FYEO

Security Assessment of the Cryptid

Identity Technologies Inc.

February 2023
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Strictly Confidential

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

Overview

Identity Technologies Inc. engaged FYEO Inc. to perform a Security Assessment of the Cryptid.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on January 19 - February 03, 2023, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues were identified during the testing period. Two issues remain open following a re-review:

- FYEO-CRY-01 – Attacker can extend a transaction they do not have the authority of
- FYEO-CRY-02 – Permissionless ApproveExecution instruction allows attackers to block another user's ExecuteTransaction instruction
- FYEO-CRY-03 – Attackers can block the execution of other user's transactions
- FYEO-CRY-04 – The inability to modify Cryptid properties may create security risk

Based on our review process, we conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

The FYEO Review Team performed a Security Assessment of the Cryptid. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a public repository at https://github.com/identity-com/cryptid with the commit hash 5b6a0630a41ba61acafa1593dd9ccc74e5cb5e95.

A re-review was carried out with the commit hash e30f06e980e3c4c6cd57121cd8389ba650a68595.

Files included in the code review

```
cryptid/
├── programs/
│   ├── cryptid/
│   │   ├── src/
│   │   │   ├── instructions/
│   │   │   │   ├── approve_execution.rs
│   │   │   │   ├── close_transaction.rs
│   │   │   │   ├── create_cryptid_account.rs
│   │   │   │   ├── direct_execute.rs
│   │   │   │   ├── execute_transaction.rs
│   │   │   │   ├── extend_transaction.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── propose_transaction.rs
│   │   │   │   ├── superuser_approve_execution.rs
│   │   │   │   └── util.rs
│   │   │   ├── state/
│   │   │   │   ├── abbreviated_account_meta.rs
│   │   │   │   ├── abbreviated_instruction_data.rs
│   │   │   │   ├── account_meta_props.rs
│   │   │   │   ├── cryptid_account.rs
│   │   │   │   ├── did_reference.rs
│   │   │   │   ├── instruction_size.rs
│   │   │   │   ├── mod.rs
│   │   │   │   ├── transaction_account.rs
│   │   │   │   └── transaction_state.rs
│   │   │   ├── util/
│   │   │   │   ├── cpi.rs
│   │   │   │   ├── mod.rs
│   │   │   │   └── seeder.rs
│   │   │   ├── error.rs
│   │   │   └── lib.rs
│   │   ├── Cargo.toml
│   │   └── Xargo.toml
│   └── middleware/
│       ├── check_pass/
│       │   ├── src/
│       │   │   └── lib.rs
│       │   ├── Cargo.toml
```

