Uncharted territories:

The untold tale of Helm Chart security

powered by snyk

Table of contents

> “For the cloud native ecosystem to reach its full potential, we need to collectively improve security. And that means applying a set of patterns, techniques, and tools that focus specifically on cloud native technologies. The Helm Project is deeply interested in the security of our core software, our charts, and the practices of the community. And we eagerly welcome a new breed of tools that help us identify and fix vulnerabilities proactively.

- ***Matt Butcher***

Helm Org Maintainer

Introduction

Helm is undoubtedly one of the most popular ways of installing software on Kubernetes today. It's widely used for deploying first party applications and has a vibrant ecosystem of shared content. Helm Charts make getting started with Kubernetes easier. If you're using a popular piece of software like PostgreSQL or Redis or GitLab, you can probably just run ` helm install` instead of starting from scratch by identifying images and writing lots of configuration. Not only does this save time, but you also benefit from the expertise of the people packaging the software and making it easily configurable.

But like any repository of third-party content, vulnerabilities in popular Helm Charts can pose a risk to lots of users at once. Helping developers use third party content securely is what we do at Snyk. We already provide developer tools to help secure open source dependencies for popular package managers (like Java, .NET, Python, Node,js, Ruby and more), as well as providing tools to detect vulnerabilities in container images.

With this report we wanted to take a look at the state of vulnerabilities in Helm Charts. The intention isn't to call out Helm as being insecure any more than any popular third party content repository is insecure. Rather, our intent is to start a conversation about better ways of securing Helm Charts across the public charts repository so even more people can benefit from Helm's ease of use. Likewise the Helm project is similarly focused on analyzing and discussing the general security posture of Helm with their recent security audit sponsored by the Cloud Native Computing Foundation.

Alongside this report we are also releasing a Helm plugin for Snyk so you can test your own Helm Charts for vulnerabilities.
We also look forward to working with the Helm community in the future by raising the visibility of vulnerabilities and ultimately on helping to fix them.

- Gareth Rushgrove
Director of Product, Snyk

TL;DR

Helm Chart

- 277 stable Helm Charts
- 68% of stable Helm Charts contain an image with a high severity vulnerability

Images

- 416 images used across stable Helm Charts
- 6 images account for nearly half of all vulnerable paths, the other 410 images account for the other half
- 15% of stable charts utilize the Bats image (dduportal/bats:0.4.0) which is the image with the most vulnerable paths. This makes the image a potential vector for attacking the ecosystem. Bats is a popular testing tool, so coming up with an exploit to compromise valuable data might be difficult.

Vulnerabilities

- The most common types of vulnerabilities were out-of-bounds reads or writes, access restriction bypass, and NULL pointer dereference.
- 40,047 vulnerabilities found when each vulnerability is counted only once per image in which it appears

Remediation

- 176 stable Helm Charts (64%) can benefit from an image upgrade
- There are 261 image upgrades that can be made across the stable Helm Charts to improve security.

Glossary

Configuration

Configuration files set specific parameters and initial settings of your application. The Kubernetes API is a powerful abstraction for building cloud native systems. But an unintended consequence of the rich API has been developers authoring large amounts of configuration, mainly in YAML. These config files, if they aren't carefully written, can introduce security risk.

Helm

The package manager for Kubernetes. Helm helps you manage Kubernetes applications. Helm is maintained by the Cloud Native Computing Foundation (CNCF) in partnership with Microsoft, Google, Bitnami, and the Helm contributor community.

Helm Chart

Charts are Helm packages and consist of a collection of files that describe a related set of Kubernetes resources. Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Image

A container image is an executable package of software that includes everything needed to run an application. Charts can incorporate a container image. The vulnerabilities discussed in this report are present in the images.

Incubation

A category of Helm Chart for charts that are under development, but do not yet meet the criteria of a stable chart. They can be shared, collaborated upon, but have a different means of installation.

Kubernetes

Kubernetes is an open source project that is widely used to automate deployment, scaling, and management of containerized applications.

Glossary

Stable

A category of Helm Chart for charts that are developmentally mature. To be considered stable, a chart must meet the requirements laid out in Helm Charts' contribution guidelines. The requirements include things like providing a secure default configuration and only including images free of majority security issues.

