number_of_guardian + 1)/ 2` signatures. The owner's signature is accepted:

```
if (methodId == CANCEL_RECOVERY_PREFIX) {
     uint numberOfSignaturesRequired =
Utils.ceil(recoveryConfigs[_wallet].guardianCount + 1, 2);
     return (numberOfSignaturesRequired, OwnerSignature.Optional);
 }
```

Figure 5.2: `modules/RecoveryManager.sol#L184-L187`

If there is only one guardian, only one signature will be required to cancel a request. Since this signature can be the owner's signature, an attacker wanting to compromise the wallet will be able to cancel any recovery request.

Exploit Scenario
Bob has a wallet with one guardian. Eve compromises Bob's wallet, so Bob's guardian starts a recovery, but Eve cancels the recovery and Bob is not able to recover his wallet.

Recommendation
Short term, if the wallet has one guardian, clarify the recovery process by either:

- Requiring `(number_of_guardian/ 2) + 1` signatures for `cancelRecovery`, or discard the owner's signature, or
- Updating the documentation and making it explicit that users need two guardians to be protected from theft.

An attacker compromising a wallet with one guardian will be able to prevent any recovery requests.

Long term, document all the modules' requirements and evaluate their robustness in case of wallet without guardian, with one guardian, or with multiple guardians. Each module's behavior in case of missing guardian or low number must be carefully considered

6. Absence of Oracle restriction is dangerous

Severity: High | Difficulty: High
Type: Access Controls | Finding ID: TOB-ARGENT-006
Target: `infrastructure/storage/TokenPriceStorage.sol`

Description
The daily transfer limit relies on a price Oracle that is controlled by the Wallet. The current implementation does not integrate any limit on price updates, which increases the severity of an Oracle compromise.

`setPriceForTokenList.setPrice` and `TokenPriceStorage.setPriceForTokenList` update the price of the tokens:

```
   function setPriceForTokenList(address[] calldata _tokens, uint256[] calldata
_prices) external override onlyManager {
        for (uint16 i = 0; i < _tokens.length; i++) {
            cachedPrices[_tokens[i]] = _prices[i];
        }
    }

    function setPrice(address _token, uint256 _price) external override onlyManager
{
        cachedPrices[_token] = _price;
    }
```

Figure 6.1: `infrastructure/storage/TokenPriceStorage.sol#L45-L53`.

These functions do not limit the values updated or the update frequency, so an attacker that compromises the Oracle's manager will gain significant control over the system.

A better design would split control over the price update in two:

- One set of functions with limited capability to update the price, callable by the Oracle's bot.
- One set of functions with full capability to update the price, callable by an account/multisig wallet that is safely stored.

Splitting the access controls reduces the risks and the impact of an Oracle compromise. Oracles have been the source of many recents hacks and must be carefully integrated (see References).

Exploit Scenario
Eve compromises the Oracle. Bob allows the relayer to execute its transactions in exchange for DAI. Eve changes the price of DAI to a very low value, executes Bob's transaction, and steals all of Bob's DAI.

Recommendation
Short term, split the Oracle price update into two sets of functions:

- Create `setPriceLimited` and `setPriceForTokenListLimited` to allow a limited update per day. These functions will be called by the Oracle's bot.
- Allow only a safety stored account or multisig wallet to call `setPrice` and `setPriceForTokenList`.

Splitting the right to update the price into two entities will decrease the overall risks associated with the Oracle.

Long term, carefully review the Oracle's bot setup and consider conducting a bot infrastructure security review.

References
- Synthetix Response to Oracle Incident
- bZx Hacked Again!
- Miners Trick Stablecoin Protocol PegNet, Turning $11 Into Almost $7M Hoard

7. Absence of two-step procedure leaves critical operations error-prone

Severity: High
Type: Data Validation
Target: `infrastructure/base/Owned.sol#L47-L51`

Difficulty: High
Finding ID: TOB-ARGENT-007

Description
`Owned.changeOwner` is a critical operation completed in one function call. This schema is error-prone and can lead to irrevocable mistakes.

`Owned.changeOwner` updates the contract's owner:

```
function changeOwner(address _newOwner) external onlyOwner {
     require(_newOwner != address(0), "Address must not be null");
     owner = _newOwner;
     emit OwnerChanged(_newOwner);
 }
```

Figure 7.1: `infrastructure/base/Owned.sol#L47-L51`.

The critical update is done in one function call, and will not be recoverable in case of a mistake.

Exploit Scenario
Eve alters the copy-and-paste feature on the Wallet mobile application. Bob wants to call `changeOwner`. Eve's alters the destination. Bob calls `changeOwner` with Eve's address, and Eve steals the wallet ownership.

Recommendation
Short term, use a two-step procedure for `Owned.changeOwner` and all irrecoverable critical operations. Using a one-step procedure for critical operations is error-prone

Long term, identify and document all possible actions and their associated risks for privileged accounts. Document and prepare adequate mitigations for everything that can happen when a wallet is compromised. Risks associated with privileged accounts must be understood and carefully reviewed

8. Absence of contract existence check in `baseWallet.invoke` may cause errors

Severity: High
Type: Data Validation
Target: `wallet/BaseWallet.sol`

Difficulty: High
Finding ID: TOB-ARGENT-008

Description
Without a check for a contract's existence in `baseWallet.invoke`, it may be incorrectly assumed that operations have been executed.

`baseWallet.invoke` calls the target with a low-level call:

```
   function invoke(address _target, uint _value, bytes calldata _data) external
moduleOnly returns (bytes memory _result) {
        bool success;
        (success, _result) = _target.call{value: _value}(_data);
        if (!success) {
            // solhint-disable-next-line no-inline-assembly
            assembly {
                returndatacopy(0, 0, returndatasize())
                revert(0, returndatasize())
            }
        }
```

Figure 8.1: `wallet/BaseWallet.sol#L117-L126`.

The Solidity documentation warns:

> *The low-level call, delegatecall, and callcode will return success if the calling account is non-existent, as part of the design of EVM. Existence must be checked prior to calling if desired.*

Figure 8.2: Solidity documentation.

As a result, if `invoke` is called on an incorrect contract destination or a destructed contract, the function will return `success`.

Exploit Scenario
Bob creates a smart contract that uses `TransferManager.callContract` to perform external operations. The contract incorrectly sets the destination of the external operations, so the call succeeds while no code is executed. As a result, Bob's smart contract reaches an incorrect state and is broken.

Recommendation

Short term, check the contract's existence before every low-level call in `baseWallet.invoke` if `_data` is not empty. Since there's no existence check for low-level calls, it may be incorrectly assumed that operations have been executed

Long term, avoid low-level calls since they lack the mitigations present in high-level calls. If they're not avoidable, carefully review the Solidity documentation, especially the Warnings section.

9. A pending address whitelist request cannot be canceled

Severity: Low
Type: Access Controls
Target: `modules/TransferManager.sol`

Difficulty: High
Finding ID: TOB-ARGENT-009

Description
Without pending address whitelist removal, the wallet can be spammed with whitelist requests. Every request gives an attacker a chance to drain the funds.

