

Fleet

Security Assessment

26.04.2021

Prepared For:
Zach Wasserman | *Fleet*
zach@fleetdm.com

Prepared By:
Paweł Płatek | *Trail of Bits*
pawel.platek@trailofbits.com

Alex Useche | *Trail of Bits*
alex.useche@trailofbits.com

[Executive Summary](#)

[Project Dashboard](#)

[Code Maturity Evaluation](#)

[Engagement Goals](#)

[Coverage](#)

[Recommendations Summary](#)

[Short Term](#)

[Long Term](#)

[Findings Summary](#)

- [1. Unhandled deferred file close operations](#)
- [2. Files and directories pre-existence attacks](#)
- [3. Possible nil pointer dereference](#)
- [4. Forcing empty passphrase for keys encryption](#)
- [5. Signature verification in fleetctl commands](#)
- [6. Redundant online keys in documentation](#)
- [7. Lack of alerting](#)
- [8. Key rotation methodology is not documented](#)
- [9. Threshold and redundant keys](#)
- [10. Database compaction function could be called more times than expected](#)
- [11. All Windows users have read access to Fleet server secret](#)
- [12. Insufficient documentation of SDDL permissions](#)

[A. Vulnerability Classifications](#)

[B. Code Maturity Classifications](#)

[C. Code Quality Recommendations](#)

[D. Semgrep Rule](#)

Executive Summary

From 19 April through 23 April 2021, Fleet engaged Trail of Bits to review the security of Orbit autoupdater. Trail of Bits conducted this assessment over the course of 1 person-weeks with 2 engineers working from the following commits:

- <https://github.com/fleetdm/orbit>: 5b0020fe
- <https://github.com/fleetdm/fleet>: 871ba394

The assessment focused on local privilege escalation attack vectors, key management functionalities, and correctness of usage of the go-tuf library. We applied manual code review composed with dynamic analysis against a local instance of the system. We also directed static analysis to detect Go-specific bugs.

The finding with the highest severity could allow local attackers to get read and write access to Orbit files by tricking users into usage of already existing directories. This could be used, for example, to interfere with package generation and can result in remote code execution on clients machines. The other high severity issue is lack of a robust alerting mechanism, which hampers detection of security incidents. Five other findings are related to the implementation and documentation of key management functionalities, with the highest severity issue being lack of key rotation and revocation methodology. Finally, two medium severity findings concern file permissions on Windows machines.

Issues found indicate that Orbit could use improvements in areas related to key management, to provide better protection against key compromises, handling of file permissions, ideally in a centralized manner, and alerting about detected malicious behaviour. We found no bugs in the usage of the go-tuf library.

We recommend to address all findings presented in this report. Update the Fleet key management code and documentation to fix the related issues. Enhance code related to file permissions to make it more centralized and uniform across the codebase. Utilize static analyzers like gosec or CodeQL to continuously improve security of the application.

Project Dashboard

Application Summary

Name	Orbit, Fleet
Version	5b0020fe, 871ba394
Type	Go
Platforms	Linux, MacOS, Windows

Engagement Summary

Dates	April 19 through April 23, 2021
Method	Whitebox
Consultants Engaged	2
Level of Effort	1 person-week

Vulnerability Summary

Total High-Severity Issues	3	■ ■ ■
Total Medium-Severity Issues	4	■ ■ ■ ■
Total Low-Severity Issues	2	■ ■
Total Informational-Severity Issues	3	■ ■ ■
Total Undetermined-Severity Issues	0	
Total	12	

Category Breakdown

Access Controls	2	■ ■
Auditing and Logging	1	■
Configuration	1	■
Cryptography	3	■ ■ ■
Data Validation	3	■ ■ ■
Undefined Behavior	2	■ ■
Total	12	

Engagement Goals

The engagement was scoped to provide a security assessment of the Orbit autoupdater application, and to verify that it used the go-tuf package in a secure manner.

Specifically, we sought to answer the following questions:

- Is the Orbit application configuring and making use of file systems rights and permissions in a secure manner in all supported operating systems?
- Could the Orbit application be leveraged for local privilege escalation attacks?
- Are there components with potential memory corruption vulnerabilities?
- Do any of the components leak sensitive information through errors or logging messages?
- Do the components handle sensitive data in a safe and cryptographically secure manner?
- Can an attacker perform unauthorized operations?
- Are there potential concurrency bugs that could lead to security concerns?
- Does the application use the go-tuf package in a secure manner?
- Is key management secure and are related procedures well documented?

Coverage

To conduct the review, we configured the Fleet server locally, created installation packages using the Orbit CLI, and installed said packages. We then proceeded to test Orbit autoupdater dynamically in Windows, Linux, and macOS. At the same time, we conducted manual static code review of the Orbit codebase and the updates.go file from the Fleet repository, while leveraging static analysis tools such as CodeQL to aid our review.

