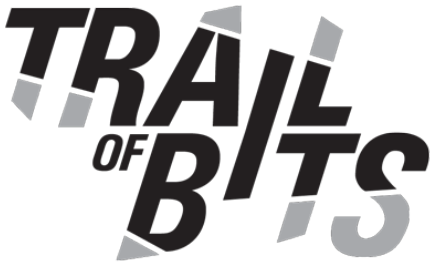

Helm

Security Assessment

August 10, 2020

Prepared For:
Matt Farina | *Helm*
matt@mattfarina.com

Matt Butcher | *Helm*
technosophos@gmail.com

Prepared By:
Dominik Czarnota | *Trail of Bits*
dominik.czarnota@trailofbits.com

Johanna Ratliff | *Trail of Bits*
johanna.ratliff@trailofbits.com

Executive Summary

From July 27 through August 5, 2020, Trail of Bits reviewed the security of Helm. We conducted this assessment over the course of three person-weeks with two engineers working from `v3.3.0-rc.1 (c2dfaa)` from the Helm repository.

In week one, we employed static analysis techniques such as `gosec`, `errcheck`, `CodeQL`, `semgrep`, and the GoLand code inspection feature while gaining a deeper understanding of the codebase through manual review. This yielded three findings ranging from informational to low severity. Additionally, we investigated the potential issue of `yaml` bombs, which resulted in pull request `go-yaml/yaml#637` to the `go-yaml` library used by Helm. This fix would address a theoretical scenario where the underlying counters used to prevent `yaml` bombs could overflow and lead to a denial of service.

In the final week, we conducted deeper analysis of certain code paths such as Helm's plugin system, template parsing and rendering, and interactions with Kubernetes clusters. This resulted in 11 more findings ranging from informational to medium severity. Most notably, findings TOB-HELM-009, TOB-HELM-011, and TOB-HELM-012 detail issues related to plugin command duplication; a path traversal in a chart configuration file that can lead to processing of malicious `yaml` files or information leaks; and a `yaml` validation deficiency that allows duplicate entries in chart repository index files.

We also reviewed the CNCF requirements criteria in Appendix E and included code quality recommendations in Appendix B to help improve and future-proof the Helm codebase.

Our assessment revealed a total of 14 findings ranging from medium to informational severity. Overall, the Helm codebase maturity could be improved. In some areas, it does not perform the necessary data validation, and in others the implementation either does not match the expected functionality or is not fully documented. These gaps can affect the security posture of the system since Helm users may make incorrect assumptions.

To improve the security posture of Helm, we recommend addressing the findings in this report, prioritizing short-term recommendations, and integrating long-term recommendations into future releases. Once fixes are applied, another assessment should be performed to ensure the fixes are adequate and do not introduce additional security risks.

Project Dashboard

Application Summary

Name	Helm
Version	v3.3.0-rc.1 (c2dfaa)
Type	Go
Platforms	MacOS, Linux, Windows

Engagement Summary

Dates	July 27, 2020–Aug 5, 2020
Method	Whitebox
Consultants Engaged	2
Level of Effort	3 person-weeks

Vulnerability Summary

Total High-Severity Issues	0	
Total Medium-Severity Issues	3	■ ■ ■
Total Low-Severity Issues	7	■ ■ ■ ■ ■ ■ ■
Total Informational-Severity Issues	3	■ ■ ■
Total Undetermined-Severity Issues	0	
Total	13	

Category Breakdown

Access Controls	1	■
Auditing and Logging	1	■
Configuration	1	■
Data Validation	9	■ ■ ■ ■ ■ ■ ■ ■ ■
Undefined Behavior	1	■
Total	13	

Engagement Goals

The engagement was scoped to provide a security assessment of Helm 3. Specifically, we sought to answer the following questions:

- Is the data processed by Helm validated properly?
- Are the filesystem permissions used by Helm secure?
- Does Helm provide the necessary logging for sensitive operations?
- Does Helm properly use basic cryptographic practices, where applicable?
- Are there any correctness issues in error-handling within Helm?
- Is there any evidence of credential issues or exposure?
- Does Helm meet CNCF requirements for secure development?