An address can be whitelisted in the `TransferManager`, which allows it to bypass the daily limit. However, there is a delay between the whitelist addition request and its completion. Presumably, this allows the owner to react if their private key is stolen and used to try to whitelist a malicious address. Unfortunately, there is no way for the owner to cancel the pending request, as only non-pending whitelisted addresses can be removed.

```
    function removeFromWhitelist(
        address _wallet,
        address _target
    )
        external
        onlyWalletOwnerOrModule(_wallet)
        onlyWhenUnlocked(_wallet)
    {
        require(isWhitelisted(_wallet, _target), "TT: target not whitelisted");
        transferStorage.setWhitelist(_wallet, _target, 0);
        emit RemovedFromWhitelist(_wallet, _target);
    }
```

Figure 9.1: `modules/TransferManager.sol#L337-L348`.

Exploit Scenario
Bob's private key is hacked by Eve. Both Bob and Eve now have access to the private key. Eve submits thousands of requests to add her address to the whitelist in order to bypass the daily limit. Bob must manually transfer all assets out of the wallet before the requests take effect. If he does not succeed, Bob will need to front-run every transfer from Eve to remove the addresses before any transfer succeeds.

Recommendation
Short term, allow owners to remove addresses that are pending but not yet whitelisted. This will prevent an attacker from compromising a key to add malicious addresses to the whitelisted accounts.

Long term, identify and document all possible actions and their associated risks for privileged accounts. Document and prepare adequate mitigations for everything that can happen when a wallet is compromised. Risks associated with privileged accounts must be understood and carefully reviewed.

10. Wallet with more than 510 guardians cannot relay transactions

Severity: Low
Type: Data Validation
Target: `modules/RelayerModule.sol`

Difficulty: High
Finding ID: TOB-ARGENT-010

Description
An integer overflow in `RelayerModule.validateSignatures` will prevent transactions from being relayed if the wallet has more than 510 guardians.

When a transaction is relayed, every signature has to be validated in `RelayerModule.validateSignatures`:

```
        for (uint8 i = 0; i < _signatures.length / 65; i++) {
            address signer = Utils.recoverSigner(_signHash, _signatures, i);
            [..]
            (isGuardian, guardians) = GuardianUtils.isGuardian(guardians,
signer);
            if (!isGuardian) {
                return false;
            }
        }
```

Figure 10.1: `modules/RelayerModule.sol#L279-L304`.

The number of signatures required depends on the module to be called, but it's generally about half the number of guardians.

The loop iterator `i` is a `uint8`. If the loop iterates over 255 values, the iterator `i` will overflow to zero, leading to an infinite loop. As a result, if the wallet has more than 510 guardians, the loop will never end and the transaction will revert due to insufficient gas.

Exploit Scenario
Bob creates a guardian service which provides hundreds of guardians per wallet. The guardians are located in different physical and logical locations, making it difficult to compromise all of the wallets. Alice, who has 600 guardians, uses Bob's service. As a result, she cannot use the gas-less transactions, and must remove Bob's service.

Recommendation
Short term, use a `uint256` for the loop iterator in `RelayerModule.validateSignatures`. Using a shorter type can lead to an infinite loop and prevent the function from being executed.

Long term, identify all loops iterating over a user-controlled value, and thoroughly test their iterations with large values. Do not use small type integers for loop iterators

11. Mismatches between documentation and code

Severity: Informational
Difficulty: Low
Type: Undefined Behavior
Finding ID: TOB-ARGENT-011
Target: `infrastructure/storage/TokenPriceStorage.sol`, `modules/RecoveryManager.sol`

Description
We observed two cases of discrepancies between documentation and code. In the first case, the documentation states that the token price is scaled by `10^(36-decimals)`, but in tests it's scaled by `10^(18-decimals)`:

```
* @notice Note that prices stored here = price per token * 10^(36-token decimals)
```

Figure 11.1: `infrastructure/storage/TokenPriceStorage.sol#L26`.

In the second case, the documentation claims that there's a delay before an ownership transfer can be triggered, but the code transfers ownership immediately:

```
/**
  * @notice Lets the owner start the execution of the ownership transfer procedure.
  * Once triggered the ownership transfer is pending for the security period before it can
be finalised.
  * @param _wallet The target wallet.
  * @param _newOwner The address to which ownership should be transferred.
  */
 function transferOwnership(address _wallet, address _newOwner) external
onlyWalletModule(_wallet) onlyWhenUnlocked(_wallet) {
     require(_newOwner != address(0), "RM: new owner address cannot be null");
     IWallet(_wallet).setOwner(_newOwner);

     emit OwnershipTransfered(_wallet, _newOwner);
```

Figure 11.2: `modules/RecoveryManager.sol#L148-L159`.

Recommendation
Short term, update the documentation in `infrastructure/storage/TokenPriceStorage.sol#L26` to reflect the correct token price scale. Also, either implement a delay for `RecoveryManager.transferOwnership`, or update its documentation. Out-of-date or incorrect documentation impedes code review and may allow issues to escape detection.

Long term, ensure the code is thoroughly tested to reflect the properties stated in the documentation. Any mismatches can lead to undefined behavior and impede code review.

12. `ApprovedTransfer` module is usable with no guardians

Severity: Informational
Type: Data Validation
Target: `modules/ApprovedTransfer.sol`

Difficulty: Undetermined
Finding ID: TOB-ARGENT-012

Description
`ApprovedTransfer` is expected to be used as a wallet with guardians. It has an undefined behavior if this condition is not met.

The `ApprovedTransfer` module is intended to be used by a wallet owner in conjunction with a majority of the wallet's guardians to perform a transfer that bypasses the daily limit. However, as shown in Figure 12.1, if the wallet has no guardians, the number of required signatures will be equal to 1. This means the owner can still use the module to bypass the daily limit.

```
        // owner  + [n/2] guardians
         uint numberOfSignatures = 1 + Utils.ceil(guardianStorage.guardianCount(_wallet), 2);
         return (numberOfSignatures, OwnerSignature.Required);
```

Figure 12.1: `modules/ApprovedTransfer.sol#L173-L175`.

Recommendation
Short term, revert in `getRequiredSignatures` if the wallet has no guardians. The concept of a majority does not apply in this case.

Long term, evaluate which modules need at least one guardian, and prevent their execution if this condition is not met. This will prevent unexpected behavior from modules if a wallet has no guardians.

13. Whitelisting any spender allows the daily limit to be bypassed

Severity: High
Type: Data Validation
Target: `contracts/modules/TransferManager.sol`

Difficulty: High
Finding ID: TOB-ARGENT-013

Description
By default, Argent wallets impose a daily transfer limit. This limit can disabled for the wallet or for certain recipients. However, disabling the daily limit for even one recipient effectively disables the daily limit for all recipients.