Vulnerability

A vulnerability, for the purpose of this report, describes a known exploitable issue present in a container image.

Vulnerability types

These are general categories used to classify vulnerabilities and describe similarities between vulnerabilities in the same category. They roughly correspond to CWEs, Common Weakness Enumeration, a community-developed list of common software security weaknesses

Vulnerability Path

A specific vulnerability may be incorporated into a chart multiple times. This is because operating system dependencies can be nested and a single dependency can be introduced multiple times. We account for the multiple vulnerability instances through a concept we call "vulnerable paths". One path is counted for every way a specific vulnerability is introduced into the project. One path to the vulnerability might be trivial to fix, while another is much more difficult. Completing the trivial fix helps secure your system, but while the other vulnerable paths are present, the vulnerability has not been eradicated.

How was Helm Chart data gathered?

For a report of this nature, the first question many readers will have involves how the data was gathered. All stable charts present in Helm Charts' GitHub repository as of the week of October 21, 2019, were considered. They were installed and tested against using a tool Snyk developed for this purpose, which can be found here. The results of this test were collated and loaded into a database to be queried and inspected for patterns and relevant insights.

Helm Charts

Helm is a popular package manager for Kubernetes. It streamlines the installation and management of Kubernetes applications. Charts are Helm packages and consist of a collection of files that describe a related set of Kubernetes resources. Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.

Helm is currently an incubating project with the Cloud Native Computing Foundation (CNCF). It is the widely adopted package manager for Kubernetes. The following statistics will give us an idea of the size and impact on the Helm community.

- The Helm Chart repository currently has 2,600 contributors and 10,700 stars on GitHub.
- The Helm website had 1,156,252 hits in the month of October 2019.
- Helm was downloaded more than 80,000 times in October 2019.

You can find and browse Helm Charts within the GitHub repository found at www.github.com/helm/charts. The charts found there, curated by Helm maintainers, are separated into two groups, stable and incubator.

Stable charts meet a set of requirements outlined in the repository's contributing guidelines. These requirements include things like following Kubernetes best practices and providing a secure default configuration. Incubator projects have not yet met one or more of the requirements.

This report will focus on the available stable Helm Charts, their associated container images, and the security vulnerabilities found in the container images.

Helm Chart landscape

Statistics are current as of October 24, 2019.

- 277 stable Helm Charts, 233 (84%) of which have an associated container image
- 188 or 68% of stable Helm Charts contain an image with a high severity vulnerability
- 33,852 operating system package dependencies across all the image instances
- 40,047 vulnerabilities found
- The average chart contains two images
- All of the charts contain a total of 416 images
- An average of 81 operating system package dependencies per image
- Current images have between 0 and 550 operating system package dependencies per image
- This corresponds to between 0 and 940 vulnerabilities per image