Coverage

Internal Helm logic and helm CLI. Using automated analysis tools such as `gosec`, `ineffassign`, `errcheck`, CodeQL, and `go vet`, we discovered the best locations to dive deeper into the code for manual analysis. Ultimately we focused on topics including (but not limited to) potential cascading errors from rollbacks and user permissiveness.

Helm upgrade and Helm rollback. Since we noticed a potential for cascading errors in a rollback scenario, we attempted many manual upgrade and rollback processes that were killed at various times during the process. We attempted to produce a situation in which a Helm rollback would fail but remain completely hidden from the user. Helm history successfully communicates the state of various upgrade and rollback scenarios even if the exact error is not known. This gives the user enough information to remedy the situation regardless of specifics (in the scenarios tested by Trail of Bits).

Decompression bombs. We assessed this by manually analyzing any places where inputs were unpacked to confirm that they could not cascade multiple levels deep. Helm could still be susceptible to an overlapping files compression bomb but the implementation of a memory limit in places where Helm takes input (such as charts) is not valid due to the variable nature of Helm releases and Kubernetes deployments.

`YAML` bombs. We tried different `YAML` bomb denial-of-service payloads against `yaml` files parsed by Helm. The `go-yaml` library used seems to properly protect against this scenario, but we sent a `go-yaml/yaml#637` pull request to improve this mitigation.

User permissions. When we analyzed permissions provided to users and processes to assess if overly permissive states could cause potential harm, we discovered that permissions are not checked for used configuration files (TOB-HELM-001) and the `MkdirAll` operation (TOB-HELM-014).

Data validation. Using various inputs, manual analysis, and debugging methods, we assessed commands and components such as the plugin system, template-rendering code, chart generation, parsing, and installation for insufficient validation that could lead to injections, path traversals, denial-of-service scenarios, and other bugs.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short term

□ **Check the permissions of important files processed by Helm and warn the user if they are too broad (e.g., readable or writable by others).** This will help protect users' configuration files from being leaked or manipulated by others, and may catch such vulnerabilities if they occur. TOB-HELM-001

□ **Merge the `validateReleaseName` and `validateMetadataName` functions into one function; then have this single function 1) check for name length and 2) match the regular expression.** If the metadata name check has different length requirements, keep the functions separate but add a sane length check. Alternatively, limit the number of match repetitions in the regular expression itself. This will prevent denial-of-service scenarios via regular expression processing functions. TOB-HELM-002

□ **Document that the `/etc/.mdns_debug` file stays open forever if it exists on Mac OS builds internally.** Documentation may expedite diagnosis of bugs related to this issue. Note: Trail of Bits will report this issue to Apple. TOB-HELM-003

□ **Validate the `helm create` command `NAME` parameter to prevent keys and values from being injected into the generated `yaml` files.** TOB-HELM-004

□ **Make sure the `helm create` command's documentation matches its implementation to prevent ambiguous scenarios.** TOB-HELM-005

□ **Change the logged message in the `helm create` command to indicate when the chart directory already exists so its content can be overwritten.** This will improve user experience of the Helm command line interface. TOB-HELM-006

□ **Fix the way Helm interacts with VCS systems so it doesn't depend on the user's VCS configuration files.** Use a VCS library to perform the necessary actions instead of executing VCS commands via shell. Alternatively, add command line flags to prevent the user's configuration files from being employed. This will help prevent unexpected behavior or failures when using Helm. TOB-HELM-007

□ **Either change the `NewForSource` function to allow for plugin names that end with archive extensions, or document the limitations for plugin URLs.** Additionally, extend

the error message to include more information about the issue to improve the Helm plugin installation experience for users. TOB-HELM-008

