F Y E O

Secure Code Review of VestigeSafe, TinyStake, & TinyStart

Vestige

November 2022
Version 1.1

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

Vestige engaged FYEO Inc. to perform a Secure Code Review of the VestigeSafe, TinyStake, & TinyStart contracts.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on September 26 - October 14, 2022, 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 have been identified during the testing period. These should be prioritized for remediation to reduce to the risk they pose:

- FYEO-VEST-01 – MaxLength reached preventing Deployment on Algorand
- FYEO-VEST-02 – API "Admin_claimSaleAmt" does not occur in the program
- FYEO-VEST-03 – API "Any_halt" does not occur in the program
- FYEO-VEST-04 – Mismatched types in Manager requirement
- FYEO-VEST-05 – Mismatched types in flexibleFunding assumption
- FYEO-VEST-06 – Mismatched types in Staking assumption
- FYEO-VEST-07 – Set Compiler to 'use strict'
- FYEO-VEST-08 – TODO: Implement additional Testing
- FYEO-VEST-09 – Trustworthy maps are not trustworthy on Algorand
- FYEO-VEST-10 – Use of `safe` Arithmetic functions

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 Secure Code Review of VestigeSafe, TinyStake, & TinyStart. 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 private repository at https://github.com/BunsanMuchi/VestigeSafe, https://github.com/vestigefi/TinyStake, and https://github.com/BunsanMuchi/TinyStart with the commit hashes c5c3401b51d0988008f779556672046818861e75, d9f2d1a137ee43b0aef32a6d048466b11d98c3fa, and aa2e3ab1b500d6dd94877a7cee94a1919c48b85a respectively.

While the Vestige team worked throughout the engagement to remediate findings, a re-review was performed on November 3, 2022, of the TinyStart repository with the commit hash 4d25590e730687d18e51a813f1ea21195a4b4db8.

Files included in the code review

vestigesafe/
├── index.mjs
└── index.rsh
 	
tinystake/
└── reach
    ├── index.rsh
    └──index.mjs
 	
tinystart/
├── index.mjs
└── index.rsh

Table 1: Scope

Technical Analyses and Findings

During the Secure Code Review of VestigeSafe, TinyStake, & TinyStart, we discovered:

- 1 finding with HIGH severity rating.
- 5 findings with LOW severity rating.
- 4 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