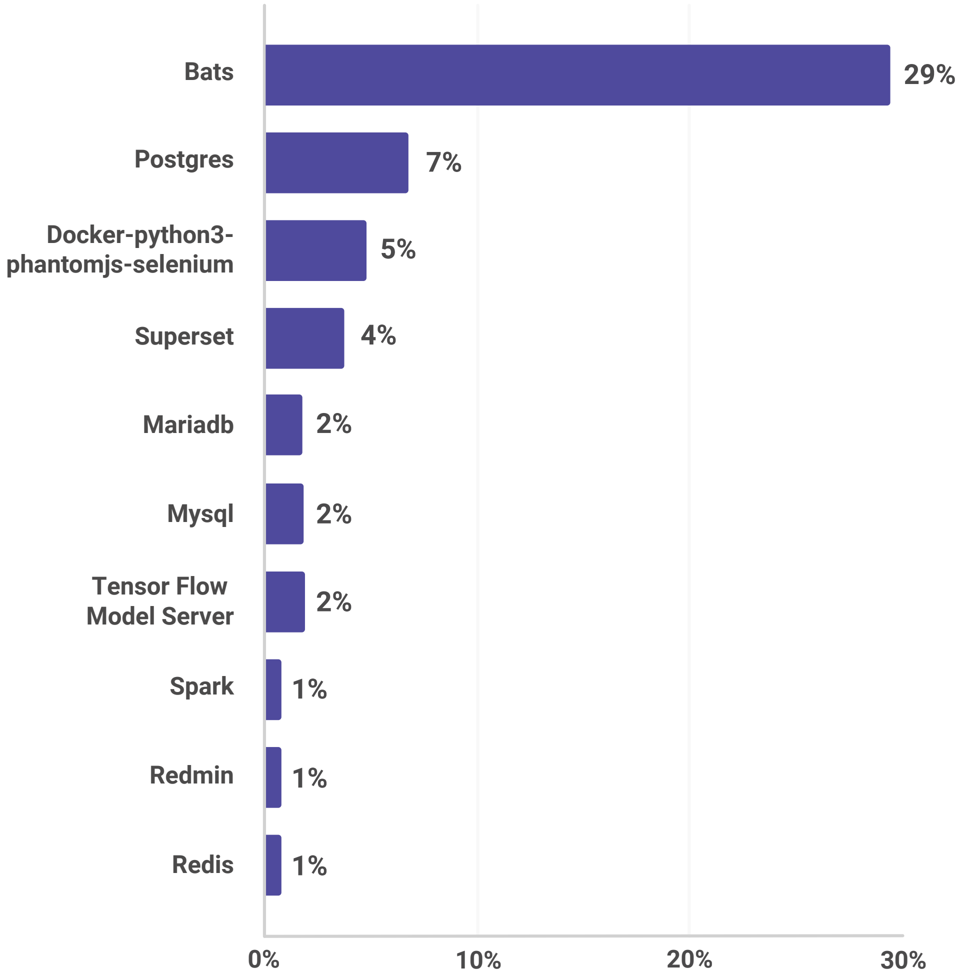
Images

As described by Docker, a container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries, and settings.

Images do not change (if a change is made, you now have a new image). This immutability makes them predictable and portable. One or more images may be included as part of a Helm Chart. These images are what Snyk uses to analyze the health of a Helm Chart.

The following images account for the largest share of the vulnerabilities found in the stable Helm Charts.

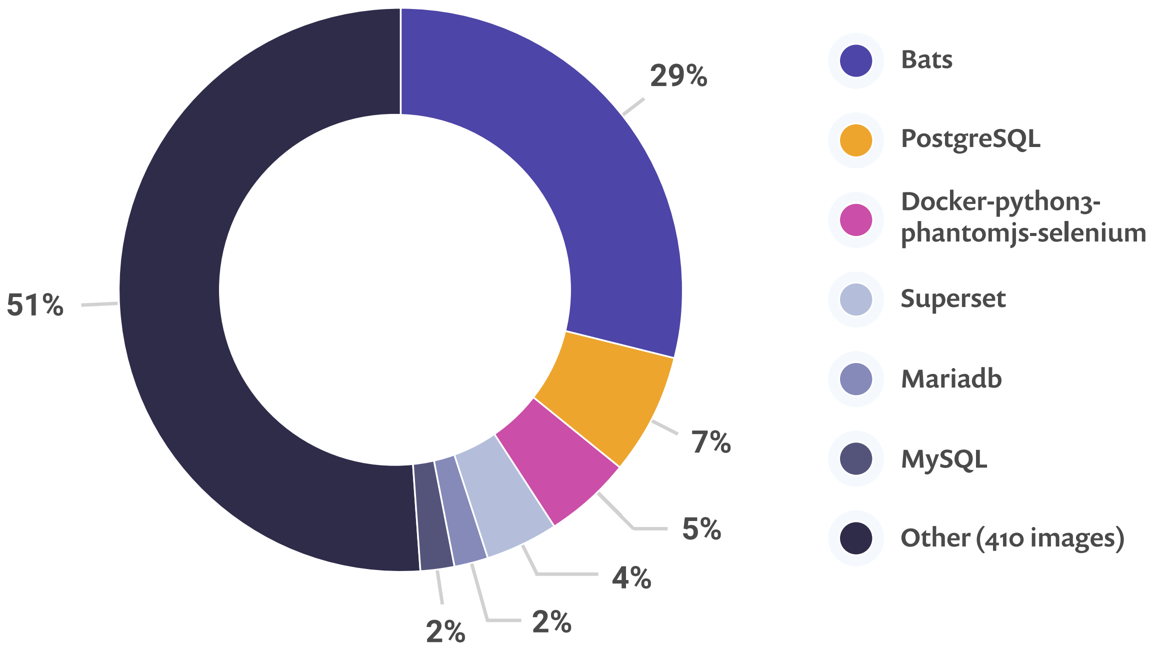
Container images percentage share of known vulnerabilities



This graph is interesting for the following reasons.

1. There are a total of 416 images found in the stable charts. With such a high number of images, one might expect that even the image responsible for the most vulnerabilities would still have a minimal overall share. What we actually see is that dduportal/bats:0.4.0 accounts for 29% of the vulnerable paths.
2. In addition to dduportal/bats:0.4.0 accounting for the plurality of vulnerable paths, a small handful of images can account for the majority. In fact, the six images shown in the chart below account for roughly the same number of vulnerable paths as the remaining 410 images found in the stable charts.

Top 6 vulnerable images



These images are not necessarily less secure than the others that we find contributing vulnerable paths in the stable charts. Instead these images are the ones that carry the heaviest vulnerability load across the stable charts. Their share of vulnerabilities can be accounted for both by their wide adoption and because the vulnerabilities found in these images often have a number of vulnerable paths.

Let’s take a closer look at the top three images.

dduportal/bats:0.4.0

Bats is a testing framework for Bash. It is reassuring that the image responsible for the plurality of vulnerable paths within the stable charts is a testing framework. On Docker Hub you will find this image described in the following way:

> *The idea is to use Docker's lightweight isolation to have a self-contained image embedding bats, any dependency, and all your tests.*

This suggests that if someone were to compromise a project through a known vulnerability in this image that they might not be able to attack the high value targets for which they are looking. Instead, they are more likely to gain access to something less valuable, like test data.

In total, 41 stable Helm Charts utilize this image. This means that the vulnerabilities in this image impact 15% of stable Helm Charts.

The following table describes the 10 vulnerability types most frequently seen with this image and the average severity score that the given vulnerability type is likely to introduce.

Common vulnerability types in Bats image

Vulnerability type	Vulnerability paths in stable charts	Average severity
Out-of-Bounds	19024	7.2
Access Restriction Bypass	5371	7.1
NULL Pointer Dereference	4182	7.1
Improper Input Validation	3731	7.3
Resource Management Errors	2788	5.2
Information Exposure	2501	5.2
Cryptographic Issues	2255	7.0
Race Condition	2050	5.4
Integer Overflow or Wraparound	1927	8.9
Security Features	779	8.1

postgres:9.6.2

PostgreSQL (often referred to as postgres) is a popular database management system. It is unsurprising that it is commonly used in Helm Charts due to its popularity. The following table describes the top 10 vulnerability types for the PostgreSQL image. Postgres and Bats (discussed previously) are different tools that solve different problems and a developer might not necessarily expect for them to have much in common with respect to the vulnerabilities that they introduce. However, the top 4 vulnerability types match between the two images and the remainder of the top 10 are close, though not matching precisely.

This image is used by 7 Helm Charts, or approximately 2.5% of the stable Helm Charts.

Common vulnerability types in Postgres image

Vulnerability type	Vulnerability paths in stable charts	Average severity
Out-of-Bounds	4718	8.1
Access Restriction Bypass	994	6.9
NULL Pointer Dereference	889	7.1
Improper Input Validation	749	6.9
Race Condition	581	5.6
Resource Management Errors	581	5.4
Cryptographic Issues	406	5.3
Information Exposure	287	5.4
Integer Overflow or Wraparound	273	9.0
Directory Traversal	252	8.1

unguiculus/docker-python3-phantomjs-selenium:v1

The image that contributes the third most vulnerable paths is called Docker-python3-phantomjs-selenium. This image adds phantom js and selenium. The top 10 vulnerability types for this image are listed in the table to the right.

This image is interesting for a few reasons.

Both PhantomJS and Selenium deal with web browser automation. They can both be used for testing — meaning this image can be thought of as similar to Bats. It isn't great that it is introducing vulnerabilities, but the vulnerabilities might be deemed acceptable because they are unlikely to expose high value targets.

Another interesting thing to consider is that PhantomJS has been archived and is no longer under active development as of March 2018. If you are using this image in your Kubernetes project, it probably makes sense to move to a new tool.