The `TransferManager` module allows for both transferring/approving tokens and transferring/approving tokens with an additional call. If the daily limit is enforced, the additional call cannot be to a token contract (as it may be a `transfer` function call).

```
function checkAndUpdateDailySpentIfNeeded(
    address _wallet,
    address _token,
    address _spender,
    uint256 _amount,
    address _contract
)
    internal
{
    if (!isWhitelisted(_wallet, _spender)) {
        // Make sure we don't call a supported ERC20 that's not whitelisted
        require(!coveredByDailyLimit(_wallet, _contract), "TM: Forbidden contract");
```

Figure 13.1: `contracts/modules/TransferManager.sol#L566-L577`.

However, notice that if the spender is whitelisted, the entire check is bypassed.

Exploit Scenario
Eve gains access to Bob's Argent wallet owner key. She wishes to steal all of the DAI in the wallet, but there is a daily limit of 10 ETH. Fortunately for her, Bob has whitelisted his close friend, Alice. Eve calls `approveTokenAndCallContract` with a spender of Alice, a target contract of DAI, and message data equal to `transfer(Eve, balanceOf(wallet))`. No daily limit is enforced because Alice's address is whitelisted, and Eve receives all of the DAI.

Recommendation
Short term, enforce that the target contract is not a spendable token even if the spender is whitelisted. With no check on the target contract, the daily limit can be bypassed.

Long term, consider conducting property testing with Echidna to ensure invariants are properly enforced. Property testing can detect access controls and state-related issues.

14. `isERC721` can be abused to bypass the daily limit

Severity: High
Type: Data Validation
Target: `modules/NftTransfer.sol`

Difficulty: High
Finding ID: TOB-ARGENT-014

Description
`NftTransfer` allows NFT tokens to be transferred. Incorrect logic on the NFT support allows the module to transfer ERC20 tokens.

`NftTransfer.transferNFT` checks if the destination is an ERC721 before calling `transferFrom`:

```
require(isERC721(_nftContract, _tokenId), "NT: Non-compliant NFT contract");
methodData = abi.encodeWithSignature(
      "transferFrom(address,address,uint256)", _wallet, _to, _tokenId);
```

Figure 14.1: `modules/NftTransfer.sol#L112-L114`.

```
function isERC721(address _nftContract, uint256 _tokenId) internal returns (bool) {
     (bool success, bytes memory result) =
_nftContract.call(abi.encodeWithSignature("supportsInterface(bytes4)",
0x80ac58cd));
     if (success && result[0] != 0x0)
         return true;

     (success, result) =
_nftContract.call(abi.encodeWithSignature("supportsInterface(bytes4)",
0x6466353c));
     if (success && result[0] != 0x0)
         return true;

     (success,) = _nftContract.call(abi.encodeWithSignature("ownerOf(uint256)",
_tokenId));
     return success;
```

Figure 14.2: `modules/NftTransfer.sol#L129-L139`.

`isERC721` checks if the target supports:

1. The interface `0x80ac58cd`,
2. The interface `0x6466353c`, and
3. A call to `ownerOf(uint256)`.

(3) is implemented with a low-level call. If the destination does not implement `ownerOf(uint256)`, but possesses a fallback function, the call will succeed. As a result, `isERC721` can return true for contracts that are not actually ERC721-compatible.

Exploit Scenario
Eve compromises Bob's wallet. Bob has all his assets in an ERC20 token that has the `supportsInterface` function and a fallback function. The token does not need any approval on `transferFrom` if `from` is the caller, and Eve uses `NftTransfer` to withdraw all the assets.

Recommendation
Short term, replace all low-level calls in `isERC721` with high-level calls. This will ensure that the function will return success only for `isERC721` contracts.

Long term, consider checking if the potential NFT contract is an ERC20 token for which the price is tracked. This will prevent the daily limit on ERC20 tokens from being abused by the NFT module. Carefully review the Solidity documentation, especially the Warnings section.

15. The daily limit does not take pending transactions into account

Severity: High
Type: Data Validation
Target: `modules/TransferManager.sol`

Difficulty: High
Finding ID: TOB-ARGENT-015

Description
Pending transactions are not taken into account in the daily limit, so an attacker can bypass the daily limit once the security period has expired. When the current daily limit is reached, new transactions are queued as pending transactions:

```
if (LimitUtils.checkAndUpdateDailySpent(limitStorage, _wallet, etherAmount)) {
    // transfer under the limit
    doTransfer(_wallet, _token, _to, _amount, _data);
} else {
    // transfer above the limit
    (bytes32 id, uint256 executeAfter) = addPendingAction(ActionType.Transfer,
_wallet, _token, _to, _amount, _data);
    emit PendingTransferCreated(_wallet, id, executeAfter, _token, _to, _amount,
_data);
```

Figure15.1: `modules/TransferManager.sol#L190-L196`.

Once the security period has expired, all pending transactions can be executed, and the new daily limit will not be checked:

```
   function executePendingTransfer(
        address _wallet,
        address _token,
        address _to,
        uint _amount,
        bytes calldata _data,
        uint _block
    )
        external
        onlyWhenUnlocked(_wallet)
    {
        bytes32 id = keccak256(abi.encodePacked(ActionType.Transfer, _token, _to,
_amount, _data, _block));
        uint executeAfter = configs[_wallet].pendingActions[id];
        require(executeAfter > 0, "TT: unknown pending transfer");
        uint executeBefore = executeAfter.add(securityWindow);

        require(executeAfter <= block.timestamp && block.timestamp <=
executeBefore, "TT: transfer outside of the execution window");
        delete configs[_wallet].pendingActions[id];
        doTransfer(_wallet, _token, _to, _amount, _data);
```

```
        emit PendingTransferExecuted(_wallet, id);
```

Figure 15.2: `modules/TransferManager.sol#L360-L379`.

As a result, if an attacker compromises a wallet, generates pending transactions, and executes them before the guardians react, he will be able to drain all the assets.

Exploit Scenario

- Bob has a wallet with 1,000 Ether, a daily limit of 1 Ether, and a security period of 24 hours. Eve compromises Bob's wallet—she knows Bob will be offline for 24 hours and won't be able to contact the wallet's guardians. Eve withdraws all the funds within 24 hours.
- Bob has a wallet with 1,000 Ether, a daily limit of 1 Ether, and a security period of 24 hours. Eve compromises Bob's mobile phone and begins a transfer of 1,000 Ether. Bob receives an SMS and an email to warn him about the transfer, but Eve deletes both messages before Bob can see them. Within 24 hours, Eve steals all of Bob's Ether.

Recommendation

Short term, update the daily limit in `TransferManager.executePendingTransfer`. Since there's no limit on pending transactions, an attacker can bypass the daily limit.

