Dapper Labs

Dapper Smart Contract Wallet

Solidity Security Review

Version: 2.0

May, 2019

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Dapper Ethereum smart contract wallet. The smart contract wallet provides an authorisation mapping, enabling fine-grained and user-sovereign control over the wallet's funds and assets (e.g. Ether, ERC223 tokens, non-fungible tokens) and implements the following features:

- **Multi-signature support (two-of-two) with a co-signing check:** co-signing addresses can be other contracts (to potentially enforce additional verification);
- **Recovery operation:** a backup transaction that removes all existing authorisations and sets a new device key as the sole administrator.

This review focuses solely on the security aspects of the Solidity implementation of the Dapper Ethereum smart contract wallet, `CoreWallet`, also including additional general recommendations and comments relating to minimising gas usage.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project.

Document Structure

The first section of this report details an overview of the functionality of the `CoreWallet` contract contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an open/closed/resolved status and a recommendation. Additional findings, that do not have direct security implications, are marked as *informational* as they may potentially be of interest.

Outputs of the automated testing that were developed during this assessment are also included for reference in the Appendix (see Test Suite).

Finally, the Appendix provides additional documentation, including a table of gas estimates for various functions within the contract suite and the severity matrix used to classify vulnerabilities within the `CoreWallet` contract.

Overview

The `CoreWallet` contract can be used in two forms:

1. Standalone full wallet (by deploying the `FullWallet` contract);
2. Cloned wallet (by calling the `deployCloneWallet` function on a `WalletFactory` contract)

The `CoreWallet` smart contract inherits functionality from a range of standards and protocols to interact with, namely:

- `ERC721` **(Non-Fungible Token Standard) [1]** allowing the interaction of this wallet with NFTs through the `ERC721Received` interface;
- `ERC223` **(ERC223 Token Standard) [2]** describes a token standard to help prevent accidental sending of tokens to contracts, by implementing a `tokenFallback` function;
- `ERC165` **(Standard Interface Detection) [3]** creates a standard method to publish and detect the interfaces implemented by a smart contract;
- `ERC725_Core` **(Subset of `ERC725`, the Ethereum Identity Standard) [4]** provides key management functionality as a small subset of the `ERC725` standard to support Blockchain identities *(no longer supported in new contract version)*.
- `ERC1271` **(Standard Signature Validation Method) [5]** provides a standard way for contracts to validate signatures *(introduced in new contract version)*.

The `CoreWallet` provides support for *1-of-1* or *2-of-2* multi-signatures. A ***signer*** is an individual entity that signs invocations and interacts with the wallet. A signer must be *authorized* to invoke actions on the wallet. Optionally, the ***signer*** can also have a ***co-signer***, which then requires signatures from the pair of ***signer***, ***co-signer*** to perform the actions and invoke functions on the wallet.

A ***signer*** can be removed from the *authorized* users by setting its ***co-signer*** to zero.

The `CoreWallet` smart contract supports four different methods of interacting with the wallet:

1. `invoke0`: used when there are no ***co-signers***;
2. `invoke1SignerSends`: used by a ***signer*** to supply the ***co-signer***'s signature;
3. `invoke1CosignerSends`: used by a ***co-signer*** to supply the ***signer***'s signature;
4. `invoke2`: to be used to explicitly provide both the ***signer*** and ***co-signer*** signatures.

The `CoreWallet` also supports chaining calls, allowing authorised users to perform multiple operations, such as sending ETH, sending tokens (e.g. ERC20, ERC223, ERC721) or administering the wallet (e.g. change authorisations). Chaining calls reduces considerably the gas costs associated with each operation.

In the event that the wallet becomes inaccessible (e.g. due to lost keys), the wallet permits a *recovery address* to set a new authorized address and increments the authorized version, invalidating the authorization of other keys. This backup feature is implemented in the `emergencyRecovery()` function.