Files included in the code review
```
│         │    └── Xargo.toml
│         ├── check_recipient/
│         │    ├── src/
│         │    │    └── lib.rs
│         │    ├── Cargo.toml
│         │    └── Xargo.toml
│         └── time_delay/
│              ├── src/
│              │    └── lib.rs
│              ├── Cargo.toml
│              └── Xargo.toml
├── Cargo.lock
└── Cargo.toml
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of the Cryptid, we discovered:

- 1 finding with HIGH severity rating.
- 1 finding with MEDIUM severity rating.
- 1 finding with LOW severity rating.
- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

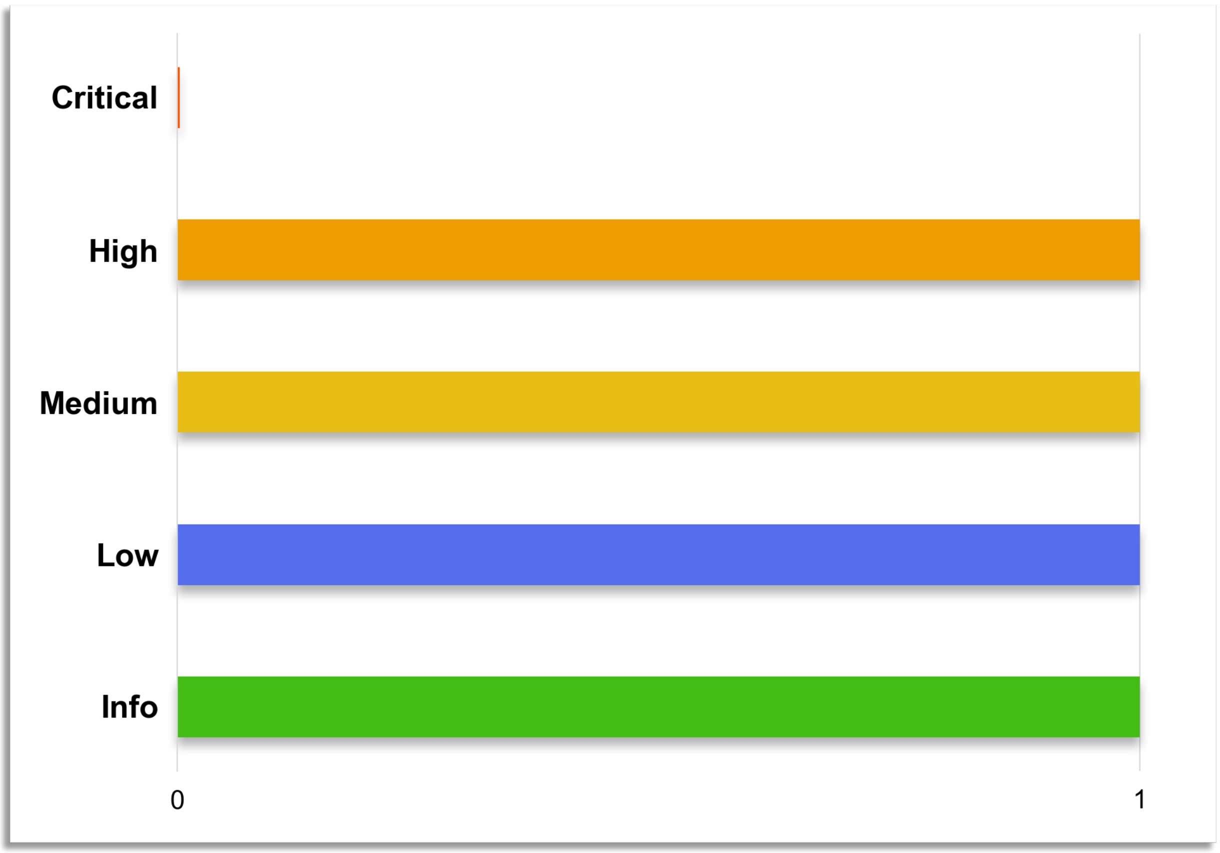

Figure 1: Findings by Severity

Findings

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-CRY-01	High	Attacker can extend a transaction they do not have the authority of
FYEO-CRY-02	Medium	Permissionless ApproveExecution instruction allows attackers to block another user's ExecuteTransaction instruction
FYEO-CRY-03	Low	Attackers can block the execution of other user's transactions
FYEO-CRY-04	Informational	The inability to modify Cryptid properties may create security risk

Table 2: Findings Overview

Technical Analysis

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

TECHNICAL FINDINGS

GENERAL OBSERVATIONS

General Strengths

The identity team was a pleasure to work with. They answered all questions the reviewer had and even spent sessions walking him through more complex areas of Cryptid.

During the review the following positive observations were made:

- It was great to see 141+ test cases attempting unique scenarios
- Extensive security checks for signer authorization
- Well-thought-out design and implementation
- Detailed comments which helped the reviewer understand areas of the code better
- Middleware components that add further constraints on the types of transactions that can be made

Summary Of Discovered Vulnerabilities.

The main theme of the discovered vulnerabilities included having an unauthorized ability to impact another user's transaction to a level that could bring a varied amount of damage to said user. The first vulnerability discovered essentially allows an attacker to block another user's ability to execute certain transactions. The second vulnerability discovered allows an attacker to extend an unexecuted transaction of another user.

ATTACKER CAN EXTEND A TRANSACTION THEY DO NOT HAVE THE AUTHORITY OF

Finding ID: FYEO-CRY-01
Severity: High
Status: Remediated

Description

The intended functionality of the ExtendTransaction instruction is to allow privileged user's (authorized signers and allowed unauthorized signers) the ability to extend a transaction. Under certain conditions, an attacker can successfully extend the transaction of another user.

Proof of Issue

File name: programs/cryptid/src/instructions/extend_transaction.rs

Line number: 140

An attacker can call the extend instruction and supply the allow_unauthorized argument with `true`