We focused on testing against potential file permission issues, and uncovering vulnerabilities that could lead to local privilege escalation in each operating system supported by Orbit. Additionally, during our code review we paid close attention to potential insecure uses of cryptography, inefficient usage of Go concurrency mechanisms such as channels and anonymous Goroutines, as well as common Go specific bugs and vulnerabilities. We also review code responsible for key generation and TUF repository updates, as well as documentation for it. The audit did not cover the security of the go-tuf library itself, nor the security of communication channels between remote machines.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

- ❑ **Consider closing files explicitly at the end of functions and checking for errors.** Alternatively, defer a wrapper function to close the file and check for errors if it makes sense. [TOB-FLT-001](#)
- ❑ **When using utilities such as `os.MkdirAll` and `ioutil.WriteFile`, check all directories in the path and the final file and validate their owner and permissions before performing operations on them.** This will help prevent situations in which sensitive information is written to a pre-existing attacker-controlled path. [TOB-FLT-002](#)
- ❑ **Add a default branch to the switch statement that will show an error message and exit.** This will prevent application from `panic`. [TOB-FLT-003](#)
- ❑ **Check if there is no keys directory before starting creation of a new repository, that is at the beginning of the `updatesInitFunc` function in the `fleet/ee/fleetctl/updates.go` file.** Abort the procedure otherwise. [TOB-FLT-004](#)
- ❑ **Add a new command line argument to the `fleetctl` command that will enable users to provide the root public key.** Use it for repository validation. Update Fleet documentation so it instructs users to double check public keys printed by the `roots` command. [TOB-FLT-005](#)
- ❑ **Rewrite the documentation to require only the timestamp key to be stored in the online environment.** [TOB-FLT-006](#)
- ❑ **Implement a notification mechanism for Orbit users, e.g.** as a system notifications or a communication channel like an email. Alert users if a serious security incident is detected, e.g. an [indefinite freeze attack](#). Print versions of the Orbit and `osquery` in the Fleet administration panel - the same versions as used by the `go-tuf`, not a version reported by the `osquery`. [TOB-FLT-007](#)
- ❑ **Implement commands in the `fleetctl` tools that will allow users to easily replace expired or compromised keys.** Make sure that users can renew only selected keys and don't have to replace all of them. Add flags that will enable control over keys expiration

times and set secure defaults for these. Do the same for metadata expiration times.
[TOB-FLT-008](#)

❑ **Extend fleetctl tool with support for customizable thresholds and multiple keys for a single role.** Document how to use this feature. Recommend users to use it for the root role keys. [TOB-FLT-009](#)

❑ **Add an additional select case at the beginning of the loop to avoid having to call the RunValueLogGC in the edge case described above.** A potential fix is shown below:
[TOB-FLT-010](#)

❑ **Restrict read access to all non-admin users to all files under C:\Program Files\Orbit\, including the Fleet server secret.** [TOB-FLT-011](#)

❑ **Add additional comments next to each SDDL string used in code describing in detail the intended permissions and trustees.** [TOB-FLT-012](#)

Long term

- ❑ **Enumerate files and directories and ensure that they have expected permissions and owners.** Build validation to ensure appropriate permissions are applied before their creation and upon their use. Ideally, this validation should be centrally defined and used throughout the application. [TOB-FLT-002](#)
- ❑ **Use static analyzers to scan the code for possible nil pointer dereferences.** [TOB-FLT-003](#)
- ❑ **Verify that keys are protected with non-empty passwords after creating them and before using them.** [TOB-FLT-004](#)
- ❑ **Clearly describe which keys should be stored in what type of an environment and when should be used.** For example, create a table with such information. This should help to prevent misinterpretation of the documentation. [TOB-FLT-006](#)
- ❑ **Consider sending a notification from the Orbit client to Fleet server when the client encounters a failure or spots an attack against him.** Create documentation instructing clients how they should respond to detected security incidents, based on the type of the error received. [TOB-FLT-007](#)
- ❑ **Implement integration with hardware security modules to further increase security of keys management.** Implement notification mechanism that will alert administrators about expired metadata files. [TOB-FLT-008](#)
- ❑ **Consider to use multiple roles keys and thresholds for not-root-role keys.** [TOB-FLT-009](#)
- ❑ **Use the Semgrep query in [Appendix D](#) to periodically check for and detect this type of issue.** [TOB-FLT-010](#)
- ❑ **Create and incorporate unit tests in the CI/CD pipeline that verify that non-privileged users are not able to read the Fleet server secret, even after changes are made to the code.** [TOB-FLT-011](#)

Findings Summary

#	Title	Type	Severity
1	Unhandled deferred file close operations	Undefined Behavior	Low
2	Files and directories may pre-exist with too broad permissions	Data Validation	High
3	Possible nil pointer dereference	Data Validation	Informational
4	Forcing empty passphrase for keys encryption	Cryptography	Medium
5	Signature verification in fleetctl commands	Data Validation	High
6	Redundant online keys in documentation	Access Controls	Medium
7	Lack of alerting mechanism	Configuration	Medium
8	Key rotation methodology is not documented	Cryptography	Medium
9	Threshold and redundant keys	Cryptography	Informational
10	Database compaction function could be called more times than expected	Undefined Behavior	Informational
11	All Windows users have read access to Fleet server secret	Access Controls	High
12	Insufficient documentation of SDDL permissions	Auditing and Logging	Low

1. Unhandled deferred file close operations

Severity: Low
Type: Undefined Behavior
Target: Orbit

Difficulty: High
Finding ID: TOB-FLT-001

Description

Several locations throughout the Orbit codebase defer file close operations after writing to a file. This may introduce undefined behavior, as the file's content may not be flushed to disk until the file is closed.

