PRESENTS

Crossplane security audit

In collaboration with the Crossplane maintainers, Open Source Technology Improvement Fund and The Linux Foundation

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 27th July 2023

Table of contents

Executive summary

This report contains the results from Crossplane's 2023 security audit carried out by Ada Logics. The audit was funded by the Cloud Native Computing Foundation and was facilitated by the Open Source Technology Improvement Fund.

The engagement was a time-based, holistic security audit that had the following high-level goals:

1. Formalise a threat model of Crossplane
2. Carry out a manual code audit of Crossplane
3. Review Crossplane's fuzzing suite against the threat model from goal #1
4. Review Crossplane's SLSA compliance

During the threat modelling goal, we found that the container image registry represented a significant attack surface for Crossplane. This guided the work in goals #2 and #3 to consider a specific set of attacks that have received increased attention in recent years: Supply chain attacks. The findings from the threat modelling combined with the manual code auditing goal resulted in several findings from the supply chain class of vulnerabilities that were reachable from the container image registry. Two of the issues were assigned CVEs with Low and High severity. At the completion of the audit, Crossplane has fixed all issues except for a single one which impacts an alpha feature.

Ada Logics added 4 fuzzers to Crossplane's fuzzing suite. These fuzzers target parts of Crossplane that we found to be exposed from attacks to the registry during the threat modelling goal of the audit. We added the fuzzers to the packages that they target in the Crossplane repository. Crossplane is integrated into OSS-Fuzz in such a way that all fuzzers from the Crossplane and Crossplane-runtime repositories are included during OSS-Fuzz build cycles. As such, the fuzzers we added in this audit run continuously as well as in Crossplane's CI.

The SLSA goal of the audit found that Crossplane performs well on all accounts except for provenance distribution along with releases. Adding provenance will allow adopters to avoid issues in their supply chain by verifying the Crossplane artefacts against provenance attestations.

Results summarised

5 new fuzzers added to Crossplane's fuzzing suite
16 issues security found
2 CVE's assigned
All issues minus 1 (in an alpha feature) are fixed at the end of the audit by the Crossplane team

Project Summary

The auditors of Ada Logics were:

Name	Title	Email
Adam Korczynski	Security Engineer, Ada Logics	Adam@adalogics.com
David Korczynski	Security Researcher, Ada Logics	David@adalogics.com

The Crossplane community members involved in audit were:

Name	Title	Email
Philippe Scorsolini	Senior Software Engineer, Upbound	philippe.scorsolini@upbound.io
Jared Watts	Community Manager, Upbound	jared@upbound.io
Jean du Plessis	Senior Engineering Manager, Upbound	jean@upbound.io
Nic Cope	Senior Principal Engineer, Upbound	negz@upbound.io

The following facilitators of OSTIF were engaged in the audit:

Name	Title	Email
Derek Zimmer	Executive Director, OSTIF	Derek@ostif.org
Amir Montazery	Managing Director, OSTIF	Amir@ostif.org

Audit Scope

The following assets were in scope of the audit.

Repository	https://github.com/crossplane/crossplane
Language	Go

Repository	https://github.com/crossplane/crossplane-runtime
Language	Go

Threat model formalisation

In this section, we present the threat modelling we did as part of the audit. Threat modelling is a process to identify potential security risks or threats to a system. We analysed Crossplane's components, data flows and attack vectors to identify how threat actors could seek to compromise Crossplane and which goals they would attempt to achieve. During the audit, threat modelling progressed alongside other efforts; We started with an initial threat model of the assets in scope, and as the audit progressed, the manual auditing contributed to a clearer understanding of the threat model. Vice versa, the threat modelling contributed to identifying more security issues in Crossplane's code base. The audit drew organically towards the threats from Crossplane's supply chain attack vectors. The audit was not a dedicated supply chain audit, however, we found this vector to be important and exposed to Crossplane. Crossplane has many characteristics of a package manager considering Debians definition:

"*A package manager keeps track of what software is installed on your computer, and allows you to easily install new software, upgrade software to newer versions, or remove software that you previously installed. As the name suggests, package managers deal with packages: collections of files that are bundled together and can be installed and removed as a group.*"
https://www.debian.org/doc/manuals/aptitude/pr01s02.en.html

Crossplane has all of these qualities which makes it an ideal target for supply chain attacks. Package managers are responsible for fetching software packages from internal or remote sources and installing them in an environment so that they can be executed later. This is by default an exposed attack surface since attackers have a wide range of techniques and tactics at their disposal to comprise the package manager itself or its users. We see Crossplane as a package manager for container images. Crossplane fetches the images from remote sources. Over the last few years the security community has formalised a series of attack vectors for software systems that handle container images from remote sources which we have included in the threat model below.

After we enumerate the attack vectors, we detail how attackers can put themselves in a position to launch an attack through this attack vector. We finally present the threat actors and their objectives that could seek to exploit security holes in Crossplane.

Supply chain attacks

In recent years, much work has been put into identifying how attackers can seek to compromise the software supply chain. Known, high-impact attacks from successful compromises of users' software supply chain have sparked an interest in understanding this attack vector which has led the community to formalising a series of supply chain specific threats against software systems that consume 3rd-party software packages. Crossplane both consumes 3rd-party packages but also implements infrastructure that facilitates the lifecycle of consuming 3rd-party software packages for the user. Both sides are critical to harden against.

Endless data attack
An attack where an attacker tricks a client into downloading an endless stream of data when the client requests it. The resulting impact is that the machine on which the client is running will run indefinitely without completing the task thus preventing the service from carrying out other tasks. Crossplane is exposed to this attack vector in case an attacker can compromise the registry from which Crossplane fetches images.

Rollback attacks
A rollback attack is where a threat actor can trick a client into installing older versions of a given software artefact. The actor would be interested in doing this in case they knew about vulnerabilities in older versions of a software artefact. They would then prevent updates of the artefact and exploit the vulnerabilities of the old version. Crossplane users specify images by either tags or digests. Specifying an image with a digest is the best security practice, however, if an attacker compromises the registry, they may be able to tamper with the image if Crossplane lacks validation against the requested digest.

Freeze attacks
This is an attack where an actor prevents a client from updating software artefacts. The threat actor does this by presenting files to the client that the client is already aware of and is tricked into concluding that there is no update to the artefact. A freeze attack can be enabled by a vulnerability that allows the actor to carry out a Man-in-the-Middle attack, where the actor intercepts requests between the client and the central repository when the client checks for updates. To update a Crossplane package, users are required to manually update manifest files which will prompt Crossplane to pull the new package. An attacker who can block upgrades will be able to launch a freeze attack against Crossplane.

Arbitrary package attacks
An arbitrary package attack is where an attacker can trick a client into downloading a software artefact of the attackers choice. In a successful arbitrary package attack, the client will not be aware that it downloaded the wrong artefact. There are several ways an

attacker could attempt this against Crossplane: Image tampering is one way since a maliciously modified image could be considered an arbitrary package. Attackers could also seek to compromise the dependencies of Crossplane packages since Crossplane pulls dependencies when installing Crossplane packages.

