

Flux

Security Assessment

November 8, 2023

Prepared for: **Hidde Beydals** Open Source Technology Improvement Fund (OSTIF)

Prepared by: Maciej Domański and Sam Alws

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.

228 Park Ave S #80688 New York, NY 10003 https://www.trailofbits.com info@trailofbits.com

Notices and Remarks

Copyright and Distribution

© 2023 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to OSTIF under the terms of the project statement of work and has been made public at OSTIF's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.



Table of Contents

About Trail of Bits	1
Notices and Remarks	2
Table of Contents	3
Executive Summary	4
Project Summary	6
Project Goals	7
Project Targets	9
Project Coverage	10
Automated Testing	11
Codebase Maturity Evaluation	12
Summary of Findings	14
Detailed Findings	15
1. SetExpiration does not set the expiration for the given key	15
2. Inappropriate string trimming function	17
3. Go's default HTTP client uses a shared value that can be modified by other	
components	18
4. Unhandled error value	20
5. Potential implicit memory aliasing in for loops	22
6. Directories created via os.MkdirAll are not checked for permissions	24
7. Directories and files created with overly lenient permissions	25
8. No restriction on minimum SSH RSA public key bit size	26
9. Flux macOS release binary susceptible to .dylib injection	28
10. Path traversal in SecureJoin implementation	30
A. Vulnerability Categories	33
B. Code Maturity Categories	35
C. Non-Security-Related Findings	37
D. Issue Occurrences	39
E. Automated Static Analysis	42
F. Fix Review Results	44
Detailed Fix Review Results	46
G. Fix Review Status Categories	48



Executive Summary

Engagement Overview

OSTIF engaged Trail of Bits to review the security of Flux, a tool for keeping Kubernetes clusters in sync with configuration sources.

A team of two consultants conducted the review from July 24 to August 4, 2023, for a total of four engineer-weeks of effort. Our testing efforts focused on the elements that are part of the General Availability release. With full access to source code and documentation, we performed static and dynamic testing of the Flux tool, using automated and manual processes.

Observations and Impact

Trail of Bits found that Flux is well structured and generally written defensively. However, we identified one undetermined-severity finding, TOB-FLUX-10, that poses an immediate risk to users if the underlying package is treated as a standalone library because its main security guarantee of preventing unauthorized read/write operations outside the root directory has been proven false.

We did not identify any other findings that present an immediate threat to Flux or its users. However, we did identify findings that could have been uncovered with more robust unit testing (TOB-FLUX-1 and TOB-FLUX-2). By expanding unit test coverage, Flux can further enhance its resilience.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that OSTIF take the following steps:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Implement static analysis tools in the CI/CD pipeline.** Implementing additional tools presented in appendix E will help automatically find issues in the code that could lead to security vulnerabilities before they are merged into the codebase.



Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	Count
High	0
Medium	0
Low	3
Informational	6
Undetermined	1

CATEGORY BREAKDOWN

Category	Count
Access Controls	1
Configuration	2
Data Validation	3
Error Reporting	1
Undefined Behavior	3



Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager	Jeff Braswell, Project Manager
dan@trailofbits.com	jeff.braswell@trailofbits.com

The following engineers were associated with this project:

Maciej Domański, Consultant	Sam Alws , Consultant
maciej.domanski@trailofbits.com	sam.alws@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
July 20, 2023	Pre-project kickoff call
July 31, 2023	Status update meeting #1
August 4, 2023	Delivery of report draft
August 4, 2023	Report readout meeting
October 20, 2023	Delivery of report draft with fix review
November 8, 2023	Delivery of comprehensive report with fix review

Project Goals

The engagement was scoped to provide a security assessment of the Flux tool. Specifically, we sought to answer the following non-exhaustive list of questions:

- Does the codebase conform to industry best practices?
- Are the system architecture and design foundationally secure?
- Are there any data exposures to or data extractions by unknown or unauthorized sources?
- Can Flux be used to deliver malicious payloads and executables?
- Does Flux correctly use the Kubernetes API extension system and other core components of the Kubernetes ecosystem?
- Does Flux securely handle credential storage and use?
- Are there appropriate access controls on critical functions?
- Are there areas within ownership and access controls that may be compromised or altered to cause adverse states, unauthorized access, or exploitation?
- Can security constraints when syncing repositories and files be bypassed?
- Can files outside the designated file structure be replaced and/or modified?
- Could the system experience a denial of service (DoS)?
- Are all inputs and system parameters validated correctly?
- Do adequate account management, security controls, and separation exist to operate the accounts safely?
- How are automated testing and validation of security controls in pipelines performed?
- Are strong sign-in mechanisms used? How long do credentials last?
- What security mechanisms are used to store secrets?
- How are account groups, permissions, and attributes provisioned securely?
- How are public and cross-account access mechanisms managed?



- How are shared resources managed and secured?
- How are service and application logging configured and monitored?
- How are data and customer information protected at rest and in transit?
- If supporting a multi-tenant environment, how is isolation implemented between the tenants? What resources are shared between tenants?
- Are access controls for cross-namespace objects implemented securely?