Errors arising from the inability to flush content to disk while closing will not be caught, and the application may assume that content was written to disk successfully. See example in figure 1.1.

```
func (s *fileStore) writeData() error {
    f, err := os.OpenFile(s.filename, os.O_RDWR|os.O_CREATE, constant.DefaultFileMode)
    if err != nil {
        return errors.Wrap(err, "open file store")
    }
    defer f.Close()
```

Figure 1.1: [orbit/pkg/update/filestore/filestore.go, lines 82 – 87.](#)

Identified occurrences of the bug are:

- [orbit/pkg/update/filestore/filestore.go, lines 82 – 87](#)
- [orbit/pkg/packaging/linux_shared.go, lines 124 – 128](#)
- [orbit/pkg/packaging/macOS.go, lines 275 – 279](#)
- [orbit/pkg/packaging/packaging.go, lines 62 – 66](#)

Exploit Scenario

The server on which the Orbit application runs has a disk that periodically fails to flush content due to a hardware failure. As a result, certain methods in the codebase sometimes fail to write content to disk. This causes undefined behavior.

Recommendation

Short term, consider closing files explicitly at the end of functions and checking for errors. Alternatively, defer a wrapper function to close the file and check for errors if it makes sense.

References

- ["Don't defer Close\(\) on writable files" blogpost](#)

2. Files and directories may pre-exist with too broad permissions

Severity: High

Type: Data Validation

Target: Orbit, Fleet

Difficulty: Medium

Finding ID: TOB-FLT-002

Description

Fleet and Orbit applications create certain file and directory paths with specific access permissions (e.g., 0700) by using the `ioutil.WriteFile`, `os.MkdirAll`, `os.OpenFile` or similar functions. These functions don't change permissions of already existing files or directories and don't return an error in such situations. This could allow an attacker to create a file or directory with broad permissions before the Fleet user can create the file or directory, which could enable the attacker to tamper with the files.

Vulnerabilities occur in the multiple places, for example:

- [fleet/ee/fleetctl/updates.go, lines 334 - 336](#)
- [fleet/ee/fleetctl/updates.go, line 338](#)
- [orbit/pkg/update/update.go, lines 186 - 190](#)
- [orbit/cmd/orbit/orbit.go, line 116](#)
- [orbit/pkg/packaging/packaging.go, lines 58 - 60](#)

Please note that the go-tuf library may also contain bugs of this type.

Exploit Scenario

An attacker has unprivileged access to the machine on which a client uses Fleet. He creates new, empty directories - `keys` and `repository` - with 0777 permissions in a directory where the Fleet user will initialize a new repository. The user executes `fleetctl updates init` command. The directories remain owned by the low-privileged user and have 0777 permissions. The attacker removes files generated by the user and replaces them with his own. The user publishes the modified repository, creates the Orbit installation package and distributes it to clients. Clients execute the attacker's files.

Recommendation

Short term, when using utilities such as `os.MkdirAll` and `ioutil.WriteFile`, check all directories in the path and the final file and validate their owner and permissions before performing operations on them. This will help prevent situations in which sensitive information is written to a pre-existing attacker-controlled path.

Long term, enumerate files and directories and ensure that they have expected permissions and owners. Build validation to ensure appropriate permissions are applied before their creation and upon their use. Ideally, this validation should be centrally defined and used throughout the application.

3. Possible nil pointer dereference

Severity: Informational

Type: Data Validation

Target: orbit/pkg/packaging/macos.go

Difficulty: N/A

Finding ID: TOB-FLT-003

Description

The Orbit packaging utility panics when run on an unsupported machine, that is on neither darwin nor linux. The bug occurs because there is no default handler in a switch statement over runtime.GOOS variable, as presented in figure 3.1.

```
// Make bom
var cmdMkbom *exec.Cmd
switch runtime.GOOS {
case "darwin":
    cmdMkbom = exec.Command("mkbom", filepath.Join(rootPath, "root"),
filepath.Join("flat", "base.pkg", "Bom"))
case "linux":
    cmdMkbom = exec.Command("mkbom", "-u", "0", "-g", "80", filepath.Join(rootPath,
"flat", "root"), filepath.Join("flat", "base.pkg", "Bom"))
}
cmdMkbom.Dir = rootPath
```

Figure 3.1: [orbit/pkg/packaging/macos.go, lines 230 - 236.](#)

Recommendation

Short term, add a default branch to the switch statement that will show an error message and exit. This will prevent application from panic.

Long term, use static analyzers to scan the code for possible nil pointer dereferences.

4. Forcing empty passphrase for keys encryption

Severity: Medium
Type: Cryptography
Target: `fleet/ee/fleetctl/updates.go`

Difficulty: High
Finding ID: TOB-FLT-004

Description

When initializing a new repository, already existing keys will be read and appended to the newly created one. Since the old keys could be created by an attacker, it opens a door for manipulation. Moreover, if the old keys are stored in encrypted form and the encryption passphrase is an empty string, the new key will also be encrypted with an empty password, even if the `FLEET_*_PASSPHRASE` environment variables were set.

