Security review of FluxCD

Technical report

Ref 18-10-21-FLX
Version 1.1
Date 2021
Prepared for FluxCD Team
Prepared by: Dr. David Korczynski, Adam Korczynski
Facilitated by: Open Source Technology Improvement Fund (OSTIF)

This report is licensed under Creative Commons 4.0 (CC BY 4.0)

1 Table of contents

1 Table of contents **1**

2 Executive summary **2**

3 Project information **3**

4 Comments on Flux documentation **4**

5 Fuzzing integration **9**

6 Review of multi-tenancy access control model **12**

7 Issues found **14**

© 2021 Ada Logics Ltd.

2 Executive summary

This report describes the results of a security assessment and fuzzer development of the FluxCD repositories made by Ada Logics and funded by the OSTIF.

2.1 Scope of audit:

The following repositories were in scope of the audit:

- https://github.com/fluxcd/flux2
- https://github.com/fluxcd/image-automation-controller
- https://github.com/fluxcd/source-controller
- https://github.com/fluxcd/helm-controller
- https://github.com/fluxcd/image-reflector-controller
- https://github.com/fluxcd/notification-controller
- https://github.com/fluxcd/kustomize-controller
- https://github.com/fluxcd/pkg

The goal of the audit was to:

1. Perform a manual audit of the Flux repositories
2. Integrate fuzzing into the Flux projects
3. Review the documentation of Flux
4. Advice on how to convert the text in https://github.com/fluxcd/flux2/pull/582 into a security model

2.2 Results

Throughout the review we found many positive elements in that the developers have clearly spent efforts in security-relevant considerations. However, we also found several issues, both in the code and in the structure of Flux, that should be improved upon.

The manual review of the source found 22 issues divided into the following severities: 1 high, 3 medium, severity issue, 3 medium severity, 13 low and 5 information. During the manual audit we also found that the code could be improved across the repositories.

We implemented fuzzers for each of the custom controllers in the project and also performed an initial integration with OSS-Fuzz. The fuzzers found several issues and also helped find inconsistent code patterns in Flux.

Our review of the documentation is that the documentation is often complicated to grasp. The documentation is mainly composed of examples, which complicates the process of getting a conceptual understanding of Flux. We also found that the security-documentation is lacking, e.g. the security considerations of adopting Flux, the threat model and attack surface, and also documentation about the access control system.

We found the text in https://github.com/fluxcd/flux2/pull/582 to present ideas that will improve the security posture of Flux, but the text is in need of work to convert it into a proper security model. The current text leaves out many questions, mixes design and implementation and does not outline a clear problem for the new security model to solve.

© 2021 Ada Logics Ltd.

3 Project information

3.1 Document history

Version	Date	Details
1.0	08/10/2021	First version presented to Flux team
1.1	18/10/2021	Flux team comments taken into account.

3.2 Contacts

Ada Logics

Contact	Position	Email address
David Korczynski	Security Researcher	david@adalogics.com
Adam Korczynski	Security engineer	adam@adalogics.com

Open Source Technology Improvement Fund

Contact	Email address
Derek Zimmer	derek@ostif.com
Amir Montazery	amir@ostif.org

Flux team

Contact	Email address
Daniel Holbach	daniel@weave.works
Michael Bridgen	michael@weave.works
Hidde Beydals	hidde@weave.works
Stefan Prodan	stefan@weave.works
Somtochi Onyekwere	somtochi@weave.works
Tamao Nakahara	tamao@weave.works
Philip Lane	philip.lane@gmail.com
Aurel Canciu	aurelcanciu@gmail.com

© 2021 Ada Logics Ltd.

4 Comments on Flux documentation

In this section we go over our review of the Flux documentation from a Security engineer's perspective.

4.1 Informational comments on documentation

We have found the documentation of Flux to be vast and extensive. One of the core approaches that the documentation takes to convey the information about Flux is through examples and use-cases. This is good in that the reader can quickly extract templates that may reflect the need of the reader.

However, one of the potential downsides of an extensive documentation rooted in examples is that the documentation becomes dispersed and even too extensive, which can make the reader wander and get lost in specifics. In such a scenario it becomes difficult to extract a holistic conceptual understanding of Flux. For example, since the documentation is largely rooted in examples it is, to an extent, left to the reader to abstract this into more general concepts.

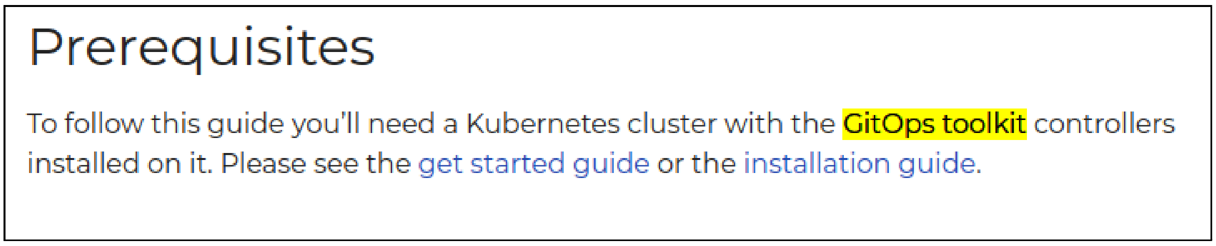
4.1.1 Recommendation

An improvement in this context would be to have clarification on end-to-end processes in Flux, similar to how Envoy Proxy has an "life of an event" documentation:
https://www.envoyproxy.io/docs/envoy/v1.19.1/intro/life_of_a_request

From a security perspective such an overview would highly improve the understanding of what trust boundaries Flux assumes and also describe the threat model of Flux. In such end-to-end documentation, it would be of high value to make it clear how the individual components relate to each other as well as describe where authentication and hardening procedures are in place.

4.2 Blurry line between "GitOps toolkit" and "Flux"

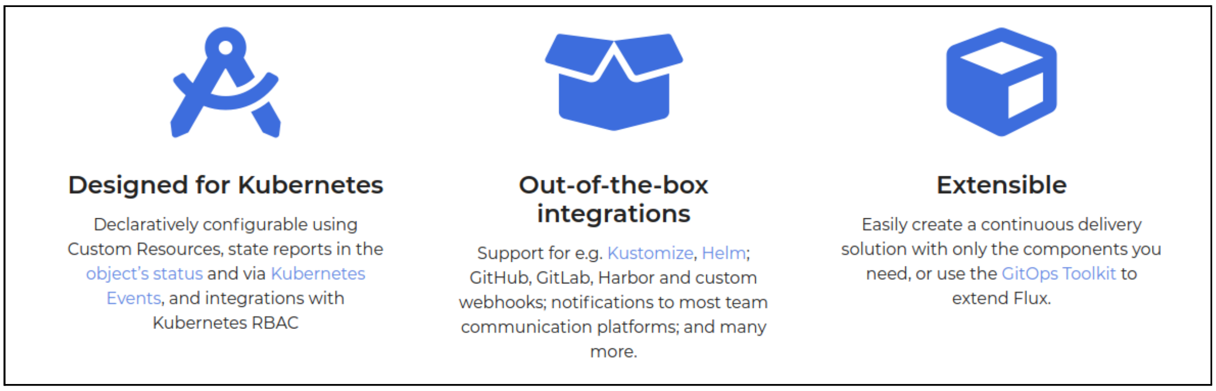
We found the difference between GitOps toolkit blurry and at the beginning of the engagement difficult to understand. For example, when going through the documentation (https://fluxcd.io/docs) sequentially, then the first mention of "GitOps toolkit" is in the second page within the "Guides" section, named "Helm Releases":
https://fluxcd.io/docs/guides/helmreleases/
On this page it says a prerequisite to the guide is to have a Kubernetes cluster with the GitOps controllers installed and then referencing the getting started and installation guides, as shown by the following Figure.


Prerequisites

To follow this guide you'll need a Kubernetes cluster with the GitOps toolkit controllers installed on it. Please see the get started guide or the installation guide.

