F Y E O

Security Assessment of did:sol

Identity Technologies Inc.

January 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 did:sol.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on December 15 - January 06, 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 mostly remediated. FYEO-SOL-04 and FYEO-SOL-05 remain open as they are dependent on functional feedback from W3C

- FYEO-SOL-01 - DID lockout can occur by using a malicious update instruction
- FYEO-SOL-02 – In the event of a key compromise, an attacker can remove the recovery method of the original DID owner
- FYEO-SOL-03 – Missing check to ensure other controllers contain the proper DID syntax
- FYEO-SOL-04 – Controller functionality deviates from W3C specifications
- FYEO-SOL-05 – Preventing the DID authority from being their own native controller deviates from the W3C DID specifications

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 did:sol. 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/sol-did with the commit hash b4f2b71a8a600009379915ea49612f9dd82d2c59.

A re-review was carried out under the commit hash ef4045d2a9a7c65b3dae5c239d5058a0acd5fd5f

Files included in the code review

```
sol-did/
└── sol-did/
    ├── cpi/
    │   ├── src/
    │   │   └── lib.rs
    │   ├── Cargo.toml
    │   └── build.rs
    ├── programs/
    │   ├── example/
    │   │   ├── src/
    │   │   │   └── lib.rs
    │   │   ├── Cargo.toml
    │   │   └── Xargo.toml
    │   └── sol-did/
    │       ├── src/
    │       │   ├── instructions/
    │       │   │   ├── add_service.rs
    │       │   │   ├── add_verification_method.rs
    │       │   │   ├── close.rs
    │       │   │   ├── initialize.rs
    │       │   │   ├── migrate.rs
    │       │   │   ├── mod.rs
    │       │   │   ├── remove_service.rs
    │       │   │   ├── remove_verification_method.rs
    │       │   │   ├── resize.rs
    │       │   │   ├── set_controllers.rs
    │       │   │   ├── set_vm_flags.rs
    │       │   │   └── update.rs
    │       │   ├── integrations/
    │       │   │   ├── is_authority.rs
    │       │   │   └── mod.rs
    │       │   ├── legacy/
    │       │   │   ├── legacy_did_account.rs
    │       │   │   └── mod.rs
    │       │   ├── state/
    │       │   │   ├── did_account.rs
    │       │   │   └── mod.rs
    │       │   ├── constants.rs
```

Files included in the code review
│ │ ├── errors.rs │ │ ├── lib.rs │ │ ├── security_txt.rs │ │ └── utils.rs │ ├── Cargo.toml │ └── Xargo.toml ├── Cargo.lock └── Cargo.toml

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Security Assessment of the Sol:Did, we discovered:

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

The following chart displays the findings by severity.