Project Targets

The engagement involved a review and testing of the targets listed below.

kustomize-controller

Repository	https://github.com/fluxcd/kustomize-controller
Version	8d9a1811655fff9a093f9c98397e2ed806876f10
Туре	Golang
Platform	Linux

source-controller

Repository	https://github.com/fluxcd/source-controller
Version	7f40be76e90b2d4afe9f8f9d7f53ac719fe1205e
Туре	Golang
Platform	Linux

notification-controller

Repository	https://github.com/fluxcd/notification-controller
Version	b80c2c4060f62af40c06fe2f6f3bef295ee56e43
Туре	Golang
Platform	Linux
flux2	
Repository	https://github.com/fluxcd/flux2
Version	44d69d6fc0c353e79c1bad021a4aca135033bce8
Туре	Golang
Platform	Linux
pkg	
Repository	https://github.com/fluxcd/pkg
Version	2a323d771e17af02dee2ccbbb9b445b78ab048e5
Туре	Golang
Platform	Linux



Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- Manually reviewing the provided repositories with a focus on the controllers with the General Availability components:
 - source-controller
 - kustomize-controller
 - notification-controller
 - o flux2
 - The pkg repository—in particular, the git/gogit/fs component
- Running static analysis tools and triaging results

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- We did not review the helm-controller, image-automation-controller, and image-reflector-controller components since they are not General Availability components.
- We did not thoroughly review the "Flux Multi-tenancy Threat Modelling" document. However, it was the basis for our assumptions and potential attack scenarios.
- We did not review unit, end-to-end, or integration tests for completeness, nor did we evaluate the fuzz testing coverage.
- We did not review whether logging information was sufficient.
- The list of outdated dependencies and deprecated methods was not included in our assessment. Instead, we focused on analyzing the code of third-party libraries while reviewing specific components.



Automated Testing

Trail of Bits uses automated techniques to extensively test the security properties of software. We use both open-source static analysis and fuzzing utilities, along with tools developed in house, to perform automated testing of source code and compiled software.

Test Harness Configuration

Tool	Description	Policy
Semgrep	A static analysis tool designed to identify bugs and specific code patterns across multiple languages	Appendix E
CodeQL	A code analysis engine developed by GitHub to automate security checks	Appendix E
TruffleHog	An open-source tool that scans Git repositories for secrets such as private keys and API tokens	Appendix E
golangci-lint	A Go linters aggregator	Appendix E

We used the following tools in the automated testing phase of this project:

Areas of Focus

Our automated testing and verification work focused on the following system properties:

- The system does not produce undefined behavior.
- The code does not contain security or quality issues.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	The application's primary purpose does not involve mathematical operations; however, as with any software, arithmetic operations are present. We found no significant issues concerning the proper use of mathematical operations.	Satisfactory
Auditing	The density and quality of logged information is sufficient. However, we did not try to verify this for all execution paths or verify whether all information required to perform incident response is always logged.	Further Investigation Required
Authentication / Access Controls	The Kubernetes role-based access controls (RBACs) follow best practices. RBAC impersonation is used to limit the permissions of tenants.	Satisfactory
Complexity Management	Overall, the Flux codebase has a logical organization and clear structures to manage the system's complexity. It is possible for a new developer to quickly understand the structure of the Flux codebase. However, we found duplicate code that uses two distinct SecureJoin implementations from different packages with nearly identical implementations (appendix C, item 4).	Satisfactory
Configuration	We found that specific components are generally configured securely. However, some directories have overly lenient permissions (TOB-FLUX-7). Additionally, consider hardening the macOS release binary against potential .dylib hijacking (TOB-FLUX-9).	Satisfactory

Cryptography and Key Management	We found no major issues related to cryptography.	Satisfactory
Data Handling	Generally, Flux takes the necessary precautions when validating most data types; however, we found that an inappropriate string trimming function is used (TOB-FLUX-2) and a minimum RSA public key bit size is not validated (TOB-FLUX-8).	Satisfactory
Documentation	User-facing documentation is thorough, with getting- started guides, setup examples, and API references. In addition, the code contains fairly thorough comments. Nevertheless, we recommend completing documentation with warnings to users about potentially dangerous options and their implications (e.g., passing a password as a CLI argument).	Satisfactory
Maintenance	While our assessment did not prioritize checking for outdated third-party dependencies, our brief analysis of certain components revealed outdated security-related libraries. Implementing govulncheck could improve maintenance efforts for Golang code.	Further Investigation Required
Memory Safety and Error Handling	Flux is written in Go, which reduces its exposure to memory safety issues. However, we found a minor issue related to an unhandled error value (TOB-FLUX-4).	Satisfactory
Testing and Verification	The codebase is verified using various tests, including fuzz tests. Due to the time constraints of the audit, we did not evaluate the thoroughness of the tests. However, we found that some functions do not work properly (TOB-FLUX-1), which could be caught with the unit test that covers the identified function. We also recommend customizing a CodeQL GitHub workflow with a more detailed security-and-quality query suite (TOB-FLUX-4).	Further Investigation Required



Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Туре	Severity
1	SetExpiration does not set the expiration for the given key	Undefined Behavior	Low
2	Inappropriate string trimming function	Data Validation	Informational
3	Go's default HTTP client uses a shared value that can be modified by other components	Undefined Behavior	Low
4	Unhandled error value	Error Reporting	Informational
5	Potential implicit memory aliasing in for loops	Undefined Behavior	Informational
6	Directories created via os.MkdirAll are not checked for permissions	Access Controls	Informational
7	Directories and files created with overly lenient permissions	Configuration	Informational
8	No restriction on minimum SSH RSA public key bit size	Data Validation	Informational
9	Flux macOS release binary susceptible to .dylib injection	Configuration	Low
10	Path traversal in SecureJoin implementation	Data Validation	Undetermined

Detailed Findings

1. SetExpiration does not set the expiration for the given key		
Severity: Low	Difficulty: High	
Type: Undefined Behavior	Finding ID: TOB-FLUX-1	
Target: source-controller/internal/cache/cache.go#163-172		

Description

The SetExpiration function does not change the expiration for the given key because it does not store the updated item back in the specific cache item (figure 1.1).