The Fleet code tries to detect if no repository exists at the path where a new one should be created - see figure 4.1. It does so by looking for repository and staged directories. However, it does not consider a keys directory to be an indication of repository existence. In fact, it reads the old keys and appends them to the generated one, as can be seen in figure 4.2. This behaviour is not a vulnerability by itself, because only the new key will be used in the `repository/root.json:roles/root/keyids` field. Although, it may cause confusion in the system and introduce new attack vectors.

```
meta, err := store.GetMeta()
if err != nil {
    return errors.Wrap(err, "get repo meta")
}
if len(meta) != 0 {
    return errors.Errorf("repo already initialized: %s", path)
}
```

Figure 4.1: [fleet/ee/fleetctl/updates.go, lines 84 - 90.](#)

```
// add the key to the existing keys (if any)
keys, pass, err := f.loadKeys(role)
if err != nil && !os.IsNotExist(err) {
    return err
}
keys = append(keys, key)
```

Figure 4.2: [go-tuf/blob/master/local_store.go, lines 327 - 332.](#)

For example, an exploitable vector is a key's passphrase overwriting. If an attacker creates keys encrypted with an empty password, saves them in the keys directory and the user initializes the repository using these keys, the Orbit will encrypt the newly generated key with the empty password - not the one set in environment variables. That's because the

go-tuf tries to decrypt files using the empty password firstly, and only if the decryption fails it will try to read the passphrase, as presented in figure 4.3.

```
// try the empty string as the password first
pass := []byte("")
if err := encrypted.Unmarshal(pk.Data, &keys, pass); err != nil {
    pass, err = f.passphraseFunc(role, false)
    if err != nil {
        return nil, nil, err
    }
    if err = encrypted.Unmarshal(pk.Data, &keys, pass); err != nil {
        return nil, nil, err
    }
}
return keys, pass, nil
```

Figure 4.3: [go-tuf/blob/master/local_store.go, lines 404 – 415.](#)

Exploit Scenario

An attacker creates a malicious keys/root.json file, similar to the one presented in figure 4.4, in a directory where the Fleet user will create a new repository. The user executes `fleetctl updates init` command. New keys are generated, encrypted with an empty password, and saved. The attacker later intercepts the new root.json file, decrypts it and attacks Fleet clients.

```
{
  "encrypted": true,
  "data": {
    "kdf": {
      "name": "scrypt",
      "params": {
        "N": 32768,
        "r": 8,
        "p": 1
      },
      "salt": "2hsBqebwy2sSsyPjHwzRoChqS5s3AdiWgaqwtGpt04A="
    },
    "cipher": {
      "name": "nacl/secretbox",
      "nonce": "WMsoB13Q4cv92MKpYsvFk0vaFW5pEvyN"
    },
    "ciphertext":
    "GUd2nqP+RPuR0r3JaakceIY+sGTmiMVb0fgPU/9R5h0gLHns0318vjsjzEG8eCXOrnJVvnQ91wxPrn957neqtqQQde1
    pcn8buArb7z7G54SHGVTnNWSzgBlqy1vgCnVipcInsy8L8ufXg8xBa/cg6xmy4SDPHJ/UWJpqCxdvCJ6IrbkXBHTw1f
    Lb9emvAwdOd+2F9gGmIU2HiC1QQ7HacU4k9ZS+wBdF5Q94bY398dq06sIHPBmWUz9b+AWJ1VONuvgfNHcKH4iiqI3vma
    2fRjtvN4qhQa1Z0BEsC+nm9U28aC4PW9xPg0f12Z7GG71YeH8p40Zj147WRswPQlhwQhTqn8tQrUBvg2RXW6cN1qWuS2
    6/ewUg1vy+BXa4pf9XIndAcyiCNL66gLcsuEkgBz3LCtACmJ3gjAMxRSomTAm6skmebGtrKdEZb/02xtmiu9X/vFQ/TJ
```

```
eBe57G9uf2XceboG31ptzKfBo8S606XiTp4ZvLzdm"  
  }  
}
```

Figure 4.4: an example root.json file encrypted with an empty password.

Recommendation

Short term, check if there is no keys directory before starting creation of a new repository, that is at the beginning of the `updatesInitFunc` function in the `fleet/ee/fleetctl/updates.go` file. Abort the procedure otherwise.

Long term, verify that keys are protected with non-empty passwords after creating them and before using them.

5. Signature verification in fleetctl commands

Severity: High
Type: Data Validation
Target: fleet/ee/fleetctl/updates.go

Difficulty: High
Finding ID: TOB-FLT-005

Description

If Fleet updates commands are executed from a malicious repository, then an attacker may force the user to read and sign his files, effectively compromising keys. That's because the user has no way to establish a root of trust - he lacks knowledge about the root public key.

The `fleetctl updates add` and `fleetctl updates timestamp` commands open the repository located in the current working directory or specified by the `--path` argument. They do not verify signatures of files in the repository. So an attacker with write access to the files - but without access to the private keys - can write arbitrary content to the files and these will be signed by the Fleet user. To protect against such attacks, the `fleetctl` may require users to provide a root public key e.g. as the command line parameter and use it to verify content of the repository prior to updating it.

The `fleetctl updates roots` command, which prints the root public key, could be additionally protected by out-of-band verification of its output. For example, users should be instructed to compute a hash of the printed public key and compare it with the expected root key ID, e.g. the one printed after repository initialization. If a repository initialization takes place after a long time from the `roots` command execution, users should be asked to securely store the ID, e.g. in their password manager or in a file with restricted access. To enable users to correctly compute an ID from a printed public key, default json encoder used in the `updatesRootsFunc` function should be replaced with the github.com/tent/canonical-json-go encoder used by the `go-tuf` library - see figure 5.1.

```
func updatesRootsFunc(c *cli.Context) error {
    [redacted]

    if err := json.NewEncoder(os.Stdout).Encode(keys); err != nil {
        return errors.Wrap(err, "encode root metadata")
    }

    return nil
}
```

Figure 5.1: [fleet/ee/fleetctl/updates.go, lines 157 - 173.](#)

Exploit Scenario

Attacker gets access to a local repository. He replaces the hash of the `orbit` binary in the `targets.json` file and waits. The Fleet user stages updates for the `osqueryd` binary - replaces

the old binary with a new one, updates hashes and other data in repository files, and signs them. Signed target.json file contains the correct hash of the osqueryd, but an incorrect, attacker controlled hash of the orbit. After that, the attacker replaces the orbit file with his malware and waits until the Fleet user publishes the updated repository.