Long term, thoroughly document the impact of each module on the daily limit. Carefully review the external calls when adding new modules, and look for flaws that would allow the daily limit to be bypassed. We found many issues in modules that allowed the daily limit to be bypassed, so this aspect must be carefully reviewed when adding new modules

16. Users can avoid paying relayers

Severity: Medium
Type: Data Validation
Target: `modules/RelayerModule.sol`

Difficulty: Medium
Finding ID: TOB-ARGENT-016

Description
In the absence of ERC20 transfer success checks, relayers may not be paid. To pay a relayer, `RelayerModule.refund` calls `transfer` through `invokeWallet`:

```
bytes memory methodData = abi.encodeWithSignature("transfer(address,uint256)",
refundAddress, refundAmount);
    invokeWallet(_wallet, _refundToken, 0, methodData);
```

Figure 16.1: `modules/RelayerModule.sol#L349-L350`.

The value return by transfer is not checked. The ERC20 standard states:

> - Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

Figure 16.2: ERC20 standard.

As a result, the transfer can fail and the relayer is not paid.

Exploit Scenario
Bob is a relayer, and provides a bot that automatically accepts transactions to be relayed as long as they are paid with a token tracked by the Argent price Oracle and do not fail. BAT is accepted, but it returns false in case of insufficient balance and does not revert. Eve has no BAT. Bob relays Eve's transactions, and expects to receive BAT tokens as compensation, but receives none. The transactions mint GasToken. As a result, Eve mints GasToken and makes Bob pay.

Recommendation
Short term, check the return value of `transfer` in `RelayerModule.refund`. If the return value is not checked, relayers might not be paid for their work.

Long term,
- Review any call to `invokeWallet` and evaluate if the return value should be checked.
- Write a guideline for transactions relayer integration. The guideline should include recommendations on what to check before accepting to relay a transaction (gas limit, price, likelihood of transaction success, ..).

17. With no zero comparison for `ecrecover` anyone can perform privileged operations for wallet without owner

Severity: Informational
Type: Cryptography
Target: `modules/common/Utils.sol`

Difficulty: High
Finding ID: TOB-ARGENT-017

Description
Since there's no `ecrecover` return value check, anyone can perform owner operations if the wallet's owner is set to `0x0`.

`Utils.recoverSigner` uses `ecrecover` to retrieve the address associated with the public key that signs a message:

```
    function recoverSigner(bytes32 _signedHash, bytes memory _signatures, uint
_index) internal pure returns (address) {
        uint8 v;
        bytes32 r;
        bytes32 s;
        // we jump 32 (0x20) as the first slot of bytes contains the length
        // we jump 65 (0x41) per signature
        // for v we load 32 bytes ending with v (the first 31 come from s) then
apply a mask
        // solhint-disable-next-line no-inline-assembly
        assembly {
            r := mload(add(_signatures, add(0x20,mul(0x41,_index))))
            s := mload(add(_signatures, add(0x40,mul(0x41,_index))))
            v := and(mload(add(_signatures, add(0x41,mul(0x41,_index)))), 0xff)
        }
        require(v == 27 || v == 28);
        return ecrecover(_signedHash, v, r, s);
```

Figure 17.1: `modules/common/Utils.sol#L31-L45`.

`Utils.recoverSigner` is used to authenticate the owner of a wallet in `RelayerModule` and `TransferManager`. `ecrecover` returns `0x0` in case of error, so if the wallet's owner is zero, anyone can execute the owner's actions.

Additionally, `Utils.recoverSigner` does not prevent attacks using signature malleability, which allows creation of a second valid signature based on an existing one. However, the codebase does prevent signature re-use.

Exploit Scenario
Bob and Alice share a wallet. All the desired functions can be executed with the guardians. They decide to remove the owner privileges, and set owner to zero. Eve steals all the assets of the wallet by sending malformed signatures.

Recommendation
Short term, revert in Utils.recoverSigner if ecrecover returns zero. Additionally, consider preventing signature malleability by reverting if s is greater than 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0. This will prevent an attacker from abusing the ecrecover edge cases.

Long term, carefully review the Solidity documentation, in particular, the Warnings section.

18. Failed transfers can decrease the daily limit

Severity: Informational
Type: Data Validation
Target: `common/BaseTransfer.sol, baseModule.sol`

Difficulty: Medium
Finding ID: TOB-ARGENT-018

Description
Without ERC20 transfer success checks, the daily limit can decrease even if no transfers are done.

To transfer tokens, the modules call `invokeWallet` in `doTransfer`:

```
   function doTransfer(address _wallet, address _token, address _to, uint256
_value, bytes memory _data) internal {
        if (_token == ETH_TOKEN) {
            invokeWallet(_wallet, _to, _value, EMPTY_BYTES);
        } else {
            bytes memory methodData =
abi.encodeWithSignature("transfer(address,uint256)", _to, _value);
            invokeWallet(_wallet, _token, 0, methodData);
        }
        emit Transfer(_wallet, _token, _value, _to, _data);
```

Figure 18.1: `common/BaseTransfer.sol#L75-L82`.

```
   function invokeWallet(address _wallet, address _to, uint256 _value, bytes memory
_data) internal returns (bytes memory _res) {
        bool success;
        (success, _res) =
_wallet.call(abi.encodeWithSignature("invoke(address,uint256,bytes)", _to, _value,
_data));
        if (success && _res.length > 0) { //_res is empty if _wallet is an "old"
BaseWallet that can't return output values
            (_res) = abi.decode(_res, (bytes));
```

Figure 18.2: `common/BaseModule.sol#L159-L163`.

`doTransfer` does not check the return value of the transfer. The ERC20 standard states:

> - Callers MUST handle `false` from `returns (bool success)`. Callers MUST NOT assume that `false` is never returned!

Figure 18.3: ERC20 standard.

If a token returns false and does not revert in case of failure, the module will consider the transfer successful. As a result, the daily limit will decrease but no funds will be transferred. Several high-profile tokens do not revert in case of failure, including BAT or REP.

Exploit Scenario
Bob uses the wallet with BAT, whichreturns false in case of insufficient balance. Bob has 1,000 BAT, and initiates a transfer of 2,000 BAT. The transfer fails, but Bob's daily limit decreases.

Recommendation
Short term, check the return value of `transfer` and `transferFrom` in `BaseTransfer.` This will prevent failed transactions to decrease the daily limit.

Long term, carefully review any standard you interact with (including the ERC20 standard). Review any call to `invokeWallet` and evaluate whether the return value should be checked. Without a return value check, unexpected behavior can occur

19. `init` functions can be called multiple times, leading to errors

Severity: Informational
Type: Data Validation
Target: All modules

Difficulty: Undetermined
Finding ID: TOB-ARGENT-019

Description
All modules' `init` functions should be called only once by each wallet, but this is not enforced on-chain.

Modules have a `init` function that is called by the wallet when the module is added. For example, `TransferManager.init` will initiate the daily limit:

```
function init(address _wallet) public override(BaseModule) onlyWallet(_wallet) {

     // setup static calls
     IWallet(_wallet).enableStaticCall(address(this),
ERC1271_ISVALIDSIGNATURE_BYTES);
     IWallet(_wallet).enableStaticCall(address(this),
ERC1271_ISVALIDSIGNATURE_BYTES32);

     if (address(oldTransferManager) == address(0)) {
         limitStorage.setLimit(_wallet, ILimitStorage.Limit(defaultLimit, 0, 0));
```

Figure 19.1: `modules/TransferManager.sol#L111-L118`.