However, “GitOps toolkit” is not mentioned on these pages.

In addition to the above observations, the current destination of the link to the gitops toolkit on the fluxcd.io page, namely https://fluxcd.io/#gitops-toolkit, does not lead anywhere. This is the link on the rightmost box on the website as shown by the following Figure:


Designed for Kubernetes

Declaratively configurable using Custom Resources, state reports in the object's status and via Kubernetes Events, and integrations with Kubernetes RBAC

Out-of-the-box integrations

Support for e.g. Kustomize, Helm; GitHub, GitLab, Harbor and custom webhooks; notifications to most team communication platforms; and many more.

Extensible

Easily create a continuous delivery solution with only the components you need, or use the GitOps Toolkit to extend Flux.

4.2.1 Recommendation

Flux uses names and concepts conservatively to easily convey the structure of the infrastructure. However, the documentation must have an equal structure including proper introduction of concepts. We recommend using clear naming of the components in the system and ensuring proper introduction without broken links or links to pages that do not clarify concepts that were promised. It would be a great assistance to have a page that unifies the nomenclature of Flux and links to details about the definition or where the concept is introduced.

4.3 Repositories and GitOps toolkit

We found that the documentation of each custom controller in Flux varies. This includes particularly important information on concepts such as running and testing the controllers. For example, an example where we found the repositories to have differences in their documentation in an area where they should be much similar is the CONTRIBUTING.md files. To show the difference in these files, we collected the titles in the Github pages and these are as follows:

Source controllers:

- Contributing
    - Certificate of origin
    - Communications
        - Installing required dependencies
            - macOS
            - Arch Linux
            - Building from source

        - How to run the test suite
    - Acceptance policy
        - Format of the commit message

Kustomize controller:

- Contributing
    - Certificate of origin
    - Communications
        - How to run the test suite
        - How to run the controller locally
    - Acceptance policy
        - Format of the commit message

Image-automation-controller

- Contributing
    - Certificate of Origin
    - Communications
        - How to run the test suite
    - Acceptance policy

Image-reflector-controller

- Contributing
    - Certificate of origin
    - Communications
        - How to run the test suite
    - Acceptance policy
        - Format of the Commit message

Notification-controller

- Contributing
    - Certificate of Origin
    - Communications
        - How to run the test suite
    - Acceptance policy
        - Format of the commit message

Helm-controller

- Contributing
    - Certificate of Origin
    - Communications
        - How to run the test suite
        - How to run the controller locally
    - Acceptance policy
        - Format of the Commit Message

By looking at these titles, we can identify the following issues:

- All information related to "how to execute the controller" is in the "communications" section.
- The files do not share the same guidelines, for example, the Kustomize and Helm controllers have sections such as "How to run the controller locally" whereas none of the others have.
- If we look deeper at the text in each of these files, we will also find differences between similar sections. For example, the Helm and Kustomize controller asks contributors to sign-off on commits whereas the others do not.
- If we look deeper at the text, Helm-controller and Kustomize-controller ask the user to sign off commits whereas the other controllers do not.

© 2021 Ada Logics Ltd.

Furthermore, we want to emphasize that from a security perspective it is much more straightforward to work with code that is well-documented in terms of how to build and run the code under analysis. At present this type of documentation is only available in the Kustomize and Helm controllers through the "*how to run the controller locally*" section. Simply launching some of the controllers locally can be difficult for someone with limited understanding of the Flux codebase.

4.3.1 Recommendation

Some observations that we made about these files which we think could improve the documentation of Flux/GitOps Toolkit is:

1. Centralise the redundant sections. This includes:
    - Certificate of Origin.
    - Do not have developer guides under the "Communications" title. Currently, all of the information related to "how to execute the controller" is under the "communications" title. We believe this should not be the case
    - Acceptance policy should be centralised
    - Format of the commit message should be centralised
2. Each of the controllers should have clear guidelines on how to build and run the projects. This includes the sections:
    - How to install dependencies
    - How to install the controller
    - How to run the test suite locally
    - How to run the controller locally

4.4 Documenting the permission system of Flux

Flux and the GitOps toolkit leverages RBAC to control permissions. Flux creates a complex RBAC set up (and the set up based on the new impersonation will be even more complex) and this role/permission set up is not well-documented. In order to get an understanding of the RBAC set up it is currently required to inspect the implementation of RBAC yaml files themselves. We consider this to be a significant issue since the RBAC roles and Flux will be deployed within a cluster running potentially sensitive operations.

4.4.1 Recommendation

Provide a conceptual description of the permission system of Flux, preferably with schematic to make it easy for a reader to quickly understand what the roles, resources and permissions are in the Flux system. Highlight in particular:

- What RBAC roles are used
- What resources are used by the roles and the permissions on these resources
- Whether Flux asks for more than necessary to ease implementation.
- If the RBAC system is modifiable to create a more secure model.
- The effect that the RBAC implementation of Flux has on the entire state of the cluster.

© 2021 Ada Logics Ltd.

4.5 Documenting the use of sensitive data by Flux in the cluster

Currently Flux provides no central documentation on the security of Flux. This is an issue as a user might expose more data than anticipated and currently has to read the implementation to identify these concepts.

4.5.1 Recommendation

Document the security context of Flux in a central location. Topics that are useful to document on such a page include:

1. The sensitive data that Flux uses.
2. The attack surface of Flux.
3. What are the consequences of a compromise of Flux/component of Flux
4. Who should be allowed to administer Flux within an Organisation
5. Security hardening of Flux, i.e. Flux can be used in many ways and what are the best practices from a security perspective and what are some of the pitfalls that can happen

Open source projects that have examples of this:

- Docker: https://docs.docker.com/engine/security/
- Envoy: https://www.envoyproxy.io/docs/envoy/latest/intro/arch_overview/security/security
    - In particular the threat model

4.6 Missing release cycle documentation

The Flux controllers provide tagged releases on Github and the Flux release procedure is documented here. However, the release procedure is missing security-relevant information such as how long versions are supported and how often to expect releases.

Examples of open source projects that describe their release cycle:

- Kubernetes: https://kubernetes.io/releases/version-skew-policy/
- Redis: https://redis.io/topics/releases
- Linux kernel: https://www.kernel.org/category/releases.html

4.6.1 Recommendation

In the release procedure documentation specify the release cycle and for how long releases are supported.

Following discussions with the Flux maintainers it was clarified that this will happen once Flux reaches general availability status. We consider this reasonable and would recommend explicitly stating this on the release procedure page.

© 2021 Ada Logics Ltd.

5 Fuzzing integration

We integrated fuzzers into all of the Flux repositories except the command line utility as the command line utility is not interesting from a fuzzing perspective. The focus was primarily to target the reconcilers of the various controllers. The reconcilers triggers when Kubernetes events happen, such as a Kubernetes object being created or modified, and will act based on:

1. The values in the reconciled object
2. The state of the Kubernetes cluster
3. The value of various other inputs, e.g. the downloading artifacts from external URLs.
4. The state of the controller itself

The goal of our fuzzers have been to focus primarily on mutating (1) but we also created some fuzzers that are based on randomising the state of (3) and (4) listed above.

The fuzzers are developed such that they can integrate with OSS-Fuzz (https://github.com/google/oss-fuzz) and this requires that the fuzzers can run in a specific runtime environment (https://google.github.io/oss-fuzz/further-reading/fuzzer-environment/). The runtime environment is different from the build environment and the fuzzers should be able to run as standalone binaries. This means we have had to implement some hacky parts in the fuzzers, namely, download various Flux artifacts within the fuzzer initialisation itself and this is all documented in the fuzzer source code. The current state of the fuzzers are:

- They are waiting to be reviewed and merged by the Flux maintainers (see PRs below).
- OSS-Fuzz integration is ready but requires merge from the Flux maintainers.
- CIFuzz integration is prepared, but the above steps must be completed first.