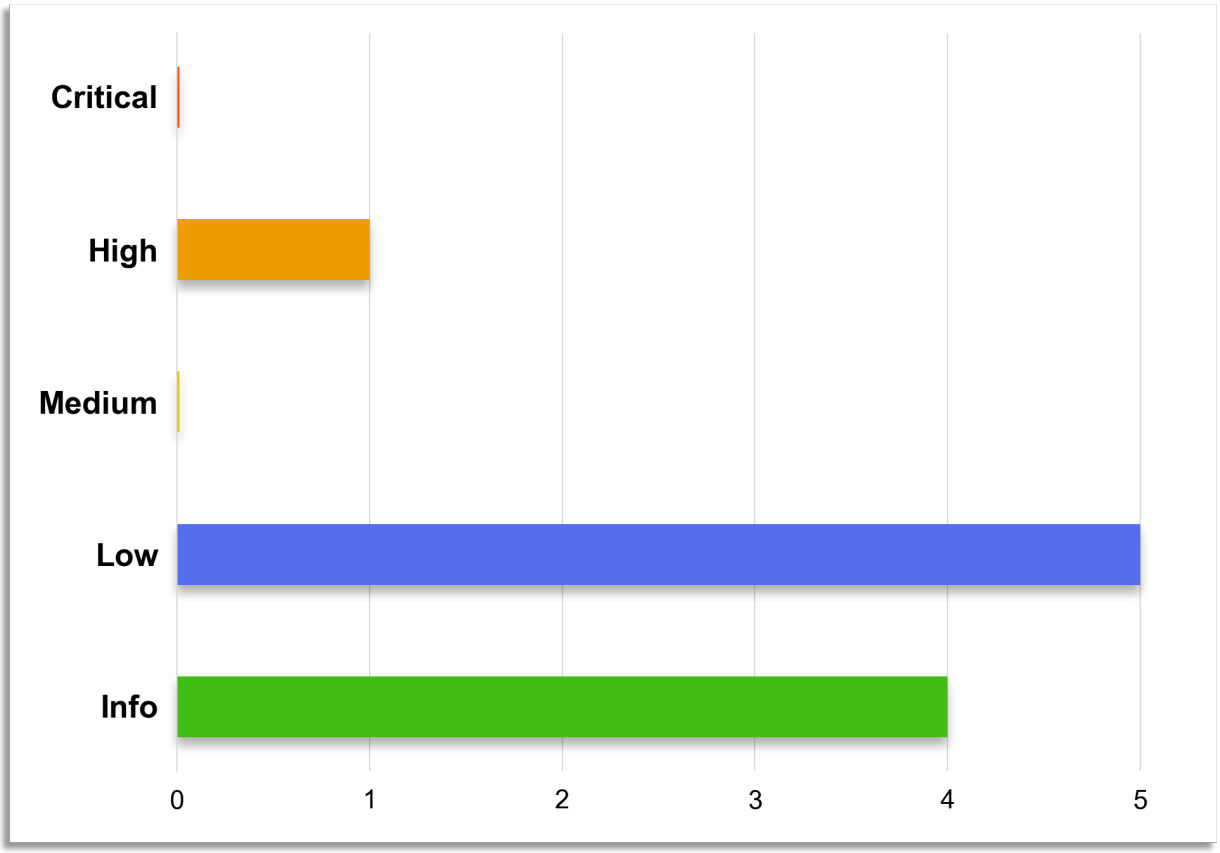

Figure 1: Findings by Severity

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-VEST-01	High	MaxLength reached preventing Deployment on Algorand
FYEO-VEST-02	Low	API "Admin_claimSaleAmt" does not occur in program
FYEO-VEST-03	Low	API "Any_halt" does not occur in the program
FYEO-VEST-04	Low	Mismatched types in Manager requirement
FYEO-VEST-05	Low	Mismatched types in flexibleFunding assumption
FYEO-VEST-06	Low	Mistmatched types in Staking assumption
FYEO-VEST-07	Informational	Set Compiler to 'use strict'
FYEO-VEST-08	Informational	TODO: Implement additional Testing
FYEO-VEST-09	Informational	Trustworthy maps are not trustworthy on Algorand
FYEO-VEST-10	Informational	Use of `safe` Arithmetic functions

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

During the code assessment of Vestige (VestigeSafe, TinyStake, TinyStart), it was observed that the Reach code was well-written and structured in a generally logical way. The development team was very good at explaining and engaging in discussions There was limited formal documentation provided during the assessment, but code comments and active developer interaction added sufficient insight that those documents would have provided. The code seems to be quickly evolving as it is in a varying state of maturity. Due to the advantageous situation of both the development team and auditing partner, daily meetings were conducted throughout the engagement. In daily meetings, many questions relating to the codebase were discussed in depth and answered directly which facilitated direct contributions and improvements by the FYEO Security Team to the code in real-time.

The assessment covered three different iterations of code state (commit hashes), which generally is not the preferred methodology, but was necessary for both code maturity as well as deployment functionality. The use of verify arithmetic and/or “safe” arithmetic operations were recommended and subsequently implemented across the program where necessary. It was also recommended that, even though not necessary in deployment, to utilize the ‘use strict’ functionality of the compiler to showcase any unexpected or odd behavior. The strict compilation through all three iterations resulted in the identification of subtle type errors, but the main insight came from the lack of additional errors, which reflects positively on the final iteration of the program code. During this process, intermediate artifacts and CFGs were constructed to facilitate lower-level analysis as well as developer visibility.