 Add appropriate validation to the installed or updated Helm plugin's command name to prevent the duplication of other commands or the injection of unexpected characters into the plugin name. Additionally, update the "Installing a Plugin" documentation to describe the plugin command name restrictions. TOB-HELM-009

 Fix the `dependencyStatus` function used by the `helm dependency list` command so it properly fetches the chart's dependencies statuses. Find archives that start with the dependency name specified in the `Chart.yaml` file, then check if the remaining name part is a version string. This will prevent the display of an unexpected "too many matches" status by the `helm dependency list` command for a chart's dependency with an "ok" status. TOB-HELM-010

 Validate that the `alias` field parsed from the chart's dependency specification is in an expected format. Such validation should prevent path traversal during template rendering. For example, it should disallow special characters like newlines that could lead to keys and value injection in `yaml` files. TOB-HELM-011

 Validate that the chart repository `index.yaml` file does not contain duplicate entries. Use a library routine that allows for strict `yaml` decoding or verification. This will prevent security issues as well as bug-prone situations that would permit an entry to be written into the index twice and allow the last duplicated entry to be used by Helm. TOB-HELM-012

 Validate that a repository already exists in Helm and error out that the user should first remove it if they want to overwrite it during the `helm repo add` command. Also, consider adding a special command line flag to trigger the overwriting of an existing repository. TOB-HELM-013

 When using utilities such as `os.MkdirAll`, check all directories in the path and validate their owner and permissions before performing operations on them. This will help prevent sensitive information from being written to a pre-existing attacker-controlled path. TOB-HELM-014

Long term

□ **Consider building a full static Helm binary that doesn't depend on system libraries** to prevent similar issues related to system libraries. TOB-HELM-003

□ **Add appropriate testing to ensure the `helm create` command performs the actions it is intended to do.** This will prevent future logic changes from introducing regressions. TOB-HELM-005

□ **Add tests to ensure the `helm` commands that use VCS systems do not rely on users' configuration files.** This will prevent future logic changes from introducing regressions. TOB-HELM-007

□ **Add tests to ensure the Helm plugins can be installed from VCS urls ending with archive extensions.** This will prevent future logic changes from introducing regressions. TOB-HELM-008

□ **Add tests for plugin installation, and updates to ensure that plugin command names are validated properly.** Plugin command names should be checked against unexpected values and overwriting of other Helm or plugin command names. TOB-HELM-009

□ **Add tests to ensure the `helm dependency list` command works properly with dependencies that have specific names as described in the finding.** This will prevent future logic changes from introducing regressions. TOB-HELM-010

□ **Add more tests to ensure that all chart `yaml` files and all possible fields are validated properly against malicious data.** This will help prevent various injection issues from appearing if the code changes. TOB-HELM-011

□ **Add tests to ensure the `helm repo add` command errors out when an already existing repository is added.** TOB-HELM-013

□ **Enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use.** Ideally, this validation should be centrally defined and used throughout the application as a whole. TOB-HELM-014

Findings Summary

#	Title	Type	Severity
1	Helm does not warn the user about important file permissions that are too broad	Auditing and Logging	Low
2	The `ValidName` kubernetes resource name regex may lead to denial of service	Data Validation	Low
3	Helm's Mac OS build keeps a file descriptor to `/etc/.mdns_debug` file open if this file exists due to bug in `mdns/lookup` library	Undefined Behavior	Informational
4	Lack of name validation in `helm create` command allows data to be injected into generated `yaml` files	Data Validation	Low
5	The `helm create` command does not overwrite files as stated in its help message	Data Validation	Low
6	The `helm create` command does not warn that a directory or file already exists	Auditing and Logging	Informational
7	Helm plugin executes VCS commands as external programs relying on user configuration	Configuration	Low
8	Plugins can't be installed from VCS if URL ends with archive extractor's extension	Data Validation	Informational
9	Plugin command name is not validated; can duplicate other plugin commands and Helm's top-level commands	Data Validation	Medium
10	The `helm dependency list` command won't print correct dependency status for certain dependency names	Data Validation	Informational
11	Path traversal through chart's dependency alias	Data Validation	Medium
12	Chart repository `index.yaml` file allows for duplicate entries	Data Validation	Medium
13	Adding `helm` repository may overwrite	Data Validation	Low

	another one without warning		
14	Directories created via `os.MkdirAll` are not checked for permissions	Access Controls	Low