5.1 Findings

During local runs of the fuzzers we found 3 issues:

- 1 slice out-of-bounds panic (Issue 13)
- 2 nil-dereference panics (Issue 15)

We predict the fuzzers will uncover more issues once the fuzzers are merged and integrated with OSS-Fuzz. Please see the following blog post on the importance of continuity in fuzzing (https://adalogics.com/blog/the-importance-of-continuity-in-fuzzing-cve-2020-28362).

5.2 Fuzzers developed

The following pull requests have been made and are all awaiting review. In total, this includes around 3500 lines of code added to the Flux repositories. The changes are all additive and thus there are no modifications to any of the Flux logic.

- https://github.com/fluxcd/image-automation-controller/pull/229
- https://github.com/fluxcd/source-controller/pull/443
- https://github.com/fluxcd/pkg/pull/150
- https://github.com/fluxcd/image-reflector-controller/pull/175
- https://github.com/fluxcd/helm-controller/pull/326
- https://github.com/fluxcd/notification-controller/pull/250
- https://github.com/fluxcd/kustomize-controller/pull/434

© 2021 Ada Logics Ltd.

- OSS-fuzz initial integration PR: https://github.com/google/oss-fuzz/pull/6539

The following table shows the name of the fuzzers as well as which controller and reconciler they target, and also contains a URL with the PR for each fuzzer.

#	Fuzzer name	Controller	Reconciler	Pull request
1	FuzzReconciler	Image Automation Controller	ImageUpdateAutomationReconciler	pull/229
2	FuzzUpdateWithSetters	Image Automation Controller	N/a*	pull/229
3	FuzzRandomGitFiles	Source Controller	GitRepositoryReconciler	pull/443
4	FuzzGitResourceObject	Source Controller	GitRepositoryReconciler	pull/443
5	FuzzHelmchartController	Source Controller	HelmRepositoryReconciler	pull/443
6	FuzzStorageArchive	Source Controller	N/a*	pull/443
7	FuzzStorageCopy	Source Controller	N/a*	pull/443
8	FuzzGetterConditions	Shared Library: Pkg	N/a*	pull/150
9	FuzzConditionsMatch	Shared Library: Pkg	N/a*	pull/150
10	FuzzPatchApply	Shared Library: Pkg	N/a*	pull/150
11	FuzzConditionsUnstructured	Shared Library: Pkg	N/a*	pull/150
12	FuzzUntar	Shared Library: Pkg	N/a*	pull/150
13	FuzzLibGit2Error	Shared Library: Pkg	N/a*	pull/150
14	FuzzEventInfof	Shared Library: Pkg	N/a*	pull/150
15	FuzzTlsConfig	Shared Library: Pkg	N/a*	pull/150
16	Fuzz	Image Reflector Controller	ImagePolicyReconciler	pull/175
17	FuzzHelmreleaseComposeValues	Helm Controller	HelmReleaseReconciler	pull/326
18	FuzzHelmreleasereconcile	Helm Controller	HelmReleaseReconciler	pull/326
19	Fuzz	Notification Controller	AlertReconciler, ReceiverReconciler	pull/250
20	Fuzz	Kustomize Controller	KustomizationReconciler	pull/434

*Does not target a reconciler

5.3 Next steps for fuzzer integration

The Flux team needs to perform several steps in order to integrate continuous fuzzing with OSS-Fuzz. All of the steps have been prepared by Ada Logics but will need to be merged etc. by the Flux team. The next steps are:

© 2021 Ada Logics Ltd.

1. **Review and merge fuzzers**. Once the fuzzers are merged, Ada Logics will switch the git source in Flux's OSS-fuzz integration from our forks to Flux's own git repositories (here). We will furthermore complete the CIFuzz set up which is supported by OSS-fuzz.
2. **Enable static linking of C dependencies**. All fuzzers must be built with static linking to be run by OSS-fuzz. This issue is already in progress by the Flux maintainers, and once it is done it should enable fuzzers that depend on libgit2 to run on OSS-fuzz.
3. **CI integration:** Once Flux is merged into OSS-fuzz, upload `main.yml` to `.github/workflows/` of each repository that contains fuzzers running on OSS-fuzz:

main.yml

```
name: CIFuzz
on: [pull_request]
jobs:
 Fuzzing:
   runs-on: ubuntu-latest
   steps:
   - name: Build Fuzzers
     id: build
     uses:
google/oss-fuzz/infra/cifuzz/actions/build_fuzzers@master
     with:
       oss-fuzz-project-name: 'fluxcd'
       language: go
   - name: Run Fuzzers
     uses: google/oss-fuzz/infra/cifuzz/actions/run_fuzzers@master
     with:
       oss-fuzz-project-name: 'fluxcd'
       language: go
       fuzz-seconds: 60
   - name: Upload Crash
     uses: actions/upload-artifact@v1
     if: failure() && steps.build.outcome == 'success'
     with:
       name: artifacts
       path: ./out/artifacts
```

© 2021 Ada Logics Ltd.

6 Review of multi-tenancy access control model

The Flux team requested that we review a set of early thoughts on a new security model of Flux, proposed in https://github.com/fluxcd/flux2/pull/582. The focus of the review was on general advice on the next steps required for turning it into a proper security model. The document we specifically review in this section is the file accessible here: Secure Impersonation

The document describes a set of ideas on a new implementation of multi-tenancy in Flux by way of secure impersonation. Overall, we welcome many of the ideas in the document. We find the goals of the document to be reasonable and we find strictly enforcing the role with which the kustomize and helm controllers operate on GitOps-managed objects to be based on the user of the object to be a security improvement. However, the document is written in a manner where concepts involve the current state significantly, but the current state is not referenced or documented in another place and instead hidden in the implementation of Flux and within the guide for managing multi-tenancy in Flux here. Furthermore, the documentation discusses both the model and implementation in an ad hoc manner, which makes it difficult to separate the two without detailed knowledge of the implementation at the time the document was written. As such, the text is written by Flux maintainers for Flux maintainers.

We consider the most important step to turning the ideas of the text into a proper security model is to define the ideas in a manner that does not require detailed knowledge of Flux that is only accessible from studying the source code. The high-level next steps to converting this into a security model overall is to properly separate documentation for:

1. The current (pre-proposal) security model of Flux
2. The problems of the existing model and what is to be tackled
3. The design of the new model
4. Argumentation as to why the new model solves the problems that were to be tackled
5. Implementation documentation for the new model

The security model should be a self-contained document that provides definitions and outlines concepts, references relevant texts and also describes the model “as-is”. The model should outline all core Flux components: the reviewed text does not mention all controllers relevant in the Flux ecosystem, e.g. it predates image-reflector-controller and image-automation-controller, so it is unclear where these fit into the overall scheme. The security model that comes out of the reviewed text should clearly outline where each of the components of Flux fit into the security model.

The focus on the security model should be to outline the conceptual artifacts relevant in the Flux infrastructure. This includes the users of the system, the roles of the users, the data handled and the permissions of the roles on the data. The Flux maintainers should emphasize on outlining where permissions change due to the impersonation and also what procedures are in place to enforce the permissions. Once this is in place, an added benefit is that it will become more clear for the Flux maintainers to document their security posture as discussed in “comments on Flux documentation” of this report.

© 2021 Ada Logics Ltd.

Following discussions with the Flux maintainers it became clear that some parts of the documentation may be inaccurate. For example, the document states "*The controller ServiceAccounts are far overprivileged for Flux API operations*" and this was contested by some Flux maintainers. We are positive about the approach to designing the model based on a problem-oriented strategy, however, there should be consensus amongst the Flux maintainers on the problems.

6.0.1 Recommendations

Our recommendation is for the Flux maintainers to compartmentalise these ideas into the separate documents described above. This will make it possible to define a model and also evaluate the soundness of it. We would also recommend clearly stating the status of the reviewed text in the document, as it is easy to interpret the text being a description of a model whereas this is indeed not the case.