While reviewing the second iteration (second commit) of the program, it was discovered that the program was too large to be deployed to the Algorand blockchain. Once this was discovered, it was agreed upon to make the necessary changes to the system so that the program met the size and computational constraints of the Algorand network.

Once this new code was implemented (third commit), which made the deployment now possible, the code review concentrated on analyzing lower-level functionality that was precluded in earlier iterations. During this final iteration, more robust testing was conducted, but it was noted that boilerplate testing code was present.

Overall, the security, robustness, and functionality of the Vestige program was iteratively improved and follows most programming best practices.

MaxLength reached preventing Deployment on Algorand

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

Description

Algorand connector was selected, which instructed the compiler to emit for deployment to the Algorand blockchain, but it was confirmed that the program's length was too large to be properly deployed. This issue was not previously known at the start of the engagement due to the use of compiler flags in the development phase.

Proof of Issue

```
WARNING: Compiler instructed to emit for Algorand, but we can statically determine that this program will not work on Algorand, because:
 * The program is too long; its length is 12961, but the maximum possible length is 8192
```

Severity and Impact Summary

The issue is quite severe as compilation and contract deployment are not possible without an architectural restructuring or code rewrite.

Recommendation

Once the issue was discovered, the development team and security partner discussed how the program could be reconstructed to fit within the maximum possible length. A solution was agreed upon to split up the funding cases into different parts so that the maximum possible length is never reached. The development team quickly introduced the fix and progress was able to continue throughout the remainder of the engagement.

References

https://developer.algorand.org/docs/get-details/parameter_tables/

API "ADMIN_CLAIMSALEAMT" DOES NOT OCCUR IN PROGRAM

Finding ID: FYEO-VEST-02
Severity: Low
Status: Remediated

Description

API was defined but is not used. This could pose issues to future functionality and perhaps allow the program to execute in unexpected ways.

Proof of Issue

File name: index.rsh
Line number: 156

```
Compiling "main"...
Verifying knowledge assertions
Verifying for generic connector
  Verifying when ALL participants are honest
  Verifying when NO participants are honest
Checked 306 theorems; No failures!
reachc: error[RAPI0000]: API "Admin_claimSaleAmt" does not occur in program

  ./index.rsh:156:30:application

  156| export const main = Reach.App(() => {

For further explanation of this error, see:
https://docs.reach.sh/rsh/errors/#RAPI0000

CallStack (from HasCallStack):
  expect_throw, called at src/Reach/AST/Base.hs:224:17 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.AST.Base
  expect_thrown, called at src/Reach/APICut.hs:162:3 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.APICut
```

Severity and Impact Summary

Unused code that is present serves no purpose and is potentially dangerous as it may cause unwanted or unexpected behavior.

Recommendation

Remove unused code from the program. If an API is not being directly used or contributing to the program functionality, it should be removed. When this issue was brought up, it was discovered that the original commit hash for the code was not the most up to date. In subsequent iterations of development, this issue has been remediated.

References

https://docs.reach.sh/rsh/errors/#RAPI0000

API "ANY_HALT" DOES NOT OCCUR IN THE PROGRAM

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

Description

API was defined but not used in the program. This could pose issues to future functionality and perhaps allow the program to execute in unexpected ways.

Proof of Issue

File name: index.rsh
Line number: 183

```
reachc: error[RAPI0000]: API "Any_halt" does not occur in program

  ./index.rsh:183:32:application

  183|   export const main = Reach.App(() => {

For further explanation of this error, see:
https://docs.reach.sh/rsh/errors/#RAPI0000

CallStack (from HasCallStack):
  expect_throw, called at src/Reach/AST/Base.hs:224:17 in reach-0-
t6a9aaBayV7nUd09kpubD:Reach.AST.Base
  expect_thrown, called at src/Reach/APICut.hs:162:3 in reach-0-
t6a9aaBayV7nUd09kpubD:Reach.APICut
```

Severity and Impact Summary

Unused code that is present serves no purpose and is potentially dangerous as it may cause unwanted or unexpected behavior.