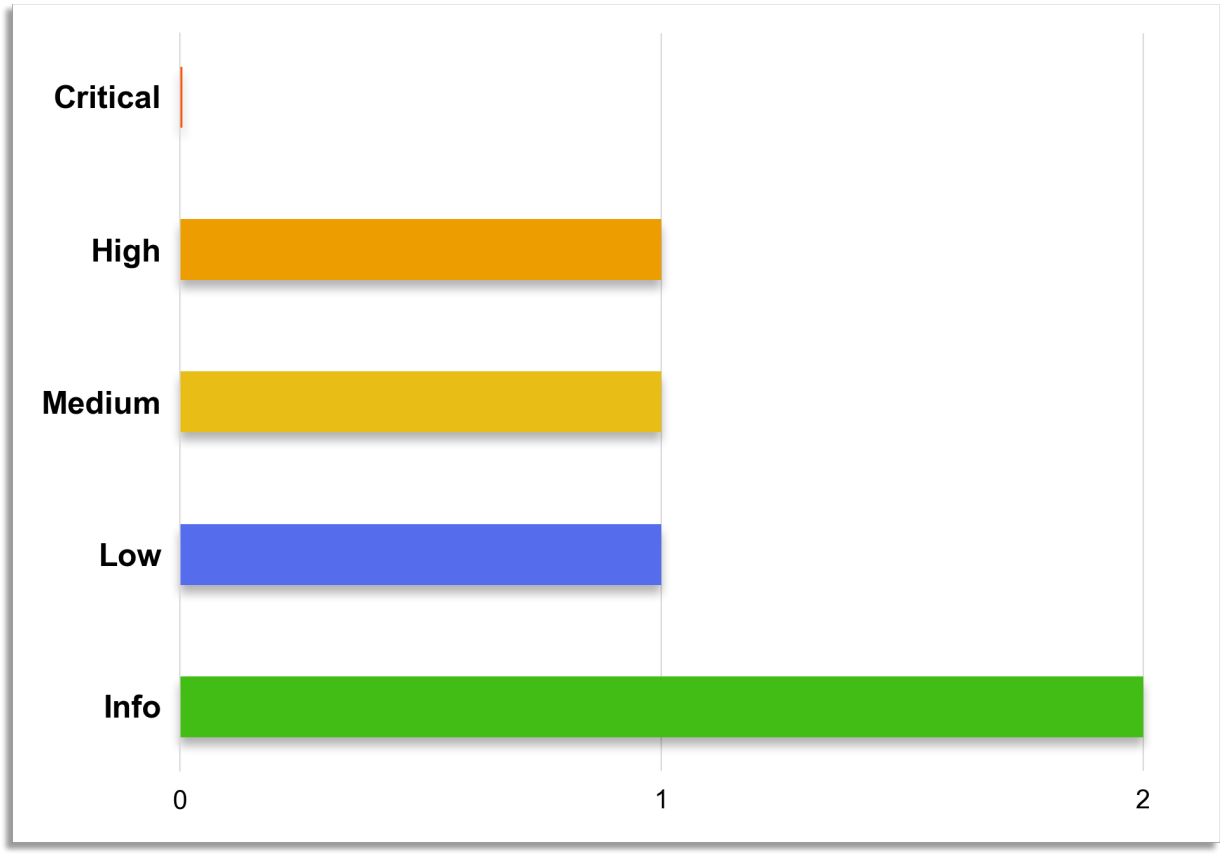

Figure 1: Findings by Severity

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-SOL-01	High	DID lockout can occur by using a malicious update instruction
FYEO-SOL-02	Medium	In the event of a key compromise, an attacker can remove the recovery method of the original DID owner
FYEO-SOL-03	Low	Missing check to ensure other controllers contain the proper DID syntax
FYEO-SOL-04	Informational	Controller functionality deviates from W3C specifications
FYEO-SOL-05	Informational	Preventing the DID authority from being their own native controller deviates from the W3C DID specifications

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

The Identity team had well-structured code with plenty of supporting documentation. The team was also very responsive whenever questions or concerns were raised throughout the engagement.

During the review, the following positive observations were made:

- Authorization checks for instructions with critical operations
- Checks for replay attacks when verifying signatures
- Validating function input (ie, the uniqueness for vectors/acceptable input)
- An extensive set of tests covering all areas of the codebase
- Assertion checks to prevent vulnerable edge cases

Summary Of Discovered Vulnerabilities

The primary areas of concern were centred around account lockout capabilities and DID recovery methods. A DID lockout occurs when all elevated permissions are removed, allowing no one the ability to modify the DID. Regarding, the DID recovery method a potential scenario can occur where a user's recovery method could be removed in the event of a key compromise. The remaining findings listed in the report discuss deviations from intended functionality and/or DID specifications.

DID LOCKOUT CAN OCCUR BY USING A MALICIOUS UPDATE INSTRUCTION

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

Description

The intended functionality for a DID includes having no ability to lock out the account by removing the last verification method. This requirement check can be found in remove_verifcation_method.rs and set_vm_flags.rs.

Proof of Issue

In the update instruction, there are no checks to ensure that the DID account data still contains an authority verification method.

```
    pub fn update(
    ctx: Context<Update>,
    update_arg: UpdateArg,
    eth_signature: Option<Secp256k1RawSignature>,
) -> Result<()> {
    // Move the business logic DidAccount struct.
    let data = &mut ctx.accounts.did_data;
    if eth_signature.is_some() {
        data.nonce += 1;
    }

    data.set_services(update_arg.services, false)?;
    data.set_verification_methods(Vec::new(),
update_arg.verification_methods)?;
    data.set_native_controllers(update_arg.native_controllers)?;
    data.set_other_controllers(update_arg.other_controllers)?;

    Ok(())
}
```

Severity and Impact Summary

Having the ability to lock out an account would prevent the DID account from operating in an intended manner. No user would be able to modify any methods of the DID since no permission verification method would exist.

Recommendation

Implement the did lockout require statement in the update function in update.rs or in the set_verification_method function in did_account.rs.

```
require!(
        data.has_authority_verification_methods(),
        DidSolError::VmCannotRemoveLastAuthority
    );
```

IN THE EVENT OF A KEY COMPROMISE, AN ATTACKER CAN REMOVE THE RECOVERY METHOD OF THE ORIGINAL DID OWNER

Finding ID: FYEO-SOL-02
Severity: **Medium**
Status: **Remediated**

Description

A user who has 2 wallets can become locked out of their DID if an attacker compromises one of the user's wallets and removes the user's recovery abilities.