Recommendation

Short term, add a new command line argument to the fleetctl command that will enable users to provide the root public key. Use it for repository validation. Update Fleet documentation so it instructs users to double check public keys printed by the roots command.

<Long term recommendation>

References

- <Any references>

6. Redundant online keys in documentation

Severity: Medium

Type: Access Controls

Target: fleetctl documentation

Difficulty: High

Finding ID: TOB-FLT-006

Description

Fleet documentation states that both snapshot and timestamp keys should be stored in the online environment for the update timestamp operation. However, the snapshot key is not required for the operation and it should be kept in the more secure, offline environment.

The documentation is located in the [4-fleetctl-agent-updates.md](#) file and can be seen in the figure 6.1. The TUF specification states that only the timestamp key should be stored in the online environment - see figure 6.2. Please note that the TUF [faq](#), section 5. Which roles can use online keys? states that "The Timestamp and Snapshot roles can use online keys", but this is not recommended setup in the Orbit context.

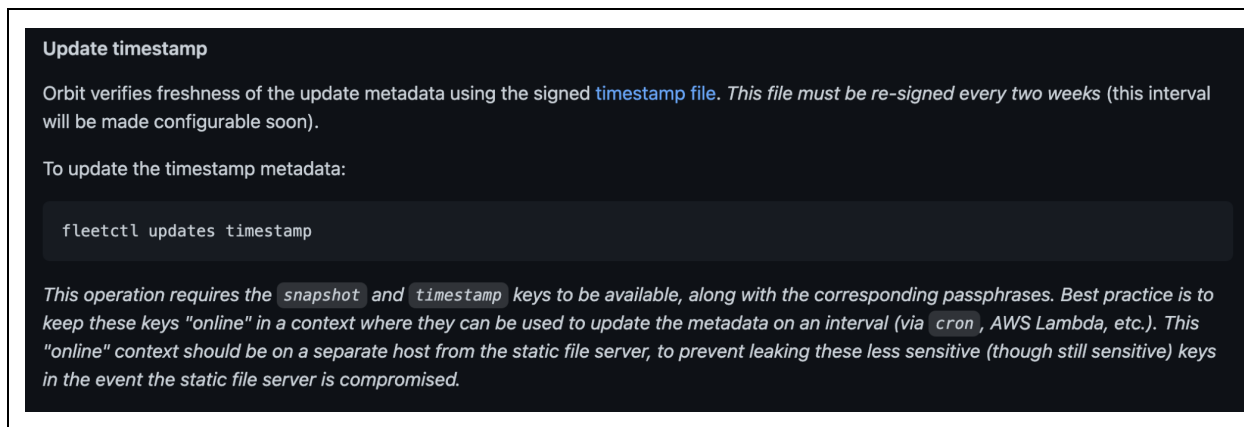


Figure 6.1: [fleet/docs/2-Deployment/4-fleetctl-agent-updates.md#update-timestamp](#).

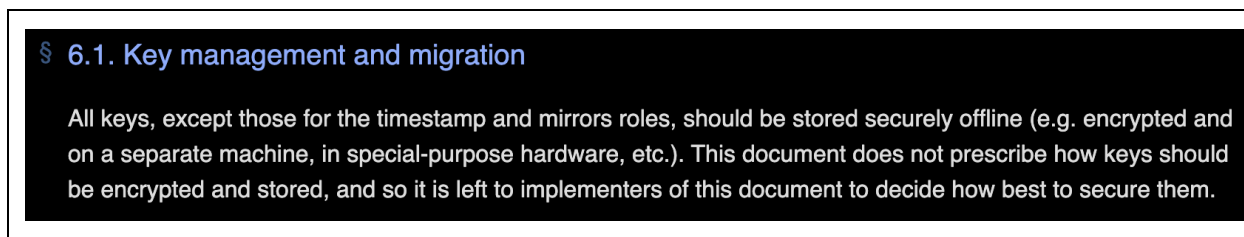


Figure 6.2: [The Update Framework - 6.1. Key management and migration](#).

Moreover, the [Initialize the repository](#) section in the documentation - presented in figure 6.3 - may be misinterpreted by users so that they may assume that only the root key should be stored only in an offline environment, whereas also snapshot and target keys should use such protection.

Exploit Scenario

An attacker compromises the online environment and steals snapshot and timestamp keys. He performs a [mix-and-match](#) attack, forcing users to downgrade the osqueryd file. He can now exploit vulnerabilities existing in the old version of the application.

Recommendation

Short term, rewrite the documentation to require only the timestamp key to be stored in the online environment.

Long term, clearly describe which keys should be stored in what type of an environment and when should be used. For example, create a table with such information. This should help to prevent misinterpretation of the documentation.

7. Lack of alerting mechanism

Severity: High
Type: Configuration
Target: Orbit, Fleet

Difficulty: Medium
Finding ID: TOB-FLT-007

Description

Thanks to The Update Framework, Orbit can detect various attacks, resulting from e.g. remote repository or a key compromise. However, it does not alert a client sufficiently about such security incidents nor does inform the Fleet server.

The only way for a client to notice that his Orbit application is not functioning properly is to periodically scan a log file generated by the application, for example `/var/log/orbit/orbit.stderr.log`.