There is no on-chain mitigation to prevent the modules from being called multiple times. While it is not currently possible to call these functions directly from the wallet, future modules might enable the call, leading to unexpected behavior.

Exploit Scenario
Eve finds a way to call `TransferManager.init` multiple times. Eve compromises Bob's wallet, and uses `TransferManager.init` to bypass the daily limit.

Recommendation
Short term, create `BaseModule.init` to prevent a wallet from calling `init` multiple times, and instead allow it to call the functions in the derived classes. Since there's no built-in check to prevent the `init` functions from being called multiple times, unexpected behavior can occur.

Long term, when adding a module, create a checklist of review questions, including:

- What functions can the module call? This includes checking for function ID collision (see Slither's `function-id` printer and Appendix E) and calls to modules and wallets.
- Does the module handle assets?

- Does the module hold assets? If so `BaseModule.recoverToken` must be disabled.
- What are the functions that anyone can call? What is the impact of these functions?
- Are the operations at initialization thoroughly reviewed (including for re-entrancies)?
- Are the new static calls legitimate and their purposes documented?

20. Relayers can discard transactions with low gas limit

Severity: Informational
Type: Data Validation
Target: `modules/RelayerModule.sol`

Difficulty: Medium
Finding ID: TOB-ARGENT-020

Description
The 1/64 gas rule can allow malicious relayers to prevent the execution of the module's function. `RelayerModule.execute` calls a module on behalf of a wallet:

```
    function execute(
        [...]
        uint startGas = gasleft();
        require(startGas >= _gasLimit, "RM: not enough gas provided");
        [...]
        (stack.success, stack.returnData) = _module.call(_data);
        refund(
            _wallet,
            startGas,
            _gasPrice,
            _gasLimit,
            _refundToken,
            _refundAddress,
            stack.requiredSignatures,
            stack.ownerSignatureRequirement);
```

Figure 20.1: `modules/RelayerModule.sol#L90-L139`.

EIP 150 introduced the 1/64 gas rule: A called contract can only consume "all but one 64th" available gas.

As a result, `_module.call(_data)` can fail due to an out-of-gas issue, while `RelayerModule.execute` will still have enough gas to end the transaction. This is more likely to happen if `refund` returns directly, and does not consume a significant amount of gas:

```
    function refund(
        [...]
        // only refund when approved by owner and positive gas price
        if (_gasPrice == 0 || _ownerSignatureRequirement !=
OwnerSignature.Required) {
            return;
        }
```

Figure 20.2: `modules/RelayerModule.sol#L317-L332`.

The user can prevent exploitation by setting a high `_gasLimit`, but they may have to pay a high fee.

Exploit Scenario
Bob creates transactions to be relayed. These transactions will not be paid to the relayer, and Bob set `_gasLimit` and `_gasPrice` to zero. Eve relays the transactions, but provides a limited amount of gas, so the transaction succeeds while the module's call is not executed. As a result, Eve disturbs Bob's operations.

Recommendation
Short term, document that a transaction can be discarded due to the 1/64 gas rule to help ensure that users will increase the transaction's gas limit if they are targeted by an attack.

Long term, carefully review the gas rules of Ethereum. These rules are frequently changing and might impact the behavior of the transaction relayer.

21. `RelayerModule` can call non-module contracts

Severity: Informational
Type: Data Validation
Target: `modules/RelayerModule.sol`

Difficulty: High
Finding ID: TOB-ARGENT-021

Description
`RelayerModule` calls modules on the behalf of a wallet's owner. Without a strict module check, an attacker can call other contracts.

`RelayerModule.execute` checks that the wallet authorizes the module and calls:

- `_module.getRequiredSignatures(_wallet, _data)`.
- `_module.call(_data)`.

```
require(isAuthorisedModule(_wallet, _module), "RM: module not authorised");
[...]
(stack.requiredSignatures, stack.ownerSignatureRequirement) =
IModule(_module).getRequiredSignatures(_wallet, _data);
[...]
(stack.success, stack.returnData) = _module.call(_data);
```

Figure 21.1: `modules/RelayerModule.sol#L107-L130`.

The wallet authorizing the transaction could be a malicious contract that would authorise any contract. If a contract does not revert on `getRequiredSignatures`, an attacker can call this contract with any `_data` from the `RelayerModule`.

In particular, if a legitimate wallet authorizes `getRequiredSignatures` to be executed as a static call, an attacker could call `RelayerModule.execute` with a malicious wallet, and set `_module` to the wallet to attack. In this situation, the attacker could drain the wallet funds by calling transfer from the `RelayerModule`.

Exploit Scenario

- Eve creates a module that authorizes `getRequiredSignatures` as a static call.
- Argent validates the module.
- Bob adds the module.
- Eve exploits `RelayerModule` to drain the wallet.

Recommendation
Short term, check that the module is (or was) registered in `ModuleRegistry`. This will prevent an attacker from abusing `RelayerModule` to execute arbitrary contracts.

Long term, when adding a module, create a checklist of review questions, including:

- What functions can the module call? This includes checking for function ID collision (see Slither's `function-id` printer and Appendix E) and calls to modules and wallets.
- Does the module handle assets?
- Does the module hold assets? If so `BaseModule.recoverToken` must be disabled.
- What are the functions that anyone can call? What is the impact of these functions?
- Are the operations at initialization thoroughly reviewed (including for re-entrancies)?
- Are the new static calls legitimate and their purposes documented?

22. Access controls are too permissive

Severity: Informational
Type: Access Controls
Target: All modules

Difficulty: High
Finding ID: TOB-ARGENT-022

Description
Current module access controls rely on strong assumptions that the modules and the wallets will behave honestly.

All the modules rely on two modifiers:

- `onlyWalletOwnerOrModule`, which allows the wallet's owner or any registered module to call this module.
- `onlyWalletModule`, which allows any registered module to call this module.

The storage contracts rely on:

- `onlyModule`, which allows any registered module to call this module.

As a result, any module can call any other module, and there is no restriction on the ones that should be authorized. Overall, this schema is too permissive and can allow a malicious or compromised module to perform critical operations.

While we haven't found a direct vulnerability exploiting the current modules, several edge cases can be broken in the future.

Exploit Scenario
- Eve creates two malicious modules, `ModuleA` and `ModuleB`.
- `ModuleA` can call a module with a function for which the ID collides with `transferFrom`. `ModuleB` implements the function.
- Argent validates the module, and Bob adds it.
- Eve uses `ModuleA` to call the `TransferManager` module, and steals Bob's assets.

Recommendation
Short term, consider restricting module calls to authorized modules, perhaps by using DSAuth. For example, `TransferManager` should be callable by the wallet's owner or the `RelayerModule`.