Recommendation

Remove unused code from the program. If an API is not being directly used or contributing to the program functionality, it should be removed. When the issue was brought up it was discovered that the original commit hash for the code was not the most up-to-date. In subsequent iterations of development, this issue has been remediated.

References

https://docs.reach.sh/rsh/errors/#RAPI0000

MISMATCHED TYPES IN MANAGER REQUIREMENT

Finding ID: FYEO-VEST-04
Severity: Low
Status: Remediated

Description

Types are mismatched. The compilation of `flexibleFunding` results in an incorrect invariant in which the expected type is `Address`, but a `Null` was received.

Proof of Issue

File name: vestigesafe-main/index.rsh

Line number: 88

```
      .api(AdminAPI.updateParam,
        (newPct, newAmt, newAddr) => {
          assume(this == adminAddr
            && newPct >= 0
            && newPct <= 10000
            && newAddr != null)
        },
each@ubuntu:~/Vestige/vestigesafe-main$ /home/reach/Vestige/reach compile
Compiling "manager"...
reachc: error[RE0088]: These types are mismatched:
  expected: Address
       got: Null

  ./index.rsh:88:24:application

  88|             && newAddr != null)

Trace:
  in [unknown function] from (./index.rsh:84:38:function exp) at
(./index.rsh:84:38:application)

For further explanation of this error, see:
https://docs.reach.sh/rsh/errors/#RE0088

CallStack (from HasCallStack):
  expect_throw, called at src/Reach/Eval/Core.hs:372:3 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.Eval.Core
  expect_, called at src/Reach/Eval/Core.hs:681:5 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.Eval.Core
```

Severity and Impact Summary

The error occurs because the comparison operator is comparing two different data types. This could lead to issues in the compilation as well as the possibility of unintended behavior.

Recommendation

When the `use-strict` flag is set for the compiler, it will raise an error because `null` doesn't have a type. With strict compilation, this error will be raised throughout the code base. Use alternatives or allow for non-strict compilation.

References

https://docs.reach.sh/rsh/errors/#RE0088

MISMATCHED TYPES IN FLEXIBLEFUNDING ASSUMPTION

Finding ID: FYEO-VEST-05
Severity: Low
Status: Remediated

Description

Types are mismatched. The compilation of `flexibleFunding` results in an incorrect invariant in which the expected type is `Token`, but a `Null` was received.

Proof of Issue

File name: tinystart/index.rsh

Line number: 1060

```
  const User = API('User', {
    buyFixed: Fun([UInt], Bool),
    //offer: Fun([UInt], Bool),
    //claimToken: Fun([], Bool),
    //claimBackOffer: Fun([], Bool),
    //endRound: Fun([], Bool)
  })
  const OffererAPI = API("OffererAPI", {
    claimFixed: Fun([], Bool),
    //claim: Fun([], Bool),
    cancelOrder: Fun([], Bool)
  })
  const Admin = API("AdminAPI", { claimFees: Fun([], Bool) })
  const Viewer = View('Viewer', { read: State })
  init();
  TokenOfferer.only(() => {
    const {
      tokenOffered,
      amountOffered,
      hasDeadline,
      deadline,
      fundingCase,
      minReceived,
      maxReceived,
      managerContract,
      tokenForFee
    } = declassify(interact.setParam());

    assume(tokenOffered != null)
    assume(tokenForFee != tokenOffered)
    assume(amountOffered > 0)
    if (!hasDeadline) { assume(minReceived == maxReceived) }
    if(hasDeadline){
```

```
        assume (minReceived > 0)
      }
      assume(amountOffered < UInt.max)
      assume(hasDeadline == false )
    });
Compiling "flexibleFunding"...
reachc: error[RE0088]: These types are mismatched:
  expected: Token
       got: Null

  ./index.rsh:1060:27:application

  1060|       assume(tokenOffered != null)

Trace:
  in [unknown function] from (./index.rsh:1047:26:function exp) at
(./index.rsh:1047:22:application)

For further explanation of this error, see:
https://docs.reach.sh/rsh/errors/#RE0088

CallStack (from HasCallStack):
  expect_throw, called at src/Reach/Eval/Core.hs:372:3 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.Eval.Core
  expect_, called at src/Reach/Eval/Core.hs:681:5 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.Eval.Core
```