In the current design, the *recovery address* is intended to be the address associated with an ephemeral key, used only once to sign the related backup transaction. This key is generated on the users device (e.g. mobile application, web application, browser extension) and is meant to be erased from the device's memory immediately after signing the backup transaction.

To incentivise the clearing of space, the wallet also employs a `recoverGas()` function that allows for the public to receive a discount of gas when freeing up old authorized accounts that were cleared when the authorization version is incremented.

Security Assessment Summary

This review was initially conducted on commit 2d68897, which contains the contract for review, `CoreWallet.sol`.

Retesting activities targeted commit 6b3784e.

This contract is leveraged in `FullWallet.sol` and `CloneableWallet.sol`, to provide two types of working wallet instances, standalone and cloned. The cloned wallets are deployed using the `WalletFactory.sol` and `CloneFactory.sol` contracts.

Additionally, to support and comply with the related ERC standards, the `CoreWallet.sol` contract also inherits the following contracts:

- `ERC721Receivable`;
- `ERC223Receiver`;
- `ERC165Query`;
- `ERC725Core` *(No longer supported in new contract version)*;
- `ERC1271` *(Introduced in new contract version)*.

There are a number of additional contracts which lie outside the scope of this review. The complete list of contracts contained in the assessed repository at commit 2d68897 are:

```
.
├── Accessory
│   └── EscapeHatch.sol
├── Batching
│   └── Batcher.sol
├── ERC165
│   ├── ERC165Checker.sol
│   ├── ERC165Query.sol
│   └── ERC165.sol
├── ERC223
│   └── ERC223Receiver.sol
├── ERC721
│   ├── ERC721Receivable.sol
│   └── ERC721Receiver.sol
├── ERC725
│   ├── ERC725Core.sol
│   └── ERC725.sol
├── Migrations.sol
├── Ownership
│   ├── HasNoEther.sol
│   └── Ownable.sol
├── Test
│   ├── Debuggable.sol
│   ├── ERC721TokenMock.sol
│   ├── Selector.sol
│   ├── SimpleWallet.sol
│   ├── StandardTokenMock.sol
│   └── ThrowOnPayable.sol
├── Wallet
│   ├── CloneableWallet.sol
│   ├── CoreWallet.sol
```

```
    └── FullWallet.sol
└── WalletFactory
    ├── CloneFactory.sol
    └── WalletFactory.sol
```

The testing team identified a total of six (6) issues during this assessment, of which:

- One (1) is classified as high risk,
- One (1) is classified as medium risk,
- Four (4) are classified as informational.

All these issues have been addressed/acknowledged by the development team.

To support this review, the testing team used the following automated testing tools:

- Rattle: `https://github.com/trailofbits/rattle`
- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the `CoreWallet` contract. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contract, including potential gas optimisations, are also described in this section and are labelled as “informational”.

Each issue is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
DAP-01	Replay Attacks on Co-signer Signed Invocations	High	Resolved
DAP-02	Outdated ERC721 Implementation	Medium	Resolved
DAP-03	ERC721 Event Log Poisoning	Informational	Resolved
DAP-04	Potential Gas Optimisation Suggestions	Informational	Resolved
DAP-05	Miscellaneous General Comments	Informational	Resolved
DAP-06	Minor Typographical Errors in Comments	Informational	Resolved

DAP-01	Replay Attacks on Co-signer Signed Invocations		
Asset	CoreWallet.sol		
Status	**Resolved:** See Resolution.		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

Due to the co-signer `nonce` not being incremented upon signing, the signed messages by co-signers are able to be replayed if the co-signer is re-assigned as another co-signer, or later assigned as a signer (with or without another co-signer). The following scenario illustrates the ability to replay:

1. **Step 1: MsgA *(Send ETH to Trent)*** — `invoke2(signer=Alice, Co-Signer=Bob, from=Alice)` *(Here MsgA has been signed by Bob with a nonce of 1)*
2. **Step 2: *Bob gets authorized as a signer and his own cosigner*** — `signer=Bob, Co-Signer=Bob`
3. **Step 3: MsgA' *(Replay to send $ETH to Trent)*** — `invoke2(signer=Bob, Co-signer=Bob, from=Trent)`

Since `nonce` is a `public` state variable, an adversary is also able to predict when they are able to replay.