```
    allow_unauthorized: bool,
```

File name: programs/cryptid/src/instructions/extend_transaction.rs

Line number: 143

If the transaction account has an `unauthorized_signer` value of `None` the authority check will be skipped

```
if let Some(unauthorized_signer) =
ctx.accounts.transaction_account.unauthorized_signer {
        require_keys_eq!(
            ctx.accounts.authority.key(),
            unauthorized_signer,
            CryptidError::KeyMustBeSigner
        );
```

Severity and Impact Summary

Any Cryptid account with super-user middleware that creates a transaction account with no unauthorized signers is vulnerable to transaction account manipulation. This vulnerability is labelled high because an authorized Cryptid user can approve and execute a maliciously extended transaction that could cause severe damage.

Recommendation

The recommendation is to allow the user to pass allow_unauthorized as true if the transaction was already given an unauthorized signer during the propose transaction instruction.

Permissionless ApproveExecution instruction allows attackers to block another user's ExecuteTransaction instruction

Finding ID: FYEO-CRY-02
Severity: Medium
Status: Remediated

Description

A permissionless instruction is when anyone on the Solana network can execute a transaction that interacts with the user's programs. In some cases, permissionless instructions can be harmless depending on the logic of the program in question. In the case of Cryptid, the permissionless ApproveExecution instruction can have negative effects on the functionality of core operations, such as the ExecuteTransaction instruction.

Proof of Issue

File name: programs/cryptid/src/instructions/approve_execution.rs

Line number: 7

There are no constraints prohibiting an attacker from invoking the ApproveExecution instruction on a user's `transaction_account` causing the `approved_middleware` state to be `cleared`

```
pub struct ApproveExecution<'info> {
    pub middleware_account: Signer<'info>,
    #[account(
        mut,
        // ensure the transaction is not approved until it is ready to be
approved, and is not executed
        constraint = transaction_account.state == TransactionState::Ready @
CryptidError::InvalidTransactionState,
    )]
    pub transaction_account: Account<'info, TransactionAccount>,
}
```

File name: programs/cryptid/src/instructions/execute_transaction.rs

Line number: 165

When the authorized user attempts to validate the transaction that an attacker approved, this require statement may cause the transaction to revert since the `approved_middlware` may not be the same as the `cryptid_account.middleware`. This would result in a successful denial of the `ExecuteTransaction` instruction.

```
require!(
        ctx.accounts.transaction_account.approved_middleware ==
cryptid_account.middleware,
        CryptidError::IncorrectMiddleware
    );
```

Severity and Impact Summary

Since the ApproveExecution instruction is permissionless, an attacker can clear the middleware approval state of an already approved transaction of a user. This prevents the user from executing their transaction. Executing transactions are a core function of Cryptid accounts. This finding is labeled as Medium instead of High because there are alternate ways for a user to execute instructions without the need of the ApproveExecute instruction. They can use the DirectExecute instruction instead.

Recommendation

A potential recommendation is to ONLY allow the proposer or permitted unauthorized signers the ability to approve the transaction. An alternate mitigation we discussed would include the Cryptid account having an ordered list of expected middleware to verify against. The last way to mitigate would be to advise users to close the transaction in question and to perform DirectExecutes in the event another user is attempting to block their transactions.

ATTACKERS CAN BLOCK THE EXECUTION OF OTHER USER'S TRANSACTIONS

Finding ID: FYEO-CRY-03
Severity: Low
Status: Open

Description

Whitelisted middleware accounts can be created by an attacker to block the execution of other users transactions.

Proof of Issue

Filename: *check_recipient/lbib.rs*

Linenumber: 16