We recommend the Flux maintainers to engage security professionals for help exclusively focused on designing and implementing the security model of Flux. The Flux maintainers must outline the state of the current model, the problems of the current model and their priorities in a new model, and then consult a security team that is specialised in designing security models and access control systems. We recommend engaging with experts, such as the CNCF Security Technical Advisory Group, on both the design of the underlying user system and also on the implementation of the security model.

© 2021 Ada Logics Ltd.

7 Issues found

In this section we list the issues found through the engagement, in particular the manual auditing of the code and results from the fuzzers.

In total, our review found 22 issues divided into:

- 1 high severity issue
- 3 medium severity issues
- 13 low severity issues
- 5 informational issues

Issue	Severity	ID
Issue 1: Arbitrary command execution via command injection in the kustomize controller by way of secrets	High	ADA-FLUX-21-01
Issue 2: Nil-dereference in image-automation controller	Low	ADA-FLUX-21-02
Issue 3: Credentials exposed in environment variables and command line arguments	Medium	ADA-FLUX-21-03
Issue 4: Use of deprecated library	Low	ADA-FLUX-21-04
Issue 5: Invalid and missing testing documentation	Informational	ADA-FLUX-21-05
Issue 6: Bug fixes do not always include regression tests	Informational	ADA-FLUX-21-06
Issue 7: Deprecated SHA-1 is used for checksums	Low	ADA-FLUX-21-07
Issue 8: Missing checksum verification	Medium	ADA-FLUX-21-08
Issue 9 Inconsistent and missing logging	Low	ADA-FLUX-21-09
Issue 10: Reading large files can crash flux with an out-of-memory bug	Low	ADA-FLUX-21-10
Issue 11: Files are opened but never closed	Low	ADA-FLUX-21-11
Issue 12: Unhandled error	Low	ADA-FLUX-21-12
Issue 13: Slice bounds out of range	Low	ADA-FLUX-21-13
Issue 14: Possible nil-deref in image-automation controller	Low	ADA-FLUX-21-14
Issue 15: Inconsistent code-styles and potential nil-dereferences	Informational	ADA-FLUX-21-15

© 2021 Ada Logics Ltd.

Issue 16: Missing return statement after error	Low	ADA-FLUX-21-16
Issue 17: File extension comparisons are case sensitive	Low	ADA-FLUX-21-17
Issue 18: Some dependencies are outdated	Informational	ADA-FLUX-21-18
Issue 19: Lack of container security options in deployed pods	Low	ADA-FLUX-21-19
Issue 20: Unhandled errors from deferred file close operations	Low	ADA-FLUX-21-20
Issue 21: x509 certificates are not used for Webex	Medium	ADA-FLUX-21-21
Issue 22: Unnecessary conditions in the code	Informational	ADA-FLUX-21-22

© 2021 Ada Logics Ltd.

7.1 Issue 1: Arbitrary command execution via command injection in the Kustomize controller by way of secrets

Severity	High
Difficulty	Medium
Target	kustomize-controller/controllers/kustomization_controller.go
Finding ID	ADA-FLUX-21-01

Description:
There are two command injections in the kustomize controller and they exist within the `apply` and `validate` functions. Consider the following code from the `validate` function:

```
cmd := fmt.Sprintf("cd %s && kubectl apply -f %s.yaml --timeout=%s --dry-run=%s
--cache-dir=/tmp --force=%t",
	dirPath, kustomization.GetUID(), kustomization.GetTimeout().String(),
validation, kustomization.Spec.Force)

if kustomization.Spec.KubeConfig != nil {
	kubeConfig, err := imp.WriteKubeConfig(ctx)
	if err != nil {
		return err
	}
	cmd = fmt.Sprintf("%s --kubeconfig=%s", cmd, kubeConfig)
} else {
	// impersonate SA
	if kustomization.Spec.ServiceAccountName != "" {
		saToken, err := imp.GetServiceAccountToken(ctx)
		if err != nil {
			return fmt.Errorf("service account impersonation failed:
%w", err)
		}

		cmd = fmt.Sprintf("%s --token %s", cmd, saToken)
	}
}

command := exec.CommandContext(applyCtx, "/bin/sh", "-c", cmd)
Command, err := command.CombinedOutput
```

The code executes a command on the system based on the content of the `cmd` string, and the cmd string is assembled based on the values in the `kustomization` object being reconciled. Some of the values that are used to assemble the `cmd` string comes from objects referenced by the `kustomization` object, including a token extracted from a secret by way

© 2021 Ada Logics Ltd.

of a service account, which in the above code is the value returned by `imp.GetServiceAccountToken`

The value returned by `imp.GetServiceAccountToken` is a string and no sanitization is done on it, so from the perspective of the above code it can in essence be an arbitrary string. As an example, consider the value " || mkdir /tmp/fromsecret2" being returned, then this string will be interpreted as a command and be executed as part of the CombinedOutput call.

A similar code pattern is present in the `validate` function.

7.1.0.1 Steps to reproduce:

To make the proof-of-concept easier to construct we comment out the following lines in **GetClient (in** kustomization_impersonation.go). To the best of our knowledge this does not have implications on the exploitability of the command injections.

```
func (ki *KustomizeImpersonation) GetClient(ctx context.Context) (client.Client,
*polling.StatusPoller, error) {
      if ki.kustomization.Spec.KubeConfig == nil {
            //if ki.kustomization.Spec.ServiceAccountName != "" {
            //     return ki.clientForServiceAccount(ctx)
            //}

            return ki.Client, ki.statusPoller, nil
      }
      return ki.clientForKubeConfig(ctx)
}
```

Then, follow the documented steps on how to run the controller locally:
https://github.com/fluxcd/kustomize-controller/blob/74f08c3f1b3ec8ab8725c7775da8d32903996835/CONTRIBUTING.md#how-to-run-the-controller-locally

Create the following secret and notice the value of the secret being similar to the example above:

```
kubectl create secret generic build-robot43-token --from-literal=token=" || mkdir
/tmp/fromsecret2"
```

Then apply the following .yaml file. The file simply sets up a service account that refers to the secret as well as a customization object that refers to the service account.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: 2015-06-16T00:12:59Z
  name: build-robot43
  namespace: default
  resourceVersion: "272500"
```

© 2021 Ada Logics Ltd.

```
automountServiceAccountToken: false
secrets:
- name: build-robot43-token
---
apiVersion: kustomize.toolkit.fluxcd.io/v1beta1
kind: Kustomization
metadata:
  name: webapp-dev43
spec:
  interval: 5m
  path: "./deploy/webapp/"
  prune: false
  sourceRef:
    kind: GitRepository
    name: webapp-latest
  serviceAccountName: build-robot43
  validation: client
  healthChecks:
    - kind: Deployment
      name: backend
      namespace: webapp
    - kind: Deployment
      name: frontend
      namespace: webapp
  timeout: 2m
```

Following this, you should see a directory `/tmp/fromsecret2` where your customization controller runs. To execute other commands you simply set the value in the secret to be something different.

We consider this to be high severity since it leads to executing an arbitrary command through a controller that runs with cluster-admin privileges. As such, this is an avenue for potential privilege escalation in that controlling the value of the custom resources leads to arbitrary execution by way of the kustomize controller.

We consider the difficulty to be medium since the set up is fairly simple, but an attacker needs to be able to create an arbitrary secret as well as being able to create the correct kustomization objects. Thus, an attacker should already have transcended various trust boundaries. In the event that a non-admin user creates a service account with a corresponding secret to be consumed by the kustomize controller, then they can effectively execute commands with the privileges of the controller.

7.1.1 Recommendation

1. Do not use `exec.CommandContext` unless strictly needed, which it does not seem to be in this case. Preferably use API calls instead.
2. Sanitize input that is used in sensitive operations.

© 2021 Ada Logics Ltd.

7.1.2 Status

A fix proposed and merged in kustomize-controller v0.15.0 and specifically this PR **https://github.com/fluxcd/kustomize-controller/pull/426**

A security advisory is published here.

© 2021 Ada Logics Ltd.

7.2 Issue 2: Nil-dereference in image-automation controller

Severity	Low
Difficulty	Medium
Target	image-automation-controller/controllers/imageupdateautomation_controller.go
Finding ID	ADA-FLUX-21-02

7.2.1 Description

Consider the code in the Reconciler function of image-automation controller:

```
// validate the git spec and default any values needed later, before proceeding
var ref *sourcev1.GitRepositoryRef
if gitSpec.Checkout != nil {
	ref = &gitSpec.Checkout.Reference
	tracelog.Info("using git repository ref from .spec.git.checkout", "ref",
ref)
} else if r := origin.Spec.Reference; r != nil {
	ref = r
	tracelog.Info("using git repository ref from GitRepository spec", "ref",
ref)
} // else remain as `nil`, which is an acceptable value for cloneInto, later.

var pushBranch string
if gitSpec.Push != nil {
	pushBranch = gitSpec.Push.Branch
	tracelog.Info("using push branch from .spec.push.branch", "branch",
pushBranch)
} else {
	// Here's where it gets constrained. If there's no push branch
	// given, then the checkout ref must include a branch, and
	// that can be used.
	if ref.Branch == "" {
		failWithError(fmt.Errorf("Push branch not given explicitly, and
cannot be inferred from .spec.git.checkout.ref or GitRepository .spec.ref"))
	}
	pushBranch = ref.Branch
	tracelog.Info("using push branch from $ref.branch", "branch", pushBranch)
}
```

In the above code that ref can be nil when the code `if ref.Branch == ""` executes, if the following conditions are true:

- `gitSpec.Checkout` is nil

- `origin.Spec.Reference` is nil
- `gitSpec.push` is nil

7.2.1.1 Steps to reproduce:

Launch the image-automation-controller locally with `make run`

Run `kubectl apply -f ./nilderef.yaml` with the following `nilderef.yaml` file:

```
apiVersion: source.toolkit.fluxcd.io/v1beta1
kind: GitRepository
metadata:
  name: sample-repo
spec:
  interval: 1m
  url: https://github.com/stefanprodan/podinfo
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: imageupdateautomation-sample
spec:
  interval: 5m
  sourceRef:
    kind: GitRepository # the only valid value, but good practice to be explicit
here
    name: sample-repo
  git:
    commit:
      author:
        name: fluxbot
        email: fluxbot@example.com
      messageTemplate: |
        An automated update from FluxBot

        [ci skip]
      signingKey:
        secretRef:
          name: git-pgp
  update:
    strategy: Setters
    path: ./cluster/sample
```

The image-automation controller will panic and crash:

```
panic: runtime error: invalid memory address or nil pointer dereference
[signal SIGSEGV: segmentation violation code=0x1 addr=0x8 pc=0x156a969]

goroutine 372 [running]:
```

© 2021 Ada Logics Ltd.

```
github.com/fluxcd/image-automation-controller/controllers.(*ImageUpdateAutomationReconciler).Reconcile(0xc00090b4c0, {0x1b31258, 0xc000548660}, {{{0xc000b9ec39, 0x17d2560}, {0xc00004c960, 0xc000754780}}})

image-automation-controller/controllers/imageupdateautomation_controller.go:214 +0xc09
sigs.k8s.io/controller-runtime/pkg/internal/controller.(*Controller).reconcileHandler(0xc00063ae60, {0x1b311b0, 0xc000a040c0}, {0x174dfa0, 0xc00071a2e0})
        ....
```

A malformed kubernetes object can crash a loop of the image-automation controller which could impact the delivery pipeline. The execution of creating objects that trigger the nil-dereference is fairly simple and given that the nilderef.yaml closely resembles a valid configuration a misconfiguration may happen.

7.2.2 Recommendation

Change the code of the if-statement where the nil-dereference happen to fail if ref is nil:

```
if (ref == nil || ref.Branch == "") {
      return failWithError(fmt.Errorf("Push branch not given explicitly, and cannot be inferred from .spec.git.checkout.ref or GitRepository .spec.ref"))
}
pushBranch = ref.Branch
tracelog.Info("using push branch from $ref.branch", "branch", pushBranch)
```

© 2021 Ada Logics Ltd.

7.3 Issue 3: Credentials exposed in environment variables and command line arguments

Severity	Medium
Difficulty	High
Target	https://github.com/fluxcd/flux2
Finding ID	ADA-FLUX-21-03

7.3.1 Description

Flux-CLI uses credentials to bootstrap many commands. These are often placed in the command line or environment variables. This means the credentials are exposed to a wider audience than intended, e.g. an internal attacker with a host position will be able to watch credentials through various utilities such as via ps. This could lead to credentials being stolen if someone has access to the host at which the command line was entered but not the access to the token of a given Github, as then the details can be leaked.

In the Kubernetes Security audit 2019 this type of attack was classified Medium in severity (https://github.com/kubernetes/community/blob/master/sig-security/security-audit-2019/findings/Kubernetes%20Final%20Report.pdf finding ID: TOB-K8S-005) and to keep consistency we follow this severity.

7.3.2 Recommendation

Make it possible to write sensitive data in way that it won't be exposed, such as: ***** or not showing any characters when being typed.

© 2021 Ada Logics Ltd.

7.4 Issue 4: Use of deprecated library

Severity	Low
Difficulty	High
Target	notification-controller/internal/server/event_server.go
Finding ID	ADA-FLUX-21-04

7.4.1 Description

Flux uses the internal io/ioutil package which was deprecated in Go 1.16 (released in february 2021) such as here:
https://github.com/fluxcd/notification-controller/blob/main/internal/server/event_server.go#L25

The io/ioutil packages was deprecated in Go 1.16 as seen in the announcement:
https://golang.org/doc/go1.16#ioutil

The deprecation was not due to security issues and as such it does not pose any immediate risk. However, the use of deprecated libraries is discouraged and can lead to situations where security issues in a library are found but never patched.

7.4.2 Recommendation

Switch from a deprecated library to a maintained library. In this case the deprecated library was replaced by other GO standards and we recommend using these.

© 2021 Ada Logics Ltd.

7.5 Issue 5: Invalid and missing testing documentation

Severity	Informational
Difficulty	N/A
Target	
Finding ID	ADA-FLUX-21-05

7.5.1 Description

The documentation for running the tests for the controllers is missing information and we had to take several more steps in order for them to work, for example on the need for creating custom resources beforehand and more.

This item is informational and does not represent a vulnerability in and of itself, however, it is imperative to ensure tests are properly run. An example of where there is no documentation on how to run the image-automation controller locally
https://github.com/fluxcd/image-automation-controller/blob/main/CONTRIBUTING.md

7.5.2 Recommendation

Provide proper documentation on how to build, test and run the Flux code.

© 2021 Ada Logics Ltd.

7.6 Issue 6: Bug fixes do not always include regression tests

Severity	Informational
Difficulty	N/A
Target	
Finding ID	ADA-FLUX-21-06

7.6.1 Description

Examples of bug fixes that do not include regression tests:

- https://github.com/fluxcd/source-controller/pull/49
- https://github.com/fluxcd/source-controller/pull/417

Issues that come up can often be introduced in different settings later on and regression testing is an important tool to prevent re-introducing known bugs and crashes.

7.6.2 Recommendation

Create regression tests for reported issues.

© 2021 Ada Logics Ltd.

7.7 Issue 7: Deprecated SHA-1 is used for checksums

Severity	Low
Difficulty	High
Target	https://github.com/fluxcd/source-controller/blob/main/controllers/storage.go
Finding ID	ADA-FLUX-21-07

7.7.1 Description

In the storage utility of the source controller checksums on the collected artifacts are calculated using SHA-1. SHA-1 is considered deprecated as collision attacks against SHA-1 are feasible.

7.7.2 Recommendation

Switch from SHA-1 to SHA-2 in checksum calculations.

7.7.3 Additional note

We want to emphasize here that it is only in the checksum calculation SHA-1 is recommended not to be used. In the HMAC implementation of Flux SHA-1 is still safe to use.

© 2021 Ada Logics Ltd.

7.8 Issue 8: Missing checksum verification

Severity	Medium
Difficulty	High
Target	https://github.com/fluxcd/source-controller/blob/main/controllers/storage.go https://github.com/fluxcd/source-controller/blob/main/controllers/gitrepository_controller.go
Finding ID	ADA-FLUX-21-08

7.8.1 Description

In the source-controller checksums are calculated when fetching artifacts from S3 stores (bucket_reconciler.go) and git repositories (gitrepository_controller.go). Specifically, the checksum is calculated here:
https://github.com/fluxcd/source-controller/blob/d7afc3596bdfc3818ed8987db029bea8a461c62c/controllers/storage.go#L272

This function is called by both the git repository reconciler and the bucket reconciler, but it is never checked in the reconcilers despite comments indicating "check integrity".

- https://github.com/fluxcd/source-controller/blob/d7afc3596bdfc3818ed8987db029bea8a461c62c/controllers/gitrepository_controller.go#L350

We assume this integrity check should be matched with data from the source where you download and that is a good idea. However, there is never any check on the checksums calculated.

7.8.2 Recommendation

Ensure proper verification of checksums.

© 2021 Ada Logics Ltd.

7.9 Issue 9 Inconsistent and missing logging

Severity	Low
Difficulty	High
Target	Flux controllers
Finding ID	ADA-FLUX-21-09

7.9.1 Description

Proper logging is important to ensure audit trails in case of breaches and, in general, ensure non-repudiation of the system in case an attack happens. However, throughout the code we found inconsistency in the way logging is handled, and often when errors occur there would be no error logging.

The ImageAutomationReconciler declares the following `failWithError` function:

```
// failWithError is a helper for bailing on the reconciliation.
failWithError := func(err error) (ctrl.Result, error) {
	r.event(ctx, auto, events.EventSeverityError, err.Error())
	imagev1.SetImageUpdateAutomationReadiness(&auto,
metav1.ConditionFalse, meta.ReconciliationFailedReason, err.Error())
	if err := r.patchStatus(ctx, req, auto.Status); err != nil {
		log.Error(err, "failed to reconcile")
	}
	return ctrl.Result{Requeue: true}, err
}
```

This function is consistently used to return from the Reconcile function in an appropriate manner with proper logging, e.g:

```
access, err := r.getRepoAccess(ctx, &origin)
if err != nil {
	return failWithError(err)
}
```

This is a good way of abstracting common logic and ensuring proper logging. However, this approach is never used in any of the other controllers and these controllers implement the logic quite differently, e.g.

Source controller:

© 2021 Ada Logics Ltd.

```
// update status with the reconciliation result
if err := r.updateStatus(ctx, req, reconciledRepository.Status); err !=
nil {
      log.Error(err, "unable to update status")
      return ctrl.Result{Requeue: true}, err
}

// if reconciliation failed, record the failure and requeue immediately
if reconcileErr != nil {
      r.event(ctx, reconciledRepository, events.EventSeverityError,
reconcileErr.Error())
      r.recordReadiness(ctx, reconciledRepository)
      return ctrl.Result{Requeue: true}, reconcileErr
}
```

ImageReflector controller:

```
// check if we are allowed to use the referenced ImageRepository
if _, err := r.hasAccessToRepository(ctx, req,
pol.Spec.ImageRepositoryRef, repo.Spec.AccessFrom); err != nil {
      imagev1.SetImagePolicyReadiness(
            &pol,
            metav1.ConditionFalse,
            "AccessDenied",
            err.Error(),
      )
      if err := r.patchStatus(ctx, req, pol.Status); err != nil {
            return ctrl.Result{Requeue: true}, err
      }
      log.Error(err, "access denied")
      return ctrl.Result{}, nil
}
```

Alert controller:

```
// validate alert spec and provider
if err := r.validate(ctx, alert); err != nil {
      meta.SetResourceCondition(&alert, meta.ReadyCondition,
metav1.ConditionFalse, meta.ReconciliationFailedReason, err.Error())
      if err := r.patchStatus(ctx, req, alert.Status); err != nil {
            return ctrl.Result{Requeue: true}, err
      }
      return ctrl.Result{Requeue: true}, err
}
```

© 2021 Ada Logics Ltd.

7.9.2 Recommendation

1. Standardise how logging should occur.
2. Use helper methods for common error handling.
3. Log whenever errors occur.
4. Log differently depending on how each controller exits the Reconcile functions.

© 2021 Ada Logics Ltd.

7.10 Issue 10: Reading large files can crash flux with an out-of-memory bug

Severity	Low
Difficulty	High
Target	Helmchart controller
Finding ID	ADA-FLUX-21-10

7.10.1 Description

There are two locations in the Helmchart controller where the controller loads a Helm chart into memory by reading all of the Helm chart using io.ReadAll without checking the size. As such, if a large Helmchart is provided this can cause the controller to be killed with an Out-Of-Memory (OOM) error. The io.ReadAll function needs to be used with care due to this issue. As such, if an attacker can taint a Helmchar chart to be large in size, then a denial-of-service attack can occur.

The code where io.ReadAll is used are here:
https://github.com/fluxcd/source-controller/blob/main/controllers/helmchart_controller.go#L329-L333

```
indexFile, err :=
os.Open(r.Storage.LocalPath(*repository.GetArtifact()))
if err != nil {
      return sourcev1.HelmChartNotReady(chart,
sourcev1.StorageOperationFailedReason, err.Error()), err
}
b, err := io.ReadAll(indexFile)
```

And here:
https://github.com/fluxcd/source-controller/blob/d7afc3596bdfc3818ed8987db029bea8a461c62c/internal/helm/repository.go#L208-L212

```
func (r *ChartRepository) DownloadIndex() error {
u, err := url.Parse(r.URL)
if err != nil {
      return err
}
u.RawPath = path.Join(u.RawPath, "index.yaml")
u.Path = path.Join(u.Path, "index.yaml")

res, err := r.Client.Get(u.String(), r.Options...)
```

© 2021 Ada Logics Ltd.

```
if err != nil {
      return err
}
b, err := io.ReadAll(res)
```

7.10.2 Recommendation

Validate size of input before reading data with `io.ReadAll` and in general avoid loading large arbitrary data into memory regardless of size. Set a strict upper limit, and perhaps a user-configurable limit. In addition to this, provide logging around the file loading, in particular when a large file is loaded, as this will be important in auditing logs.

© 2021 Ada Logics Ltd.

7.11 Issue 11: Files are opened but never closed

Severity	Low
Difficulty	High
Target	https://github.com/fluxcd/source-controller/blob/main/controllers/helmchart_controller.go
Finding ID	ADA-FLUX-21-11

7.11.1 Description

There are two occurrences in Helmchart controller where a file is opened but never closed here and here. This can lead to resource leaks and exhaustion of available file descriptors in the process. An example of the code pattern is shown here:

```
indexFile, err :=
os.Open(r.Storage.LocalPath(*repository.GetArtifact()))
if err != nil {
      return sourcev1.HelmChartNotReady(chart,
sourcev1.StorageOperationFailedReason, err.Error()), err
}
b, err := io.ReadAll(indexFile)
```

7.11.2 Recommendation

Fix the file descriptor leak by appropriately closing opened files. This can be performed with a deferred indexFile.Close() operation, however, notice issue 20 in this document on handling deferring file close operations.

7.12 Issue 12: Unhandled error

Severity	Low
Difficulty	High
Target	image-automation-controller/controllers/imageupdateautomation_controller.go
Finding ID	ADA-FLUX-21-12

7.12.1 Description

Unhandled error in the image-automation controller, despite the error being assigned to a variable, which can lead to undefined behaviour.

