The Update Framework (TUF) Security Assessment

Kolide

October 18, 2017 – Version 1.0

Prepared for
Mike Arpaia

Prepared by
Mason Hemmel
Jeff Dileo

Executive Summary

Synopsis

During the summer of 2017, Kolide engaged NCC Group to conduct a security assessment of their implementation of a client for TUF, The Update Framework. Two NCC Group consultants performed this assessment between August 28 and September 1. Along with a Docker Notary server, this client allows osquery to automatically and securely update itself. This report, in conjunction with NCC Group's prior assessments of Notary and osquery, completes a trifecta of security assessments for a fully functional suite of interdependent secure endpoint management technologies.

This engagement was structured as an implementation assessment of Kolide's TUF client. The full client, as sourced from Kolide's Github page, was in scope. In addition to the source to the client, Kolide also actively gave feedback and collaborated with NCC Group consultants. NCC Group performed a manual review of the code, supplemented by light dynamic testing, with a focus on identifying coding and logical errors, misuse or misapplication of cryptographic techniques, unsafe functions, and other vulnerabilities that would permit an attacker to disrupt the intended flow of the program.

Though NCC Group did find some issues that could affect users in less-secure environments, the assessment did not uncover any findings that affect the security of the processes for receiving and verifying updates.

Scope

The testing team spent ten person-days assessing the provided Golang source code for Kolide's TUF client. As noted above, while the client relies on Docker Notary and is intended for use with osquery, neither Notary nor osquery were included in the scope of this assessment. As a result, NCC Group did not explore issues that may arise when combining these projects together. The team spent the majority of the testing time verifying that attackers could not forge updates from a privileged position, maliciously structure updates to cause unexpected behavior in the client, or cause a client's view of its state relative to the wider world to be inaccurate. In addition, NCC Group verified the client against TUF's specification.

NCC Group performed this assessment using TUF's threat model as a guide.[1] As such, vulnerabilities relating to the following were not in scope:

- The content of updates (apart from their authenticity)
- The process for applying updates
- The use of automated signatures for any TUF role
- Recovery from error states
- Errors arising from clock synchronization
- Other details relating to specific instances of usage

Key Findings

NCC Group's assessment highlighted two areas for further improvement. As noted earlier, these findings do not directly impact the client's core functionality. They are included for the purpose of ensuring consistently safe use further downstream once the project has been opened to the public. These findings are as follows:

- The client writes its backups to predictable file locations without checking those locations for symbolic links. An attacker who anticipated the write location for a given backup could create a symbolic link from this location to a privileged resource. By this means, the attacker could hijack the backup process to write to areas that they could not ordinarily access. More details concerning this finding can be found in finding NCC-KOLI-001-002 on page 6.
- Default configurations allow users to access updates over plaintext HTTP, removing confidentiality and authenticity guarantees from the network transport layer. More details concerning this finding can be found in finding NCC-KOLI-001-001 on page 7.

Limitations

NCC Group did not have the opportunity to explore the use of delegates,[2] as none were implemented when the assessment was performed. Due to time constraints, NCC Group was unable to perform fuzzing of the parsing and interpretation logic of TUF's various document and file formats.

[1] See section 1.5, "Goals", under https://github.com/theupdateframework/tuf/blob/224e30183ce2feba9ebe184f4a8aa339a018e126/docs/tuf-spec.md#1-introduction

[2] See discussion of delegated trust roles in section 2.1.2 under https://github.com/theupdateframework/tuf/blob/224e30183ce2feba9ebe184f4a8aa339a018e126/docs/tuf-spec.md#2-system-overview

Project Dashboard

Target Metadata

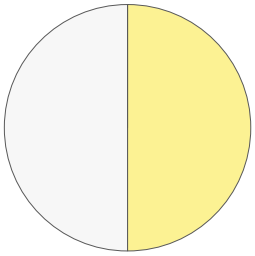
Name	The Update Framework (TUF)
Type	Secure Update Fraemwork
Platforms	Go

Engagement Data

Type	Security assessment
Method	Code-assisted
Dates	2017-08-28 to 2017-09-01
Consultants	2
Level of effort	2 person-weeks

Finding Breakdown

Critical Risk issues	0
High Risk issues	0
Medium Risk issues	0
Low Risk issues	1
Informational issues	1
Total issues	2



Category Breakdown

Configuration	1
Other	1

Key

Critical High Medium Low Informational

Strategic Recommendations

During the course of this engagement, NCC Group determined that the following high-level recommendations may enhance the security posture of Kolide's TUF client. The individual recommendations described in the Finding Details section should still be reviewed and implemented in order to address the findings described in this report.

Integrate fuzzing into test pipeline

An essential element of TUF is its suite of custom JSON-based file formats. They describe and define cryptographic keys, roles, and trust structures, along with settings and configurations. A flaw in the parsing or processing of any of these could violate TUF's security goals.

Custom file formats such as those used by TUF can describe any number of data structures and configurations—both valid and invalid. Fuzzing is a technique that involves programmatically generating inputs for a computer program, then monitoring for unexpected behavior. This process occurs automatically, and thus quickly covers many more possibilities than could ever be tested manually. Accordingly, the best method for ensuring that a robust logic parses these formats would be fuzz testing. Kolide should consider using a coverage-guided fuzzing framework, such as go-fuzz, [3] to ensure strong code coverage.