1. Helm does not warn the user about important file permissions that are too broad

Severity: Low
Type: Auditing and Logging
Target: `various places in Helm codebase`

Difficulty: Medium
Finding ID: TOB-HELM-001

Description
Helm does not check if the permissions of `$KUBECONFIG` or other configuration files passed to it are too broad. This may allow user configuration files to be leaked without the user noticing.

This issue can be tested by changing permissions of the used files such as `~/.kube/config` or other used files (i.e., additional configuration files specified with `helm install -f`) to `0777` while using Helm. Helm does not log a warning in those scenarios.

Exploit Scenario
Alice creates a `yaml` file with permissions that are too broad to override values in later installed charts. Eve, another user on the same machine, accesses the file and steals important information from it or tampers with it to set up a backdoor in charts installed by Alice.

Recommendation
Short term, check the permissions of important files processed by Helm and warn the user if they are too broad (e.g., readable or writable by others). This will help protect users' configuration files from being leaked or manipulated by others, and may catch such vulnerabilities if they occur.

2. The `ValidName` Kubernetes resource name regex may lead to denial of service

Severity: Low | Difficulty: Low
Type: Data Validation | Finding ID: TOB-HELM-002
Target: `helm/pkg/action/action.go and helm/pkg/lint/rules/template.go`

Description
The `ValidName` regular expression (Figures 2.1-2) used for validating Kubernetes resource names is vulnerable to a regular expression denial of service (ReDoS).

Appendix C shows a proof of concept of this issue and regex matching timings for payloads of different length. This regex is used in the `validateReleaseName` and `validateMetadataName` functions (Figures 2.3-4). They are used to validate the Kubernetes resource names either passed to Helm commands as CLI arguments or present in linted charts' `yaml` files.

```
var ValidName =
regexp.MustCompile(`^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`)
```

Figure 2.1: The `ValidName` regex. (helm/pkg/action/action.go#L65-L75 also defined in helm/pkg/lint/rules/template.go#L46)

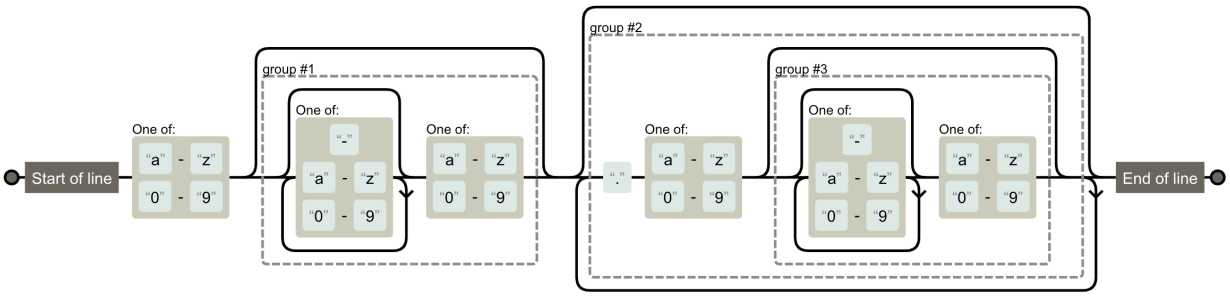

Figure 2.2: Visualization of the `ValidName` regex (regexper.com).