Long term, when adding a module, create a checklist of review questions, including:

- What functions can the module call? This includes checking for function ID collision (see Slither's `function-id` printer and Appendix E) and calls to modules and wallets.

- Does the module handle assets?
- Does the module hold assets? If so `BaseModule.recoverToken` must be disabled.
- What are the functions that anyone can call? What is the impact of these functions?
- Are the operations at initialization thoroughly reviewed (including for re-entrancies)?
- Are the new static calls legitimate and their purposes documented?

23. Absence of legitimate wallet list may cause errors

Severity: Informational
Type: Access Controls
Target: modules/common/BaseModule.sol

Difficulty: High
Finding ID: TOB-ARGENT-023

Description
Without a legitimate wallet list, modules can't determine if the caller is a wallet created by the `WalletFactory`.

Several modifiers assume that the caller is a wallet:

```
   /**
     * @notice Throws if the sender is not the target wallet of the call.
     */
    modifier onlyWallet(address _wallet) {
        require(msg.sender == _wallet, "BM: caller must be wallet");
        _;
    }

    /**
     * @notice Throws if the sender is not the owner of the target wallet.
     */
    modifier onlyWalletOwner(address _wallet) {
        require(isOwner(_wallet, msg.sender), "BM: must be wallet owner");
        _;
    }

    /**
     * @notice Throws if the sender is not an authorised module of the target
wallet.
     */
    modifier onlyWalletModule(address _wallet) {
        require(isAuthorisedModule(_wallet, msg.sender), "BM: must be a wallet
module");
        _;
    }
```

Figure 23.1: `modules/common/BaseModule.sol#L54-L76`.

There is no check to ensure that the contract meant to be a wallet was created with the `WalletFactory`, so the wallet can be any arbitrary contract and partly mimic the expected behavior of a wallet.

Exploit Scenario
Bob has a gas-less transaction bot. The bot accepts any wallet. Eve abuses the bot with a fake wallet to mint gas tokens.

Recommendation
Short term, use `WalletFactory` to track which contracts were generated by the factory. Check that the wallet is present in `onlyWallet`, `onlyWalletOwner`, and `onlyWalletModule`. Begin with the list of known legitimate wallets. This will prevent attackers from calling the modules with malicious wallets

Long term, properly design mitigations to prevent malicious wallets or modules from calling the contracts. Access controls should distinguish legitimate calls from malicious ones

24. No events for critical operations

Severity: Informational
Type: Auditing and Logging
Target: All

Difficulty: Low
Finding ID: TOB-ARGENT-024

Description
Several critical operations do not trigger events, so it's difficult to review the correct behavior of the contracts once they're deployed.

Critical operations that would benefit from triggering events include:

- Migrating the daily limit.
- Disabling the daily limit.
- Acquiring a new CDP.

Additionally, while these operations emit events, the validity of an event is not guaranteed:

- Refunding a relayer: The refund may fail if the wallet's balance is insufficient.

Users and blockchain monitoring systems will not be able to easily detect suspicious behaviors without events.

Exploit Scenario
Bob is building an off-chain monitoring system and is unable to accurately and consistently track the state of the protocol due to missing or inaccurate events.

Recommendation
Short term, add events for all critical operations to monitor the contracts and detect suspicious behavior.

Long term, consider using a blockchain monitoring system to track any suspicious behavior in the contracts. The system relies on the correct behavior of several contracts, so a monitoring system that tracks critical events would quickly detect compromised system components.

25. `RelayerModule` does not enforce the expected gas price

Severity: Informational
Type: Data Validation
Target: `modules/RelayerModule.sol`

Difficulty: High
Finding ID: TOB-ARGENT-025

Description
Relayed transactions contain a gas price set by the transaction's sender. `RelayerModule` does not check that this price matches the actual gas price of the executed transaction, so relayers can set a lower gas price to earn an unexpected benefit.

`RelayerModule.execute` has a `_gasPrice` parameter:

```
/**
 [...]
 * @param _gasPrice The gas price to use for the gas refund.
 [...]
 */
 function execute(
 [...]
     uint256 _gasPrice,
 [...]
 )
```

Figure 25.1: `modules/RelayerModule.sol#L78-L100`.

`_gasPrice` is used to compute how many assets should be refunded to the relayer:

```
refundAmount = Utils.min(gasConsumed, _gasLimit).mul(_gasPrice);
```

Figure 25.2: `modules/RelayerModule.sol#L341`.

There is no on-chain check to ensure that `_gasPrice` actually matches the transaction's gas price. A malicious relayer could send the transaction with a lower gas price, which would delay the mining transaction while the relayer makes a profit.

Exploit Scenario
Bob wants to relay a transaction with a gas price of 1,000. Eve relays the transaction and sets a gas price of 10. As a result, the transaction is delayed, and Eve earns more in the refund than she spent.

Recommendation
Short term, check for `tx.gasprice >= _gasPrice` in `RelayerModule.execute` to prevent relayers from reducing the gas price.

Long term, change the refund algorithm to better manage an environment with adversarial relayers and consider creating economic incentives that encourage relayers to behave well

26. Wallet's owner can be a guardian

Severity: Informational
Type: Access Controls
Target: `modules/RelayerModule.sol`

Difficulty: High
Finding ID: TOB-ARGENT-026

Description
The wallet works on the assumption that its owner is not a guardian. This assumption can be broken if the owner is transferred to a guardian.

`GuardianManager.addGuardian` checks that the new guardian is not an owner:

```
require(!isOwner(_wallet, _guardian), "GM: target guardian cannot be owner");
```

Figure 26.1: `modules/GuardianManager.sol#L86`.

`BaseWallet.setOwner` does not have a similar check:

```
function setOwner(address _newOwner) external override moduleOnly {
     require(_newOwner != address(0), "BW: address cannot be null");
     owner = _newOwner;
     emit OwnerChanged(_newOwner);
 }
```

Figure 26.2: `wallet/BaseWallet.sol#L105-L109`.

As a result, it is possible for an owner to be a guardian if:

1. A guardian is set as an owner through `RecoveryManager.finalizeRecovery`.
2. A guardian is set as an owner through `RecoveryManager.transferOwnership`.

While (2) can be easily prevented, we do not recommend preventing (1) from happening, as it could allow an attacker to prevent the recovery of a wallet by making the recovered owner a guardian.

Recommendation
Short term,

- Prevent `RecoveryManager.transferOwnership` from being called if the owner is a guardian, so the wallet's owner is not a guardian invariant to be broken.
- Document that `RecoveryManager.finalizeRecovery` allows a guardian to be the owner. The wallet's owner is not a guardian invariant to be broken using `RecoveryManager.finalizeRecovery`.

