// HALBORN

C3 - PyTeal Updates

Algorand Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **October 10th, 2023 - October 27th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	10/27/2023
0.2	Draft Review	10/27/2023
1.0	Remediation Plan	11/16/2023
1.1	Remediation Plan Review	11/16/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

C3 engaged Halborn to conduct a security assessment on their PyTeal smart contracts beginning on October 10th, 2023 and ending on October 27th, 2023. The security assessment was scoped to the vesting module provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided around one month for the engagement and assigned two full-time security engineers to verify the security of the merge requests. The security engineers are blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this assessment is to:

- Ensure that the **smart contracts** operates as intended.
- Identify potential security issues with the PyTeal smart contract.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks that were accepted and addressed by the C3 team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the custom modules. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of structures and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the assessment :

- Research into architecture and purpose.
- Static Analysis of security for scoped repository, and imported functions. (e.g., semgrep)
- Manual Assessment for discovering security vulnerabilities on code-base.
- Ensuring correctness of the codebase.
- Dynamic Analysis on files and smart contracts related to the **C3**.

An assessment was conducted on the provided PyTeal code to identify any weaknesses, vulnerabilities, and non-compliance to Algorand best practices.

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

This review was scoped to the on the **C3 contracts-unified** repository.

IN-SCOPE REPOSITORY & COMMIT :

Halborn conducted a second follow-up assessment on the C3 protocol's new updates :

- 5564078483b2cee270822463f133a121b64cadae
- 1efaaaacc3887860e91878cb6c97f6b87d8e4d8a
- 6c4dff07036b985d96ef4fa7e1008c59e9bfb577
- 4693cb7aab8597b54e1c9fe431ff8870c24978e1
- 3f0eb69b116e36c489549156d591ea08b86e821f
- 6c5107f4628efd08f5b9f7d1e34c470e666dcc17
- 80be35fa1914cda98977cb26c889c94da25f5cf3
- 1da6ccec707ff7f7a82a6126c65c77fb3cc36f07
- ae47ddac4003db0c0ead036b2ddc1455dbea0825
- bf1f699b20decf859dca75edcaf9a2a0add15574

---

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	1	0

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SIGNATURE MISMATCH DUE TO PROGRAM UPDATE	Medium (6.9)	RISK ACCEPTED
(HAL-02) INSUFFICIENT VALIDATION OF PRICECASTER VALUES	Low (3.1)	SOLVED - 11/16/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) SIGNATURE MISMATCH DUE TO PROGRAM UPDATE - MEDIUM (6.9)

Description:

During a recent program update, there were modifications made to the `getDataToSign` function and the `MethodSignature` for `ADD_ORDER_SIG`. The prior signature mechanism, which was reliant on specific byte structures, has undergone changes. The method in question now includes an additional `Buffer.from("(C3.IO)0")` prefix and has seen changes in its parameter structure. Additionally, the `ADD_ORDER_SIG` has seen alterations in its parameter list.

The implications of these changes are significant for systems or integrations that rely on the previous signature format. Any system that has stored, or processes based on, the prior signature will find these signatures invalidated. This can disrupt transaction verifications, potentially leading to failed transactions or system inconsistencies where signature validation plays a pivotal role.

Code Location:

6c5107f4628efd08f5b9f7d1e34c470e666dcc17

Listing 1

```
- ADD_ORDER_SIG = MethodSignature("add_order(byte[],((address,byte[32],uint64),byte[],byte[],uint8,byte[],byte[],byte[]),((address,byte[32],uint64),byte[],byte[],uint8,byte[],byte[],byte[])[],uint64)void")
+ ADD_ORDER_SIG = MethodSignature("add_order(address,((address,byte[32],uint64),byte[],byte[],uint8,byte[],address,byte[]),((address,byte[32],uint64),byte[],byte[],uint8,byte[],address,byte[])[],uint64)void")
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:M/A:L/D:M/Y:N/R:N/S:U (6.9)

Recommendation:

When possible, maintain a layer of backward compatibility for a defined period. This can be achieved by supporting both the old and new signature methods and gradually phasing out the old method.

Remediation Plan:

RISK ACCEPTED: The C3 Team has formally acknowledged and accepted the risks associated with the identified issue. As part of this acceptance, the team commits to implementing strict measures to ensure that no signature remains intact during the system upgrades process. This approach is intended to mitigate any potential security vulnerabilities that may arise as a result of the identified issue.

4.2 (HAL-02) INSUFFICIENT VALIDATION OF PRICECASTER VALUES - LOW (3.1)

Description:

The code snippet reads a price entry from the price caster object and sets the normalized_price as the price. However, there is no validation to ensure that the retrieved price value is within expected bounds or even a valid number.

Code Location:

Listing 2

```
        constants.set(cast(abi.ReturnedValue, GlobalStateHandler.
↳ get_constants())),
        pricecaster.set(constants.pricecaster_id),
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:L/A:L/D:N/Y:N/R:N/S:U (3.1)

Recommendation:

Before using the pricecaster value in any calculations, validate it against expected constraints. This might include checks to ensure that the value is within a reasonable range, is not a negative number, is not NaN, etc.

Remediation Plan:

SOLVED: The C3 Team solved the issue by implementing need validations in the pricecaster codebase.

RECOMMENDATIONS OVERVIEW

RECOMMENDATIONS OVERVIEW

- The recent program update led to modifications in the signature structure, potentially affecting systems relying on the previous format. This could result in transaction verification issues and system inconsistencies.
- The current system retrieves a price value without ensuring its validity or ensuring it’s within expected bounds. This oversight could lead to potential miscalculations or system errors.

THANK YOU FOR CHOOSING

// HALBORN