Typosquatting attacks
A typosquatting attack is when an attacker tricks a client into downloading malicious software artifacts by typing a name that closely resembles the safe artifacts name but is still different. Typosquatting attacks are hard for Crossplane to harden against, given that Crossplane trusts the package name.

Crossplane's supply chain attack vectors

An attacker can take several positions to launch a supply chain attack. In this part, we present how an attacker could obtain such a position.

Compromising the registry
An attacker can compromise the registry and control the data the registry responds to requests made to it. This level of compromise allows the attacker to return any data to Crossplane.

Compromising the package in the registry
An attacker can compromise the package that exists in the registry. This requires the attacker to compromise either the registry itself or the maintainer of the package. A successful attack would require that neither the user nor the package manager validate the package at download time. This could for example be enabled by compromising the vendor account that uploads and manages a given package at the registry.

Compromising the infrastructure delivering the package
An attacker can compromise the infrastructure delivering the package to Crossplane. There are two important pieces of infrastructure to Crossplane in this attack vector:

1. The cloud providers' infrastructure. An attacker could compromise the cloud providers infrastructure to get access to the users cluster.
2. Crossplane's go-containerregistry dependency. Crossplane relies on the `github.com/google/go-containerregistry` library to fetch images. Vulnerabilities in go-containerregistry that Crossplane can reach could impact Crossplane in the same manner as if the code lived in Crossplane's own repository.

Crossplane threat scenario

In this section, we exemplify the above observations with a threat scenario. The purpose is to consider a tangible series of observations that an attacker would make to look for vulnerabilities to exploit in Crossplane. The previous sections in our threat modelling have taken a high-level approach, and this section aims to make these more tangible with an example threat scenario.

Crossplane infrastructure teams manage packages through container image references. The infrastructure team will input a reference to Crossplane which will then download an image from a registry specified in the reference. For example, when creating a Provider or Configuration, the user will specify a package which is the image reference:

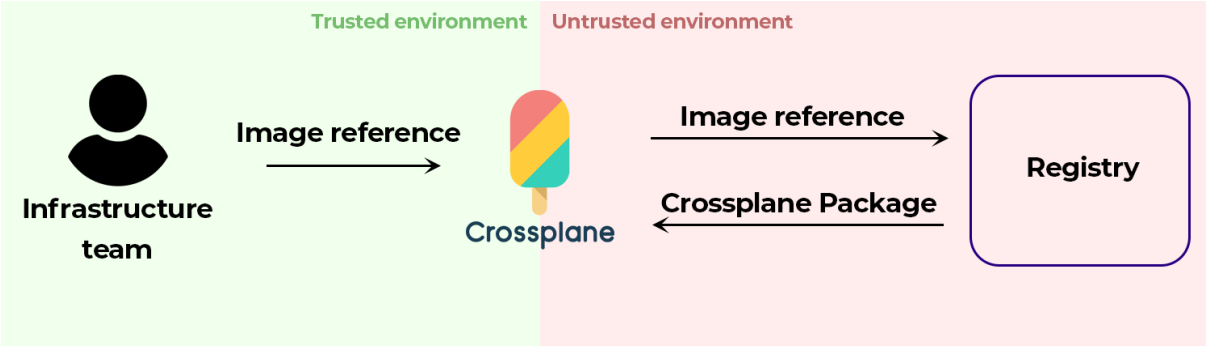
type PackageSpec

```
type PackageSpec struct {
    // Package is the name of the package that is being requested.
    Package string `json:"package"`

    // RevisionActivationPolicy specifies how the package controller should
    // update from one revision to the next. Options are Automatic or Manual.
    // Default is Automatic.
    // +optional
    // +kubebuilder:default=Automatic
    RevisionActivationPolicy *RevisionActivationPolicy `json:"revisionActivationPolicy,omitempty"`

    // RevisionHistoryLimit dictates how the package controller cleans up old
    // inactive package revisions.
    // Defaults to 1. Can be disabled by explicitly setting to 0.
    // +optional
    // +kubebuilder:default=1
    RevisionHistoryLimit *int64 `json:"revisionHistoryLimit,omitempty"`
```

https://pkg.go.dev/github.com/crossplane/crossplane/apis/pkg/v1beta1#PackageRevisionSpec

The Crossplane revision controller will fetch the image defined in the `PackageSpec` from its registry. As such, when Crossplane deals with container images, there are two inputs to Crossplane one of which the Crossplane admin controls, and one of which the admin does not control.

1. The infrastructure team inputs the image reference to Crossplane. In this step, there is no change in trust.
2. Crossplane sends a request to the registry which responds with the Crossplane package. In this step, the trust flows high to low when Crossplane sends the request to the registry and low to high when the registry responds with the package:



In this workflow, Crossplane expects the registry to correctly and safely handle the request from Crossplane to the registry and to return the correct image based on the reference. Even if the infrastructure team has specified the correct image reference using a digest which is more secure, the attack surface from the registry exists. For example, if an attacker has compromised the registry, they can choose to ignore the digest and return an arbitrary data blob to Crossplane. This makes the registry an ideal target for supply chain attacks. Attackers will look for ways to compromise registries as well as missing hardening in Crossplane's Package fetching routines.
If an attacker has compromised the registry, they can potentially fully control the package that Crossplane returns. In this scenario, Crossplane should be hardened against every possible attack vector from the registry, which includes the supply chain attacks listed above as well as non-supply chain-specific privilege escalations.

Crossplane Claims

Claims in Crossplane are declarative requests for a given cloud resource or service. They allow Platform Engineers to define the specific resources that developers can provision and access, while also allowing developers to define their desired characteristics and parameters for those permitted resources. When a user makes a claim, Crossplane provisions and manages the requested resources based on the provided specifications. This separation of concerns is a major advantage of Crossplane which is intended to allow safe self-service provisioning of resources by developers.

A vital security assumption for claims is that users consume the resources that they believe they consume. If an attacker can cause that assumption to be wrong, they can obtain an advantageous security position. For example, a threat actor could interfere with how Crossplane handles claims to attempt to tamper with the resource that the user is expecting. By causing Crossplane to supply a different resource than the requested one, the threat actor would be able to breach the security of Crossplane.

A serious breach related to composite resources is the case where a user - who is not a cluster admin - can assume the role of the platform team and obtain privileges to configure the CompositeResourceDefinitions of the cluster. This could lead to full cluster breach as the threat actor could replace existing CompositeResourceDefinitions with malicious ones or add new CompositeResourceDefinitions that could be used to further escalate privileges. This attack vector is particularly severe if the user can do this by way of a claim since claims are the API that limited privilege users are meant to use and must be secured against privilege escalations.

Threat actors

A threat actor is an individual or group that intentionally attempts to exploit vulnerabilities, deploys malicious code, or compromise or disrupt a Crossplane deployment, often for financial gain, espionage, or sabotage.
We identify the following threat actors for Crossplane. A threat actor can assume multiple profiles from the table below; for example, a fully untrusted user can also be a contributor to a 3rd-party library used by Crossplane.