Finally, we should consider how many charts are utilizing this image. Currently only a single stable Helm chart (keycloak@4.10.1) uses this image.

Common vulnerability types in docker-python3-phantomjs-selenium image

Vulnerability type	Vulnerability paths in stable charts	Average severity
Out-of-Bounds	2846	7.3
Resource Management Errors	1597	7.0
NULL Pointer Dereference	631	7.7
Resource Exhaustion	491	7.3
Improper Input Validation	455	7.1
Allocation of Resources Without Limits or Throttling	240	7.2
Missing Release of Resource after Effective Lifetime	195	4.9
Integer Overflow or Wraparound	189	8.1
Information Exposure	149	6.2
Access Restriction Bypass	139	6.5

This final point gives us a reality check with respect to these vulnerabilities. These vulnerabilities deeply impact a single chart, but are not widely felt across the stable charts.

Now that we have taken a quick look at the images that account for the most vulnerable paths, it is helpful to consider the vulnerabilities themselves.

Vulnerabilities

This table shows us which vulnerabilities are seen repeatedly in the stable charts. You will find the vulnerability type, the percentage share, a link to the vulnerability in the Snyk database, the CVSS score, and the severity rating.

Six of the top 10 vulnerabilities are medium severity. This is nice to see because many organizations will find that a medium severity vulnerability is a tolerable risk, at least initially. This means that these common vulnerabilities can potentially be a lower priority to fix, freeing resources to fix more pernicious issues.

If you want to know more about these individual vulnerabilities, please be sure to click through the link which will take you to the entry in Snyk's database.

Most commonly occuring vulnerabilities in stable Helm Charts

Vulnerability	Count	CVSS	Severity
Access Restriction Bypass	4039	7.8	☠ high
Resource Management Errors	2628	4.3	⚠ medium
Out-of-bounds Read	2234	4.4	⚠ medium
Out-of-bounds Read	2234	6.3	⚠ medium
NULL Pointer Dereference	2131	7.5	☠ high
NULL Pointer Dereference	2131	5.5	⚠ medium
Out-of-Bounds	1964	4.0	⚠ medium
Access Restriction Bypass	1964	9.8	☠ high
Race Condition	1964	4.7	⚠ medium
Credentials Management	1813	9.8	☠ high