There are two valid exploitation scenarios:

1. The cosigner becomes a cosigner for another party;
2. The cosigner becomes a signer and cosigner for themselves.

Attack scenarios are provided in `tests/test_replay.py`.

Recommendations

There are two possible solutions to mitigate the replay attacks on the `CoreWallet` smart contract.

1. Integrate the nonce of the cosigner into the messages.
   - By utilising the nonce of both the cosigner and the signer, as well as incrementing accordingly, the cosigner's signed message will only be valid for the combination of `<signer, cosigner>` nonce pair. Once this has occurred, then `nonces[cosigner]++` will force the signed message to become invalid;
   - The drawback for this proposed solution is that the cosigner is then blocked from performing any other transactions that may increment the nonce. This means that a cosigner will be able to deny the transaction execution of any message they cosigned and messages will be required to come in order. (Nonces may be out of sync and messages will not get through).
2. Utilise a `message_nonce`.
   - By using a *message nonce*, you will only validate the signature on that wallet for that message;
   - The negative impact on this proposed method is that two competing messages from disjoint signers will be "conflicting" and only one message will come through successful. This creates a competition/race between messages.

Resolution

The development team fixed this vulnerability in commit 6b3784e by including the signing address as part of the signature data. This effectively ties the signature nonce to the signing address creating a unique signature for each signing address and signer nonce.

DAP-02	Outdated ERC721 Implementation		
Asset	ERC721Receivable.sol		
Status	**Resolved:** See Resolution.		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The Dapper Ethereum smart contract wallet (`CoreWallet` , deployed in its cloned and full versions) inherits the `ERC721Receiver` contract and as such, implements the `onERC721Received()` function.

This function in non-compliant with the `ERC721` standard as it does not take the right arguments as specified by the standard:

- **ERC721 Standard [1]:** `onERC721Received(address, address, uint256, bytes)` ;
- **Dapper implementation (ERC721Receiver):** `onERC721Received(address, uint256, bytes)` ;

This function is to be called by ERC721 contracts when a `safeTransferFrom` is made to a contract address. Typically, these contracts would implement a function which verifies that the recipient address, when a contract, is compliant with the `ERC721TokenReceiver` interface, expecting the `onERC721Received()` function of the Dapper contract to return `0x150b7a02` (equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`).

Since the Dapper wallet returns `0xf0b9e5ba` (equals to `bytes4 (keccak256("onERC721Received(address,uint256,bytes)"))`), any ERC721 transfer to a Dapper smart contract wallet would effectively fail.

This is illustrated in our test suite, refer to `tests/test_erc721.py` .

Recommendations

Change the `ERC721Receiver` and `ERC721Receivable` contracts to comply with the ERC721 standard. Specifically, change the `onERC721Received()` function to take an additional `address` (i.e. the address calling the `safeTransferFrom()`).

Resolution

The new version of the contract in commit 6b3784e now supports both the final ERC721 specification (`ERC721ReceiverFinal`), and the previous one, `ERC721ReceiverDraft` (used for example by the *CryptoKitties* contract).

DAP-03	ERC721 Event Log Poisoning
Asset	ERC721Receivable.sol
Status	**Resolved:** See Resolution.
Rating	Informational

Description

The Dapper Ethereum smart contract wallet (`CoreWallet` , deployed in its cloned and full versions) inherits the `ERC721Receiver` contract and therefore implements the `onERC721Received()` function, which when called emits the `ERC721Received` event.

This function can be externally called by any Ethereum address, resulting in `ERC721Received` events being arbitrarily generated.