```
pub fn create(
        ctx: Context<Create>,
        recipient: Pubkey,
        bump: u8,
        previous_middleware: Option<Pubkey>,
    ) -> Result<()> {
        ctx.accounts.middleware_account.recipient = recipient;
        ctx.accounts.middleware_account.authority =
*ctx.accounts.authority.key;
        ctx.accounts.middleware_account.bump = bump;
        ctx.accounts.middleware_account.previous_middleware =
previous_middleware;
        Ok(())
    }
```

a) An attacker can create a whitelisted middleware

Filename: *instructions/approve_execution.rs*

Linenumber: 35

```
ctx.accounts.transaction_account.approved_middleware =
        Some(*ctx.accounts.middleware_account.key);
```

b) An attacker can approve the execution of another users transaction account

Filename: *instructions/execute_transaction.rs*

Linenumber: 165

```
  require!(
        ctx.accounts.transaction_account.approved_middleware ==
cryptid_account.middleware,
        CryptidError::IncorrectMiddleware
    );
```

c) When an authorized user attempts to execute the transaction, it will revert due to the `approve_middleware` being incorrect

Severity and Impact Summary

Depending on the middleware configurations, many transactions can be blocked from performing transaction executions.

Recommendation

The two potential fixes could involve:

- Tracking the whole chain of expected middleware accounts
- Middleware accounts are required to perform checks before a CPI call takes place. For example, it could check that approve is not called via CPI if it is to an unrelated CryptidAccount.

THE INABILITY TO MODIFY CRYPTID PROPERTIES MAY CREATE SECURITY RISK

Finding ID: FYEO-CRY-04
Severity: Informational
Status: Open

Description

Cryptid allows for super-user middleware accounts to approve transactions of unauthorized signers. When you create a Cryptid account, you specify the amount of super user middleware accounts you would like to have. After you create this Cryptid account, there is no way to edit or revoke these super-user middleware accounts.

Proof of Issue

File name: programs/cryptid/src/instructions/create_cryptid_account.rs

Line number: 40

When using the CreateCyrptidAccount instruction, the Cryptid creator can provision an arbitrary amount of super-user middleware accounts. There are no other instructions that permit the modification or revocation of a superuser.

```
pub fn create_cryptid_account(
    ctx: Context<CreateCryptidAccount>,
    middleware: Option<Pubkey>,
    superuser_middlewares: Vec<Pubkey>,
    controller_chain: Vec<Pubkey>,
    index: u32,
    did_account_bump: u8,
) -> Result<()> {
    require_gt!(index, 0, CryptidError::CreatingWithZeroIndex);
    ctx.accounts.cryptid_account.middleware = middleware;
    ctx.accounts.cryptid_account.index = index;
    ctx.accounts.cryptid_account.superuser_middleware = superuser_middlewares;
```

Severity and Impact Summary

The more superusers that are created without the ability to be modified, the higher the risk that one of the super user's could become a malicious actor. This could happen in the form of a key compromise or good actor turned bad. In both of these scenarios, there is no way to revoke the super-user account of the malicious actor from the Cryptid account.

Recommendation

If Cryptid is intended for users and organizations to allow others the ability to have elevated abilities via a super-user middleware account , the recommendation is to implement a way to “burn” and recreate the account in the event of a compromise.

OUR PROCESS

METHODOLOGY

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

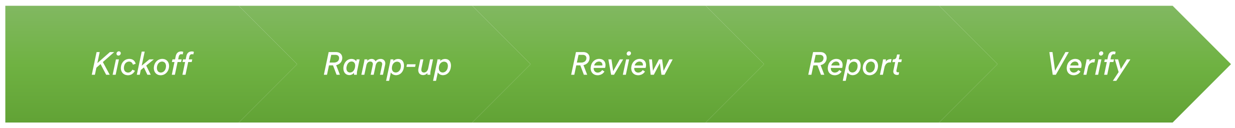

Figure 2: Methodology Flow

KICKOFF

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It’s an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

RAMP-UP

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren’t security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations`