GETTY/IO

Security Audit for:

KleverChain (Code Review)

Version 0.1: September 9th, 2022

About US

Getty/IO is an innovative remote IT security consulting firm that has the expertise to audit and recommendation to build secure web and mobile products. We specialize in modern Javascript technologies, lean and highly scalable backends, and blockchain technologies, which help to build the secure product for our customers worldwide.

Born in South America, GETTY has become the largest remote development firm in the region. We are a global company helping startups and enterprises from all around the world scale their development teams by providing them with the top remote developers and consultants:

- Blockchain - Security Review & Audit
- Smart Contract - Security Review & Audit
- Applications - Security Review & Audit

Our professionals are able to integrate to our Customer's teams and add value from day one. Agile methodologies, experience helping dozens of teams build products, and constant coaching and mentoring ensure all our engagements help our customers become successful.

In our hands, your entire process is safe, without the hassle, and as seamless as it can be. With the collaboration of our team of experts, you can expect to achieve much more. We are present in the United States, Canada, Portugal, Estonia, and Brazil.

Abstract

Getty/IO has been appointed by the Klever Team to carry out the audit of KleverChain, his main blockchain. **Klever** is a decentralized p2p and self-custody wallet and **KFI** is the Klever blockchain governance token, to be used in Klever's blockchain parameters and kApp upgrades and the **KLV** token is the fuel of the blockchain, used to incentive people to participate in the blockchain throughout proof of stake blockchain's consensus mechanism as also burn it to pay networks fees.

This document presents the results of an Internal security audit for the KleverChain. This test aimed to identify security vulnerabilities that could negatively affect the systems under the scope, the data they handle, and consequently the business. They were simulated in a systematic way, attacks that were specifically tailored for the engagement's scope to test the resilience against real-life attack scenarios based on a black-box approach.

The analysis focused on vulnerabilities especially related to implementation, and on issues caused by architectural or design errors.

For each vulnerability discovered during the assessment, it was attributed a risk severity rating. The issue's severity classification is based on the potential it presents to provide means for fraud, data leakage, and other harmful events that may bring a direct adverse impact to the business.

Methodology

Tests were conducted using risk factors, such as probability and impact. Each test and tool that has been used was focused on vulnerability's complexity and how it could be mitigated.

		IS Risk Vision		
Exploitation Complexity	Low	Medium	High	Critical
	Medium	Low	Medium	High
	High	Low	Low	High
		Low	Medium	High
		Impact		

Tools and Vectors

The following tools and vectors were applied:

- Fuzzers: Bed, Rfuzz (Ruby), Sfuzz, fuzzing auxiliary modules of Metasploit, Spike (kit for developing fuzzers), etc.
- Brute force: John the Ripper, Hashcat, etc.
- Web applications: W3AF, Websecurify, Accunetix, Metasploit scanning auxiliary modules, CGI
- Scanner, ASP-Auditor, Oscanner, proxies such as Fiddler2 or Webscarab, Firefox browser plugins such as hacking toolbar, Tamper Data, User-Agent switcher, etc.
- Manual search in vulnerability repositories such as CVE, OSVD or NVD.
- Manual code analysis with the aim of finding weaknesses and developing bad practices, making use of editors, debuggers, and decompilers.
- Manual attacks: open port stress tests, SQL injections, CSS, RFI, overflows buffer detection, directory listing, web proxies (ZAP, Burp Suite, and webscarab), etc.
- Network communications analysis, Tshark, Ettercap, Wireshark, etc.
- In addition, this audit phase can be completed by using tools that automate the security analysis process of devices under studies, such as the well-known Nessus scanner and its GPL equivalent OpenVas.

Tests Performed

- Look for hardcoded credentials
- Bind to all interfaces
- Audit the use of unsafe block
- Audit errors not checked
- Audit the use of ssh.InsecureIgnoreHostKey function
- Url provided to HTTP request as taint input
- Profiling endpoint is automatically exposed
- Converting strconv.Atoi result to int32/int16
- Detect io.Copy instead of io.CopyN when decompression
- Detect http.Dir('/') as a potential risk
- Detect ReadHeaderTimeout not configured as a potential risk
- Usage of Rat.SetString in math/big with an overflow
- Use of net/http serve function that has no support for setting timeouts
- SQL query construction using format string