Furthermore, the `_from` , `_tokenId` and `_data` event parameters can be forged to any arbitrary value, allowing attackers to potentially replicate and use existing asset IDs (e.g. valid *cryptokitties* token IDs), which could generate confusion for DApp users.

Please refer to our test suite for an example (see `tests/test_event_poisoning.py`).

The front-ends potentially consuming these events (e.g. mobile application, web application, browser extension) are outside the scope of this asssessment.

Recommendations

Traditionally, DApps rely on these event logs to perform certain actions or update UI (e.g. notify users that a particular NFT was received). Ensure this is as expected. Do not rely on the `ERC721Received` event in the front-ends (DApps) consuming the wallet logs.

Consider implementing the following `require` statement in the `onERC721Received()` :

```
require(msg.sender.doesContractImplementInterface(0x150b7a02));
```

This would ensure that only `ERC721` compliant contracts can call this function and trigger the `ERC721Received` event emission. Please note that this additional restriction can be bypassed by creating a malicious contract which complies with the `ERC721` interface and implements an `external` function (e.g. `generateLogInWallet` which calls `wallet.onERC721Received()`).

Another more restrictive approach could be to whitelist the contracts authorised to call the `onERC721Received()` function.

Resolution

The two `onERC721Received` functions no longer emit the `ERC721Received` event log in the updated version of the assessed contract at commit 6b3784e.

DAP-04	Potential Gas Optimisation Suggestions
Asset	CoreWallet.sol
Status	**Resolved:** See Resolution.
Rating	Informational

Description

This section suggests changes to the `CoreWallet` that would potentially have a positive impact on gas costs.

1. **Invoke1* Functions:** The `invoke1SignerSends` and `invoke1CosignerSends` allows respectively ***signers*** and ***co-signers*** to submit a signature along with the transaction data to be invoked. These functions are meant to be used when a ***signer*** is assigned a ***co-signer*** (i.e. when `requiredCosigner != signer`). However, it is possible for a ***signer*** which does not use a ***co-signer*** to use the invoke1 functions (instead of `invoke0`). When this happens, the invoke1SignerSends function unnecessarily performs a `keccak256` hash and an `ecrecover` to retrieve the ***co-signer*** address.
2. **Authorizations Mapping Change:** The `authorizations` mapping is a `uint256 => uint256` mapping. While it is necessary to map `uint256` variables (required for tracking `authVersion`), `authorizations` could be changed to a `uint256 => address` mapping. This will result in a significant gas cost decrease when `invoke2` is called, as it would no longer be required to cast `authorizations[authVersion + uint256(signer)]` into an address. Please note that implementing this suggestion would slightly increase the gas costs related to other functions. Refer to the appendix (Gas Costs) for a detailed gas cost analysis.
3. **Unnecessary `ERC165` Checker:** The `CoreWallet` contracts inherits `ERC165Query` contract, which implements the public `doesContractImplementInterface` function. This function can be used to verify if a contract supports a specific interface (as per the `ERC165` standard). This functionality is therefore shipped with every wallet instance (standalone full wallets and cloned wallets) and does not appear necessary for the purpose of this wallet. Removing this inheritance would save gas on wallet deployment.

Recommendations

Consider implementing the following recommendations to optimise gas costs:

1. Change the `invoke1*` functions to only verify signatures when necessary by modifying the `require` on line [371] and line [414] to an `if` statement similar to:

```
if(requiredCosigner != msg.sender) {
    // verify signature with ecrecover
}
```

2. Change the `authorizations` mapping to a `uint256 => address` mapping;
3. Remove the `ERC165Query` inheritance from the `CoreWallet` smart contract.

Resolution

Each point has been acknowledged by the authors. Action was taken for (3), which was resolved in commit 6b3784e.

DAP-05	Miscellaneous General Comments
Asset	CoreWallet.sol
Status	**Resolved:** See Resolution.
Rating	Informational

Description

This section describes general observations made by the testing team during this assessment.