The `validateReleaseName` function also validates the length of the matched string, which could prevent the ReDoS case. However, this check is performed after matching the regex, which allows a longer payload to be processed and causes the problem. Additionally, this function is used with input that comes from command-line arguments. Considering that operating systems limit the length of CLI arguments, it is unlikely that the `validateReleaseName` function can currently be used to trigger the DoS.

```
func validateReleaseName(releaseName string) error {
        if releaseName == "" {
                return errMissingRelease
        }
```

```
        if !ValidName.MatchString(releaseName) || (len(releaseName) > releaseNameMaxLen) {
                return errInvalidName
        }

        return nil
}
```

Figure 2.3: The `validateReleaseName` *function. (helm/pkg/action/upgrade.go#L145-L155)*

```
func validateMetadataName(obj *K8sYamlStruct) error {
        // This will return an error if the characters do not abide by the standard OR if the
        // name is left empty.
        if validName.MatchString(obj.Metadata.Name) {
                return nil
        }
        return fmt.Errorf("object name does not conform to Kubernetes naming requirements:
%q", obj.Metadata.Name)
}
```

Figure 2.4: The `validateMetadataName` *function. (helm/pkg/lint/rules/template.go#L166-L173)*

Exploit Scenario

Eve prepares a malicious chart with the following command:

```
python3 -c 'open("Chart.yaml", "w").write("name: "+"a"*1_100_000_000+".")'
```

Even then sends this 1GB `Chart.yaml` file to Alice which lints it via the `helm lint` command and causes a denial of service on Alice's machine.

Recommendation

Short term, merge the `validateReleaseName` and `validateMetadataName` functions into one function; then have this single function 1) check for name length and 2) match the regular expression. If the metadata name check has different length requirements, keep the functions separate but add a sane length check. Alternatively, limit the number of match repetitions in the regular expression itself. This will prevent denial-of-service scenarios via regular expression processing functions.

3. Helm's Mac OS build keeps a file descriptor to `/etc/.mdns_debug` file open if this file exists due to bug in `mdns/lookup` library

Severity: Informational
Type: Undefined Behavior
Target: `Helm Mac OS build`

Difficulty: High
Finding ID: TOB-HELM-003

Description
A Mac OS Helm build tries to access a `/etc/.mdns_debug` file as one of the first files it opens (Figure 3.1). If this file exists, it is then left open, most likely unnecessarily. This can lead to unexpected behavior if, for example, Helm is run with limited resources or if it's assumed that a certain file descriptor corresponds to a specific resource.

It seems this issue comes from Apple's `mdns` lookup library, which opens the `/etc/.mdns_debug` file during its initialization to check some debug flags and never closes it (Figure 3.2).

```
$ sudo opensnoop -n helm
dtrace: system integrity protection is on, some features will not be available

  UID    PID COMM          FD PATH
  501  57574 helm           5 .
  501  57574 helm           5 /dev/dtracehelper
  501  57574 helm           5 /dev/urandom
  501  57574 helm           5 /etc/.mdns_debug
  501  57574 helm           6 /Users/dc/tob/helm/bin
  501  57574 helm           6 /Users/dc/tob/helm
  501  57574 helm           6 /Users/dc/tob
  501  57574 helm           6 /Users/dc
  501  57574 helm           6 /Users
  501  57574 helm           6 /Users/dc/tob/helm/bin
  501  57574 helm          -1 /Users/dc/tob/helm/bin/Info.plist
  501  57574 helm           8 /Users/dc/.kube/config
  ...
```

Figure 3.1: Files accessed by Helm on a Mac OS build, inspected with the opensnoop tool.

```
#define MDNS_DEBUG_FILE "/etc/.mdns_debug"

// (...)

static void
_mdns_init(void)
{
        pthread_atfork(_mdns_atfork_prepare, _mdns_atfork_parent, _mdns_atfork_child);

        if (getenv("RES_DEBUG") != NULL) _mdns_debug |= MDNS_DEBUG_STDOUT;
        int fd = open(MDNS_DEBUG_FILE, O_RDONLY, 0);
        errno = 0;

        if (fd >= 0)
        {
                int i, n;
```

```
                char c[5];
                memset(c, 0, sizeof(c));
                n = read(fd, c, 4);

                for (i = 0; i < n; i++)
                {
                        if ((c[i] == 'o') || (c[i] == 'O')) _mdns_debug |= MDNS_DEBUG_STDOUT;
                        if ((c[i] == 'e') || (c[i] == 'E')) _mdns_debug |= MDNS_DEBUG_STDERR;
                        if ((c[i] == 'a') || (c[i] == 'A')) _mdns_debug |= MDNS_DEBUG_ASL;
                }
        }
}
```

