Quantstamp

Sequence - Eiffel

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	Enclave Infra
Timeline	2024-03-20 through 2024-03-22
Language	Go, Dockerfile
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Specification	Sequence WaaS: Trusted Third Party Upgrade Protocols
Source Code	• https://github.com/0xsequence/eiffel #aed2359
Auditors	• Andy Lin Senior Auditing Engineer • Gelei Deng Auditing Engineer

Documentation quality	Medium
Test quality	Undetermined
Total Findings	2 Fixed: 2
High severity findings	0
Medium severity findings	0
Low severity findings	1 Fixed: 1
Undetermined severity findings	0
Informational findings	1 Fixed: 1

Summary of Findings

We conducted an audit focusing on the `eiffel` repository, which provides crucial tools like a base Docker image, a script for building the Enclave Image File (EIF), and a special `pid1` Go executable. This executable helps start essential services such as the `chronyd` daemon. Our main goal was to check if PCR0, a type of hash, can be trusted to verify the security of the software environment, known as an enclave. The Sequence team, who developed this, wants to ensure that any application using their tools can consistently produce the same reliable PCR0 hash if the starting conditions remain the same, so that users could validate the integrity of the services running in the enclave.

For our review, we began with hands-on tests to verify our initial assumptions. We built the sample app on different operating systems (MacOS, Ubuntu, Windows) and found that PCR0 was always the same, proving it works as intended. Changing the code or the setup of the app also resulted in changes to PCR0, which demonstrates that the hash can detect differences. In addition to these tests, we explored how AWS's Nitro Enclaves and its security checks work, focusing on PCR values and AWS KMS's role in ensuring an enclave's integrity. We examined the code and settings to ensure everything was correct. Our review confirmed that the Sequence team's approach is solid, but we also identified some areas that could be improved and a potential issue that might affect operations.

Last but not least, for the future audit of the `waas-authenticator`, or any app that uses the `eiffel` base image, we recommend checking the following:

- The `layout.yaml` file should not have conflicts with the `bootstrap.yaml`, as it is unclear which file will take precedence.
- The `layout.yaml` must define both the `cmd` and `env`.
- The `cmd` must start with `/sbin/pid1` according to the current design, unless there is a specific reason to bypass the `pid1/main.go`.
- The Dockerfile of the app should use a pinned version of the base image, for example, `FROM ghcr.io/0xsequence/eiffel:v0.3.0@sha256:d4aa946d7eb587e0554123efc3eaa5830a1428b0325ea239fe489e372f573dfe`.
- Sequence team should provide valid, verifiable approach for users to examine the PCR0 value of running enclaves with the app to the users.

Fix Review Update: All issues have been addressed. The team resolved them using a more elegant code style than we recommended and went further by locking additional dependency versions beyond what we had pointed out. We have tested the fix for SEQ-1, and it now works as expected.

ID	DESCRIPTION	SEVERITY	STATUS
SEQ-1	Risk of Zombie Instance	Low	Fixed
SEQ-2	Dependency Version Not Pinned	Informational	Fixed

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

Also, we did not review dependencies used in the project, such as `aws-nitro-enclaves-cli`. The security of the dependencies are out of the scope of this review.

Possible issues we looked for included (but are not limited to):

- Dockerfile setup
- Infrastructure architecture
- Nitro enclave security concerns
- KMS integration

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and analysis that includes the following:
    1. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

Scope

Files Included

Repo: https://github.com/0xsequence/eiffel(676c6c43d6b165059a84be66b8b5e0d887dc2a20) Files: *

Files Excluded

Repo: https://github.com/0xsequence/eiffel(676c6c43d6b165059a84be66b8b5e0d887dc2a20) Files: .github

Findings

SEQ-1 Risk of Zombie Instance

Low Fixed

Update

The team fixed the issue in commit `83d49069` by combining the use of `errgroup.WithContext()` and `exec.CommandContext()`. The current implementation leverages the Go context. `errgroup.WithContext()` ensures that any goroutine exiting with an error will trigger the cancellation of the context. Meanwhile, `exec.CommandContext()` will interrupt and stop any other spawned executions when the context is cancelled.

File(s) affected: `pid1/main.go`

Description: The `main()` function initiates commands, with the first command being a hardcoded `chronyd`, and others can be passed as argument inputs. The function triggers these commands (`cmd.Start()`) and spawns goroutines to wait for the commands to return or finish

(`cmd.Wait()`). At the end of the function, it uses `wg.Wait()` to wait for all commands to finish. If any of the commands errors out, it uses `log.Fatal()` to log the error and exit the program.

The issue arises if one of the commands is a long-running program, it will never return, leading `wg.Wait()` to wait indefinitely. If any of the commands fail during this period, the program will continue running and not exit. There is a risk that the enclave becomes a zombie instance, as the main command it aims to run might have already stopped, while the main `pid1` program is still waiting for other long-running commands (e.g., `chronyd`) to finish.

Exploit Scenario: Here is an example,

1. The `pid1` triggers the `example` command and the `chronyd` daemon.
2. The `example` command crashes somehow.
3. The `pid1` will not stop, and the instance becomes a zombie that cannot perform its intended function anymore.

Recommendation: Consider a different design that allows the program to stop if any of the spawned commands error out. For instance, one could use a for-loop with `select` to help stop the program earlier:

```
for i := 0; i < num_of_spawned_go_routines; i++ {
    select {
    case err := <-done:
        if err != nil {
            log.Fatal(err.Error())
        }
    }
}
```

SEQ-2 Dependency Version Not Pinned

Informational Fixed

Update

The team pinned the versions of the Docker image dependencies in commit `3cf4ed3` . Not only is the specified dependency (`aws-nitro-enclaves-image-format`) pinned, but the versions of the `rust` and `alpine` images are also locked.

File(s) affected: `Dockerfile`

Description: The main `Dockerfile` clones the `aws-nitro-enclaves-image-format` repository from the main branch in the following line: `RUN git clone --depth 1 -b main https://github.com/aws/aws-nitro-enclaves-image-format.git /workspace` . The setup does not pin the dependency to a specific version, which might risk having indeterministic builds in the future.

During the initial audit, we have discussed this with the team, and they seem to have plans to replace this specific dependency repository soon. So, the issue might become irrelevant after that.

Recommendation: After checking the dependency repository and noting it does not have release versions, we suggest pinning it to a specific git commit hash.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Automated Analysis

N/A

Changelog

- 2024-03-22 - Initial report
- 2024-03-26 - Fix review report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.