1. **Reliance on client side security:** The security of the backup/recovery process of the Dapper Ethereum smart contract wallet is highly dependent on the users' devices. In the current design, the *recovery address* is intended to be the address associated with an ephemeral key, used only once to sign the related backup transaction. This key is generated on the users device (e.g. mobile application, web application, browser extension) and is meant to be erased from the device's memory immediately after signing the backup transaction. A compromised device (smartphone, workstation) could result in the ephemeral key being stolen when generated and used. Furthermore, successful tailored attacks targeting the front-ends using this smart contract could also result in the disclosure of this ephemeral key.
2. **Unused state variables in ERC725:** The `CLAIM_SIGNER_KEY` and `ENCRYPTION_KEY` are not used in any of the `CoreWallet` functions. The testing team assumes that no encryption operations related to the `ERC725` standard are expected to be performed by the wallet. Furthermore, the interface defined in the `ERC725/ERC725.sol` is not compliant with the actual `ERC725` standard as it does not specify the `execute` and `approve` functions.
3. **Public function can be made External:** The `keyHasPurpose` function is declared with a `public` visibility. Since there are no internal calls made to this function, it would be possible to change its declaration to be an `external` function. Please note that the testing team did not measure significant gas savings by doing so.
4. **Invoke1 and Invoke2 functions can be called by any address:** By design, the Dapper smart contract wallets allow anyone to submit transaction data and signatures. These transactions are adequately processed if the signature(s) match(es) the related authorizations.

Recommendations

Ensure these are as expected.

Resolution

Each point has been acknowledged by the authors. Action was taken for (2) and (3), which were resolved in commit 6b3784e (the ERC725 standard is no longer supported in the updated smart contract).

DAP-06	Minor Typographical Errors in Comments
Asset	CoreWallet.sol
Status	**Resolved:** See Resolution.
Rating	Informational

Description

This section highlights small and insignificant typos found as a bi-product of this review. This is included in an attempt to improve the overall code readability. Specifically, the following minor typographical errors were noticed in comments:

- line [149]: `recieves` → `receives`
- line [150]: *"This `method` is fired"* → *"This `event` is fired"*
- line [170]: `cosiging` → `cosigning`

Recommendations

Consider updating the related comments.