Proof of Issue

See the following scenario between Bob's wallet X and wallet Y. Wallet X is used to primarily interact with the DID and wallet y is used as a recovery key for the DID. If the Bob's wallet X is compromised by an attacker, then the attacker with control of wallet X can remove wallet Y's ability to recover the DID.

This occurs when the attacker removes the verification method of wallet y.

```
pub fn remove_verification_method(
    ctx: Context<RemoveVerificationMethod>,
    fragment: String,
    eth_signature: Option<Secp256k1RawSignature>,
) -> Result<()> {
    let data = &mut ctx.accounts.did_data;
    if eth_signature.is_some() {
        data.nonce += 1;
    }

    let _ = data.remove_verification_method(&fragment);

    // prevent lockout
    require!(
        data.has_authority_verification_methods(),
        DidSolError::VmCannotRemoveLastAuthority
    );

    Ok(())
}
```

Severity and Impact Summary

Having non-recoverable options for a user's decentralized identity could result in a user losing their DID. This scenario could result in an attacker performing continued identity theft without the ability to be stopped.

Recommendation

The FYEO and Identity teams discussed two potential solutions. One of the proposed solutions we discussed was to modify the `capabilityInvocation` functionality and to require other verification keys to have fewer privileges. The second potential solution is to leverage a null recovery key that can never be removed. This would give the owner of a DID the ability to recover their DID in the event of a key compromise.

MISSING CHECK TO ENSURE OTHER CONTROLLERS CONTAIN THE PROPER DID SYNTAX

Finding ID: FYEO-SOL-03
Severity: Low
Status: Remediated

Description

`sol:did` controllers must follow the correct DID syntax(`"did:" method-name ":" method-specific-id`) in order to conform to DID specifications. In the set_other_controllers function there is no check to ensure that the controller passed in, and follows the correct DID specification.

Proof of Issue

In the set_other_controllers function in did_account.rs there is one require statement. This statement checks to ensure that other controllers don't have the `sol:did` prefix. There is no check to ensure a proper did prefix

```
 require!(
            check_other_controllers(&self.other_controllers),
            DidSolError::InvalidOtherControllers
        );
pub fn check_other_controllers(controllers: &[String]) -> bool {
    controllers.iter().all(|did| !is_did_sol_prefix(did))
}
```

Severity and Impact Summary

The syntax of a did:documents must be followed, in order for did documents to be resolved properly. DID documents that contain controllers with improper syntax will be rejected during this resolution process.

Recommendation

Implement a `require` statement to ensure that the passed-in controller starts with the proper DID syntax.

CONTROLLER FUNCTIONALITY DEVIATES FROM W3C SPECIFICATIONS

Finding ID: FYEO-SOL-04
Severity: **Informational**
Status: **Open**

Description

The W3C specs state that a DID controller, "has the capability to make changes to a DID document". If a DID is set as a controller and it doesn't have the proper permissions, it will not have the authority to modify the DID document.

Proof of Issue

The `find_authority` function below is used to validate when a key is authorized to make changes to a DID. There are no checks to provide authority for controllers because there is no direct relationship between verification methods and controllers.

```
    pub fn find_authority(
        &self,
        key: &[u8],
        filter_types: Option<&[VerificationMethodType]>,
        filter_fragment: Option<&String>,
    ) -> Option<&VerificationMethod> {
        // msg!("Checking if key {:?} is an authority", key,);
        self.verification_methods(
            filter_types,
            Some(VerificationMethodFlags::CAPABILITY_INVOCATION),
            Some(key),
            filter_fragment,
        )
        .into_iter()
        .next()
    }
```

Severity and Impact Summary

For other controllers, this requirement cannot be fully satisfied, due to the inability to validate other controllers on chain. Any native controllers that are added without given `capabilityInvocation` privileges won't have the ability to modify DIDs

Recommendation

Make updates to the sol:did specifications, detailing the restrictions of other controllers. Native controllers will need to include some "relationship to verification" methods in order to fulfil controller requirements.

PREVENTING THE DID AUTHORITY FROM BEING THEIR OWN NATIVE CONTROLLER DEVIATES FROM THE W3C DID SPECIFICATIONS

Finding ID: FYEO-SOL-05
Severity: **Informational**
Status: **Open**

Description

When setting a native controller, there is a required statement that prevents the initial DID authority from being its own DID controller. In the W3C DID specifications, the DID can be its own controller. This requirement in sol:did deviates from DID standards.

Proof of Issue

```
require!(
            !self.native_controllers.contains(&own_authority),
            DidSolError::InvalidNativeControllers,
        );
```

Severity and Impact Summary

While this issue doesn't create breaking changes to the functionality of DID operations, it deviates from core DID architecture.

Recommendation

Remove the set controllers require statement to allow for a DID authority to become its own controller.

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