The SetExpiration function retrieves the corresponding item from the cache and assigns it to the item variable (figure 1.1, line 165). Then it updates the item's expiration time by setting its Expiration field to the current time plus the provided expiration duration (figure 1.1, line 170). Finally, the lock on the cache is released without the prior cache update (figure 1.1, line 171), so any subsequent access to the cache item with the given key will not see the updated expiration set by SetExpiration.

```
163
       func (c *cache) SetExpiration(key string, expiration time.Duration) {
164
            c.mu.Lock()
            item, ok := c.Items[key]
165
            if !ok {
166
                   c.mu.Unlock()
167
168
                   return
169
            }
            item.Expiration = time.Now().Add(expiration).UnixNano()
170
171
            c.mu.Unlock()
172
       }
```

Figure 1.1: The SetExpiration function responsible for setting the expiration for the given key (source-controller/internal/cache.go#163-172)

Exploit Scenario

A developer intentionally places sensitive data with a specific expiration date in the cache. An attacker gains access to confidential information because the sensitive data has not expired. This allows the attacker to further compromise the system.

Recommendations

Short term, explicitly assign the updated item variable back to the c.Items map before releasing the lock (figure 1.2).



```
func (c *cache) SetExpiration(key string, expiration time.Duration) {
    c.mu.Lock()
    if item, ok := c.Items[key]; ok {
        item.Expiration = time.Now().Add(expiration).UnixNano()
            c.Items[key] = item
    }
    c.mu.Unlock()
}
```

Figure 1.2: The proposed fix that updates the expiration time correctly

Long term, extend unit tests in the cache_test.go file to cover the SetExpiration function.



2. Inappropriate string trimming function		
Severity: Informational	Difficulty: High	
Type: Data Validation	Finding ID: TOB-FLUX-2	
Target: notification-controller/internal/server/receiver_handlers.go#71-77		

The handlePayload function fails to remove a specific substring as intended because its implementation uses the strings.TrimLeft function (figure 2.1). The incoming HTTP request URL (r.RequestURI) is passed to the strings.TrimLeft function with the apiv1.ReceiverWebhookPath parameter, which is set to /hook (figure 2.1, line 74). The goal is to remove this specific substring from r.RequestURI. However, due to the use of strings.TrimLeft, all occurrences of the specified characters, instead of just the exact substring, are removed from the left side of the string. Consequently, the handling request path is incorrectly logged (figure 2.1, line 76).

```
71 func (s *ReceiverServer) handlePayload() func(w http.ResponseWriter, r
*http.Request) {
72 return func(w http.ResponseWriter, r *http.Request) {
73 ctx := context.Background()
74 digest := url.PathEscape(strings.TrimLeft(r.RequestURI,
apiv1.ReceiverWebhookPath)) // apiv1.ReceiverWebhookPath = "/hook"
75
76 s.logger.Info(fmt.Sprintf("handling request: %s", digest))
```

Figure 2.1: The use of strings.TrimLeft in the handlePayload function (notification-controller/internal/server/receiver_handlers.go#71-77)

Recommendations

Short term, fix the handlePayload function to properly remove substrings from the remote URL using strings.TrimPrefix function.

Long term, implement unit tests for all string-parsing functions. In the CI/CD pipeline, introduce the **golangci-lint** tool that uses the **Staticcheck** tool with the **SA1024** check.



3. Go's default HTTP client uses a shared value that can be modified by other components

Severity: Low	Difficulty: High	
Type: Undefined Behavior	Finding ID: TOB-FLUX-3	
Target:flux2/pkg/manifestgen/install/install.go#91–97, flux2/pkg/manifestgen/install/install.go#118–125		

Description

Go's default HTTP client uses a shared http.DefaultClient value that can be modified by other application components, which leads to unexpected behavior. In the case of Flux, the issue arises in the GetLatestVersion and ExistingVersion functions, where the timeout is modified.

91 // 0	SetLatestVersion calls the GitHub API and returns the latest released version
92 f	<pre>func GetLatestVersion() (string, error) {</pre>
93	ghURL := "https://api.github.com/repos/fluxcd/flux2/releases/latest"
94	<pre>c := http.DefaultClient</pre>
95	c.Timeout = 15 * time.Second
96	
97	res, err := c.Get(ghURL)

Figure 3.1: The GetLatestVersion function that uses http.DefaultClient (flux2/pkg/manifestgen/install/install.go#91-97)

```
118 func ExistingVersion(version string) (bool, error) {
    // (...)
123 ghURL :=
fmt.Sprintf("https://api.github.com/repos/fluxcd/flux2/releases/tags/%s", version)
124 c := http.DefaultClient
125 c.Timeout = 15 * time.Second
```

Figure 3.2: The ExistingVersion function that uses http.DefaultClient (flux2/pkg/manifestgen/install/install.go#118-125)

Exploit Scenario

An attacker introduces a malicious library into the Flux codebase that can modify the shared http.DefaultClient value. By manipulating this value, the attacker orchestrates DoS attacks, disrupting the software's normal operation.



Recommendations

Short term, avoid using the shared http.DefaultClient value and instead use the go-cleanhttp package to ensure that the HTTP client configuration remains unaffected by other parts of the application.

Long term, periodically audit other global values that may impact different components within Flux.