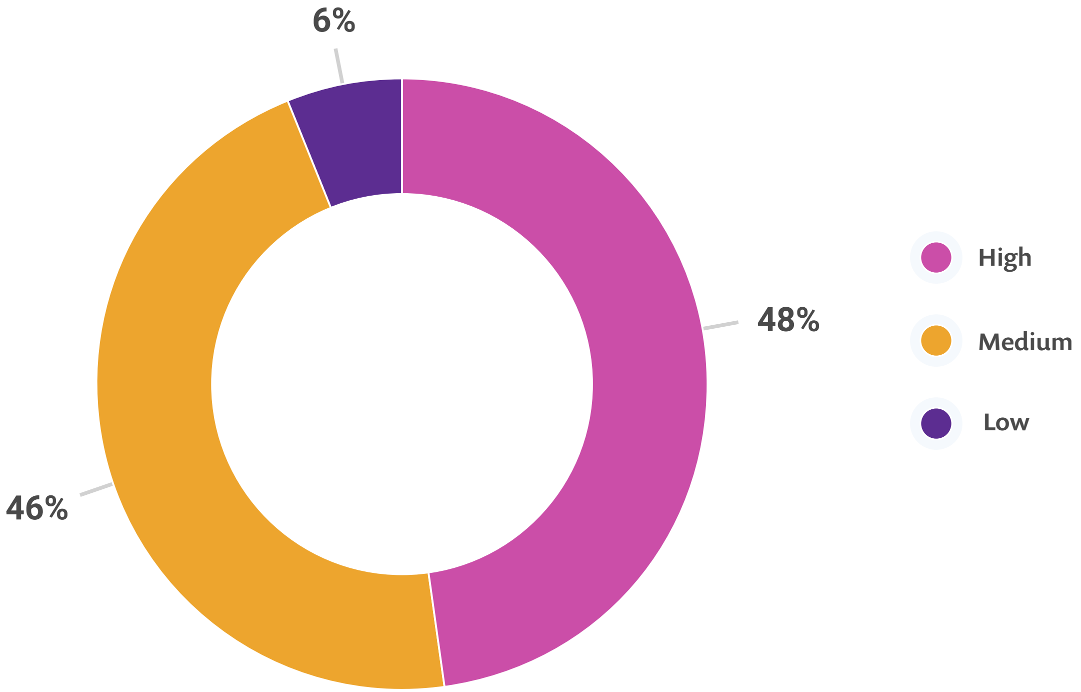
Severities

Now that we have taken a closer look at the more common vulnerabilities, it would be good to understand them from a global scale. The following chart shows the proportion of the vulnerabilities rated as high severity, medium severity, and low severity.

Unfortunately, high severity vulnerabilities are the most common across the Helm Charts. However, more than half the vulnerabilities in the ecosystem are either medium or low severity. Risk tolerance will vary across teams and projects, but in general, if you are using a helm chart you can expect to see a high severity vulnerability. 68% of the 277 stable Helm Charts include a high severity vulnerability.

All of the charts considered in this report are "stable" meaning that they have met specific criteria for inclusion. One criteria that must be met is that the images used "should not have any major security vulnerabilities". This is outlined in the Helm Charts' contribution guidelines.

Vulnerability severity ratings



This criteria may be more aspirational than practical for the following reasons:

1. “Major security vulnerabilities” is not defined. Does a known vulnerability with a high cvss score meet that standard? If not, what would meet the standard?
2. This requirement is outlined in the contribution guidelines. This generally suggests that “major security vulnerabilities” are not acceptable at time of contribution—however the security health of an image is not static. An image free of known vulnerabilities one day may be compromised the next day. But no one from the chart maintainer to a chart user may know that the security status has changed.

It isn’t reasonable to expect an image to contain no known vulnerabilities. However, it is incongruent to represent these charts as stable, with no major security vulnerabilities when 68% of the stable charts use an image with a high severity vulnerability.

A Helm Chart is a powerful tool, but it is in the user’s interest to know what vulnerabilities they are introducing into their project through the use of a given chart. It is likely that the risk is well within the user’s tolerance, but it is better for a user to know about the risks rather than making an assumption about the security of a chart just because it is classified as “stable”.