Actor	Description	Have already escalated privileges
Fully untrusted users	Users that have not been granted any privileges.	No
Platform authors	Users that are responsible for configuring Crossplane, e.g., installing providers/configurations, creating compositions, setting up cloud credentials, etc.	No
Platform consumers	App developers who consume the platform API through claims. They have limited permissions via RBAC. They do not have permission to install a provider for example.	No
Contributors to 3rd-party dependencies	Contributors to dependencies used by Crossplane.	No
Well-funded criminal groups	Organised criminal groups that often have either political or economic goals.	No

Attacker objectives

In this section, we enumerate common objectives observed in the wild amongst attacks against cloud-native systems that apply to Crossplane.

Steal compute power

Threat actors may seek to compromise a Crossplane deployment to steal computational resources. This is a common attack vector for cloud-based infrastructure, and there are numerous examples of such attacks in the wild[1] [2].

The attacker will often attempt to generate new workloads that run cryptominers. Attackers will often use the Monero crypto miners, since there are many images available publicly to deploy Monero mining workloads[3].

Further privilege escalation

Attackers look for vulnerabilities that can allow them to further manifest their position even after having already elevated their privileges. One way is to extend the timeframe of the elevated position by finding a way to maintain their position even after the initial vulnerability has been patched. At the time of this audit, attackers are actively leveraging misconfigurations in RBAC to gain a foothold into Kubernetes clusters in the wild and obtain persisted elevated privileges[4]. Another way the attacker could seek to further their position is by way of horizontal privilege escalation. This could be done by attacking other users of the Crossplane deployment to run code on the victims machine, steal secrets, pretend to be the victim in other attacks, or something else. A desirable goal for any attack is to escalate privileges all the way to root. As such, an attacker can seek to chain vulnerabilities starting with a low-impact vuln and escalate horizontally with other higher-impact vulns that require existing privileges to exploit.

Denial of service

Attackers may seek to gain a competitive advantage by launching denial-of-service attacks against Crossplane deployments. The motivation here could be financial or to disrupt a competitor's research and development efforts. Denial of service attacks can also damage the users reputation or put the users in a position where they breach contractual agreements with their partners, thus making them vulnerable to legal action.

---

1 https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/malicious-docker-hub-container-images-cryptocurrency-mining

2 https://blog.aquasec.com/container-vulnerability-dzmlt-dynamic-container-analysis

3 https://unit42.paloaltonetworks.com/cryptojacking-docker-images-for-mining-monero/

4 https://blog.aquasec.com/leveraging-kubernetes-rbac-to-backdoor-clusters

Steal secrets

Attackers may seek to steal secrets from Crossplane users to obtain confidential, high-value data or access to other systems that the user manages.

Cluster hardening

Ada Logics assessed Crossplane's hardening against NSA's Kubernetes hardening guide. We used Kubescape[5] to scan Crossplane's YAML manifests against Kubescape's NSA framework setting. Crossplane scores at 100% compliance against NSA's Kubernetes hardening guide, meaning that no resources failed with security issues of any kind.

These are the findings from Kubescape against both Crossplane and Crossplane-runtime.

Controls: 24 (Failed: 0, Passed: 24, Action Required: 0
Failed Resources by Severity: Critical — 0, High — 0, Medium — 0, Low — 0

Severity	Control name	Failed resources	All Resources	% Compliance-score (if failed)
Critical	API server insecure port is enabled	0	0	-1%
Critical	Disable anonymous access to Kubelet service	0	0	-1%
Critical	Enforce Kubelet client TLS authentication	0	0	-1%
High	Resource limits	0	0	-1%
High	Applications credentials in configuration files	0	0	-1%
High	Host PID/IPC privileges	0	0	-1%
High	HostNetwork access	0	0	-1%
High	Insecure capabilities	0	0	-1%
High	Privileged container	0	0	-1%
High	CVE-2021-25742-nginx-ingress-snippet-annotation-vu…	0	0	-1%
Medium	Exec into container	0	0	-1%
Medium	Non-root containers	0	0	-1%
Medium	Allow privilege escalation	0	0	-1%
Medium	Ingress and Egress blocked	0	0	-1%
Medium	Automatic mapping of service	0	0	-1%

[5] https://github.com/kubescape/kubescape

	account			
Medium	Cluster-admin binding	0	0	-1%
Medium	Container hostPort	0	0	-1%
Medium	Cluster internal networking	0	0	-1%
Medium	Linux hardening	0	0	-1%
Medium	CVE-2021-25741 - Using symlink for arbitrary host ...	0	0	-1%
Medium	Secret/ETCD encryption enabled	0	0	-1%
Medium	Audit logs enabled	0	0	-1%
Low	Immutable container filesystem	0	0	-1%
Low	PSP enabled	0	0	-1%
	Resource Summary	0	0	0.00%

We recommend Crossplane to integrate Kubescape into Crossplane's CI pipeline to maintain the same high standard over time.

Fuzzing

Crossplane has a fuzzing suite covering both the Crossplane main repository as well as the Crossplane-runtime repository. The fuzzers run continuously on OSS-Fuzz, which runs the fuzzers around 143 billion times every month.

OSS-Fuzz fuzzing stats from 30th April to 30th May 2023:

Fuzzer name	# of times executed (millions)	Fuzzing hours
FuzzDag	29,548	1,642.6
FuzzFindXpkgInDir	8,715	1,188.2
FuzzForCompositeResourceClaim	18,778	1,143.8
FuzzForCompositeResourceXcrd	24,347	1,349.7
FuzzNewCompositionRevision	4,532	1,147.4
FuzzPackageRevision	16,263	1,288.5
FuzzParse	1,492	1,500.3
FuzzPatchApply	500	1,017.3
FuzzPropagateConnection	17,782	1,079.9
FuzzTransform	21,525	1,354
Total	143,482	12,711.7

An important element of fuzzing is continuity: The fuzzers need to keep running to keep testing for bugs and build up their corpora. Crossplane's OSS-Fuzz build has had no downtime since the completion of its fuzzing audit[6]. Crossplane maintains the fuzzers in the Crossplane and Crossplane-runtime repositories which allows the maintainers to quickly fix any breakages should they arise.

In addition to the fuzzers running continuously by way of OSS-Fuzz, Crossplane also runs them in the CI pipeline through OSS-Fuzz's CIFuzz[7] on pull requests. As such, the fuzzers test for low-hanging, easy-to-find issues on pull requests before they are merged into the project. An advantage of using OSS-Fuzz's CIFuzz is that it starts the fuzz job with the full corpus that OSS-Fuzz has accumulated, thereby allowing the fuzzers to reach far into their targets at every run.

[6] https://blog.crossplane.io/fuzzing-security-audit/
[7] https://google.github.io/oss-fuzz/getting-started/continuous-integration/

ADA Logics added 4 new fuzzers to Crossplane's fuzzing suite and improved the `FuzzDag` fuzzer by increasing its coverage.