Consider hardening for likely downstream use cases

Kolide intends to release the source code for its TUF client under an open license and should be aware that this step changes their security outlook. Rather than a single-purpose project, their TUF client will now likely be used in projects that differ substantially from osquery.

The client's security model should begin to take into account the varying needs of downstream users and developers who integrate this code into their projects. For instance, downstream projects integrating this code will likely not have the benefit of osquery's robust security model. Nevertheless, the client's default functionality should be misuse resistant to avoid weakening the security posture of dependent projects. Misuse resistance requires attention to issues such as those identified during this assessment, which could introduce vulnerabilities in projects with lesser security preparedness. Ultimately, if future development follows this defense-in-depth approach, the client itself will benefit along with its end users downstream.

[3] https://github.com/dvyukov/go-fuzz

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 8.

Title	ID	Risk
TUF Backups May Be Written to Unsafe Directories	002	Low
Default Client Configuration Can Be Misused	001	Informational

Finding Details

Finding	**TUF Backups May Be Written to Unsafe Directories**
Risk	**Low** Impact: Low, Exploitability: Medium
Identifier	NCC-KOLI-001-002
Category	Other
Location	https://github.com/kolide/updater/blob/ea5f80f528eadab1a3faf25bbd59197802eff383/tuf/persistence.go#L46
Impact	Attackers could delete or modify crucial TUF files, or use symlinks to write to files they ordinarily do not have access to, such as '/etc/nologin', resulting in denial of service or worse.
Description	When backing up TUF repositories, TUF uses the `copy()` function to write from the existing path to a new, predictable one. Unlike the process for saving TUF roles, this process saves to a directory that has been input by the user as the `LocalRepoPath`. While there is a comment in the client's source suggesting that this should be a directory with 0600 permissions, this is not enforced. If the user were to simply proceed as normal and create a folder giving group members write permissions, an attacker can tamper with the backup process. For instance, if an attacker were to create a symbolic link with the same name as the file to which the backup will be written, the backup's write operation would then follow the symbolic link and effectively overwrite the file to which the link points. Depending on the file written to, this could have negative effects on the system. Though this finding is unlikely to apply to the osquery deployment of this client, which would likely create its root directory with recommended permissions, it is included here in consideration of downstream adopters of this client with less-secure environments. This is explained in further depth in the Strategic Recommendations on page 4.
Recommendation	Upon first creating the TUF repository, emit an error if the `LocalRepoPath` does not have 0600 permissions. In addition, stop and emit an error when backing up if the filename to which the backup will be written already exists. This functionality should be switchable, as it is possible that some users will want to backup to symlinks or similar structures.

Finding	**Default Client Configuration Can Be Misused**
Risk	**Informational** Impact: None, Exploitability: None
Identifier	NCC-KOLI-001-001
Category	Configuration
Location	https://github.com/kolide/updater/blob/ea5f80f528eadab1a3faf25bbd59197802eff383/tuf/client.go#L303
Impact	Attackers could silently tamper with content users receive if they make use of the default client with an insecure endpoint.
Description	An unsophisticated user is most likely to use the default client without any modification. To protect this type of user, NCC Group considers it a best practice to ensure that default clients be unimpeachably secure. However, the default client transparently fulfills insecure requests in the form of plaintext HTTP connections. Such connections have no confidentiality, authenticity, or integrity controls, allowing network attackers to stream arbitrary content back to the client without any way for the user to detect this attack. The support for HTTP connections arises from the client's use of the `http.Client` class, which transparently fulfills HTTP connections by design. In addition, past versions of Golang will negotiate insecure TLS options. Specifically, Go versions prior to 1.5 would negotiate SSLv3 and RC4 ciphersuites by default,[4,5] and all CBC ciphersuites prior to version 1.8 were vulnerable to Lucky 13.[6,7]
Recommendation	Before making the connection with `http.Client`, use the `url` package to parse the URL and check that it uses the HTTPS scheme, emitting an error otherwise. To protect older clients while negotiating TLS, explicitly use the current default TLS ciphersuite configuration.

[4] https://github.com/golang/go/commit/604fa4d5a149c334ff0bd5d191c4c4e29f75545d
[5] https://github.com/golang/go/commit/d26fdf295ed1f0154f65110b17ac6ecf4300dad4
[6] http://www.isg.rhul.ac.uk/tls/Lucky13.html
[7] https://github.com/golang/go/commit/f28cf8346c4ce7cb74bf97c7c69da21c43a78034

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems have. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

Access Controls	Related to authorization of users, and assessment of rights.
Auditing and Logging	Related to auditing of actions, or logging of problems.
Authentication	Related to the identification of users.
Configuration	Related to security configurations of servers, devices, or software.
Cryptography	Related to mathematical protections for data.
Data Exposure	Related to unintended exposure of sensitive information.
Data Validation	Related to improper reliance on the structure or values of data.
Denial of Service	Related to causing system failure.
Error Reporting	Related to the reporting of error conditions in a secure fashion.
Patching	Related to keeping software up to date.
Session Management	Related to the identification of authenticated users.
Timing	Related to race conditions, locking, or order of operations.