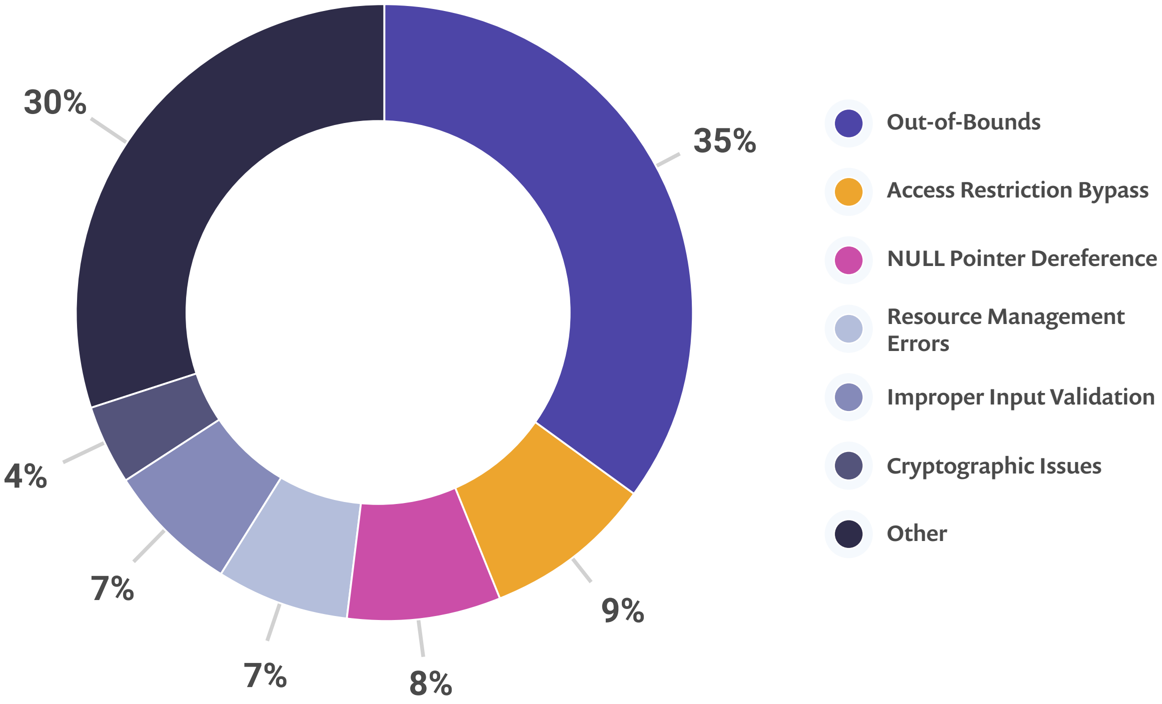
Types

In addition to the severity of the vulnerable paths, it is important to consider the different vulnerability types present in the stable helm charts.

The graph to the right shows the share different types of vulnerabilities account for across the stable Helm Chart.

There were 185,999 vulnerable paths across the stable Helm Charts and dozens of reported vulnerability types. However, three vulnerability types make up almost half of the total vulnerable paths: out-of-bounds vulnerabilities, access restriction bypasses, and NULL pointer dereferences. Each of these are discussed below.

Vulnerability types



Out-of-bounds

Out-bounds vulnerabilities come in two types — read and write. Both vulnerability types involve accessing data outside of the intended buffer. Out-of-bounds read vulnerabilities can only read information that is already there. It is a problem because that is data that the developer may not want to surface. An out-of-bounds write vulnerability can write data outside of the intended buffer, which can produce undefined or unexpected results.

Out-of-bounds read

For an out-of-bounds read vulnerability, concerns either center around the exposure of sensitive data or an improper read crashing the system. This type of problem can be combated by careful handling of input data. This may include a "whitelist" system, only accepting input that can match to the list. Any input that does not match to the whitelist is not displayed. This approach is likely to work better than a "blacklist" because it is easier and more predictable to define acceptable data than it is to define all conceivable types of unacceptable data.

Out-of-bounds write

Concerns associated with an out-of-bounds write include data corruption, a crash, or code execution. The first two concerns are also problems for out-of-bounds read vulnerabilities and are discussed above. The possibility of malicious code execution makes the out-of-bounds write vulnerabilitiy the more severe of the two.

This kind of vulnerability can be managed by checking your buffer size to make sure that you don't have anything unexpected. Additionally, you can make sure that the destination buffer size is equal to the source buffer size, or truncate input strings after a reasonable length before passing to other functions.

Access restriction bypass

Access restriction bypass vulnerabilities are the second most common type of vulnerability found in the images used by stable Helm Charts.

An access restriction bypass can occur a number of different ways. First, the system may not correctly check the identity of a user. Someone other than an account-holder will be able to access their data or their privileges.

Another type of access restriction bypass involves a user being able to perform an action in the system that they should not be able to. This kind of behavior can happen when privileges are either inappropriately assigned or insufficiently checked. This can expose sensitive data or lead to unexpected behavior.

Finally, accountability may be bypassed. If a system needs to track a user's actions, but a user is able to bypass that, the user could perform malicious operations and fly under the radar.

Developers can approach these problems on two fronts: specification and enforcement. Specification involves being thoughtful and deliberate about how permissions are assigned. Where these problems are known to exist within a system that you are using, you should consider supplementing them with your own logic and checks.

Enforcement problems occur when the program fails to adhere to the guidelines that an administrator sets. If there is a known enforcement problem, it would be wise to write and run tests to ensure that your highly sensitive data and functionality are protected.

NULL pointer dereference

A NULL pointer dereference occurs when a pointer with value NULL is treated as though it pointed to a valid memory area. A NULL pointer dereference results in a software failure. This problem can result in an exploitation if an attacker uses the stack trace to gain information about the software to plan an attack or if the exception allows a bypass of security checks.

All NULL pointer dereferences are unwelcome in a system, because they cause a process to fail. Whether the failure can be leveraged by an attacker is another question. Some proportion of them will be susceptible, but not all of them. When we consider the share of vulnerability paths made up of NULL pointer dereferences, it is heartening because only a small share of those paths are likely to be a problem from a security standpoint. We should avoid this issue if at all possible for the sake of reliability more so than security.

Remediation

As important as it is to understand the known vulnerabilities in stable Helm Charts, it is only an academic exercise unless we also talk about remediation. With respect to remediation, there is both good and bad news.

Let's discuss the bad news first. Currently none of the vulnerable images used in the stable Helm Charts have an available patch. Additionally, only 16% of vulnerabilities can be remediated through an image upgrade.

However, there is good news.

- 176 stable Helm Charts (64%) can benefit from an image upgrade. Not every vulnerability can be fixed, but the overall security health of these charts can be improved with an upgrade or upgrades.
- There are 261 image upgrades that can be made across the stable Helm Charts to improve security.

There is still a lot to do with respect to improving security across stable Helm Charts, but it is heartening to know that there are actionable items as of this writing.

Helm security

Just as Helm Charts are divided into incubating and stable, CNCF projects are divided into incubating and graduated. Graduated projects have met standards and are sufficiently mature for wide adoption. As of the writing of this report, Helm is currently classified as an incubating project and version 3.0 was released on November 13, 2019. Helm is seeking to move from incubation to graduation soon. One of the graduation criteria is to undergo a third party security audit, which Helm 3.0 has now successfully completed. The results of their audit are now publicly available and can be accessed through the Helm community's GitHub repository. Though one vulnerability was found and remediated, the report was very encouraging both with respect to Helm's general security posture and to the manual code audit that was performed. Congratulations to the Helm community for completing this important step towards graduation. This report helps security minded people adopt Helm with confidence.

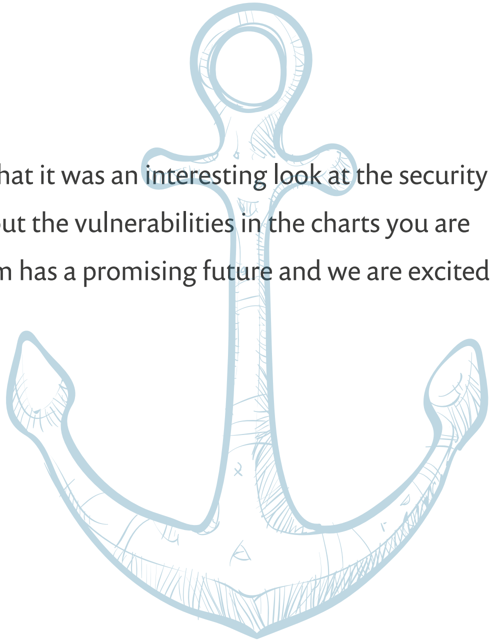
Another important aspect to consider when using a Helm Chart is your project configurations. The Kubernetes API is a powerful abstraction for building cloud native systems. But an unintended consequence of the rich API has been developers authoring by hand large amounts of configuration, mainly in YAML. This can be a security concern. Fortunately, there are a few things you can do to help your project stay secure and Helm has done a good job documenting these steps.

The default installation of a widely used version of Helm (2.14.3) applies **no security configurations**. This means that unless you are working against a cluster with no or very few security concerns, you need to invest some effort into thinking about the correct security configuration for your project. We suggest following the best practices outlined in the Helm documentation. If you are using the newly released version 3.0, we also recommend making use of the Helm provenance tools to verify the integrity and origin of a package.

We also recommend the use of conftest or similar tools to write configuration tests, so you can be confident in the configuration files that are in production.

Conclusion

Thank you for reading our report on Helm Chart security. We hope that it was an interesting look at the security implications of a popular and growing project. If you are curious about the vulnerabilities in the charts you are using, we encourage you to try our new plugin. We believe that Helm has a promising future and we are excited to help people use it securely.



snyk

Use open source. Stay secure.

Twitter: @snyksec
Web: https://snyk.io

Office info

London

1 Mark Square
London EC2A 4EG

Tel Aviv

40 Yavne st., first floor

Boston

WeWork 9th Floor
501 Boylston St
Boston, MA 02116

Report author

Hayley Denbraver (@hayleydenb)

Report contributors

Sarah Conway (@sarahkconway)
Jeff McLean
Gareth Rushgrove (@garethr)

Report design

Growth Labs (@GrowthLabsMKTG)