References

- hashicorp/go-cleanhttp—wrapping functions for accessing "clean" Go http.Client values
- PoC showing shared global variable used by the default HTTP client



4. Unhandled error value		
Severity: Informational	Difficulty: High	
Type: Error Reporting	Finding ID: TOB-FLUX-4	
Target:flux2/cmd/flux/events.go#129-138		

The eventsCmdRun function in the flux2 repository ignores an error value returned by a call to the getRows function. This can result in incorrect error reporting to the user.

```
129
        rows, err := getRows(ctx, kubeclient, clientListOpts, refListOpts,
showNamespace)
130
        if len(rows) == 0 {
             if eventArgs.allNamespaces {
131
                    logger.Failuref("No events found.")
132
133
             } else {
134
                    logger.Failuref("No events found in %s namespace.",
*kubeconfigArgs.Namespace)
135
             }
136
137
             return nil
138
        }
```

Figure 4.1: Ignored err value (flux2/cmd/flux/events.go#129-138)

The getRows function returns a nil value in the rows variable whenever it returns an error, which means the if statement's condition on line 130 will be satisfied. The if statement body will incorrectly report to the user that no events were found, rather than printing the err value.

Recommendations

Short term, add an err != nil check and modify the eventsCmdRun function to handle error values accordingly (print an error message and then return err), as shown in the following figure:

```
rows, err := getRows(ctx, kubeclient, clientListOpts, refListOpts, showNamespace)
if err != nil {
    logger.Failuref("Error while getting rows: %s", err)
    return err
}
if len(rows) == 0 {
    if eventArgs.allNamespaces {
        logger.Failuref("No events found.")
```

```
} else {
    logger.Failuref("No events found in %s namespace.",
*kubeconfigArgs.Namespace)
    }
    return nil
}
```

Figure 4.2: Fixed code snippet

Long term, ensure that there are no other places in the Flux codebase where error values are ignored. Adding CodeQL to the project CI/CD with the queries: security-and-quality option will allow the go/useless-assignment-to-local query to catch similar issues.



5. Potential implicit memory aliasing in for loops		
Severity: Informational	Difficulty: High	
Type: Undefined Behavior	Finding ID: TOB-FLUX-5	
Target: Various locations		

Throughout the Flux codebase, loop range values are passed by reference to functions. This reference is unstable and is updated at each iteration of the for loop. Here are two examples:

```
for _, resource := range resources.Items {
    if err := s.annotate(ctx, &resource); err != nil {
```

Figure 5.1: Example of memory aliasing in a for loop (notification-controller/internal/server/receiver_handlers.go#411-412)

```
for _, i := range list.Items {
    if !bucket.GetArtifact().HasRevision(i.Status.ObservedSourceArtifactRevision)
{
        reqs = append(reqs, reconcile.Request{NamespacedName:
        client.ObjectKeyFromObject(&i)})
```

Figure 5.2: Example of memory aliasing in a for loop (source-controller/internal/controller/helmchart_controller.go#1312-1314)

We did not find any examples where this results in a security problem. However, it is generally a very unsafe practice; if any of these function calls preserved their input values (e.g., by storing them in structs), the stored value would be changed while the for loop was iterating.

A full list of occurrences of this issue can be found in appendix D.

Recommendations

Short term, replace these references with more permanent ones. Here are two possible ways to do this:

```
for i, v := range l {
    // option 1: reference the entry in the list
    // the reference still only lasts as long as the list does
```



```
foo(&l[i])
// option 2: copy the value before calling the function
vClone := v
foo(&vClone)
}
```

Figure 5.3: Safer ways to pass a reference to a function

Long term, implement the **gosec** tool in the project CI/CD to catch potential issues with Golang.

References

• Beware of Implicit Memory Aliasing in Go For Loop



6. Directories created via os.MkdirAll are not checked for permissions

Severity: Informational	Difficulty: High	
Type: Access Controls	Finding ID: TOB-FLUX-6	
Target: Various locations		

Description

Flux creates certain directory paths with specific access permissions by using the os.MkdirAll function. This function does not perform any permission checks when a given directory path already exists. This would allow a local attacker to create a directory with broad permissions before Flux could create the directory with narrower permissions, possibly allowing the attacker to later tamper with the files.

A full list of occurrences of this issue can be found in appendix D.

Exploit Scenario

Eve has unprivileged access to a container running a Flux controller. Eve introduces new directories or paths with 0777 permissions before the Flux code does so. Eve then deletes and forges files in that directory to change the result of further code executed by the Flux controller.

Recommendations

Short term, when using functions such as os.MkdirAll, os.WriteFile, or outil.WriteFile, check all directories in the path and validate their owner and permissions before performing operations on them. This will help avoid situations where sensitive information is written to a preexisting attacker-controlled path.

Long term, enumerate files and directories for their expected permissions, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the application as a whole.

7. Directories and files created with overly lenient permissions		
Severity: Informational	Difficulty: High	
Type: Configuration	Finding ID: TOB-FLUX-7	
Target: Various locations		

Flux creates various directories and files with overly lenient permissions. This would allow an attacker with unprivileged access to edit, delete, and read files, interfering with Flux controllers' operations.

if err := os.MkdirAll(abs, 00755); err != nil {

Figure 7.1: Example of a directory created with overly lenient permissions (pkg/tar/tar.go#167)

err = os.WriteFile(path, out, 00644)

Figure 7.2: Example of a file created with overly lenient permissions (kustomize-controller/internal/decryptor/decryptor.go#505)

A full list of occurrences of this issue can be found in appendix D.

Recommendations

Short term, generally use permissions of 0750 or less for directories and 0600 or less for files.

Long term, enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the application as a whole.