The Fleet server does not receive any information about errors from the Orbit. It can't detect if enrolled clients are attacked. It also doesn't report versions of the Orbit used by clients.

Exploit Scenario 1

Attacker finds a weakness in the Orbit and exploits it to downgrade Orbit running on clients' machines. Exploits the downgraded version using a more severe attack. Neither Fleet administrators nor clients are aware about the attack.

Exploit Scenario 2

Attacker compromises the remote repository. He stops publishing new updates. Clients are running an outdated, vulnerable version of the Orbit and are not aware of this fact.

Recommendation

Short term, implement a notification mechanism for Orbit users, e.g. as a system notifications or a communication channel like an email. Alert users if a serious security incident is detected, e.g. an [indefinite freeze attack](#). Print versions of the Orbit and osquery in the Fleet administration panel - the same versions as used by the go-tuf, not a version reported by the osquery.

Long term, consider sending a notification from the Orbit client to Fleet server when the client encounters a failure or spots an attack against him. Create documentation instructing clients how they should respond to detected security incidents, based on the type of the error received.

8. Key rotation methodology is not documented

Severity: Medium
Type: Cryptography
Target: fleetctl

Difficulty: High
Finding ID: TOB-FLT-008

Description

One of the goals of The Update Framework is that “all keys must be easily and safely revocable. Trusting new keys for a role must be easy”. However, the Fleet documentation does not provide information about how to safely rotate expired keys or revoke compromised ones.

Keeping Fleet administrators rotating keys means that a technology and policy is in place to distribute new and old keys, and that they know how to handle the situation if keys are compromised. The opposite - not having keys rotation trained - could delay administrators' response time in case of a security breach and increase the period of clients being vulnerable to attacks.

Based on the NIST's [“Recommendation for Key Management”](#) publication, section 5.3. Cryptoperiods, Google's [guide](#), and AWS's [guide](#) - with a note that the two later guides refer to symmetric keys - we recommend to set key expiration times between 1 to 3 years. The exact time should be configurable by end users, depending on the specific requirements and capabilities, e.g. availability of a secure environment. Moreover, the rotation period should be adjusted to comprehend thresholds and redundant keys - see finding [TOB-FLT-009](#).

Based on the TUF [faq](#), section 8. How often should metadata expire?, Uptane's [best practices recommendations](#), and Notary [recommendations](#), we recommend to set timestamp metadata expiration to a short time period, e.g. 1 day - this should ensure that clients are not running outdated applications longer than a day. For the rest metadata files, we recommend setting a longer period, between 1 to 3 years. The root metadata may have longer expiration time than the storage and target files. The storage metadata may have a shorter validity period, to protect against scenarios when timestamp key and remote repository are both compromised. The metadata expiration times should depend on the client-specific keys storage setup and should be correlated with the cryptoperiods of the corresponding keys. Especially, all expiration times should be less than or equal to the root keys cryptoperiod.

Exploit Scenario

An attacker compromises some of the Fleet keys. Administrators learn about this fact and try to revoke the keys. They have no instruction how to do that, so the operation is delayed. The attacker has enough time to compromise clients and install a persistent backdoor.

Recommendation

Short term, implement commands in the `fleetctl` tools that will allow users to easily replace expired or compromised keys. Make sure that users can renew only selected keys and don't have to replace all of them. Add flags that will enable control over keys expiration times and set secure defaults for these. Do the same for metadata expiration times.

Long term, implement integration with hardware security modules to further increase security of keys management. Implement notification mechanism that will alert administrators about expired metadata files.

9. Threshold and redundant keys

Severity: Informational
Type: Cryptography
Target: Orbit, Fleet

Difficulty: N/A
Finding ID: TOB-FLT-009

Description

The Update Framework supports roles with multiple (redundant) keys and a threshold trust. These features can be leveraged for additional protection of the keys.

For example, instead of a single root key [replicated to a multiple locations](#), multiple keys can be generated and distributed across different USB drives (or other devices) and the Orbit can be instructed to trust any one of these. Such setup should be useful when recovering from keys compromise - if only some of the keys were identified to be stolen by attackers, the still-trustful can be leveraged to reestablish the trust. However, such a feature is probably not supported by the TUF. This setup will also make it easier to detect which device was compromised when one of the keys was noticed to be used maliciously.

The threshold feature can be used to further increase the security. For example, at least two root keys may be required in the signing process. Now both can be stored on separate devices and compromising one of them won't be enough for an attacker to attack the system. Combining this feature with redundant keys will allow the implementation of the M-of-N scheme (similar to a secret sharing algorithms): there are N valid keys and M of these are required to meet the threshold. This gives both protection against a single device compromise and backup keys.

Other keys than the for the root role also can benefit from redundancy and the threshold. For example, the timestamp role may require two keys and the keys can now be retrieved from two places e.g., from a database and an environment variable.

Recommendation

Short term, extend fleetctl tool with support for customizable thresholds and multiple keys for a single role. Document how to use this feature. Recommend users to use it for the root role keys.

Long term, consider to use multiple roles keys and thresholds for not-root-role keys.

10. Database compaction function could be called more times than expected

Severity: Informational
Type: Undefined Behavior
Target: Orbit

Difficulty: N/A
Finding ID: TOB-FLT-010

Description

The orbit application created a background loop to call the compaction method for BadgerDB. In some edge cases, the way this logic was configured could result in more calls to RunValueLogGC than expected.

In the code below, if the closeChan is written to at the same time as the ticker executes, then it would up to the Go scheduler to randomly pick a case to select. As a result, it may choose to run b.DB.RunValueLogGC(compactionDiscardRatio) one extra time unnecessarily instead of exiting the loop. While this may not result in any specific issues, this could result in undefined behavior in the future as the code base is further developed.

```
go func() {
    ticker := time.NewTicker(compactionInterval)
    defer ticker.Stop()
    for {
        select {
            case <-b.closeChan:
                return

            case <-ticker.C:
                if err := b.DB.RunValueLogGC(compactionDiscardRatio); err != nil &&
!errors.Is(err, badger.ErrNoRewrite) {
                    log.Error().Err(err).Msg("compact badger")
                }
        }
    }
}()
```

Figure 10.1: [orbit/pkg/database/database.go, lines 50 – 64.](#)

Recommendation

Short term, add an additional select case at the beginning of the loop to avoid having to call the RunValueLogGC in the edge case described above. A potential fix is shown below:

```
for {
    // handle cases where both ticker.C and closeChan are written to at the same time
    select {
        case <-b.closeChan:
```



```

        return
    default: // we add default here so that the select doesn't block
    }

    if err := b.DB.RunValueLogGC(compactionDiscardRatio); err != nil && !errors.Is(err,
badger.ErrNoRewrite) {
        Log.Error().Err(err).Msg("compact badger")
    }

    select {
    case <-b.closeChan:
        return
    case <-ticker.C:
    }
}

```

Figure 10.2: Potential fix to avoid potential extra call to RunValueLogGC.

Long term, use the Semgrep query in [Appendix D](#) to periodically check for and detect this type of issue.

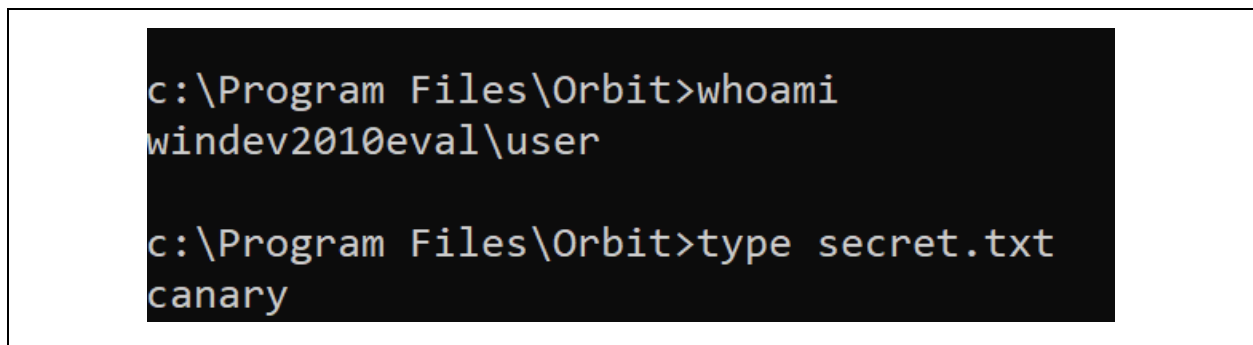
11. All Windows users have read access to Fleet server secret

Severity: High
Type: Access Controls
Target: Orbit

Difficulty: Medium
Finding ID: TOB-FLT-011

Description

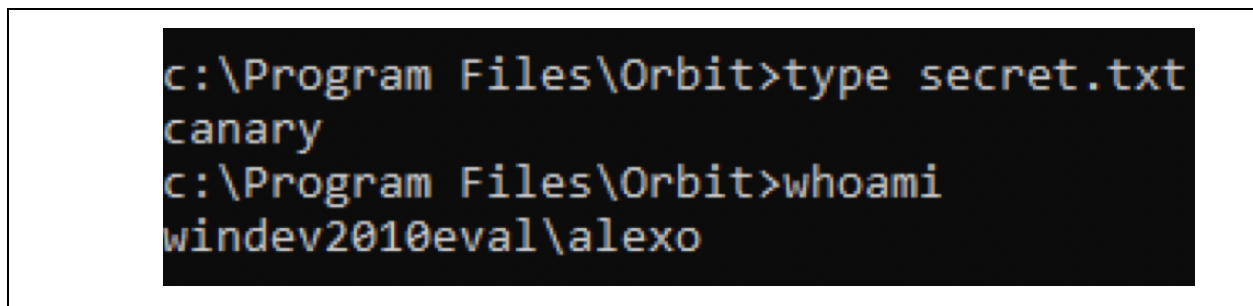
After installation of the orbit package on Windows machines, the installer stores the enrollment secret in `c:\Program Files\Orbit\secret.txt`. This secret is readable by anyone with a standard Windows user account in the machine. This is inconsistent with macOS and Linux, where non-root users do not have read access to the secret. This appeared to be due to insufficient restrictions applied by the Security Descriptor Definition Language (SDDL) strings used to configure access rights in Windows hosts.



```
c:\Program Files\Orbit>whoami
windev2010eval\user

c:\Program Files\Orbit>type secret.txt
canary
```

Figure 11.1: The secret was readable by the user that installed orbit through the msi installer.



```
c:\Program Files\Orbit>type secret.txt
canary

c:\Program Files\Orbit>whoami
windev2010eval\alexo
```

Figure 11.2: The secret was readable by another user without administrator rights with access to the same machine.