Figure 3.2: The `_mdns_init` *function that opens the* `/etc/.mdns_debug` *file and never closes it. (opensource.apple.com/source/Libinfo/Libinfo-476/lookup.subproj/mdns_module.c)*

Recommendation

Short term, document that the `/etc/.mdns_debug` file stays open forever if it exists on Mac OS builds internally. Documentation may expedite diagnosis of bugs related to this issue. Note: Trail of Bits will report this issue to Apple.

Long term, consider building a full static Helm binary that doesn't depend on system libraries to prevent similar issues related to system libraries.

4. Lack of name validation in `helm create` command allows data to be injected into generated `yaml` files

Severity: Low
Type: Data Validation
Target: `Helm create command`

Difficulty: Medium
Finding ID: TOB-HELM-004

Description
The `helm create` command, which generates a chart directory and files, doesn't validate the passed `NAME` argument. An attacker may include newlines in the `NAME` argument and inject keys and values into the generated `yaml` files. Figure 4.1 shows an example injection:

```
$ ./helm create "$(printf "xxx\ninjected_key: injected_value")"
Creating xxx
injected_key: injected_value

$ head -n4 xxx$'\n'injected_key:\ injected_value/*.yaml
==> xxx
injected_key: injected_value/Chart.yaml <==
apiVersion: v2
name: xxx
injected_key: injected_value
description: A Helm chart for Kubernetes

==> xxx
injected_key: injected_value/values.yaml <==
# Default values for xxx
injected_key: injected_value.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.
```

Figure 4.1: Proof of concept of `helm create` *command* `NAME` *parameter injection.*

Exploit Scenario
Alice sets up a web application that allows her to create and test charts. Eve uses the form and injects a malicious payload to the generated yamls through the name parameter.

Recommendation
Short term, validate the `helm create` command `NAME` parameter to prevent keys and values from being injected into the generated `yaml` files.

5. The `helm create` command does not overwrite files as stated in its help message

Severity: Low
Type: Data Validation
Target: `Helm create command`

Difficulty: Low
Finding ID: TOB-HELM-005

Description
The `helm create` command generates a chart directory and files. According to its help message (Figure 5.1), if the files in the chart directory already exist, the conflicting files will be overwritten. However, the current implementation does not overwrite any files and just skips them if they exist (Figure 5.2).

```
This command creates a chart directory along with the common files and
directories used in a chart.

// (...)

'helm create' takes a path for an argument. If directories in the given path
do not exist, Helm will attempt to create them as it goes. If the given
destination exists and there are files in that directory, conflicting files
will be overwritten, but other files will be left alone.
```

Figure 5.1: The `helm create` *command's help message (*`helm create --help`*).*

```
func Create(name, dir string) (string, error) {
	// (...)
	for _, file := range files {
		if _, err := os.Stat(file.path); err == nil {
			// File exists and is okay. Skip it.
			continue
		}
		if err := writeFile(file.path, file.content); err != nil {
			return cdir, err
		}
	}
	// Need to add the ChartsDir explicitly as it does not contain any file OOTB
	if err := os.MkdirAll(filepath.Join(cdir, ChartsDir), 0755); err != nil {
		return cdir, err
	}
	return cdir, nil
}
```

Figure 5.2: The `Create` *function. (*helm/pkg/chartutil/create.go#L601-L609*)*

Exploit Scenario
Alice uses `helm create mychart` to create a chart and modifies it. Later, she decides to recreate the chart and invokes the `helm create mychart` command, assuming it will overwrite existing files as stated in the help message. Alice then only modifies some of the files and is left with an unexpected state.