8. No restriction on minimum SSH RSA public key bit size		
Severity: Informational	Difficulty: High	
Type: Data Validation	Finding ID: TOB-FLUX-8	
Target: flux2/internal/flags/rsa_key_bits.go		

Flux does not restrict a user from creating a Kubernetes secret for Git authentication using a dangerous SSH RSA public key bit size (figure 8.1). A user can create a configuration with a 16-bit key size (figure 8.2), which is insecure because an attacker can easily brute force the correct private key that matches the public key.

```
var defaultRSAKeyBits = 2048
type RSAKeyBits int
// (...)
func (b *RSAKeyBits) Set(str string) error {
      if strings.TrimSpace(str) == "" {
             *b = RSAKeyBits(defaultRSAKeyBits)
             return nil
      }
      bits, err := strconv.Atoi(str)
      if err != nil {
             return err
      }
      if bits == 0 || bits%8 != 0 {
             return fmt.Errorf("RSA key bit size must be a multiples of 8")
      }
      *b = RSAKeyBits(bits)
      return nil
}
```

Figure 8.1: The Set function responsible for the --ssh-rsa-bits parameter validation (flux2/internal/flags/rsa_key_bits.go#25-47)

```
$ flux create secret git podinfo-auth \
      --url=ssh://git@github.com/stefanprodan/podinfo \
      --export --ssh-rsa-bits 16 --ssh-key-algorithm=rsa
apiVersion: v1
kind: Secret
metadata:
    name: podinfo-auth
    namespace: flux-system
stringData:
```



```
identity: |
    -----BEGIN PRIVATE KEY-----
MDoCAQAwDQYJKoZIhvcNAQEBBQAEJjAkAgEAAgMAsDkCAwEAAQICMZECAgDlAgIA
xQICAJUCAgCRAgFd
    -----END PRIVATE KEY-----
identity.pub: |
    ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAAwCwOQ==
```

Recommendations

Short term, implement a strict minimum requirement of 1024 bits for the SSH RSA public key size. This will ensure that users cannot create Kubernetes secrets with dangerously small key sizes, such as the 16-bit example shown in figure 8.2. By enforcing a larger key size, the system's security will significantly improve because it will be much more resistant to brute-force attacks.

Long term, periodically review other Flux arguments to ensure they do not allow insecure configurations.



Figure 8.2: The flux command to create a Kubernetes secret for Git authentication using a 16-bit RSA public key

9. Flux macOS release binary susceptible to .dylib injection		
Severity: Low	Difficulty: High	
Type: Configuration	Finding ID: TOB-FLUX-9	
Target: flux process		

The Flux macOS release binary does not have Hardened Runtime restrictions enabled (figure 9.1), making the binary vulnerable to a .dylib file injection attack. A .dylib injection attack allows an attacker to inject a custom dynamic library (.dylib) into a process, potentially leading to, for example, unauthorized access to sensitive information.

\$ brew install fluxcd/tap/flux \$ codesign -dvvv `which flux` /usr/local/bin/flux: code object is not signed at all

Figure 9.1: Installing the official release of Flux by Homebrew and using the codesign tool to check whether the binary has the kSecCodeSignatureEnforcement flag enabled

```
$ cat inj.c
#include <stdio.h>
// The constructor attribute causes the function to be called automatically before
before main() is called
__attribute__((constructor))
static void customConstructor(int argc, const char **argv)
{
    printf("Successfully injected dylib\n");
}
# Exporting the DYLD_INSERT_LIBRARIES environment variable to inject dynamic
libraries into other running processes
$ export DYLD_INSERT_LIBRARIES=`pwd`/inj.dylib
$ flux
Successfully injected dylib
Command line utility for assembling Kubernetes CD pipelines the GitOps way.
(...)
```

Figure 9.2: The proof of concept showing that the custom .dylib file can be successfully injected into the flux process

Exploit Scenario

An attacker gains access to a target user's machine and crafts a malicious .dylib to steal passwords from the standard Flux input. Then the attacker sets the DYLD_INSERT_LIBRARIES environment variable in the .zshrc file to the path of the crafted



.dylib. The user executes the flux bootstrap github command with the --token-auth parameter and provides a GitHub personal access token through standard input. As a result, the hijacked access token is sent to the attacker.

Recommendations

Short term, sign the release macOS Flux binaries and verify that the code signature flags include the kSecCodeSignatureEnforcement flag to ensure the Hardened Runtime protects the binary. The code signature flags are displayed in the CodeDirectory line when running the codesign command (figure 9.3):

- A 0x0 flag indicates that the binary has a standard code signature without additional features.
- A 0x10000 flag (kSecCodeSignatureEnforcement) indicates that the application has implemented runtime hardening policies.

```
$ codesign -dvvv `which kubectl`
Executable=/Applications/Docker.app/Contents/Resources/bin/kubectl
Identifier=kubectl
Format=Mach-0 thin (x86_64)
CodeDirectory v=20500 size=431283 flags=0x10000(runtime) hashes=13472+2
location=embedded
```

Figure 9.3: An example that uses the codesign tool to show a hardened kubectl binary

Long term, implement automatic checks in the project CI/CD pipeline to ensure the release binary has Hardened Runtime restrictions enabled.

References

- DYLIB Injection in Golang apps on Apple silicon chips
- A Deep Dive into Penetration Testing of macOS Applications (Part 2)

10. Path traversal in SecureJoin implementation		
Severity: Undetermined	Difficulty: Undetermined	
Type: Data Validation	Finding ID: TOB-FLUX-10	
Target: pkg/git/gogit/fs/join.go		

The SecureJoinVFS function in pkg/git/gogit/fs is meant to join two paths, root and unsafePath, with the condition that the returned path must be scoped within root. However, it is possible for an attacker to cause the function to return a path outside the root directory by crafting a symlink in the root directory. This compromises the methods on the OS struct in the pkg/git/gogit/fs library.

Here is a portion of the code for SecureJoinVFS:

```
99 // Absolute symlinks reset any work we've already done.
100 if filepath.IsAbs(dest) {
101 if !fi.IsDir() && strings.HasPrefix(dest,
root+string(filepath.Separator)) {
102 return filepath.Clean(dest), nil
103 }
104 path.Reset()
105 }
```

Figure 10.1: Code snippet from SecureJoinVFS (pkg/git/gogit/fs/join.go#L99-L105)

The if statements on lines 100 and 101 check that dest (the destination of a symlink) is an absolute path that has root/ as a prefix. In this case, dest is returned. However, it is possible for dest to both begin with root/ and not be a child of root. For instance, /tmp/rootDir/../a.txt begins with /tmp/rootDir/ but is not a descendent of /tmp/rootDir/ (it resolves to /tmp/a.txt).

Here is a proof of concept showing how an attacker could write to a file outside the root directory:

```
$ # STATE OF THE FILESYSTEM BEFORE MAIN.GO IS RUN; NOTE THE SYMLINK IN ROOTDIR
$ pwd
/tmp/poc
$ ls -1 rootDir
total 0
```



```
lrwxr-xr-x 1 sam wheel 42 Aug 2 17:25 file.txt ->
/tmp/poc/rootDir/../unrelatedDir/pwned.txt
$ ls -l unrelatedDir
total 0
$ # MAIN.GO SHOULD LEAVE EVERYTHING OUTSIDE OF ROOTDIR UNTOUCHED, SINCE IT USES THE
SECURE FILE SYSTEM
$ cat main.go
package main
import ("fmt"
       "github.com/fluxcd/pkg/git/gogit/fs"
      "os")
func main() {
      // Secure file system rooted in rootDir
      my_os := fs.New("/tmp/poc/rootDir")
      // Open file.txt and write "hello" to it; shouldn't affect anything outside
of rootDir
      f, err := my_os.OpenFile("file.txt", os.O_APPEND|os.O_CREATE|os.O_WRONLY,
0600)
      if err != nil {
             fmt.Println(err)
             return
      }
      _, err = f.Write([]byte("hello\n"))
      if err != nil {
             fmt.Println(err)
             return
      }
      err = f.Close()
      if err != nil {
             fmt.Println(err)
             return
      }
      // To indicate that we haven't hit any errors
      fmt.Println("success")
}
$ go run main.go
success
$ ls -l rootDir
total 0
lrwxr-xr-x 1 sam wheel 42 Aug 2 17:25 file.txt ->
/tmp/poc/rootDir/../unrelatedDir/pwned.txt
$ ls -l unrelatedDir
total 8
```

```
-rw----- 1 sam wheel 6 Aug 2 17:27 pwned.txt
$ cat unrelatedDir/pwned.txt
hello
$ # A file in unrelatedDir got written to because of the malicious symlink
```

Figure 10.2: Proof of concept to demonstrate breaking out of SecureJoin root directory

This issue will be high severity when the pkg/git/gogit/fs library is considered on its own because its main security guarantee is that it should not be possible to read or write outside the root directory. However, due to the time-boxed nature of this audit, we did not determine whether there is a way to exploit this vulnerability to affect Flux as a whole.

Recommendations

Short term, remove the return statement in figure 10.1, line 102; the loop should continue even when a symlink with an absolute path is hit, and the return statement at the end of the function (line 114) is not susceptible to this vulnerability.

Long term, expand unit tests to catch similar issues.



A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	Description
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	Description
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	Description
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Configuration	The configuration of system components in accordance with best practices
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Data Handling	The safe handling of user inputs and data processed by the system
Documentation	The presence of comprehensive and readable codebase documentation
Maintenance	The timely maintenance of system components to mitigate risk
Memory Safety and Error Handling	The presence of memory safety and robust error-handling mechanisms
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage

Rating Criteria	
Rating	Description
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.

Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.



C. Non-Security-Related Findings

This appendix contains findings that do not have immediate or obvious security implications. However, they may facilitate exploit chains targeting other vulnerabilities or may become easily exploitable in future releases.

1. Case-insensitive string comparison is done using strings.ToLower function and == operator. This results in a significant increase in both computational and memory complexity. This is because strings.ToLower will allocate a new string and compute the full lowercase version of the string, even if the first characters of the strings do not match. Use strings.EqualFold for comparing strings instead. Also, add the Staticcheck tool with the SA6005 check to the CI/CD to identify similar issues.

```
if strings.ToLower(event) == strings.ToLower(e) {
```

Figure C.1: Example of case-insensitive string comparison using strings.ToLower (notification-controller/internal/server/receiver_handlers.go#167)

The above file includes three instances of this type of comparison. We did not find this issue anywhere in the codebase aside from this file.

2. Useless assignment. The following assignment has no effect since the function returns immediately afterward and can be removed.

```
template = template[1:]
return fmt.Errorf("--filter-extract is malformed")
```

```
Figure C.2: Useless assignment
(flux2/cmd/flux/create_image_policy.go#186-187)
```

3. Calling defer in a for loop. Using a defer statement inside a for loop could cause unexpected conditions because the deferred function is called when the function exits, not at the end of each loop iteration. Delete the temporary directory at the end of the loop instead of using defer.

```
for _, obj := range objects {
    // (...)
    defer cleanupDir(tmpDir)
```

Figure C.3: Using defer in a for loop (flux2/internal/build/diff.go#89-119)

4. Use of two different, nearly identical, SecureJoin functions. The pkg/git/gogit/fs/osfs_os.go file uses both pkg/git/gogit/fs.SecureJoin



and github.com/cyphar/filepath-securejoin.SecureJoin, which have nearly identical implementations.

```
func (fs *0S) Chroot(path string) (billy.Filesystem, error) {
      joined, err := securejoin.SecureJoin(fs.workingDir, path)
      if err != nil {
             return nil, err
      }
      return New(joined), nil
}
// (...)
func (fs *OS) abs(filename string) (string, error) {
      if filename == fs.workingDir {
             filename = "/"
      } else if strings.HasPrefix(filename,
fs.workingDir+string(filepath.Separator)) {
             filename = strings.TrimPrefix(filename,
fs.workingDir+string(filepath.Separator))
      }
      return SecureJoin(fs.workingDir, filename)
}
```

Figure C.4: Use of two SecureJoin functions from different packages that have the same implementation (pkg/git/gogit/fs/osfs_os.go#218-263)

5. Use of the filepath.Join function followed by the insideWorkingDirEval function instead of SecureJoin. The Lstat and Readlink functions in the pkg/git/gogit/fs/osfs_os.go file use filepath.Join to join two directories and then call insideWorkingDirEval to ensure that the resulting path is within the root directory. However, this is what the SecureJoin function does; the logic should be simplified to a single SecureJoin call, and the insideWorkingDirEval and insideWorkingDir helper functions should be removed.

Here is a full list of locations affected by TOB-FLUX-5:

- source-controller/internal/controller/helmchart_controller.go:131
 4
- source-controller/internal/controller/helmchart_controller.go:128
 4
- source-controller/internal/controller/helmchart_controller.go:125
 4
- notification-controller/internal/server/receiver_handlers.go:412
- notification-controller/internal/server/event_handlers.go:127
- notification-controller/internal/server/event_handlers.go:69
- notification-controller/internal/server/event_handlers.go:65
- notification-controller/internal/controller/alert_controller.go:2 06
- flux2/pkg/uninstall/uninstall.go:328
- flux2/pkg/uninstall/uninstall.go:307
- flux2/pkg/uninstall/uninstall.go:293
- flux2/pkg/uninstall/uninstall.go:279
- flux2/pkg/uninstall/uninstall.go:265
- flux2/pkg/uninstall/uninstall.go:251
- flux2/pkg/uninstall/uninstall.go:237
- flux2/pkg/uninstall/uninstall.go:223
- flux2/pkg/uninstall/uninstall.go:209
- flux2/pkg/uninstall/uninstall.go:195
- flux2/pkg/uninstall/uninstall.go:181
- flux2/pkg/uninstall/uninstall.go:167

- flux2/pkg/uninstall/uninstall.go:153
- flux2/pkg/uninstall/uninstall.go:139
- flux2/pkg/uninstall/uninstall.go:117
- flux2/pkg/uninstall/uninstall.go:104
- flux2/pkg/uninstall/uninstall.go:91
- flux2/pkg/uninstall/uninstall.go:78
- flux2/pkg/uninstall/uninstall.go:65
- flux2/pkg/uninstall/uninstall.go:52

Here is a full list of locations affected by TOB-FLUX-6:

- flux2/pkg/manifestgen/manifest.go:46
- flux2/pkg/manifestgen/install/manifests.go:95
- source-controller/internal/controller/storage.go:125
- source-controller/internal/controller/storage.go:614
- source-controller/internal/fs/fs.go:90
- source-controller/pkg/azure/blob.go:228
- source-controller/pkg/gcp/gcp.go:121
- pkg/oci/client/internal/fs/fs.go:90
- pkg/tar/tar.go:119
- pkg/tar/tar.go:167
- pkg/git/gogit/fs/osfs_os.go:130
- pkg/git/gogit/fs/osfs_os.go:242

Here is a full list of locations affected by TOB-FLUX-7:

- kustomize-controller/internal/decryptor/decryptor.go:505
- flux2/internal/build/diff.go:176

- flux2/internal/build/diff.go:170
- flux2/cmd/flux/manifests.embed.go:41
- pkg/testserver/artifact.go:170
- pkg/oci/client/build.go:148
- pkg/tar/tar.go:167
- pkg/tar/tar.go:119
- pkg/helmtestserver/server.go:66
- pkg/git/internal/e2e/utils.go:274



E. Automated Static Analysis

This appendix describes the setup of the automated analysis tools used during this audit.

Though static analysis tools frequently report false positives, they detect certain categories of issues, such as memory disclosures, misspecified format strings, and the use of unsafe APIs, with essentially perfect precision. We recommend periodically running these static analysis tools and reviewing their findings.

golangci-lint

We installed the golangci-lint tool by following the installation guide.

To analyze the codebase using golangci-lint, we navigated to the root directory of the target and executed the following command:

```
golangci-lint run --enable-all
```

If the --enable-all option is too noisy, specific linters can be disabled using the -D
<name_of_linter> option. It is also possible to run only selected linters using the
--disable-all -E <gosec | staticcheck | nakedret | ...other_linters> option.

Some underlying linters may require a successful build of the Go project. They may silently ignore Go packages that are not yet built or have failing builds.

Semgrep

To install Semgrep, we used pip by running python3 -m pip install semgrep.

To run Semgrep on the codebase, we ran the following command in the root directory of the project (running multiple predefined rules simultaneously by providing multiple --config arguments):

```
semgrep --config "p/trailofbits" --config "p/ci" --config
"p/security-audit" --config "p/semgrep-go-correctness"
--metrics=off
```

Semgrep Pro Engine includes cross-file (interfile) and cross-function (interprocedural) analysis. To run Semgrep with the Pro Engine, we used the following commands:

```
semgrep login
semgrep install-semgrep-pro
semgrep --pro --config "p/default" --metrics off
```

We recommend integrating Semgrep into the project's CI/CD pipeline. To thoroughly understand the Semgrep tool, refer to our Trail of Bits Testing Handbook, where we aim to



streamline Semgrep use and improve security testing effectiveness. Also, consider doing the following:

- Limit Semgrep to show results of only error-level severity by using the --severity ERROR flag.
- Focus first on rules with high confidence and medium- or high-impact metadata.
- Use the SARIF format (by using the --sarif Semgrep argument) with the SARIF Viewer for Visual Studio Code extension. This will make it easier to review the analysis results and drill down into specific issues to understand their impact and severity.