Resolution

The typographical errors listed above have been fixed in the related comments in commit 6b3784e.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `pytest` framework was used to perform these tests and the output is given below.

```
tests/test_deploy.py::test_deploy                                                PASSED  [1%]
tests/test_deploy.py::test_factory_deploys                                       PASSED  [3%]
tests/test_erc165.py::test_erc165_compliance                                     PASSED  [5%]
tests/test_erc20.py::test_erc20_token_transfer                                   PASSED  [7%]
tests/test_erc223.py::test_erc223_token_transfers                                PASSED  [9%]
tests/test_erc721.py::test_fail_erc721_token_transfer                            PASSED  [11%]
tests/test_fullwallet_basic.py::test_init                                        PASSED  [13%]
tests/test_fullwallet_basic.py::test_fallback_pay                                PASSED  [15%]
tests/test_fullwallet_basic.py::test_onlyInvoked_modifier                        PASSED  [17%]
tests/test_fullwallet_basic.py::test_emergencyRecovery                           PASSED  [19%]
tests/test_fullwallet_basic.py::test_invoke0_recovery                            PASSED  [21%]
tests/test_fullwallet_basic.py::test_invoke0_setAuthorized                       PASSED  [23%]
tests/test_fullwallet_basic.py::test_supports_interface[0xd202158d-True]         PASSED  [25%]
tests/test_fullwallet_basic.py::test_supports_interface[0xc0ee0b8a-True]         PASSED  [26%]
tests/test_fullwallet_basic.py::test_supports_interface[0x01ffc9a7-True]         PASSED  [28%]
tests/test_fullwallet_basic.py::test_supports_interface[0xffffffff-False]        PASSED  [30%]
tests/test_fullwallet_basic.py::test_supports_interface[0xdeadbeef-False]        PASSED  [32%]
tests/test_fullwallet_basic.py::test_supports_interface[0x00000000-False]        PASSED  [34%]
tests/test_fullwallet_basic.py::test_recoverGas                                  PASSED  [36%]
tests/test_fullwallet_basic.py::test_key_has_purpose                             PASSED  [38%]
tests/test_fullwallet_basic.py::test_invoke1_cosigner_sends                      PASSED  [40%]
tests/test_fullwallet_basic.py::test_invoke1_signer_sends                        PASSED  [42%]
tests/test_fullwallet_basic.py::test_invoke2                                     PASSED  [44%]
tests/test_gas_benchmark.py::test_gas_costs                                      PASSED  [46%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[0]                 PASSED  [48%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[1]                 PASSED  [50%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[2]                 PASSED  [51%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[3]                 PASSED  [53%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[4]                 PASSED  [55%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[5]                 PASSED  [57%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[6]                 PASSED  [59%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[7]                 PASSED  [61%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[8]                 PASSED  [63%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[9]                 PASSED  [65%]
tests/test_gas_cost.py::test_gas_saving_with_account_clearing[10]                PASSED  [67%]
tests/test_log_poisoning.py::test_event_poisoning                                PASSED  [69%]
tests/test_recovery.py::test_emergencyRecovery_multiple_times                    PASSED  [71%]
tests/test_recovery.py::test_all_events_omitted                                  PASSED  [73%]
tests/test_recovery.py::test_new_recovery_can_recover                            PASSED  [75%]
tests/test_recovery.py::test_bug_cosigner_to_0                                   PASSED  [76%]
tests/test_replay.py::test_replay_cosigner_different                             PASSED  [78%]
tests/test_replay.py::test_replay_cosigner_is_signer                             PASSED  [80%]
tests/test_replay.py::test_replay_money_send                                     PASSED  [82%]
tests/test_replay.py::test_can_deplete_wallet                                    PASSED  [84%]
tests/test_set_authorized.py::test_set_authorized_from_signer                    PASSED  [86%]
tests/test_set_authorized.py::test_set_authorized_from_cosigner                  PASSED  [88%]
tests/test_wallet_chaining.py::test_invoke0_can_chain                            PASSED  [90%]
tests/test_wallet_chaining.py::test_invoke1_can_chain                            PASSED  [92%]
tests/test_wallet_chaining.py::test_invoke2_can_chain                            PASSED  [94%]
tests/test_wallet_chaining.py::test_basic_chaining                               PASSED  [96%]
tests/test_wallet_chaining.py::test_chain_authorizing                            PASSED  [98%]
tests/test_wallet_chaining.py::test_chain_and_revert                             PASSED  [100%]
```

Another set of comprehensive tests is provided in the `test-scenarios` folder.

Appendix B Gas Costs

This section provides detailed gas cost information for both standalone full wallets and cloned wallets.

Standalone (Full) Wallet

	Existing Version (gas)	Authorizations mapping change (gas)	keyHasPurpose visibility change (gas)	ERC165 Query Removal (gas)
Deployment	2944888	2878276	2944888	2818672
invoke0 - setAuthorized	53341	53373	53341	53253
invoke0 - send ETH	34518	34540	34518	34474
invoke0 - transfer ERC20	57818	57840	57818	57774
invoke0 - transfer ERC223	59075	59097	59075	59031
invoke0 - transfer ERC721	121771	121793	121771	121727
invoke1 - setAuthorized	68303	68338	68303	68237
invoke1 - send ETH	49307	49332	49307	49285
invoke1 - transfer ERC20	57796	57821	57796	57774
invoke1 - transfer ERC223	59053	59078	59053	59031
invoke1 - transfer ERC721	121826	121851	121826	121804
invoke2 - setAuthorized	62224	62066	62224	62158
invoke2 - send ETH	58213	58219	58213	58191
invoke2 - transfer ERC20	66655	66661	66655	66633
invoke2 - transfer ERC223	67958	67964	67958	67936
invoke2 - transfer ERC721	115751	115757	115751	115729