New fuzzers

FuzzPTFComposer
Adds a fuzzer that calls several APIs in the same order that the composite controller does. Tests if any of the APIs can be disrupted in a production-like scenario.
PR: https://github.com/crossplane/crossplane/pull/4198

FuzzRevisionControllerPackageHandling
Tests the Revision controllers handling of Crossplane packages. The fuzzerscreates a Revision and resolves its dependencies.
PR: https://github.com/crossplane/crossplane/pull/4199

FuzzRenderClusterRoles
Tests the RBAC managers rendering of cluster roles.
PR: https://github.com/crossplane/crossplane/pull/4201

FuzzRenderRoles
Tests the RBAC managers rendering of cluster roles.
PR: https://github.com/crossplane/crossplane/pull/4202

Improved fuzzers
Ada Logics added more calls to the DAG fuzzer which were not covered prior to the security audit. This improvement resulted in increased coverage.
PR: https://github.com/crossplane/crossplane/pull/4200

Issues found

Here we present the issues that we identified during the audit.

#	ID	Title	Severity	Fixed
1	ADA-XP-23-1	Denial of service from 3rd-party vulnerability	Moderate	Yes
2	ADA-XP-23-2	Possible OOM in internal/xpkg	Low	Yes
3	ADA-XP-23-3	Possible OOM when reading Xfn command output	Low	Yes
4	ADA-XP-23-4	Possible OOM when reading Xfn command output	Low	Yes
5	ADA-XP-23-5	Dockerfiles do not follow best practices regarding COPY vs ADD	Low	Yes
6	ADA-XP-23-6	Insufficient documentation on security implications of using tags for remote images	Low	Yes
7	ADA-XP-23-7	Crossplane's OCI store does not validate for invariants in images	Moderate	No
8	ADA-XP-23-8	gPRC connection not closed	Low	Yes
9	ADA-XP-23-9	Possible endless data attack in Crossplane's Image backend	Moderate	Yes
10	ADA-XP-23-10	Denial of service from malicious Crossplane package	Moderate	Yes
11	ADA-XP-23-11	Possible image tampering from missing image validation	High	Yes
12	ADA-XP-23-12	Possible endless data attack in ProviderRevision controller	Moderate	Yes
13	ADA-XP-23-13	Possible endless data attack in Crossplane's OCI store I	Moderate	Yes
14	ADA-XP-23-14	Possible endless data attack in Crossplane's OCI store II	Moderate	Yes
15	ADA-XP-23-15	Possible endless data attack in Crossplane's OCI store III	Moderate	Yes
16	ADA-XP-23-16	Denial of service from large image	Low	Yes

Issues found in alpha features

Several of the issues were found in alpha components of Crossplane. These issues have been scored by the same metrics as issues found in non-alpha components. The issues found in alpha components are:

#	ID	Title	Severity	Fixed
3	ADA-XP-23-3	Possible OOM when reading Xfn command output	Low	Yes
4	ADA-XP-23-4	Possible OOM when reading Xfn command output	Low	Yes
7	ADA-XP-23-7	Crossplane's OCI store does not validate for invariants in images	Moderate	No
8	ADA-XP-23-8	gPRC connection not closed	Low	Yes
13	ADA-XP-23-13	Possible endless data attack in Crossplane's OCI store I	Moderate	Yes
14	ADA-XP-23-14	Possible endless data attack in Crossplane's OCI store II	Moderate	Yes
15	ADA-XP-23-15	Possible endless data attack in Crossplane's OCI store III	Moderate	Yes

CVEs

The following issues were assigned CVEs:

#	ID	Title	CVE	Severity
11	ADA-XP-23-11	Possible image tampering from missing image validation	CVE-2023-38495	High
16	ADA-XP-23-16	Denial of service from large image	CVE-2023-37900	Low

ADA-XP-23-1: Denial of service from 3rd-party vulnerability

ID	ADA-XP-23-1
Component	Crossplane OCI store
Severity	Moderate
Fixed: Yes	

If an attacker can cause Crossplane to fetch an image with a maliciously-crafted manifest, the attacker can cause a DoS condition for Crossplane from an unrecoverable OOM panic. This will crash the Crossplane controller and will cause a temporary DoS of the node affecting other services.

The root cause of the issue was in a 3rd-party dependency. The issue has been fixed. Ada Logics has requested issuance of a CVE which has not been confirmed.

The attacker will need to compromise the registry from which Crossplane fetches an image or by tricking the Crossplane user into consuming an image by way of dependency confusion or typosquatting attack vectors. To launch a targeted attack, the threat actor will need to know which images their victim consumes in their Crossplane deployment. Alternatively, they can launch a non-targeted attack by compromising images of popular and widely-used images.

Once the attacker has compromised the registry, the attack vector is fairly simple, and crafting the malicious manifest is trivial.

ADA-XP-23-2: Possible OOM in internal/xpkg

ID	ADA-XP-23-2
Component	Crossplane Packages
Severity	Low
Fixed in: https://github.com/crossplane/crossplane/pull/4232	

`github.com/crossplane/crossplane/blob/master/internal/xpkg.Build()` is susceptible to an OOM condition on the highlighted line below if `buf` on the highlighted line contains a large buffer.

https://github.com/crossplane/crossplane/blob/498b2fcbcc5323676d48048944e8a2a6d67cd87a/internal/xpkg/build.go#L82-L117

```
82  func Build(ctx context.Context, b parser.Backend, p parser.Parser, l parser.Linter)
    (v1.Image, error) {
83          // Get YAML stream.
84          r, err := b.Init(ctx)
85          if err != nil {
86                  return nil, errors.Wrap(err, errInitBackend)
87          }
88          defer func() { _ = r.Close() }()
89
90          // Copy stream once to parse and once write to tarball.
91          buf := new(bytes.Buffer)
92          pkg, err := p.Parse(ctx, annotatedTeeReadCloser(r, buf))
93          if err != nil {
94                  return nil, errors.Wrap(err, errParserPackage)
95          }
96          if err := l.Lint(pkg); err != nil {
97                  return nil, errors.Wrap(err, errLintPackage)
98          }
99
100         // Write on-disk package contents to tarball.
101         tarBuf := new(bytes.Buffer)
102         tw := tar.NewWriter(tarBuf)
103
104         hdr := &tar.Header{
105                 Name: StreamFile,
106                 Mode: int64(StreamFileMode),
107                 Size: int64(buf.Len()),
108         }
109         if err := tw.WriteHeader(hdr); err != nil {
110                 return nil, errors.Wrap(err, errTarFromStream)
111         }
112         if _, err = io.Copy(tw, buf); err != nil {
113                 return nil, errors.Wrap(err, errTarFromStream)
114         }
```

```
115         if err := tw.Close(); err != nil {
116                 return nil, errors.Wrap(err, errTarFromStream)
117         }
```

In the case of an excessively large `buf` on line 112, Go will perform a sigkill, but before doing so, the machine running Crossplane will experience a temporary DoS condition that will affect other services as well.