- SQL query construction using string concatenation
- Use of unescaped data in HTML templates
- Audit use of command execution
- Poor file permissions used when creating a directory
- Poor file permissions used when creation file or using chmod
- Creating tempfile using a predictable path
- File path provided as taint input
- File path traversal when extracting zip archive
- Poor file permissions used when writing to a file
- Unsafe defer call of a method returning an error
- Detect the usage of DES, RC4, MD5 or SHA1
- Look for bad TLS connection settings
- Ensure minimum RSA key length of 2048 bits
- Insecure random number source (rand)
- Import blocklist: crypto/md5
- Import blocklist: crypto/des
- Import blocklist: crypto/rc4
- Import blocklist: net/http/cgi
- Import blocklist: crypto/sha1
- Implicit memory aliasing in RangeStmt

Audit Dashboard

Project	KleverChain
Auditors	Wesley Silva Luis Araujo
Assets	KleverChain Code
Networks	Node Mainnet /Node Testnet

Date	2022-008-01 to 2022-09-08

Issues Found

	Low	Medium	High	Critical
Open	**0**	**0**	**4**	**0**
Resolved	**0**	**0**	**0**	**0**

Results

Item	/Klever/projects/klever-go/node/heartbeat/componentHandler/heartbeatHandler.go
Description	Use of weak random number generator (math/rand instead of crypto/rand).

Evicence	`209: func (hbh *HeartbeatHandler) startSendingHeartbeats(ctx context.Context) {`
	`210:  r := rand.New(rand.NewSource(time.Now().Unix()))`
	`211:  cfg := hbh.arg.HeartbeatConfig`

Solution	Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CyptGenRandom on Windows, or hw_rand() on Linux.
Risk Factor	High
Assets	KleverChain Code
Resolved	No. Very low probability to be exploited.

Item	/Klever/projects/klever-go/network/p2p/libp2p/rand/seedRandReader.go
Description	Use of weak random number generator (math/rand instead of crypto/rand)
Evicence	36: 37: randomizer := rand.New(rand.NewSource(srr.seedNumber)) 38:
Solution	Use functions or hardware which use a hardware-based random number generation for all crypto. This is the recommended solution. Use CyptGenRandom on Windows, or hw_rand() on Linux.
Risk Factor	High
Assets	KleverChain Code
Resolved	No. Very low probability to be exploited.

Item	/Klever/projects/klever-go/core/metrics.go
Description	Potential hardcoded credentials

Evicence	57: // MetricNoncesPassedInCurrentEpoch is the metric that tells the number of nonces passed i 58: const MetricNoncesPassedInCurrentEpoch = "klv_nonces_passed_in_current_epoch" 59:
Solution	If the software must contain hard-coded credentials or they cannot be removed, perform access control checks and limit which entities can access the feature that requires the hard-coded credentials. For example, a feature might only be enabled through the system console instead of through a network connection.
Risk Factor	High
Assets	KleverChain Code
Resolved	No. Very low probability to be exploited.

Item	/Klever/projects/klever-go/core/metrics.go
Description	Potential hardcoded credentials
Evicence	54: // MetricSlotsPassedInCurrentEpoch is the metric that tells the number of slots passed in 55: const MetricSlotsPassedInCurrentEpoch = "klv_slots_passed_in_current_epoch" 56:
Solution	If the software must contain hard-coded credentials or they cannot be removed, perform access control checks and limit which entities can access the feature that requires the hard-coded credentials. For example, a feature might only be enabled through the system

	console instead of through a network connection.
Risk Factor	High
Assets	KleverChain Code
Resolved	No. Very low probability to be exploited.

Conclusion

After using some tools and testing the environment described in this document, We have not found critical or high vulnerabilities.

All the security requirements have been archived during the architecture and coding phase.

Appendix

Severity

Low	Low issues are generally subjective in nature or potentially deal with topics like "best practices" or "readability". Minor issues will in general not indicate an actual problem or bug in code. The maintainers should use their own judgment as to whether addressing these issues improves the codebase.
Medium	Medium issues are generally objective in nature but do not represent actual bugs or security problems. These issues should be addressed unless there is a clear reason not to.
High	High issues will be things like bugs or security vulnerabilities. These issues may not be directly exploitable or may require a certain condition to arise

	in order to be exploited. Left unaddressed, these issues are highly likely to cause problems with the operation of the contract or to lead to a situation that allows the system to be exploited in some way.