Severity and Impact Summary

The error occurs because the comparison operator is comparing two different data types. This could lead to issues in the compilation as well as the possibility of unintended behavior.

Recommendation

When the `use-strict` flag is set for the compiler, it will raise an error because `null` doesn't have a type. With strict compilation, this error will be raised throughout the code base. Use alternatives or allow for non-strict compilation.

References

https://docs.reach.sh/rsh/errors/#RE0088

MISMATCHED TYPES IN STAKING ASSUMPTION

Finding ID: FYEO-VEST-06
Severity: Low
Status: Remediated

Description

Types are mismatched. The compilation of `main` results in an incorrect invariant in which the expected type is `Token`, but a `Null` was received.

Proof of Issue

File name: tinystake-main/index.rsh

Line number: 148

```
     assume(rewardToken != tokForFee)

    if (isVanillaStake == true) {
      assume(stakeToken == null)

    }
    else { assume(rewardToken != stakeToken)
    assume(stakeToken != tokForFee)}
reach@ubuntu:~/Vestige/tinystake-main/reach$ /home/reach/Vestige/reach compile
reachc: error[RE0088]: These types are mismatched:
  expected: Token
       got: Null

  ./index.rsh:148:25:application

  148|       assume(stakeToken == null)

Trace:
  in [unknown function] from (./index.rsh:134:20:function exp) at
(./index.rsh:134:16:application)

For further explanation of this error, see:
https://docs.reach.sh/rsh/errors/#RE0088

CallStack (from HasCallStack):
  expect_throw, called at src/Reach/Eval/Core.hs:372:3 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.Eval.Core
  expect_, called at src/Reach/Eval/Core.hs:681:5 in reach-0-
ILn78w7tAsfKMai7G6d7pp:Reach.Eval.Core
Compiling "main"...
```

Severity and Impact Summary

The error occurs because the comparison operator is comparing two different data types. This could lead to issues in the compilation as well as the possibility of unintended behavior.

Recommendation

When the `use-strict` flag is set for the compiler, it will raise an error because `null` doesn't have a type. With strict compilation, this error will be raised throughout the code base. Use alternatives to `null` or allow for non-strict compilation.

References

https://docs.reach.sh/rsh/errors/#RE0088

SET COMPILER TO 'USE STRICT'

Finding ID: FYEO-VEST-07
Severity: **Informational**
Status: **Remediated**

Description

Recommend using `use strict` to enable additional checks. Using this mode is helpful for finding unexpected problems and forgotten variables that don't need to be included. In strict mode, unused variable checks are enabled for the current scope and if a variable is declared but not used, an error will be emitted at compilation.

Proof of Issue

File name: /vestigesafe-main/index.rsh /tinystake-main/index.rsh /tinystart-main/index.rsh
Line number: 1

```
'reach 0.1'
'use strict'
```

Severity and Impact Summary

There is no major impact to the program. This is informational knowledge that allows for better insight into the functionality of the Reach program.

Recommendation

The `use strict` feature is recommended even though it will reject some code that is normally accepted. It also can limit the dynamic capability of the Reach type system.

References

https://docs.reach.sh/rsh/compute/#rsh_'use%20strict'

TODO: Implement additional Testing

Finding ID: FYEO-VEST-08
Severity: **Informational**
Status: **Remediated**

Description

Boilerplate testing code was present in the codebase. Some testing was limited due to the maturity of the code and the iterative pace of development. At the conclusion of the engagement, the team was able to conduct fairly robust testing on `vestigesafe-main` contract due to the tireless efforts of the development team but could have benefited from additional testing each time code was updated beyond the core functionalities.