Table 1: Standalone Full wallet gas costs for various code changes

Cloned Wallet

	Existing Version (gas)	Authorizations mapping change (gas)	keyHasPurpose visibility change (gas)	ERC165 Query Removal (gas)
Deployment	138674	138922	138674	138652
invoke0 - setAuthorized	54921	54953	54921	54833
invoke0 - send ETH	35311	35333	35311	35267
invoke0 - transfer ERC20	43623	43645	43623	43579
invoke0 - transfer ERC223	44880	44902	44880	44836
invoke0 - transfer ERC721	107646	107668	107646	107602
invoke1 - setAuthorized	69884	69919	69884	69818
invoke1 - send ETH	50037	50062	50037	50015
invoke1 - transfer ERC20	50037	50062	50037	50015
invoke1 - transfer ERC223	59876	59901	59876	59854
invoke1 - transfer ERC721	122783	122808	122783	122761
invoke2 - setAuthorized	63846	63862	63846	63780
invoke2 - send ETH	58792	58798	58792	58770
invoke2 - transfer ERC20	67420	67426	67420	67398
invoke2 - transfer ERC223	68805	68811	68805	68783
invoke2 - transfer ERC721	116688	116694	116688	116666

Table 2: Cloned wallet gas costs for various code changes

Appendix C Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

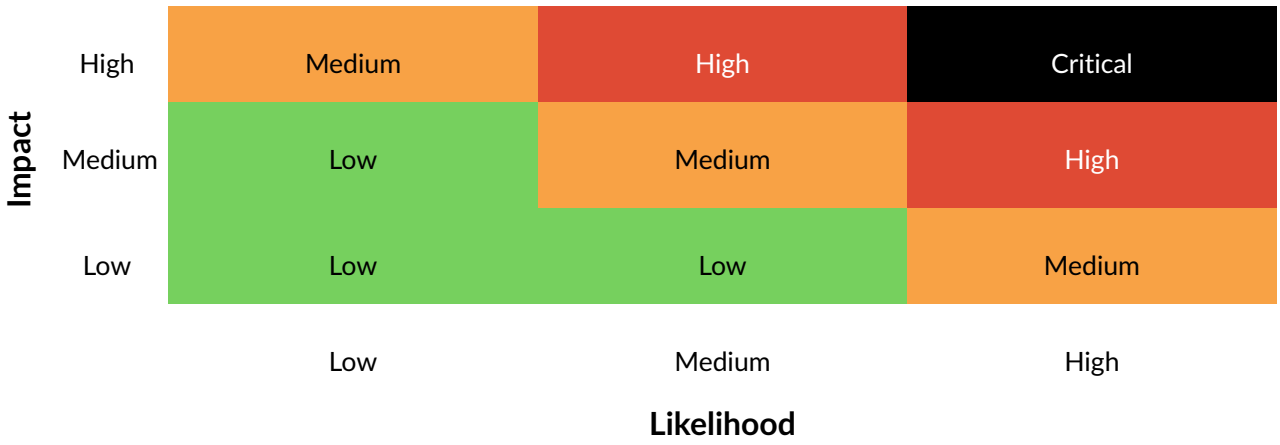

Table 3: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] ERC-721 Non-Fungible Token Standard. Github, Available: https://github.com/ethereum/EIPs/issues/721.

[2] ERC-223 Token Standard. Github, Available: https://github.com/ethereum/EIPs/issues/223.

[3] ERC-165 Standard Interface Detection. Github, Available: https://github.com/ethereum/EIPs/issues/881.

[4] ERC-725 Identity. Github, Available: https://github.com/ethereum/EIPs/issues/725.

[5] ERC-1271 Standard Signature Validation Method. Github, Available: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1271.md.