Long term, thoroughly document the system's state transition, and highlight the owner and guardian update. The system relies on the correct state transition, so clear documentation will aid code review and prevent new issues.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking. or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for

	client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

- **Refrain from using integers smaller than 256 bits when possible:** When updating token prices, a `uint16` is used as the type for the loop variable. There's no performance benefit to using an integer that's smaller than 256 bits, so consider switching it to a `uint256`.
- **When data is spread across multiple input arrays, validate that the lengths are equal:** When token prices are updated, the token address and token price are located in two separate arrays of different lengths.
- **When inserting unique data into a list or map, validate that no duplicates exist:** The `GuardianStorage` allows duplicate guardians to be added and nonexistent guardians to be removed. In the future, ensure that no duplicates are added and only guardians that exist can be removed.
- **Remove redundant code when it no longer serves its purpose:** At the time of writing, CDPs can no longer be migrated from SCD to MCD, so the code to migrate CDPs can be removed from `MakerV2Loan`.
- **Identify tokens based on address, not symbol:** When repaying a borrow from Compound, the symbol of the `cToken` is compared with the literal "`cETH`" to determine whether the `cToken` is the `cEther` token. However, comparing token symbols can be inadequate since token symbols are not guaranteed to be unique.
- **Check the return data length before accessing return data:** When the owner of an non-EOA guardian is checked, the return data area of memory is read before checking whether sufficient data was returned. In the future, consider validating that the expected number of bytes was received first.

D. Token Integration Checklist

The following checklist provides recommendations for interacting with arbitrary tokens. Every unchecked item should be justified and its associated risks understood.

For convenience, all Slither utilities can be run directly on a token address, such as:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken
```

General security considerations

Ensure that:

- ❑ **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (aka "level of effort"), the reputation of the security firm, and the number and severity of the findings.
- ❑ **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on `blockchain-security-contacts`.
- ❑ **They have a security mailing list for critical announcements.** Their team should advise users (like you!) when critical issues are found or when upgrades occur.

ERC conformity

Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to confirm that:

- ❑ **`Transfer` and `transferFrom` return a boolean.** Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- ❑ **`The name, decimals, and symbol` functions are present if used.** These functions are optional in the ERC20 standard and might not be present.
- ❑ **`Decimals` returns a `uint8`.** Several tokens incorrectly return a `uint256`. If this is the case, ensure the value returned is below 255.
- ❑ **The token mitigates the known ERC20 race condition.** The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens.
- ❑ **The token is not an ERC777 token and has no external function call in `transfer` and `transferFrom`.** External calls in the transfer functions can lead to re-entrancies.

Slither's `slither-prop` utility generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to confirm that:

- ❑ **The contract passes all unit tests and security properties from `slither-prop`.** Run the generated unit tests, then check the properties with Echidna and Manticore.

Finally, there are certain characteristics that are difficult to identify automatically. Review these conditions manually:

- ❑ **`Transfer` and `transferFrom` should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- ❑ **Potential interest earned from the token must be taken into account.** Some tokens distribute interest to token holders. This interest might be trapped in the contract if not taken into account.

Contract composition

Ensure that:

- ❑ **The contract avoids unnecessary complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's `human-summary` printer to identify complex code.
- ❑ **The contract uses `SafeMath`.** Contracts that do not use `SafeMath` require a higher standard of review. Inspect the contract manually for `SafeMath` usage.
- ❑ **The contract has only a few non–token-related functions.** Non–token-related functions increase the likelihood of an issue in the contract. Use Slither's `contract-summary` printer to broadly review the code used in the contract.

Owner privileges

Ensure that:

- ❑ **The token is not upgradeable.** Upgradeable contracts might change their rules over time. Use Slither's `human-summary` printer to determine if the contract is upgradeable.
- ❑ **The owner has limited minting capabilities.** Malicious or compromised owners can abuse minting capabilities. Use Slither's `human-summary` printer to review minting capabilities, and consider manually reviewing the code.
- ❑ **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pauseable code manually.
- ❑ **The owner cannot blacklist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features manually.
- ❑ **The team behind the token is known and can be held responsible for abuse.** Contracts with anonymous development teams, or teams that reside in legal shelters, should require a higher standard of review.

Token scarcity

Issues of token scarcity require manual review. Check for these conditions:

- ❑ **No user owns most of the supply.** If a few users own most of the tokens, they can influence operations based on the tokens' repartition.
- ❑ **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- ❑ **The tokens are located in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange can compromise the contract relying on the token.
- ❑ **Users understand the associated risks of large funds or flash loans.** Contracts relying on the token balance must carefully take into consideration attackers with large funds or attacks via flash loans.

E. Entry Points Collision Detection with Slither

Trail of Bits created a script using Slither, a static analysis framework, to detect modules that share the same entry points. This script can be used to review the addition of modules and prevent incorrect access control.

Entry points collision

Modules can share the same entry point if:

- They have the same function.
- They have two functions that result in the same function ID.

If one module were to allow a call to an unexpected module, all access control of the wallet could be compromised (see TOB-ARGENT-022).

How to use the script

Currently, the wallet codebase is split into three compilation units. To run the script, first export each compilation unit to a `zip` using `crytic-compile`:

- `crytic-compile [target] [additional flags] --export-zip [name.zip]`
- Use the following `name.zip`:
  - `Infrastructure-0.5.zip`
  - `Infrastructure.zip`
  - `modules.zip`

```
from prettytable import PrettyTable
from slither import Slither
from slither.utils.function import get_function_id
from crytic_compile import compile_all

slithers = []
for compilation_unit in ["infrastructure-05.zip", "infrastructure.zip",
"modules.zip"]:
    compilation = compile_all(compilation_unit)
    assert len(compilation) == 1
    slithers.append(Slither(compilation[0]))

table = PrettyTable(["Contract,", "Signature", "ID", "File"])
explored = set()
id_saw = dict()

for sl in slithers:
    for contract in sl.contracts:
```

```
        for function in contract.functions_declared:
            if function.modifiers:
                if any((m.name in ['onlyWalletOwnerOrModule', 'onlyWalletModule']
for m in function.modifiers)):
                    if function.canonical_name in explored:
                        continue
                    explored.add(function.canonical_name)

                    sig = function.full_name

                    function_id = get_function_id(sig)
                    if function_id in id_saw and id_saw[function_id] !=
function.canonical_name:
                        print(f'Collision found {function.canonical_name}
{id_saw[function_id]}')

                    id_saw[function_id] = function.canonical_name
                    table.add_row([function.contract.name, sig, hex(function_id),
function.source_mapping_str])