ADA-XP-23-3: Possible OOM when reading Xfn command output

ID	ADA-XP-23-3
Component	Composite Functions
Severity	Low
Fixed in: https://github.com/crossplane/crossplane/pull/4217	

This issue affects an alpha feature in Crossplane.

If an attacker can control the input to an Xfn, they can trigger an OOM DoS issue. The attacker would need to cause the Xfn to create a large Stdout buffer, and the OOM would happen when Crossplane reads it entirely into memory on the highlighted line:

https://github.com/crossplane/crossplane/blob/b01a0f8c0830f038514714baf24deb7ef21bbad4/cmd/xfn/spark/spark.go#L81

```
81    func (c *Command) Run() error {
...           ...
132           p := oci.NewCachingPuller(h, store.NewImage(c.CacheDir),
      &oci.RemoteClient{})
133           img, err := p.Image(ctx, r, FromImagePullConfig(req.GetImagePullConfig()))
134           if err != nil {
135                 return errors.Wrap(err, errPull)
136           }
137
138           // Create an OCI runtime bundle for this container run.
139           b, err := s.Bundle(ctx, img, runID,
      FromRunFunctionConfig(req.GetRunFunctionConfig()))
140           if err != nil {
141                 return errors.Wrap(err, errBundleFn)
142           }
143
144           root := filepath.Join(c.CacheDir, ociRuntimeRoot)
145           if err := os.MkdirAll(root, 0700); err != nil {
146                 _ = b.Cleanup()
147                 return errors.Wrap(err, errMkRuntimeRootdir)
148           }
...           ...
157           cmd := exec.CommandContext(ctx, c.Runtime, "--root="+root, "run",
      "--bundle="+b.Path(), runID)
158           cmd.Stdin = bytes.NewReader(req.GetInput())
159
160           out, err := cmd.Output()
161           if err != nil {
162                 _ = b.Cleanup()
```

```
                return errors.Wrap(err, errRuntime)
        }
        if err := b.Cleanup(); err != nil {
                return errors.Wrap(err, errCleanupBundle)
        }

        rsp := &v1alpha1.RunFunctionResponse{Output: out}
        pb, err = proto.Marshal(rsp)
        if err != nil {
                return errors.Wrap(err, errMarshalResponse)
        }
        _, err = os.Stdout.Write(pb)
        return errors.Wrap(err, errWriteResponse)
}
```

ADA-XP-23-4: Possible OOM when reading Xfn command output

ID	ADA-XP-23-4
Component	Composite Functions
Severity	Low
Fixed in: https://github.com/crossplane/crossplane/pull/4217	

This issue affects an alpha feature in Crossplane.

This issue is similar to ADA-XP-23-3 in root cause, attack vector and impact.

https://github.com/crossplane/crossplane/blob/b01a0f8c0830f038514714baf24deb7ef21bbad4/internal/xfn/container_linux.go#L83

```
83    func (r *ContainerRunner) RunFunction(ctx context.Context, req
      *v1alpha1.RunFunctionRequest) (*v1alpha1.RunFunctionResponse, error) {
...           ...

138           stdio, err := StdioPipes(cmd, r.rootUID, r.rootGID)
139           if err != nil {
140                   return nil, errors.Wrap(err, errCreateStdioPipes)
141           }
142
143           b, err := proto.Marshal(req)
144           if err != nil {
145                   return nil, errors.Wrap(err, errMarshalRequest)
146           }
147           if err := cmd.Start(); err != nil {
148                   return nil, errors.Wrap(err, errStartSpark)
149           }
150           if _, err := stdio.Stdin.Write(b); err != nil {
151                   return nil, errors.Wrap(err, errWriteRequest)
152           }
153
...
157           if err := stdio.Stdin.Close(); err != nil {
158                   return nil, errors.Wrap(err, errCloseStdin)
159           }
160
...
163           stdout, err := io.ReadAll(stdio.Stdout)
164           if err != nil {
165                   return nil, errors.Wrap(err, errReadStdout)
166           }
167
168           stderr, err := io.ReadAll(stdio.Stderr)
```

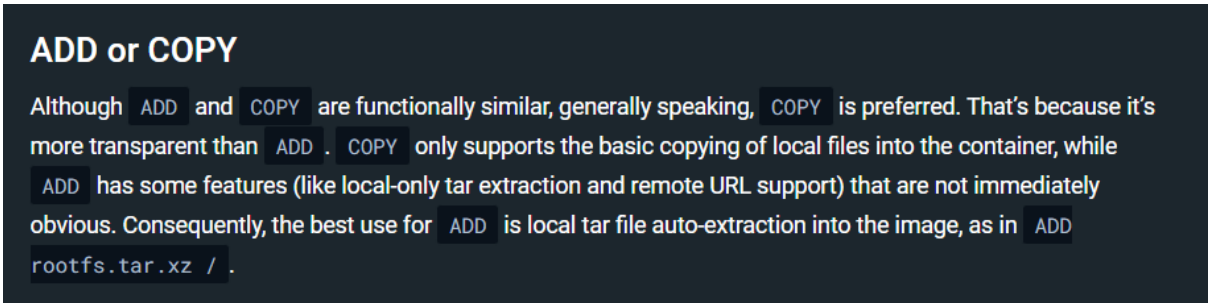
```
169         if err != nil {
170                 return nil, errors.Wrap(err, errReadStderr)
171         }
172
...         ...
180 }
```

ADA-XP-23-5: Dockerfiles do not follow best practices regarding COPY vs ADD

ID	ADA-XP-23-5
Component	Crossplane Cluster Images
Severity	Low
Fixed in: https://github.com/crossplane/crossplane/pull/4173	

Dockers best practices documentation recommends using `COPY` instead of `ADD`:

https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#add-or-copy



ADD or COPY

Although `ADD` and `COPY` are functionally similar, generally speaking, `COPY` is preferred. That's because it's more transparent than `ADD`. `COPY` only supports the basic copying of local files into the container, while `ADD` has some features (like local-only tar extraction and remote URL support) that are not immediately obvious. Consequently, the best use for `ADD` is local tar file auto-extraction into the image, as in `ADD rootfs.tar.xz /`.

Crossplane does not follow this best practices the following places:

https://github.com/crossplane/crossplane/blob/192a09266e10f1873020201043b5782cdef8ce8f/cluster/images/xfn/Dockerfile

```
14  ADD bin/${TARGETOS}\_${TARGETARCH}/xfn /usr/local/bin/
```

https://github.com/crossplane/crossplane/blob/19be2ce66814569829d13e9ecc3ba2cfc6b245bb/cluster/images/crossplane/Dockerfile

```
6   ADD bin/$TARGETOS\_$TARGETARCH/crossplane /usr/local/bin/
7   ADD crds /crds
```

ADA-XP-23-6: Insufficient documentation on security implications of using tags for remote images

ID	ADA-XP-23-6
Component	Crossplane Documentation
Severity	Low
Fixed in: https://github.com/crossplane/docs/pull/465	

The Crossplane documentation recommends referencing images with digests rather than tags. However, the documentation does not mention the security concerns of referencing by tags.