Exploit Scenario

An attacker with local access to the file system where the Orbit data is stored reads the `secret.txt` file. He then exploits a flaw in the system breaking connection between the original osquery and a Fleet server. He uses the secret to enroll in the Fleet server. The server is connected to a malicious, fake osquery. Attacker responds with untrue data for Fleets osquery requests without the Fleet noticing.

Recommendation

Short term, restrict read access to all non-admin users to all files under C:\Program Files\Orbit\, including the Fleet server secret.

Long term, create and incorporate unit tests in the CI/CD pipeline that verify that non-privileged users are not able to read the Fleet server secret, even after changes are made to the code.

12. Insufficient documentation of SDDL permissions

Severity: Low
Type: Auditing and Logging
Target: Orbit

Difficulty: Medium
Finding ID: TOB-FLT-012

Description

The Orbit source code relied on SDDL strings to configure file permissions for Orbit files installed in Windows computers through the MSI package. SDDL strings are unintuitive and complex, and their complicated syntax can make it easy for developers to create invalid or incorrect permissions when updating them. However, very few details were provided in the documentation or code comments in regard to the intent of the SDDL strings used in the code.

```
if cur.XMLName.Local == "File" {
    // This SDDL copied directly from osqueryd.exe after a regular
    // osquery MSI install. We assume that osquery is getting the
    // permissions correct and use exactly the same for our files.
    // Using this cryptic string seems to be the only way to disable
    // permission inheritance in a WiX package, so we may not have
    // any option for something more readable.
    sddl := "O:SYG:SYD:P(A;OICI;FA;;;SY)(A;OICI;FA;;;BA)(A;OICI;0x1200a9;;;BU)"
    if cur.Attrs.Get("Name") == "secret.txt" {
        // This SDDL copied from properly configured file on a Windows
        // 10 machine. Permissions are same as below but with read
        // access removed for regular users.
        sddl = "O:SYG:SYD:PAI(A;;FA;;;SY)(A;;FA;;;BA)"
    }
    cur.Children = append(cur.Children, xmlNode(
        "PermissionEx",
        xmlAttr("Sddl", sddl),
    ))
}
```

Figure 12.1: [orbit/pkg/packaging/wix/transform.go, lines 80 – 99.](#)

Exploit Scenario

A developer needs to make a change to the SDDL strings and, given the lack of sufficient documentation on the intended permissions configured through the SDDL strings, they mistakenly grant users more access than necessary to sensitive files such as the Fleet server secret.

Recommendation

Short term, add additional comments next to each SDDL string used in code describing in detail the intended permissions and trustees.

A. Vulnerability Classifications

Vulnerability Classes	
Class	Description
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 protecting the privacy or integrity of 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
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	Description
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important

Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	Description
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	Description
Access Controls	Related to the authentication and authorization of components.
Arithmetic	Related to the proper use of mathematical operations and semantics.
Assembly Use	Related to the use of inline assembly.
Centralization	Related to the existence of a single point of failure.
Upgradeability	Related to contract upgradeability.
Function Composition	Related to separation of the logic into functions with clear purpose.
Front-Running	Related to resilience against front-running.
Key Management	Related to the existence of proper procedures for key generation, distribution, and access.
Monitoring	Related to use of events and monitoring procedures.
Specification	Related to the expected codebase documentation.
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.).

Rating Criteria	
Rating	Description
Strong	The component was reviewed and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Code Quality Recommendations

Usage of IsNotExist

There were two instances where the function IsNotExist was used to check whether a file or path existed. Per the [Go documentation](#), new code should use `errors.Is(err, os.ErrNotExist)` for the same purposes. The two instance are listed below

```
if err := os.Rename(localPath, localPath+".old"); err != nil && !os.IsNotExist(err) {  
    return errors.Wrap(err, "rename old")  
}
```

Figure C.1: [/orbit/pkg/update/update.go#L230-L232](#)

```
if err != nil && !os.IsNotExist(err) {  
    return errors.Wrap(err, "stat file store")  
} else if os.IsNotExist(err) {  
    // initialize empty  
    s.metadata = metadataMap{}  
    return nil  
}
```

Figure C.2: [/orbit/pkg/update/filestore/filestore.go#L57-L63](#)

Unchecked type assertion

The following code performs a type assertion without validating its correctness. `http.DefaultTransport.(*http.Transport)` returns `nil`, `false` for the assertion, `Clone()` is called on a `nil` pointer, which causes a panic.

```
func New(opt Options) (*Updater, error) {  
    transport := http.DefaultTransport.(*http.Transport).Clone()  
}
```

Figure C.3: [/orbit/pkg/update/update.go#L61-L62](#)

D. Semgrep rule to detect redundant Go channel selection

The following Semgrep rule can be used to detect instances of the bug described in [TOB-FLT-010](#).

```
rules:
- id: undeterministic-function-execution
  patterns:
  - pattern-either:
    - pattern: |
      $TICKER := time.NewTicker(...)
      ...
      for {
        ...
        select {
          case <-$DONECHAN: return
          case <-ticker.C: ...
        }
      }
    - pattern-not: |
      $TICKER := time.NewTicker(...)
      ...
      for {
        select {
          case <-$DONECHAN: return
          default:
        }
        ...
        select {
          case <-$DONECHAN: return
          case <-$TICKER.C: ...
        }
      }
  message: Logic executed as a result of ticker $TICKER may execute more times than desired
  when both channels are written to at the same time
  severity: WARNING
  languages: [go]
```

Figure D.1: Semgrep Rule.