print(table)
```

Figure E.1: Slither script.

The script will:

- Iterate over all the compilation units, and look for functions that use the `onlyWalletOwnerOrModule` or `onlyWalletModule` modifier.
- Print a warning if two functions have the same function ID.
- Print a table that summarizes all function IDs.

For example:

- `TransferManager` and `ApprovedTransfer` share:
    - `openLoan(address,address,uint256,address,uint256)`.
    - `approveTokenAndCallContract(address,address,address,uint256,address,bytes)`.
    - `approveWethAndCallContract(address,address,uint256,address,bytes)`.
    - `changeLimit(address,uint256)`.
- `MakerV2Loan` and `CompoundManager` share:
    - `openLoan(address,address,uint256,address,uint256)`.
    - `addCollateral(address,bytes32,address,uint256)`.
    - `removeCollateral(address,bytes32,address,uint256)`.

    - `addDebt(address,bytes32,address,uint256)`.
    - `removeDebt(address,bytes32,address,uint256)`.
    - `closeLoan(address,bytes32)`.

We recommend running the script when reviewing the addition of a module to:

- Confirm that the entry points shared are expected and documented.
- Confirm that calls can reach these entry points.

F. Fix Log

Argent addressed issues 1–25 in their codebase as a result of our assessment. Each of the fixes present in PR149 was verified by Trail of Bits on September 4, 2020.

#	Title	Type	Severity	Status
1	Absence of validation on exchange pairs allows users to bypass daily limit	Data Validation	High	Fixed
2	Uncapped slippage when swapping tokens allows users to bypass daily limit	Data Validation	High	Risks accepted
3	Uncapped slippage when buying DAI allows users to bypass daily limit	Data Validation	High	Fixed
4	Absence of `chainID` validation allows signatures to be re-used across forks	Access Controls	High	Fixed
5	Attacks can prevent the recovery of wallets that have a single guardian	Access Controls	High	Fixed
6	Absence of Oracle restriction is dangerous	Access Controls	High	Fixed
7	Absence of two-step procedure leaves critical operations error-prone	Data Validation	High	Risks accepted
8	Absence of contract existence check in `baseWallet.invoke` may cause errors	Data Validation	High	Risks accepted
9	A pending address whitelist request cannot be canceled	Access Controls	Low	Fixed
10	Wallet with more than 510 guardians cannot relay transactions	Data Validation	Low	Fixed
11	Mismatches between	Undefined	Informational	Fixed

	documentation and code	Behavior		
12	`ApprovedTransfer` module is usable with no guardians	Data Validation	Informational	Fixed
13	Whitelisting any spender allows the daily limit to be bypassed	Data Validation	High	Fixed
14	`isERC721` can be abused to bypass the daily limit	Data Validation	High	Partially fixed
15	Pending transactions do not take the daily limit into account	Data Validation	High	Risks accepted
16	Users can avoid paying relayers	Data Validation	Medium	Fixed
17	With no zero comparison for `ecrecover` anyone can perform privileged operations for wallet without owner	Cryptography	Informational	Fixed
18	Failed transfers can decrease the daily limit	Data Validation	Informational	Fixed
19	`init` functions can be called multiple times, leading to errors	Data Validation	Informational	Risks accepted
20	Relayers can discard transactions with low gas limit	Data Validation	Informational	Risks accepted
21	`RelayerModule` can call non-module contracts	Data Validation	Informational	Risks accepted
22	Access controls are too permissive	Access Controls	Informational	Risks accepted
23	Absence of legitimate wallet list may cause errors	Access Controls	Informational	Risks accepted
24	No events for critical operations	Auditing and Logging	Informational	Fixed
25	`RelayerModule` does not enforce the expected gas price	Data Validation	Informational	Risks accepted
26	Wallet's owner can be a guardian	Access Controls	Informational	Fixed

Detailed fix log

TOB-ARGENT-001: Absence of validation on exchange pairs allows users to bypass daily limit
Fixed. Only whitelisted tokens can be exchanged.

TOB-ARGENT-002: Uncapped slippage when swapping tokens allows users to bypass daily limit
Risks accepted. Argent stated:

Decreasing the daily limit when assets are swapped in TokenExchanger is not an option since not being constrained by the daily limit it is the main reason to have a separated module for swapping tokens. We also believe that using Argent's Oracle to ensure that the swapped amounts are within an expected range would not fix the problem since it can only limit the damage of an attack but cannot enforce it to be under the daily limit (e.g. capping at 200% will protect ~50% of the wallet's assets... but this may still be way above the daily limit set for that wallet). Instead we will document the exploit scenario and list it as a current limitation of our model.

TOB-ARGENT-005: Attacks can prevent the recovery of wallets that have a single guardian
Fixed.

On October 5th, the online documentation was updated to highlight the exploit scenario.

TOB-ARGENT-007: Absence of two-step procedure leaves critical operations error-prone
Risks accepted. Argent stated:

The current owner of all our infrastructure contracts is a 2-of-3 multisig which we believe is sufficient.

TOB-ARGENT-008: Absence of contract existence check in `baseWallet.invoke` may cause errors
Risks accepted. Argent stated:

Since the issue does not affect the current set of modules and the fix will add some extra gas cost to every operation of the wallet we have decided to not address that in the current version.

TOB-ARGENT-014: isERC721 can be abused to bypass the daily limit
Partly fixed.

`isERC721` was replaced with a check that the contract is not a known ERC20. For defense-in-depth, we recommend adding an additional check to ensure that the contract is not a registered module.

TOB-ARGENT-015: Pending transactions do not take into account the daily limit
Risks accepted. Argent stated:

This is not an issue but a current feature of our security model. We will update the specs to make it clearer.

TOB-ARGENT-017: With no zero comparison for `ecrecover` anyone can perform privileged operations for wallet without owner
Fixed. The return of `ecrecover` is checked against zero. However, we also recommend preventing signature malleability for defense-in-depth.

TOB-ARGENT-019: `init` functions can be called multiple times, leading to errors
Risks accepted. Argent stated:

Adding a flag to check that a module has been initialized for a wallet would add a lot of gas cost when creating or upgrading wallets (i.e. 1 storage slot per wallet per module in a naive implementation). Since it is not currently possible to call directly these init functions from the wallet we believe it is not worth the added gas cost.

TOB-ARGENT-020: Relayers can discard transactions with low gas limit
Risks accepted. Argent stated:

The check require(startGas >= _gasLimit, "RM: not enough gas provided") is there to prevent exactly the 1/64 gas rule exploit.

TOB-ARGENT-021: `RelayerModule` can call non-module contracts
Risks accepted. Argent stated:

This is not possible since creating a wallet or upgrading a wallet always checks that the modules are registered in the ModuleRegistry.

TOB-ARGENT-022: Access controls are too permissive
Risks accepted. Argent stated:

Addressing that would introduce an hardcoded dependency between modules which would make them impossible to upgrade individually. We believe the benefit is not worth the added complexity.

TOB-ARGENT-023: Absence of legitimate wallet list may cause errors
Risks accepted. Argent stated:

We are ok with that and have designed our code to not make any assumptions on the caller.

TOB-ARGENT-025: `RelayerModule` does not enforce the expected gas price
Risks accepted. Argent stated:

We are ok with that for now since we currently relay all the transactions of our users. We will revisit that once we integrate with a network of relayers in the future

TOB-ARGENT-026: Wallet's owner can be a guardian
Fixed. TOB-ARGENT-026 was reported after the initial report delivery, and was fixed in 151.