When referencing an image by tag, users may receive a different image than they expect, and it is harder to validate the received image. This makes it easy for an attacker to tamper with an image and avoid detection in case of a successful attack against the Crossplane user.

Crossplane users consuming packages from a private registry are also affected by this since attackers can compromise the registry and change the underlying image that a tag refers to.

We recommend making it explicit in the documentation that referencing images by tag has considerate security downsides.

ADA-XP-23-7: Crossplane's OCI store does not validate for invariants in images

ID	ADA-XP-23-7
Component	Crossplane's OCI store
Severity	Moderate
Fixed: No	

This issue affects an alpha feature in Crossplane. It has not been fixed by the completion of the security audit, but Crossplane is working on a fix here: https://github.com/crossplane/crossplane/pull/4214

Crossplane's OCI store does not validate layers, config and manifests of images. Crossplane uses github.com/google/go-containerregistry for the underlying image interface. go-containerregistry has a validation API for images which is available here: https://github.com/google/go-containerregistry/blob/ca48523123223c5ea478501e6cb1a91a64a8a049/pkg/v1/validate/image.go#L30.

This API validates the layers, configs and manifests of an image and returns an error if any step of the validation fails.

Crossplane uses this validation API when writing a supplied image to the OCI store: https://github.com/crossplane/crossplane/blob/498b2fcbcc5323676d48048944e8a2a6d67cd87a/internal/oci/store/store.go#L158

```
145  func (i *Image) Image(h ociv1.Hash) (ociv1.Image, error) {
146          uncompressed := image{root: i.root, h: h}
147
148          ...
149          // return an error.
150          oi, err := partial.UncompressedToImage(uncompressed)
151          if err != nil {
152                  return nil, errors.Wrap(err, errPartial)
154          }
154
...          ...
158          return oi, errors.Wrap(validate.Image(oi, validate.Fast), errInvalidImage)
     }
159
```

This API is called in before writing the image to the local store here:

https://github.com/crossplane/crossplane/blob/498b2fcbcc5323676d48048944e8a2a6d67cd87a/internal/oci/store/store.go#L168

```
162 func (i *Image) WriteImage(img ociv1.Image) error {
163         d, err := img.Digest()
164         if err != nil {
165                 return errors.Wrap(err, errGetDigest)
166         }
167
168         if _, err = i.Image(d); err == nil {
169                 // Image already exists in the store.
170                 return nil
171         }
...              ...
216 }
```

This usage of the go-containerregistrys validation API will not catch validation errors, since Crossplane does not catch errors. As such, if any layers in the image have been tampered with, Crossplane will not validate and catch this to prevent the user from consuming the image.

ADA-XP-23-8: gPRC connection not closed

ID	ADA-XP-23-8
Component	Crossplane's OCI store
Severity	Low
Fixed in: https://github.com/crossplane/crossplane/pull/4174	

This issue impacts an alpha-feature in Crossplane.

Crossplane keeps the gRPC connection of composite functions open in case any of the passed options fail:

https://github.com/crossplane/crossplane/blob/001d0577249ebb4fcb0d315bf7eb452d5dc5b1ad/internal/controller/apiextensions/composite/composition_ptf.go#L793-L820

```
793  func RunFunction(ctx context.Context, fnio *iov1alpha1.FunctionIO, fn
     *v1.ContainerFunction, o ...ContainerFunctionRunnerOption)
     (*iov1alpha1.FunctionIO, error) {
794          in, err := yaml.Marshal(fnio)
795          if err != nil {
796                  return nil, errors.Wrap(err, errMarshalFnIO)
797          }
789
799          target := DefaultTarget
800          if fn.Runner != nil && fn.Runner.Endpoint != nil {
801                  target = *fn.Runner.Endpoint
802          }
803
804          conn, err := grpc.DialContext(ctx, target,
     grpc.WithTransportCredentials(insecure.NewCredentials()))
805          if err != nil {
806                  return nil, errors.Wrap(err, errDialRunner)
807          }
808
809          req := &fnv1alpha1.RunFunctionRequest{
810                  Image:             fn.Image,
811                  Input:             in,
812                  ImagePullConfig:   ImagePullConfig(fn),
813                  RunFunctionConfig: RunFunctionConfig(fn),
814          }
815
816          for _, opt := range o {
817                  if err := opt(ctx, fn, req); err != nil {
818                          return nil, errors.Wrap(err, errApplyRunFunctionOption)
819                  }
820          }
821
822          rsp, err :=
```

```
    fnv1alpha1.NewContainerizedFunctionRunnerServiceClient(conn).RunFunction(ctx, req)
823         if err != nil {
824                 // TODO(negz): Parse any gRPC status codes.
825                 _ = conn.Close()
826                 return nil, errors.Wrap(err, errRunFnContainer)
827         }
828
829         if err := conn.Close(); err != nil {
830                 return nil, errors.Wrap(err, errCloseRunner)
831         }
832
...         ...
835         out := &iov1alpha1.FunctionIO{}
836         return out, errors.Wrap(yaml.Unmarshal(rsp.Output, out), errUnmarshalFnIO)
837 }
```

ADA-XP-23-9: Possible endless data attack in Crossplane's Image backend

ID	ADA-XP-23-9
Component	Crossplane's OCI store
Severity	Moderate
Fixed in: https://github.com/crossplane/crossplane/pull/4348	

Crossplane's image backend is susceptible to an endless data attack from an unrestricted loop through manifest layers in Crossplane packages. To launch this attack, the attacker needs to craft an image containing a manifest with a high number of layers. Then, they need to either compromise the registry or trick the user into consuming the image by way of dependency confusion or typosquatting attack vectors. The impact will be that the revision controller will be stuck in an infinite loop and be prevented from finishing the task and doing other subsequent tasks.

The issue has its root cause in the image backend:

https://github.com/crossplane/crossplane/blob/f32e27f375b0218ccaf49e072c5661d71f368131/internal/controller/pkg/revision/imageback.go#LL94C1-L110C4

```
94          // Fetch image from registry.
95          img, err := i.fetcher.Fetch(ctx, ref,
    v1.RefNames(n.pr.GetPackagePullSecrets())...)
96          if err != nil {
97                  return nil, errors.Wrap(err, errFetchPackage)
98          }
99          // Get image manifest.
100         manifest, err := img.Manifest()
101         if err != nil {
102                 return nil, errors.Wrap(err, errGetManifest)
103         }
104         // Determine if the image is using annotated layers.
105         var tarc io.ReadCloser
106         foundAnnotated := false
107         for _, l := range manifest.Layers {
108                 if a, ok := l.Annotations[layerAnnotation]; !ok || a !=
    baseAnnotationValue {
109                         continue
110                 }
```

An attacker can add a high number of layers to the manifest which can cause Crossplane to go into an infinite loop.

