



# Helm v3

November 4, 2020

Prepared For:

Matt Farina | *Helm*

[matt@mattfarina.com](mailto:matt@mattfarina.com)

Matt Butcher | *Helm*

[technosophos@gmail.com](mailto:technosophos@gmail.com)

Prepared By:

Stefan Edwards | *Trail of Bits*

[stefan.edwards@trailofbits.com](mailto:stefan.edwards@trailofbits.com)

Brian Glas | *Trail of Bits*

[brian.glas@trailofbits.com](mailto:brian.glas@trailofbits.com)

# Introduction

From July 20, 2020 to July 31, 2020 Trail of Bits (“the assessment team”) undertook a threat model of the Helm to help Cloud Native Computing Foundation (“CNCF,” “the client” or “the implementation team”) understand wider design concerns within the system. The assessment included ten identified components across three trust zones, and resulted in a total of nine findings, ranging in severity from Medium to Low.

The client indicated that using the same Control Families as previous CNCF audits was preferred, and that no further security policy frameworks would be needed.

Helm is a simple system with few connections and components. The discovery phase of the assessment identified 10 components across multiple cloud providers, trust zones, and security levels. The following components were reviewed by the assessment team:

- Helm Client
- Helm Library
- Helm Cache
- Helm Plugins
- Repositories YAML file
- Helm Chart Repository & OCI
- Index Files
- Helm Charts
- Provenance Files
- Helm Configs

The assessment team believes that no additional components, connections, and risks lurk in the Helm system, as it is well documented and understood by the implementation team. We believe that this threat model accurately reflects the risks identified in the system as discussed with the implementation team. The remainder of this report is split into three sections:

1. This introduction, including [Key Findings](#) and the [Report Position](#)
2. Descriptions of components and their connections, and an analysis of them
3. Specific security findings by area

## Key Findings

Helm allows developers to package and reuse software for deployment on Kubernetes clusters. In this role, the correct and secure description, composition, and application of user's packages is critical. However, we noted few weaknesses in the design of the Helm system, which was well designed and documented. We have grouped into two key areas: **cryptographic** and **audit** controls.

**Cryptographic controls:** The Helm system consumes data from several different sources, including remote hosts. This data is coalesced into Kubernetes descriptions, and applied to clusters. However, the system does not apply cryptographic controls to ensure that data always transits secure channels or that it has not been tampered with from the original source to the end consumer. By enforcing channel security, via TLS, and applying integrity controls, either via provenance files or another mechanism such as The Update Framework (TUF), Helm can ensure that files have not been tampered with or exposed at any point in the consumption lifecycle.

**Audit controls:** Helm does log the provenance, hash, or other information consumed whilst building Kubernetes descriptions. A developer may have to manually reconstruct the locations and hashes of various components, without additional information from Helm itself. Adding logging, that shows where files were downloaded from, what version, and hash, will help developers and other defenders in debugging issues and responding to incidents, should they occur.

## Report Position

Helm is a small, well-designed system, with few security controls and design decisions that arose from organic decisions that made sense during product development. This report attempts to catalog many of the discussions captured within the threat modeling meeting processes.

The remainder of this report analyzes components, trust zones, data flows, threat actors, controls, and findings of the Helm threat model. This was a point-in-time assessment, and reflects the state of Helm at the time of the assessment, rather than any current or future state.

<b>Introduction</b>	<b>2</b>
Key Findings	3
Report Position	3
<b>Methodology</b>	<b>5</b>
<b>Components</b>	<b>6</b>
<b>Trust Zones</b>	<b>7</b>
Trust Zone Connections	8
<b>Threat Actors</b>	<b>9</b>
Threat Actor Paths	10
<b>Security Control Analysis</b>	<b>11</b>
<b>Dataflow Diagrams</b>	<b>14</b>
<b>Findings Summary</b>	<b>15</b>
TM01. Index files do not support signing	17
TM02. Repository connections do not enforce TLS	18
TM03. Open Container Initiative integration does not support provenance	19
TM04. Repository credentials are stored in plain text	20
TM05. Repository files do not check for widely-scoped permissions	21
TM06. Missing audit logs	22
TM07. lookup may be used to exfiltrate sensitive data	23
TM08. Index files may be stale	24
TM09. Missing system-wide accept & deny lists	25
TM10. Values from Kubernetes are assumed to be correct	26
<b>Appendix A: Timeline of Meetings</b>	<b>27</b>
Helm v3 23-JUL-2020	27
Helm v3 24-JUL-2020	28
Helm v3 03-AUG-2020	30

## Methodology

This document is the result of two person-weeks of effort from both the assessment and implementation teams. It is a control-focused threat model, with each discovered component reviewed against the control frameworks mentioned by the implementation team and refined in discussion with the team.

Performing a threat model and architecture review on a system as complex as Helm is fraught with challenges. First, we held a series of meetings between the assessment and implementation teams to discover as many components as possible. Next, we attempted to uncover the processes and controls that were designed in place to document the current system state and potential future recommendations. Lastly, we discussed a number of threat scenarios, with the actors specified below in [Threat Actors](#).

## Components

The following components were in scope for the threat modeling discussions:

Component Name	Description
Helm Client	The Golang binary that users directly interact with on the command line
Helm Library	The Golang library that implements most of Helm's features, including retrieval, parsing, and validation of components
Helm Cache	The local storage, on an end-user's machine, of cached interstitial Helm objects, such as charts
Helm Plugins	Various helpers that extend Helm's functionality, typically written in Golang or shell scripts
Repositories YAML file	The local configuration of an end-user's Helm repositories, including credentials, URLs, and the like
Helm Chart Repository & OCI Registries	The actual storage location of Helm Charts, on a remote service, that includes additional files such as Indexes, Charts, and Provenance files
Index Files	The file that states what Charts the repository holds, as well as additional metadata such as version information
Helm Charts	The actual packages' definitions within Helm, that describes the package, additional dependencies, and metadata such as version information
Provenance Files	Files which add signing and other information for Helm Charts (but not Repositories or Index files) to ensure that Charts have not been tampered with, and is based off of GnuPG
Helm Configs	Additional pieces of configuration data, or the result of composing Helm Charts, stored in Kubernetes secrets store

## Trust Zones

Components are situated within trust zones: logical boundaries made up of components that have similar or related sensitivity. We identified the following trust zones:

Trust Zone Name	Description	Included Components
Repository	The remote service or services which host Helm Chart repositories and store various files, such as Charts, Indexes, and so on	Helm Chart Repository & OCI Registry, Index File, Helm Charts, Provenance File
Developer's Machine	A developer's machine, which is applying and composing a Helm Chart	Helm Client, Helm Library, Helm Cache, Helm Plugins, Repositories YAML File
Cluster	The developer's target cluster	Helm Configs

## Trust Zone Connections

Trust zones become useful when data that flows between zones is understood.

Originating Zone	Destination Zone	Data Description	Connection Type	Authentication Type
Repository	Developer's Machine	Helm Charts, Indexes, Provenance, and other files which describe packages	Various	Opportunistically authenticated, based on connection
Developer's Machine	Repository	The creation, update, or deletion of Helm Charts, Indexes, Provenance, and other files which describe packages	Various	Opportunistically authenticated, based on connection
Developer's Machine	Cluster	Reconstituted Kubernetes definitions, built from Charts	HTTPS	Per-cluster Authentication, usually password- or token-based
Cluster	Developer's Machine	Kubernetes Secrets	HTTPS	Per-cluster Authentication, usually password- or token-based



## Threat Actors

Similar to establishing trust zones, defining malicious actors ahead of time is useful in determining which protections, if any, are necessary to mitigate or remediate a vulnerability. We use these actors in all subsequent findings from the threat model. Additionally, we define other “users” of the system who may be impacted by or enticed to undertake an attack.

For example, in a confused deputy attack such as [Cross-Site Request Forgery](#), a normal user is both the victim and the potential direct attacker, even though a secondary attacker entices the user to undertake the action.

Actor	Description
Malicious Internal User	A user, such as an administrator or developer, who uses their privileged position or stolen credentials maliciously against the system.
Internal Attacker	An attacker who transits one or more trust boundaries, i.e., an attacker with container access.
External Attacker	An attacker who is external to the system and is unauthenticated.
Administrator	An actual administrator of the system, tasked with operating and maintaining the cluster as a whole.
Developer	An application developer who deploys an application to a cluster, either directly or via another user (such as an administrator).
End User	An external user of an application hosted by a cluster.

## Threat Actor Paths

Defining attackers' paths through the various zones is useful when analyzing potential controls, remediations, and mitigations within the current architecture:

Actor	Originating Zone	Destination Zone	Description
Malicious Internal User	Developer's Machine	Repository	Malicious Internal Users will seek to modify Charts, delete Provenance files, and otherwise tamper with the code that a Developer or other User will run on the cluster, thereby potentially giving the Malicious Internal User access to the Cluster
	Developer's Machine	Cluster	Similarly, a Malicious Internal User may seek to modify Repository's YAML file, tamper with Provenance or Charts, in order to silently modify Packages that are destined for the Cluster
Internal Attacker	Developer's Machine	Cluster	An Internal Attacker, with access either to the Developer's Machine or the Repository, will seek to modify Charts or other assets, in order to gain access to the Cluster
	Repository	Cluster	
External Attacker	Repository	Cluster	An External Attacker may know or guess an external dependency that internal packages utilize, in order to inject malicious code or packages into the Kubernetes definitions that are sent to the server

## Security Control Analysis

The [Committee on National Security Systems \(CNSSI\) 4009](#) defines “security controls” as the management, operational, and technical controls (i.e., safeguards or countermeasures) prescribed for an information system to protect the confidentiality, integrity, and availability of the system and its information. Controls are grouped by a type or *family*, which collects controls along logical groupings such as authentication or cryptography. Our assessment focused on the following control families primarily from NIST 800-53r4, supplemented with a few additional categories:

Code	Control Family	Description
AC	Access Controls	Related to authorization of users and assessment of rights
AU	Auditing and Logging	Related to auditing of actions or logging of problems
AN	Authentication	Related to the identification of users
CM	Configuration	Related to security configurations of servers, devices or software
CR	Cryptography	Related to protecting the privacy or integrity of data
DE	Data Exposure	Related to unintended exposure of sensitive information
DV	Data Validation	Related to improper reliance on the structure or values of data
DS	Denial of Service	Related to causing system failure
ER	Error Reporting	Related to the reporting of error conditions in a secure fashion
PA	Patching	Related to keeping software up to date
SM	Session Management	Related to the identification of authenticated users
TI	Timing	Related to race conditions, locking or order of operations

UB	Undefined Behavior	Related to undefined behavior triggered by the program
----	--------------------	--

Our review assessed the controls along the following criteria:

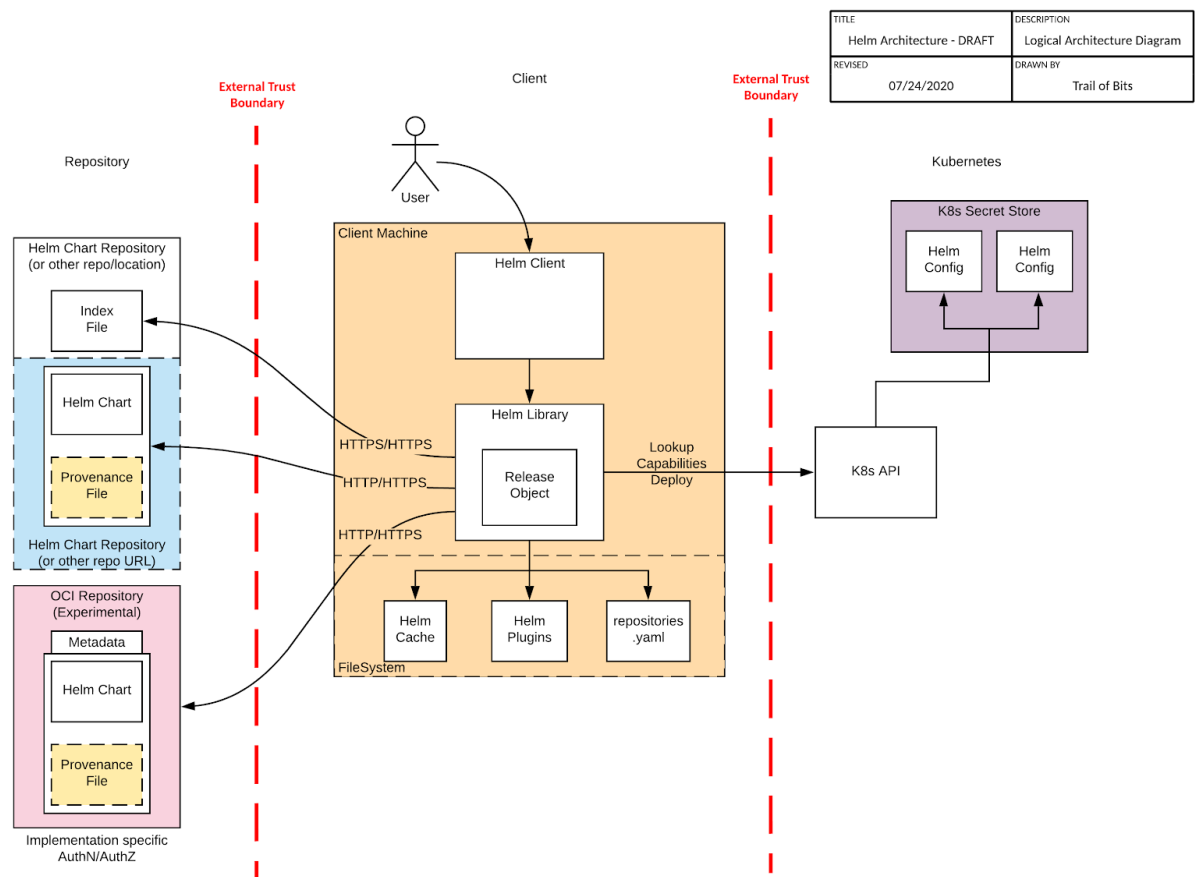
Strength	Description
<b>Strong</b>	Controls were well implemented, centrally located, not bypassable, and robustly designed
<b>Good</b>	Controls were well implemented, but may be weakened by vulnerabilities or are diffuse in location
<b>Acceptable</b>	Controls were implemented to the baseline industry standards and guidelines, but could be strengthened
<b>Weak</b>	Controls were either partly unimplemented, applied, or contained flaws in their design or location
<b>Missing</b>	An entire family of control was missing from a component
<b>Not Applicable</b>	This control family is not needed for protecting the component

Resulting in the following control analysis table:

Control Family	Strength	Description
Access Control	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Access Control
Auditing and Logging	<b>Missing</b>	Helm does not have Auditing or Logging to show progenitor information for either security or debugging purposes
Authentication	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Authentication
Configuration	<b>Good</b>	Helm has a strong default configuration, and uses operating system protections where possible, and offloads as much as possible to standard cluster components, such as secrets stores, for all other configuration

Cryptography	<b>Weak</b>	Helm generally does not need cryptography, as the vast majority of its cryptographic needs may be offloaded to other controls, such as TLS for channel-level security. However, there are a number of locations wherein Helm could use cryptographic controls, such as keyed-hash message authentication codes (HMACs) to protect content integrity more rigorously
Data Exposure	<b>Good</b>	Helm generally follows best practices for storing data, and limits potential Data Exposure issues by relying on inherited controls, like Role-Based Access Control (RBAC) from Kubernetes itself. Minor issues with potential Data Exposure were noted in the assessment
Data Validation	<b>Good</b>	Helm generally follows best practices for storing data, and limits potential Validation issues by relying on inherited controls from Kubernetes itself. Minor issues with potential Data Validation were noted in the assessment
Denial of Service	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Denial of Service
Error Reporting	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Error Reporting
Patching	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Patching
Session Management	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Session Management
Timing	<b>Not Applicable</b>	Helm itself is not an arbiter for access to any privileged resources, especially with the removal of Tiller in Helm v3. As such, it relies on external systems, such as the Kubernetes cluster and repositories, to enforce Timing
Undefined Behavior	<b>Not Applicable</b>	Helm does not define a language, but rather simply uses externally-defined languages such as Golang's template language and YAML. As such, these are inherited controls that Helm has little control over

# Dataflow Diagrams



## Findings Summary

Our discussion with the development team identified ten issues, ranging in severity from Medium to Informational. Further investigation should be made to review other potential missing or weak security controls.

All findings were rated by two metrics:

- Severity, meaning “how bad” the finding was in an uncategorized fashion.
- Difficulty, meaning “how hard” is the finding to remediate by the organization.

Once the implementation team has accepted these findings, they should appropriately recategorize the findings with their chosen risk assessment and management framework.

#	Title	Type	Severity
1	Index files do not support signing	Cryptography	Medium
2	Repository connections do not enforce TLS	Cryptography	Medium
3	Open Container Initiative integration does not support provenance	Cryptography	Medium
4	Repository credentials are stored in plain text	Data Exposure	Low
5	Repositories do not check for widely-scoped permissions	Data Validation	Low
6	Missing audit logs	Auditing & Logging	Low
7	Lookup may be used to exfiltrate sensitive data	Data Exposure	Low
8	Index files may be stale	Timing	Informational

9	Missing system-wide accept & deny lists	Data Validation	Informational
10	Values from Kubernetes are assumed to be correct	Data Validation	Informational



## TM01. Index files do not support signing

Severity: Medium

Type: Cryptography

Component(s): Index File

Difficulty: High

Finding ID: TOB-HELM-TM01

### Description

Helm uses Index Files to store information about a repository on a remote server. The Index file contains information about packaged Charts, and allows remote users to know which packages and versions are hosted by the Repository. However, Index Files themselves cannot be signed, meaning an attacker with Local position could modify the contents of the Index File to provide malicious or modified versions of a package, or to remove additional protections such as Provenance files.

### Justification

The severity is Medium for the following reasons:

- An attacker could modify packages or other files with impunity
- This would allow an attacker to install additional software or components within the cluster
- High-security clusters will likely use compensating controls such as additional authentication or file system restrictions, lowering the overall Severity

The difficulty is High for the following reasons:

- Developers and Defenders must know, distribute, and use signatures
- Additional infrastructure must be developed for the signing of files beyond Charts

### Recommendation

Short term, provide users on guidance to secure and audit their Repository services, including secondary controls that may be used to secure Index Files and other Helm assets.

Long term, include a signature scheme within the Helm ecosystem that does not rely upon third-party tools such as GnuPG. This should include a simple way for Developers and Administrators to generate, distribute, and verify keys and signatures; and ensure that users who wish to have additional security surrounding package security may do so in a seamless fashion.

## TM02. Repository connections do not enforce TLS

Severity: Medium

Type: Cryptography

Component(s): Repository

Difficulty: Low

Finding ID: TOB-HELM-TM02

### Description

Helm Repositories are a central location for Charts: they allow Developers to store and retrieve packages from a central location and reuse by other developers. Part of Repositories' flexibility is that Helm is generally neutral to the underlying transfer mechanism, as "getters," in the form of plugins, handle the actual communications between the Helm Library and Repository. However, by default Helm does not attempt to enforce TLS when it is available, meaning an attacker with Network or Adjacent position could potentially man-in-the-middle (MiTM) the connection.

### Justification

The severity is Medium for the following reasons:

- An attacker with Network or Adjacent positioning could modify packages or other files with impunity
- This would allow an attacker to install additional software or components within the cluster
- High-security clusters will likely use compensating controls such as additional authentication or communication restrictions, lowering overall Severity for those clusters

The difficulty is Low for the following reasons:

- Go supports TLS out of the box, and can also enforce certificate pinning

### Recommendation

Short term, warn users that Helm will not default to TLS connections unless specified and alert them that Charts and other data is sent in the clear via this connection.

Longer term, upgrade all connections to TLS whenever possible, and require users to use an additional flag to downgrade to clear-text connections. This will ensure that users do not accidentally expose data wherever possible, and will require extra steps to bypass secure channels. Furthermore, when TLS is enforced, allow users to either use certificate pinning or trust on first use (TOFU). This will ensure that, even if an attacker had sufficient position and a valid certificate, that the Helm getter will not interact with servers that have been compromised or otherwise had their certificate changed.

## TM03. Open Container Initiative integration does not support provenance

Severity: Medium  
Type: Cryptography  
Component(s): OCI

Difficulty: High  
Finding ID: TOB-HELM-TM03

### Description

Helm is experimenting with integrating Open Container Initiative (OCI) registries, which provide a richer interface for storing and updating Charts. However, Helm's integration of OCI registries does not support provenance files, meaning that an attacker with sufficient position could modify Charts and related files undetected.

### Justification

The severity is Medium for the following reasons:

- An attacker with sufficient position could modify packages or other files with impunity
- This would allow an attacker to install additional software or components within the cluster
- High-security clusters will likely use compensating controls such as additional authentication or file system restrictions, lowering overall Severity for those clusters

The difficulty is High for the following reasons:

- Developers and Defenders must know, distribute, and use signatures
- Additional infrastructure must be developed for the signing of files beyond Charts

### Recommendation

Support the full signature lifecycle for all operations, regardless of where it is consumed from. This should include Provenance or future solutions such as moving towards Notary or TUF.

Additionally, if possible, Provenance should be extended to be the same across registries and easily verifiable.

## TM04. Repository credentials are stored in plain text

Severity: Low

Difficulty: Medium

Type: Data Exposure

Finding ID: TOB-HELM-TM04

Component(s): Repository YAML

### Description

Helm uses a local Repository YAML file to store repository metadata. This data includes the type of repository, connection details, and credentials. However, this credential data is stored in plain text, and could be retrieved by an attacker with Local position.

### Justification

The severity is Low for the following reasons:

- The Repository YAML file is stored in a secure location on the Developer's Machine
- An attacker must gain sufficient position to access the credentials
- An attacker with this level of access would likely target other areas first, beyond credentials

The difficulty is Medium for the following reasons:

- A single system must be agreed upon in order to support all installations.
- This should require minimal setup and either use operating system resources or be robust enough to work on multiple operating systems.

### Recommendation

Consider storing credentials in a different format. This could include using a library that stores credentials in Keychain, or uses an alternative storage mechanism, such as YAML and [age](#). In either case, removing plain-text credentials will reduce the risk that credentials are exposed to unintended third parties, and ensure that only the intended parties have access to repository credentials.

## TM05. Repository files do not check for widely-scoped permissions

Severity: Low

Difficulty: Low

Type: Data Exposure

Finding ID: TOB-HELM-TM05

Component(s): Repository Files

### Description

Helm uses a local Repository YAML file to store repository metadata. This data includes the type of repository, connection details, and credentials. However, this file is not checked for permissions prior to use. An attacker with sufficient access could have modified permissions or otherwise tampered with the file system attributes of the file prior to Helm's operations.

### Justification

The severity is Low for the following reasons:

- The Repository YAML file is stored in a secure location on the Developer's Machine
- An attacker must gain sufficient position to access the credentials
- An attacker with this level of access would likely target other areas first, beyond repositories

The difficulty is Medium for the following reasons:

- A single system must be agreed upon in order to support all installations.
- This should require minimal setup and either use operating system resources or be robust enough to work on multiple operating systems.

### Recommendation

Check that the Repository YAML file matches some known-good set of permissions and ownership prior to operation, and warn the user if there is a mismatch. For example, the file should be owned and readable by only the current user.

## TMo6. Missing audit logs

Severity: Low

Type: Auditing and Logging

Component(s): Helm Library

Difficulty: Low

Finding ID: TOB-HELM-06

### Description

Helm allows Developers to compose packages into Kubernetes definitions that are sent to a cluster. In order to do this, Helm will fetch Charts from known repositories, and retrieve dependencies defined within those Charts in a similar fashion. However, Helm does not log data, such as where a Dependency or Chart is retrieved from, what was the hash of the Chart, and so on, in a location that can be used to audit where and when a Chart was used. An attacker could use this to install a Chart with a duplicate name on a repository they have access to, in order to trick Helm into installing dependencies or Charts the User did not intend.

### Justification

The severity is Low for the following reasons:

- Users can manually inspect the result prior to application to cluster
- An attacker must have a secondary exploit in order to have sufficient position in order to impact this risk
- In and of itself, this risk is not a vulnerability, but rather increases the likelihood that an attack would go unnoticed

The difficulty is Low for the following reasons:

- Audit logs for Chart composition can be stored locally, and include simple information such as where as chart originated and what the hash of the Chart was
- This can be stored in YAML, which is already consumed by Helm, and easily accessible to most users

### Recommendation

Store the details about a Chart's final product in some easily-consumable format for users' own auditing. This should include the origin location, version, hash, and other details about the Chart's processed during composition, and provide a User with the ability to audit the contents and information about a Chart, prior to applying the result to a cluster.

## TM07. lookup may be used to exfiltrate sensitive data

Severity: Low

Type: Data Exposure

Component(s): Helm Library

Difficulty: High

Finding ID: TOB-HELM-TM07

### Description

Helm includes a template function, `lookup`, that allows users to query for the state of a Kubernetes cluster. This information can then be used to write a Kubernetes definition, such as a `podspec`, which includes sensitive information in a request to an external source. For example, an attacker could append sensitive information from the cluster in a URL used to retrieve an Image within Kubelet.

### Justification

The severity is Low for the following reasons:

- An attacker could use this to exfiltrate Kubernetes secrets
- An Internal Attacker with this level of access could already exfiltrate
- However, an External Attacker could use this technique to gain access to additional information

The difficulty is High for the following reasons:

- The attack is bipartite, and requires Defender's have visibility in two places, Helm and Kubernetes
- Different accept/deny lists may be in use at the Helm and Kubernetes levels

### Recommendation

Warn users that the `lookup` may be problematic, and warn them of its use. This could be paired with [TOB-HELM-TM06: Missing audit logs](#), or another mechanism. In either case, the user should be warned that the `lookup` function was called, and allow the user to see where it was used. There may be some potential for determining the context of a secret's use within the resulting Kubernetes definition; for example, warning users when a secret is used in a location that may leak like an Image fetch in a `podspec`. Lastly, consider allowing users to disable `lookup` under certain circumstances, such as transitive dependencies.

## TM08. Index files may be stale

Severity: Informational

Type: Timing

Component(s): Helm Library, Index File

Difficulty: High

Finding ID: TOB-HELM-TM08

### Description

When adding a Repository, Helm caches a copy of the Repository's Index File. This allows Helm to locally check which repositories need to be accessed for which Charts, without accessing a remote resource. However, Helm does not refresh Index Files until a dependency build takes place, meaning that Index Files may be out of date.

### Recommendation

Consider adding a time to live to all Index Files. This will ensure that no operation is undertaken on a file that is out of date, and that users may be warned when a repository should be accessed for a fresh copy of the Index File. Additionally, this will ensure that users are aware of the latest version of a package.



## TM09. Missing system-wide accept & deny lists

Severity: Informational

Type: Data Validation

Component(s): Throughout the system

Difficulty: High

Finding ID: TOB-HELM-TM09

### Description

Helm allows users to add Repositories as needed. This means that users can import packages from a wide range of sources and include a large amount of software. However, Administrators may wish to restrict users to a single repository, or define a known-good location that all Charts should be retrieved from. This would match similar processes supported by other systems, such as Maven Nexus.

### Recommendation

Consider adding support for accept and deny lists as a configuration option, potentially stored within Kubernetes itself. This will allow Helm to match similar filtering done within Kubernetes, such as providing Kubelet with a private image registry. In this way, Administrators can define a central Helm Repository with known-good Charts, and provide a centralized mechanism for utilizing only this Repository.

## TM10. Values from Kubernetes are assumed to be correct

Severity: Informational  
Type: Data Validation  
Component(s): Helm Library

Difficulty: High  
Finding ID: TOB-HELM-TM10

### Description

Helm has the ability to query Clusters for the state of various components. This can include things such as the values of secrets, or any other data that is accessible to the User issuing the query. Helm then can operate on this data, enriching Charts via value propagation or simply displaying the data to the user. However, data from Kubernetes is assumed to be valid, and is not further validated by the Helm Library. Whilst this is generally a safe assumption, it may not always be true, and could potentially lead to Data Validation issues, such as split interpretation of data.


### Recommendation

Always validate all data as strictly as possible whenever possible. This should include validating the composition, shape, and length of the data, as well as validating that things such as references can and should be operated upon. Additionally, assume that data from external sources, including Kubernetes itself, have potentially been tampered with or are otherwise incorrect. In this way Helm can catch Data Validation issues before the manifest in larger problems for the Cluster as a whole.

## Appendix A: Timeline of Meetings

Trail of Bits conducted eight threat modeling meetings via video conference with the implementation team to help gain an understanding of the processes and architecture that are integral to the Helm application. Most of the meetings were 60-90 minutes in duration and multiple members of the implementation team and Trail of Bits were in attendance.


### Helm v3 23-JUL-2020

- arch is good
- index.yaml
- add repo:
  - gets index.yaml
  - caches
  - if requested
  - charts can live anywhere
  - chart + revision
    - foo, revisions: 0.1, 0.2
    - typically relative, can be absolute path
  - index file is basically the repo
  - provenance must live w/ chart
  - **!** I don't check if it is out of date, save for dependency build is happening
- **!** F index isn't signed, can't be signed
- doesn't index by SHA or the like
- not idempotent for builds
  - Issue No. 3612 in Helm Repo
- index file compromise can change everything
- Provenance file uses PGP
  - few people use provenance
  - not checked by default
  - not using signing
  - ex: Bitnami doesn't sign anything
  - missing practical guidance on CI/CD
-  Add new repo
  - helm repo add
  - need access to the system
  - list of repos in yaml file with auth creds

- helm env w/ repo list
- XDG, Library on macOS
- only stores username/password if provided (as in basic auth)
- mTLS supported
- [see text notes](misc notes.txt)
- **!** F add mode check for repo (as in, check that it's 0600 like SSH does for authorized\_keys)
- 🕸 ask: to have storage of chart info in k8s & etcd, s pods can be told where to get things, like maven repos
- all secrets are stored w/ namespace pattern ala sh.helm.v1.secret\_name
- use labels & annotations for security
- stored w/in namespace
- release record
  - gzip tarball: chart rendered via template
  - what user supplied + chart supplied (like passwords)
- 🕸 Add RBAC white paper
- 🕸 check out we give arch diagram back
- uses secrets properly:
  - objects in charts (pod manifest)
  - secrets
  - checks against k8s schema & CRD schema, errors out if that mismatches
- **!** F HTTPS only for chart repos
  - not enforcing HTTPS/TLS for charts & repos
  - S3 "getter", FTP, &c
  - getter fetches

## Helm v3 24-JUL-2020

- OCI: ACR, Google Container Reg, Docker...
  - allows storage of many object types
  - docker distribution is the basis
  - Josh Gilipsky
  - simpler and supports layering (request same URL with different MIME types to get different layers of data)
  - open container initiative
  - distribution spec
  - store charts & provenance file

- layers for object storage of the same
  - mime/type + hash
  - no searching b/c as per spec
- notary: images don't have portable signatures between OCI registries
  - uses provenance to help elide this
- will use OCI as the future repo for code hub, blob store
- very similar to helm repos
- new moment behind OCI support
  - 1 year time frame
  - more political than technical
- supports conditional hash check like Docker
- **!** F doesn't support provenance yet
  - doesn't support Notary or TUF
  - provenance is just PGP
  - wait for Notary V2
  - 🐞 Helm v4 would support OCI & deprecate Provenance/Helm repos
  - not public, can propose
- k8s API interaction
  -  in V3: lookup resources you have access to (ConfigMaps...)
  - secrets
  - capabilities (do I have access to apps-v2?)
- helm template: does not use lookup
  - capabilities uses k8s compliance
- helm test: smoke tests
- discovery is checked w/ kube config
- helm FS:
  - index, plugins are cached
- **!** F No audit log for where a chart/image comes from
  - largely rely on k8s logging upstream
  - **!** R up the verbosity, like hash pulled from & where
  - only pull from certain places
- **!** F no system-wide deny list for repos
  - guard rails
  - "only pull from these helm repos..."
  - maven nexus...

Helm v3 31-JUL-2020

- Arch diagram

- risk model assumptions
  - protect helm repository
  - audit charts
- template command attacks
- helm binary with minikube
- **!** F checks on records out of k8s are missing
  - storage
  - secrets
  - release records
  - known issue

## Helm v3 03-AUG-2020

1. capabilities {object, CRDs, resources load balancers}
  2. template: execute incoming user code in a sandbox
  3. lookup: lookup secret
- wait... lookup has access to secrets...
  - can we lookup then exfil data?
    - **!** F yes
    - lookup -> write bodspec with image in name/hash -> image fetch with secret data
  - templates have access to secrets
  - restricted to where kubelet can pull from
    - **!** R make sure kubelet is restricted for image repos
  - 🕸 no cert pinning for chart repos