CodeQL

We installed CodeQL by following CodeQL's installation guide.

Next, we ran the following command to create the project database for the Golang repository:

codeql database create codeql.db --language=go

We then ran the following command to query the database:

```
codeql database analyze codeql.db -j 10 --format=csv
--output=codeql_tob_go.csv -- go-developer-happiness go-lgtm-full
go-security-and-quality go-security-experimental
```

We also used private Trail of Bits query packs.

TruffleHog

We used TruffleHog to detect sensitive data such as private keys and API tokens in the repositories' Git histories.

To detect sensitive information in the fluxcd GitHub organization, we used the following command:

trufflehog github --org=fluxcd --only-verified

The --only-verified flag specifies that only findings marked as "verified" should be included in the scan results. This helps filter out false positives and focuses on confirmed instances of sensitive information.



F. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

On October 20, 2023, Trail of Bits reviewed the fixes and mitigations implemented by the Flux team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the 10 issues described in this report, Flux has resolved seven issues, has partially resolved one issue, and has not resolved the remaining two issues. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	SetExpiration does not set the expiration for the given key	Resolved
2	Inappropriate string trimming function	Resolved
3	Go's default HTTP client uses a shared value that can be modified by other components	Resolved
4	Unhandled error value	Resolved
5	Potential implicit memory aliasing in for loops	Resolved
6	Directories created via os.MkdirAll are not checked for permissions	Unresolved
7	Directories and files created with overly lenient permissions	Partially Resolved
8	No restriction on minimum SSH RSA public key bit size	Resolved
9	Flux macOS release binary susceptible to .dylib injection	Unresolved

10	Path traversal in SecureJoin implementation	Resolved

Detailed Fix Review Results

TOB-FLUX-1: SetExpiration does not set the expiration for the given key

Resolved in PR #1185 on the source-controller repository. This PR adds a statement that reassigns the relevant index in the c.Items map, allowing the modified expiration value to be preserved.

TOB-FLUX-2: Inappropriate string trimming function

Resolved in PR #590 on the notification-controller repository. This PR replaces the call to the strings.TrimLeft function with a call to the strings.TrimPrefix function.

TOB-FLUX-3: Go's default HTTP client uses a shared value that can be modified by other components

Resolved in PR #4182 on the flux2 repository. This PR modifies the relevant code to use the default client provided by the hashicorp/go-cleanhttp library. Unlike the default client provided by Go's http library, this client does not share the global state with other clients.

TOB-FLUX-4: Unhandled error value

Resolved in PR #4181 on the flux2 repository. This PR adds a check on the error value returned by the getRows function.

TOB-FLUX-5: Potential implicit memory aliasing in for loops

Resolved in PR #1257 on the source-controller repository, PR #627 on the notification-controller repository, and PR #4329 on the flux2 repository. These PRs fix the implicit memory aliasing problems, sometimes by copying list elements and sometimes by passing references to list elements (instead of to loop variables).

TOB-FLUX-6: Directories created via os.MkdirAll are not checked for permissions

Unresolved. OSTIF provided the following context for this finding's fix status:

We have analyzed the occurrences and concluded that they all target paths within directories created using os.MkdirTemp.

Since multiple programs or goroutines invoking this function simultaneously won't select the same or preexisting directory, and the directory's existence is short-lived, any potential exploit would need to be time-based and meticulously crafted to run in parallel with the program's execution.

Although we experimented with a solution like https://github.com/hiddeco/safeos, we have determined that the combination of the above approach and the environment in which our applications operate doesn't justify the maintenance and cost associated with such a solution.



TOB-FLUX-7: Directories and files created with overly lenient permissions

Partially resolved in PR #663 on the pkg repository. This PR fixes the occurrence of this issue in the pkg/git/internal/e2e/utils.go file. However, the other occurrences of this issue (see appendix D) remain unresolved.

OSTIF provided the following context for this finding's fix status:

Some overly lenient permissions persist because imposing breaking changes, such as revisions derived from file checksums, could create issues for downstream consumers. We are committed to resolving these in an upcoming minor release where feasible.

TOB-FLUX-8: No restriction on minimum SSH RSA public key bit size

Resolved in PR #4177 on the flux2 repository. This PR adds a strict minimum of 1024 bits for the RSA public key size.

TOB-FLUX-9: Flux macOS release binary susceptible to .dylib injection

Unresolved. OSTIF provided the following context for this finding's fix status:

We are currently in the challenging process (for an open-source project) of obtaining an Apple Developer Account to enable us to leverage a solution such as quill (https://github.com/anchore/quill) for code signing and notarization of our macOS binaries.

Once we secure this account, we are committed to implementing this with high priority.

TOB-FLUX-10: Path traversal in SecureJoin implementation

Resolved in PR #650 on the pkg repository and PR #31 on the go-billy/osfs repository. PR #650 removes the pkg/git/gogit/fs library and replaces references to it with references to its upstream go-billy/osfs library. PR #31 adds changes made in pkg/git/gogit/fs to the go-billy/osfs repository, using a corrected implementation of the SecureJoinVFS function in the pjbgf/filepath-securejoin repository (later changed in PR #34 to the cyphar/filepath-securejoin repository). Notably, these implementations of SecureJoinVFS do not contain the erroneous return statement described in TOB-FLUX-10.



G. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	5	
Status	Description	
Undetermined	The status of the issue was not determined during this engagement.	
Unresolved	The issue persists and has not been resolved.	
Partially Resolved	The issue persists but has been partially resolved.	
Resolved	The issue has been sufficiently resolved.	