```
var signingEntity *openpgp.Entity
if gitSpec.Commit.SigningKey != nil {
      signingEntity, err = r.getSigningEntity(ctx, auto)
}
```

7.12.2 Recommendation

We propose to change the above code to the following:

```
var signingEntity *openpgp.Entity
if gitSpec.Commit.SigningKey != nil {
    if signingEntity, err = r.getSigningEntity(ctx, auto); err != nil {
        return failWithError(err)
    }
}
```

© 2021 Ada Logics Ltd.

7.13 Issue 13: Slice bounds out of range

Severity	Low
Difficulty	High
Target	https://github.com/fluxcd/image-automation-controller/blob/main/pkg/update/setters.go#L162
Finding ID	ADA-FLUX-21-13

7.13.1 Description

An issue was found by the `FuzzUpdateWithSetters` fuzzer of the image-automation-controller. The following stack trace is produced after a few minutes of fuzzing:

```
panic: runtime error: slice bounds out of range [:-4]

goroutine 17 [running, locked to thread]:
github.com/fluxcd/image-automation-controller/pkg/update.UpdateWithSetters(0x2704df0, 0x3bd17d0, 0xc0001b70b0, 0x17, 0xc0001b7188, 0x17, 0xc00034f680, 0x1, 0x1, 0x0, ...)
        /image-automation-controller/pkg/update/setters.go:162 +0x12e9
github.com/fluxcd/image-automation-controller/controllers.FuzzUpdateWithSetters(0x5922c00, 0x17f, 0x17f, 0x0)
        /image-automation-controller/controllers/fuzz.go:388 +0x65b
main.LLVMFuzzerTestOneInput(0x5922c00, 0x17f, 0x4b0001)

github.com/fluxcd/image-automation-controller/controllers/go.fuzz.main/main.go:35 +0x66
==8== ERROR: libFuzzer: deadly signal
    #0 0x4b20d0 in __sanitizer_print_stack_trace
(/fuzzers/FuzzUpdateWithSetters+0x4b20d0)
    #1 0x45da28 in fuzzer::PrintStackTrace()
(/fuzzers/FuzzUpdateWithSetters+0x45da28)
    #2 0x443a63 in fuzzer::Fuzzer::CrashCallback()
(/fuzzers/FuzzUpdateWithSetters+0x443a63)
    #3 0x7fd3900c38df  (/lib/x86_64-linux-gnu/libpthread.so.0+0x138df)
    #4 0x521ca0 in runtime.raise runtime/sys_linux_amd64.s:163

NOTE: libFuzzer has rudimentary signal handlers.
      Combine libFuzzer with AddressSanitizer or similar for better crash reports.
SUMMARY: libFuzzer: deadly signal
MS: 2 CrossOver-InsertByte-; base unit: a00037129c43765719c94c4c994e5652daa461b8
artifact_prefix='./'; Test unit written to
./crash-e93c32a8ad39fe95da43be27e31c5fb4d720efb9
```

The issue is present on this line:

```
name := image[:len(image)-len(tag)-1]
```

7.13.2 Recommendation

Fix the issue by checking the length of the `image` and `tag`.

© 2021 Ada Logics Ltd.

7.14 Issue 14: Possible nil-deref in image-automation controller

Severity	Low
Difficulty	High
Target	https://github.com/fluxcd/image-automation-controller/blob/main/controllers/imageupdateautomation_controller.go
Finding ID	ADA-FLUX-21-14

7.14.1 Description

A possible nil-pointer dereference exists in the image update automation controller. Consider the following code:

```
if rev, err := commitChangedManifests(tracelog, repo, tmp, signingEntity, author,
messageBuf.String()); err != nil {
        if err == errNoChanges {
                r.event(ctx, auto, events.EventSeverityInfo, "no updates made")
                debuglog.Info("no changes made in working directory; no commit")
                statusMessage = "no updates made"
                if lastCommit, lastTime := auto.Status.LastPushCommit,
auto.Status.LastPushTime; lastCommit != "" {
                        statusMessage = fmt.Sprintf("%s; last commit %s at %s",
statusMessage, lastCommit[:7], lastTime.Format(time.RFC3339))
                }
        } else {
                return failWithError(err)
        }
}
```

When the `lastTime.Format` is called it is not certain that `lastTime` is not a nil-pointer. As such, similar to a check on `lastCommit` there should be a check on whether `lastTime` is nil.

7.14.2 Recommendation

We did not pursue a proof-of-concept for this potential bug due to timing constraints, but recommend for the developers to review the issue as through our analysis it is triggerable.

© 2021 Ada Logics Ltd.

7.15 Issue 15: Inconsistent code-styles and potential nil-dereferences

Severity	Informational
Difficulty	N/A
Target	Controllers
Finding ID	ADA-FLUX-21-15

7.15.1 Description

Flux is composed of projects across different repositories and there is often similar logic happening across the controllers but performed in quite different ways. This leads to a more complex overall codebase and can make it difficult to reason about properties of the code.

Event recording and checking status of similar elements in the controllers is performed differently. This came up as an issue through fuzzing due to nil-pointer dereferences. Each of the controllers rely on an EventRecoder, and the way these EventRecorder variables are used differs between the controllers. Some controllers check for nil-status and others do not. The HelmRelease reconciler and the Kustomization reconciler assume that the `EventRecorder` is not nil in their respective event() implementations, whereas the other controls do not:

Helm Release Reconciler
https://github.com/fluxcd/helm-controller/blob/main/controllers/helmrelease_controller.go#L739

```
func (r *HelmReleaseReconciler) event(ctx context.Context, hr
v2.HelmRelease, revision, severity, msg string) {
      r.EventRecorder.Event(&hr, "Normal", severity, msg)
      objRef, err := reference.GetReference(r.Scheme, &hr)
      if err != nil {
            logr.FromContext(ctx).Error(err, "unable to send event")
            return
      }
```

Kustomize Reconciler
https://github.com/fluxcd/kustomize-controller/blob/main/controllers/kustomization_controller.go#L788

```
func (r *KustomizationReconciler) event(ctx context.Context,
kustomization kustomizev1.Kustomization, revision, severity, msg string,
metadata map[string]string) {
      log := logr.FromContext(ctx)
      r.EventRecorder.Event(&kustomization, "Normal", severity, msg)
```

```
	objRef, err := reference.GetReference(r.Scheme, &kustomization)
	if err != nil {
		log.Error(err, "unable to send event")
		return
	}
```

Image Update Automation Reconciler
https://github.com/fluxcd/image-automation-controller/blob/main/controllers/imageupdateautomation_controller.go#L732

```
func (r *ImageUpdateAutomationReconciler) event(ctx context.Context,
auto imagev1.ImageUpdateAutomation, severity, msg string) {
	if r.EventRecorder != nil {
		r.EventRecorder.Event(&auto, "Normal", severity, msg)
	}
```

Git Repository Reconciler
https://github.com/fluxcd/source-controller/blob/main/controllers/gitrepository_controller.go#L427

```
func (r *GitRepositoryReconciler) event(ctx context.Context, repository
sourcev1.GitRepository, severity, msg string) {
	log := logr.FromContext(ctx)

	if r.EventRecorder != nil {
		r.EventRecorder.Eventf(&repository, "Normal", severity, msg)
	}
```

Recommendation

The same code pattern should be used across the controllers. Through our analysis we determined the EventRecorder cannot be nil using the current `main.go` files and thus the nil check should be removed.

© 2021 Ada Logics Ltd.

7.16 Issue 16: Missing return statement after error

Severity	Low
Difficulty	High
Target	image-automation-controller/controllers/imageupdateautomation_controller.go
Finding ID	ADA-FLUX-21-16

7.16.1 Description

In the Image-automation reconciler an issue is present where the code continues after an error occurs, leading to undefined behaviour:

```
// Here's where it gets constrained. If there's no push branch
// given, then the checkout ref must include a branch, and
// that can be used.
if ref.Branch == "" {
      failWithError(fmt.Errorf("Push branch not given explicitly, and
cannot be inferred from .spec.git.checkout.ref or GitRepository
.spec.ref"))
}
pushBranch = ref.Branch
tracelog.Info("using push branch from $ref.branch", "branch",
pushBranch)
```

The problem is that the failWithError branch does not itself return from the reconcile function and the code should instead be `return failWithError`

7.16.2 Recommendation

Change the code to `return failWithError`

© 2021 Ada Logics Ltd.

7.17 Issue 17: File extension comparisons are case sensitive

Severity	Low
Difficulty	High
Target	https://github.com/fluxcd/kustomize-controller/blob/main/controllers/kustomization_generator.go#L195-L197
Finding ID	ADA-FLUX-21-17

7.17.1 Description

In the lines
https://github.com/fluxcd/kustomize-controller/blob/main/controllers/kustomization_generator.go#L195-L197

A check happens on the file extension for YAML files:

```
extension := filepath.Ext(path)
if !containsString([]string{".yaml", ".yml"}, extension) {
      return nil
```

However, the check is case sensitive. We recommend either documenting this or allowing file extensions that are not purely limited to lowercase extensions. The problem is if a user configures their set up with yaml files based on different syntactic extensions they will suspect their yaml files to erroneously assume their files are used.

7.17.2 Recommendation

Normalise file extensions when processing them to allow uppercase letters in extensions or clarify that files must be lowercase.

© 2021 Ada Logics Ltd.

7.18 Issue 18: Some dependencies are outdated

Severity	Informational
Difficulty	N/A
Target	N/A
Finding ID	ADA-FLUX-21-18

7.18.1 Description

Some of the dependencies of various go modules are not up to date and archived. This includes:

Name & fluxcd version	Import location	Flux version	Latest upstream tag
Go-retryablehttp	https://github.com/fluxcd/kustomize-controller/blob/74f08c3f1b3ec8ab8725c7775da8d32903996835/go.mod#L19	v0.6.8	0.7.0
go-grpc	https://github.com/fluxcd/kustomize-controller/blob/74f08c3f1b3ec8ab8725c7775da8d32903996835/go.mod#L27	v1.38.0	v1.41.0
howeyc/gopass	https://github.com/fluxcd/kustomize-controller/blob/74f08c3f1b3ec8ab8725c7775da8d32903996835/go.mod#L20	v.0.0.0-2	Archived
filippio.io/age	https://github.com/fluxcd/kustomize-controller/blob/74f08c3f1b3ec8ab8725c7775da8d32903996835/go.mod#L8	v1.0.0-beta7	v1.0.0.0
minio-go	https://github.com/fluxcd/source-controller/blob/d7afc3596bdfc3818ed8987db029bea8a461c62c/go.mod#L24	v7.0.10	v7.0.14
sigs.k8s.io/controller-runtime	https://github.com/fluxcd/source-controller/blob/d7afc3596bdfc3818ed8987db029bea8a461c62c/go.mod#L35	v0.9.5	v0.10.1
sigs.k8.io/yaml	https://github.com/fluxcd/kustomize-controller/blob/74f08c3f1b3ec8ab8725c7775da8d32903996835/go.mod#L35	v1.2.0	v1.3.0
go-limiter	https://github.com/fluxcd/notification-controller/blob/8ff5f75a255d7dd0677590510ab1abf1d33b4f85/go.mod#L25	v0.6.0	v0.7.2

© 2021 Ada Logics Ltd.

Using outdated versions may result in vulnerabilities in the event of lacking security updates and bug fixes. In this case we did not investigate further if any of the versions were updated due to security issues.

7.18.2 Recommendation

We recommend in the short term to update the packages and in the long term to ensure all dependencies are up to date by using automated dependency checking.

© 2021 Ada Logics Ltd.

7.19 Issue 19: Lack of container security options in deployed pods

Severity	Low
Difficulty	High
Target	Flux controllers
Finding ID	ADA-FLUX-21-19

7.19.1 Description

The Deployments of the Flux controllers lack container security options that mitigate privilege escalation risks. These are hardening options that we recommend using and Flux already makes use of `allowPrivilegeEscalation: falseallowPrivilegeEscalation: false` on all of its controllers. However, in addition to the `allowPrivilegeEscalation` option Flux could harden its containers by:

- Dropping all Linux capabilities and enabling those needed
- Filtering syscalls by way of Seccomp

Docker drops many Linux capabilities by default but keeps others for convenience. Flux can harden its containers by having Docker drop all privileges a root user's process can perform on a system and enabling only those needed. See here for details.

Seccomp filtering is a way of limiting the available system calls and as of v1.19 Kubernetes has support for specifying Seccomp policies through the use of `seccompProfile` in the `securityContext` of pods. Please see here for details.

7.19.2 Recommendation

Ensure that the pod deployed by Flux has appropriate hardening applied through the use of dropping Linux capabilities and syscall filtering.

© 2021 Ada Logics Ltd.

7.20 Issue 20: Unhandled errors from deferred file close operations

Severity	Low
Difficulty	High
Target	./source-controller/controllers/storage.go ./source-controller/pkg/sourceignore/sourceignore.go ./kustomize-controller/controllers/kustomization_impersonation.go ./kustomize-controller/internal/sops/pgp/keysource.go
Finding ID	ADA-FLUX-21-20

7.20.1 Description

Throughout the codebase there are places where file close operations are deferred within a function where a file is being written to, e.g:

```
localPath := s.LocalPath(*artifact)
f, err := os.Open(localPath)
if err != nil {
  return err
}
defer f.Close()

// untar the artifact
untarPath := filepath.Join(tmp, "unpack")
if _, err = untar.Untar(f, untarPath); err != nil {
  return err
}
```

This can lead to undefined behaviour since any errors returned by the `f.Close()` operation are ignored. This can have consequences in the event a close operation fails and the data has not yet been flushed to the file, which the rest of the code will assume it to be. For a detailed discussion on this, please see here.

7.20.2 Recommendation

Ensure that errors from `f.Close()` are handled.

© 2021 Ada Logics Ltd.

7.21 Issue 21: x509 certificates are not used for Webex

Severity	Medium
Difficulty	High
Target	notification-controller/internal/notifier/webex.go
Finding ID	ADA-FLUX-21-21

7.21.1 Description

In the alert-controller the Webex hook takes an x509 certificate as argument but does not use it here:

```
// NewWebex validates the Webex URL and returns a Webex object
func NewWebex(hookURL, proxyURL string, certPool *x509.CertPool)
(*Webex, error) {
	_, err := url.ParseRequestURI(hookURL)
	if err != nil {
		return nil, fmt.Errorf("invalid Webex hook URL %s", hookURL)
	}

	return &Webex{
		URL:      hookURL,
		ProxyURL: proxyURL,
	}, nil
}
```

This is despite the certPool being used by the Webex notifier here:

```
if err := postMessage(s.URL, s.ProxyURL, s.CertPool, payload); err !=
nil {
	return fmt.Errorf("postMessage failed: %w", err)
}
```

7.21.2 Recommendation

Ensure certificate is used or clarify why it is not in the documentation.

© 2021 Ada Logics Ltd.

7.22 Issue 22: Unnecessary conditions in the code

Severity	Informational
Difficulty	N/A
Target	https://github.com/fluxcd/image-automation-controller/blob/7ec4e6150ee0b6f4770a3265161cd98108c06603/controllers/imageupdateautomation_controller.go#L253-L264
Finding ID	ADA-FLUX-21-22

7.22.1 Description

In the following code, the `auto.Spec.Update` is not needed, and if indeed `auto.Spec.Update` was nil then the first conditional statement would panic due to a nil-dereference.

```
manifestsPath := tmp
if auto.Spec.Update.Path != "" {
      tracelog.Info("adjusting update path according to
.spec.update.path", "base", tmp, "spec-path", auto.Spec.Update.Path)
      if p, err := securejoin.SecureJoin(tmp, auto.Spec.Update.Path);
err != nil {
            return failWithError(err)
      } else {
            manifestsPath = p
      }
}

switch {
case auto.Spec.Update != nil && auto.Spec.Update.Strategy ==
imagev1.UpdateStrategySetters:
      // For setters we first want to compile a list of _all_ the
```

7.22.2 Recommendation

Remove the unnecessary nil-pointer check code.

© 2021 Ada Logics Ltd.