ADA-XP-23-10: Denial of service from malicious Crossplane package

ID	ADA-XP-23-10
Component	Revision image backend
Severity	Moderate
Fixed in: https://github.com/google/go-containerregistry/pull/1742	

If Crossplane's image backend does not find an annotated file, it will call go-containerregistry's `Extract()` API passing the fetched image:

https://github.com/crossplane/crossplane/blob/f32e27f375b0218ccaf49e072c5661d71f368131/internal/controller/pkg/revision/imageback.go#L131-L133

```
131         if !foundAnnotated {
132                 tarc = mutate.Extract(img)
133         }
```

When go-containerregistry extracts the image, it decompresses each layer and ultimately copies the contents from a tar Reader to a tar Writer:

https://github.com/google/go-containerregistry/blob/03b86570e60f261bf4a79b903d73e03cb862910b/pkg/v1/mutate/mutate.go#L331-L340

```
332                     if !tombstone {
332                             if err := tarWriter.WriteHeader(header); err != nil {
333                                     return err
334                             }
335                             if header.Size > 0 {
336                                     if _, err := io.CopyN(tarWriter, tarReader, header.Size); err != nil {
337                                             return err
338                                     }
339                             }
340                     }
```

When copying the file, there is no upper limit to the file size which allows an attacker to compress a large layer that will crash go-containerregistry and thereby Crossplane with an out of memory panic.

ADA-XP-23-11: Possible image tampering from missing image validation

ID	ADA-XP-23-11
Component	Revision image backend
Severity	High
Fixed in: https://github.com/crossplane/crossplane/pull/4370	

Crossplane's image backend is susceptible to an image tampering attack due to insufficient validation of Crossplane packages. An attacker can launch an attack against Crossplane by compromising the registry from which Crossplane fetches the image. The attacker could then replace the image that the user is referencing and trick Crossplane users into fetching a tampered image.

Crossplane users can reference images by tags or digests. Digests are the more secure way to reference, however, in this case, even if the user referenced an image by digest, the user had insufficient guarantee that the image they fetched was the correct one.

Crossplane fixed the issue by validating both the image itself as well as each layer in the fetched image manifest.

CVE-2023-38495 has been assigned this finding.

ADA-XP-23-12: Possible endless data attack in ProviderRevision controller

ID	ADA-XP-23-12
Component	ProviderRevision controller
Severity	Moderate
Fixed in: https://github.com/crossplane/crossplane/pull/4347	

A Crossplane user that can send a request to the ProviderRevision controller can launch an endless data attack against the controller if the request has a high number of policy rules.

The reconciler validates incoming requests by way of `ValidatePermissionRequests()`, https://github.com/crossplane/crossplane/blob/5abc44508f88d892535fc8608d5e35ad5dfb5a0a/internal/controller/rbac/provider/roles/reconciler.go#L303, which loops through the requests policy rules:

https://github.com/crossplane/crossplane/blob/f32e27f375b0218ccaf49e072c5661d71f368131/internal/controller/rbac/provider/roles/requests.go#L174-L193

```
174  func (v *ClusterRoleBackedValidator) ValidatePermissionRequests(ctx
     context.Context, requests ...rbacv1.PolicyRule) ([]Rule, error) {
175          cr := &rbacv1.ClusterRole{}
176          if err := v.client.Get(ctx, types.NamespacedName{Name: v.name}, cr); err !=
     nil {
177                  return nil, errors.Wrap(err, errGetClusterRole)
178          }
179
180          t := newNode()
181          for _, rule := range Expand(cr.Rules...) {
182                  t.Allow(rule.path())
183          }
184
185          rejected := make([]Rule, 0)
186          for _, rule := range Expand(requests...) {
187                  if !t.Allowed(rule.path()) {
188                          rejected = append(rejected, rule)
189                  }
190          }
191
192          return rejected, nil
193  }
```

If the length of requests is sufficiently large, a threat actor can cause Crossplane into spending excessive time on this loop and can use that to prevent the controller from performing other tasks, such as upgrading Providers. An attacker could leverage that to prevent Crossplane from upgrading to a secure Provider package while exploiting a known vulnerability in the current version of the package.

ADA-XP-23-13: Possible endless data attack in Crossplane's OCI store I

ID	ADA-XP-23-13
Component	Crossplane's OCI store
Severity	Moderate
Fixed in: https://github.com/crossplane/crossplane/pull/4203	

This issue affects an alpha feature in Crossplane.

Crossplane's OCI store is susceptible to an endless data attack if the image that Crossplane fetches has a high number of layers. An attacker can launch the attack against Crossplane by compromising the registry and replacing the original image with a tampered image containing a high number of layers or by tricking a Crossplane user into fetching the image by way of dependency confusion or typosquatting attack vectors.

The root cause of the issue is that the Crossplane loops through the layers of the fetched image without an upper limit. An attacker can add a high number of layers to the manifest which can cause Crossplane to go into an infinite loop.

https://github.com/crossplane/crossplane/blob/b01a0f8c0830f038514714baf24deb7ef21bbad4/internal/oci/store/overlay/store_overlay.go#L166

```
166 func (c *CachingBundler) Bundle(ctx context.Context, i ociv1.Image, id string, o
    ...spec.Option) (store.Bundle, error) {
167         cfg, err := i.ConfigFile()
168         if err != nil {
169                 return nil, errors.Wrap(err, errReadConfigFile)
170         }
171
172         layers, err := i.Layers()
173         if err != nil {
174                 return nil, errors.Wrap(err, errGetLayers)
175         }
176
177         lowerPaths := make([]string, len(layers))
178         for i := range layers {
179                 p, err := c.layer.Resolve(ctx, layers[i], layers[:i]...)
180                 if err != nil {
181                         return nil, errors.Wrap(err, errResolveLayer)
182                 }
183                 lowerPaths[i] = p
184         }
```

ADA-XP-23-14: Possible endless data attack in Crossplane's OCI store II

ID	ADA-XP-23-14
Component	Crossplane's OCI store
Severity	Moderate
Fixed in: https://github.com/crossplane/crossplane/pull/4203	

This issue affects an alpha feature in Crossplane.