Proof of Issue

```
Starting backends...
file:///app/index.mjs:18
    backend.Alice(ctcAlice, {
            ^

TypeError: backend.Alice is not a function
    at file:///app/index.mjs:18:13
    at processTicksAndRejections (node:internal/process/task_queues:96:5)
ERROR: 1
```

Severity and Impact Summary

Unfinished test code that is either generic code from the framework or indicated by the TODO tag may cause unwanted behavior or not properly test specified components.

Recommendation

Testing was implemented on much of the core functionality of the contract. This testing was interactive and covered most of the general testing philosophy of Reach. The developer used all the Reach provided tools (1-3) and, while sufficient, could have been bolstered by the addition of 4-5. (See Reference below)

References

https://docs.reach.sh/guide/testing/#guide-testing

Trustworthy maps are not trustworthy on Algorand

Finding ID: FYEO-VEST-09
Severity: **Informational**
Status: **Remediated**

Description

This program was compiled with trustworthy maps, but maps are not trustworthy on Algorand because they are represented with local state. A user can delete their local state at any time by sending a ClearState transaction. The only way to use local state properly on Algorand is to ensure that a user doing this can only 'hurt' themselves and not the entire system.

Proof of Issue

```
WARNING: Compiler instructed to emit for Algorand, but the conservative analysis found these potential problems:
 * This program was compiled with trustworthy maps, but maps are not trustworthy on Algorand, because they are represented with local state. A user can delete their local state at any time, by sending a C
learState transaction. The only way to use local state properly on Algorand is to ensure that a user doing this can only 'hurt' themselves and not the entire system.
```

Severity and Impact Summary

The warning occurs because of the architectures present in Algorand and the Reach connectors. This is informational and should be considered a priority when designing smart contracts.

Recommendation

User functionality on the program should be assessed from the point of view that the local state can be cleared and that the resulting actions can only negatively affect themselves and not the functionality or security of the entire system.

References

https://developer.algorand.org/docs/get-details/dapps/smart-contracts/apps/

USE OF 'SAFE' ARITHMETIC FUNCTIONS

Finding ID: FYEO-VEST-10
Severity: **Informational**
Status: **Remediated**

Description

Recommend using safe to enable additional runtime checks with this operation. Using this functionality is recommended in cases that the operation isn't statically verified. There are differences between `safeMulDiv` and `verifyMuldiv`. While `safeMulDiv` behaves the same as `muldiv`, `verifyMuldiv` is required to verify `muldiv` when using `verifyArithmetic`.

Proof of Issue

File name: /vestigesafe-main/index.rsh /tinystake-main/index.rsh /tinystart-main/index.rsh

Line number: varies

```
export const manager = Reach.App(() => {
  setOptions({ connectors: [ALGO] });
  setOptions({ untrustworthyMaps: true });

setOptions({verifyArithmetic: true})

  const Admin = Participant('Admin', {
    setParam: Fun([],
      Object({
        pctgAmt: UInt,
        tokenToPay: Token,
        amtToPay: UInt,
        adminAddr: Address
      })),
    ready: Fun([], Null)
  })
```

Severity and Impact Summary

Without the use of `safe` there are no runtime checks generated for this operation, which leaves it open to the possibility of arithmetic errors such as overflow, etc.

Recommendation

During the August 2022 release, the addition of safe computation was an added functionality to the Reach compiler. It is recommended to use safe computation in arithmetic functions, when possible, to ensure runtime checks are implemented for the associated arithmetic operations. Available in release candidate 0.1.11-rc.7 and later 2022/08/07: Added safe* and veri* arithmetic functions, e.g. `safeAdd` and `veriAdd.30`

References

https://docs.reach.sh/rsh/compute/#safemuldiv

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