This issue is similar to ADA-XP-23-13 in attack vector, impact and root cause but affects the handling of uncompressed layers.

https://github.com/crossplane/crossplane/blob/d0d7527f92869a780a49d25d15fea0ed54f3bf0b/internal/oci/store/uncompressed/store_uncompressed.go#L89

```
89  func (c *Bundler) Bundle(ctx context.Context, i ociv1.Image, id string, o
    ...spec.Option) (store.Bundle, error) {
90          cfg, err := i.ConfigFile()
91          if err != nil {
92                  return nil, errors.Wrap(err, errReadConfigFile)
93          }
94
95          layers, err := i.Layers()
96          if err != nil {
97                  return nil, errors.Wrap(err, errGetLayers)
98          }
99
100         path := filepath.Join(c.root, id)
101         rootfs := filepath.Join(path, store.DirRootFS)
102         if err := os.MkdirAll(rootfs, 0700); err != nil {
103                 return nil, errors.Wrap(err, errMkRootFS)
104         }
105         b := Bundle{path: path}
106
107         for _, l := range layers {
108                 tb, err := l.Uncompressed()
109                 if err != nil {
110                         _ = b.Cleanup()
111                         return nil, errors.Wrap(err, errOpenLayer)
112                 }
113                 if err := c.tarball.Apply(ctx, tb, rootfs); err != nil {
114                         _ = tb.Close()
115                         _ = b.Cleanup()
116                         return nil, errors.Wrap(err, errApplyLayer)
117                 }
```

```
118                 if err := tb.Close(); err != nil {
119                         _ = b.Cleanup()
120                         return nil, errors.Wrap(err, errCloseLayer)
121                 }
122         }
```

An attacker can add a high number of layers to the manifest which can cause Crossplane to go into an infinite loop when Crossplane reads the layers of the image.

ADA-XP-23-15: Possible endless data attack in Crossplane's OCI store III

ID	ADA-XP-23-15
Component	Crossplane's OCI store
Severity	Moderate
Fixed in: https://github.com/crossplane/crossplane/pull/4203	

This issue affects an alpha feature in Crossplane.

This issue is similar to ADA-XP-23-13 in attack vector, impact and root cause but affects the routine that writes the fetched image to the OCI store.

https://github.com/crossplane/crossplane/blob/498b2fcbcc5323676d48048944e8a2a6d67cd87a/internal/oci/store/store.go#L162

```
162  func (i *Image) WriteImage(img ociv1.Image) error {
...          ...
201
202          layers, err := img.Layers()
203          if err != nil {
204                  return errors.Wrap(err, errGetLayers)
205          }
206
207          g := &errgroup.Group{}
208          for _, l := range layers {
209                  l := l // Pin loop var.
210                  g.Go(func() error {
211                          return i.WriteLayer(l)
212                  })
213          }
214
215          return errors.Wrap(g.Wait(), errWriteLayers)
216  }
```

ADA-XP-23-16: Denial of service from large image

ID	ADA-XP-23-16
Component	Crossplane-runtime parser
Severity	Low
Fixed in: https://github.com/crossplane/crossplane/pull/4358	

Crossplane-runtimes parser is vulnerable to a DoS attack. The root cause of the issue is that `k8s.io/apimachinery/pkg/util/yaml.(*YAMLReader).Read()` will exhaust memory if reading from a large reader. The Crossplane-runtime parser parses an image from a remote registry using Kubernetes' yaml parser:

https://github.com/crossplane/crossplane-runtime/blob/511b39fa560d40daecec2f9288850f92cc62e14b/pkg/parser/parser.go#L94

```
94  func (p *PackageParser) Parse(_ context.Context, reader io.ReadCloser) (*Package, error) { //nolint:gocyclo // Only at 11.
95          pkg := NewPackage()
96          if reader == nil {
97                  return pkg, nil
98          }
99          defer func() { _ = reader.Close() }()
100         yr := yaml.NewYAMLReader(bufio.NewReader(reader))
101         dm := json.NewSerializerWithOptions(json.DefaultMetaFactory, p.metaScheme, p.metaScheme, json.SerializerOptions{Yaml: true})
102         do := json.NewSerializerWithOptions(json.DefaultMetaFactory, p.objScheme, p.objScheme, json.SerializerOptions{Yaml: true})
103         for {
104                 bytes, err := yr.Read()
105                 if err != nil && !errors.Is(err, io.EOF) {
```

When parsing a large reader, Go will react by performing a sigkill and as a result will also crash any application using the parser. The machine will experience a temporary DoS before Go performs the sigkill.

This is an issue for Crossplane's Revision controller which parses images fetched from remote registries: https://github.com/crossplane/crossplane/blob/90b27fed1c877f4e4d2e62dbba6e26505b38271b/internal/controller/pkg/revision/reconciler.go#L478.

An untrusted image could crash the Revision controller and cause a denial of service for other users of the cluster.

The following minimized PoC illustrates the issue:

```
package main

import (
        "bufio"
        "fmt"
        "io"
        "bytes"
        "k8s.io/apimachinery/pkg/util/yaml"
)

func main() {
        b := bytes.Repeat([]byte("a"), 9000000000)
        r1 := bytes.NewReader(b)
        r2 := bytes.NewReader(b)
        r3 := bytes.NewReader(b)
        r4 := bytes.NewReader(b)
        r := io.MultiReader(r1, r2, r3, r4)

        bRead := bufio.NewReader(r)
        yr := yaml.NewYAMLReader(bRead)
        fmt.Println("Reading...")
        _, _ = yr.Read()
        fmt.Println("The end")
}
```

This program will not print “The end”. The console will print:

```
Reading...
signal: killed
```

CVE-2023-37900 has been assigned this issue.
GHSA: https://github.com/crossplane/crossplane/security/advisories/GHSA-68p4-95xf-7gx8

SLSA

SLSA is a framework for assessing the supply chain security posture of projects. The current version of SLSA is v1.0 which specifies a series of requirements related to the build platform for software releases as well as the provenance attestation. SLSA evaluates a project based on four security levels. Level 0 has no requirements as we do not include that in the table below.

The SLSA framework is useful to protect against a series of real-world supply chain attack vectors, for example, during the SolarWinds attack, attackers compromised the build platform of a software vendor - SolarWinds - and injected malicious code that the vendor then distributed to its customers. The provenance statement helps users defend against typosquatting attacks or attacks as well as consuming packages from mirrors instead of the main and intended packages. These are known attack vectors; recently researchers found 1,652 malicious packages disguised as legitimate packages[8]. SLSA compliance is therefore an important factor of Crossplane's overall security posture and should be seen as an ongoing effort to achieve and maintain a solid integration with SLSAs specification. This will help Crossplane defend against a series of well-known - by users and malicious actors - supply chain attack vectors.

Our overall assessment is that Crossplane performs well against requirements for the build platform but is lacking the provenance statement. The provenance is a large and important part of the SLSA framework. Crossplane is currently lacking a compliant provenance statement which brings compliance to a low level. Crossplane is performing well in other areas such as the build platform for release artifacts. We recommend adding the provenance generation via SLSA's official Github Actions workflows: https://github.com/slsa-framework/slsa-github-generator. The SLSA community is currently working on a framework, Bring Your Own Builder (BYOB), which includes level 3 compliance out of the box.

Our assessment for each criteria:

Requirement	L1	L2	L3
Provenance generation			
Provenance Exists	⛔	⛔	⛔
Provenance is Authentic		⛔	⛔

[8] https://sysdig.com/blog/analysis-of-supply-chain-attacks-through-public-docker-images/

Provenance is Unforgeable			⛔
Isolation			
Hosted		✓	✓
Isolated			✓