Recommendation

Short term, make sure the `helm create` command's documentation matches its implementation to prevent ambiguous scenarios.

Long term, add appropriate testing to ensure the `helm create` command performs the actions it is intended to do. This will prevent future logic changes from introducing regressions.

6. The `helm create` command does not warn that a directory or file already exists

Severity: Informational
Type: Auditing and Logging
Target: `Helm create command`

Difficulty: Low
Finding ID: TOB-HELM-006

Description
If the `helm create` command generates a chart directory and files even though they already exist, the command should overwrite them. However, there's no information that the chart directory already exists or that conflicting files are being overwritten (Figure 6.1).

The description assumes the issue from Finding TOB-HELM-005 is not present.

```
$ ./helm create xxx
Creating xxx
$ ./helm create xxx
Creating xxx
```

Figure 6.1: Executing the `helm create` *command twice on the same chart name.*

Recommendation
Short term, change the logged message in the `helm create` command to indicate when the chart directory already exists so its content can be overwritten. This will improve user experience of the Helm command line interface.

7. Helm executes VCS commands as external programs relying on user configuration

Severity: Low
Type: Configuration
Target: `Helm plugin command`

Difficulty: High
Finding ID: TOB-HELM-007

Description
The `helm plugin` command can install or update plugins from version control systems (VCS) by executing VCS commands as external processes. The executed VCS programs depend on user configuration files, which may lead to failures or other unexpected behavior.

Figure 7.1 shows an example git configuration that disables fast-forward merges. When such a configuration is present and a plugin is installed from git and updated, no further updates will work due to merge conflicts.

```
[merge]
    ff = false
[pull]
    ff = false
```

Figure 7.1: An example `.gitconfig` that breaks consecutive updates of Helm plugins installed from git repositories.

Recommendation
Short term, fix the way Helm interacts with VCS systems so it doesn't depend on the user's VCS configuration files. Use a VCS library to perform the necessary actions instead of executing VCS commands via shell. Alternatively, add command line flags to prevent the user's configuration files from being employed. This will help prevent unexpected behavior or failures when using Helm.

Long term, add tests to ensure the `helm` commands that use VCS systems do not rely on users' configuration files. This will prevent future logic changes from introducing regressions.

8. Plugins can't be installed from VCS if URL ends with archive extractor's extension

Severity: Informational
Type: Data Validation
Target: `Helm plugin command`

Difficulty: Low
Finding ID: TOB-HELM-008

Description
As seen in Figure 8.1, the `helm plugin install` command installs a Helm plugin from a given source by checking if it's:

1. A local reference.
2. A remote http archive.
3. A version control system (VCS) path.

This order and the way the `isRemoteHTTPArchive` function performs its check (Figure 8.2) make it impossible to install a plugin from repositories whose URLs end with `.tgz` or `.tar.gz`.

```
// NewForSource determines the correct Installer for the given source.
func NewForSource(source, version string) (Installer, error) {
	// Check if source is a local directory
	if isLocalReference(source) {
		return NewLocalInstaller(source)
	} else if isRemoteHTTPArchive(source) {
		return NewHTTPInstaller(source)
	}
	return NewVCSInstaller(source, version)
}
```

Figure 8.1: The `NewForSource` *function.*
(helm/pkg/plugin/installer/installer.go#L65-L74)

```
// Extractors contains a map of suffixes and matching implementations of extractor to return
var Extractors = map[string]Extractor{
	".tar.gz": &TarGzExtractor{},
	".tgz":    &TarGzExtractor{},
}

// isRemoteHTTPArchive checks if the source is a http/https url and is an archive
func isRemoteHTTPArchive(source string) bool {
	if strings.HasPrefix(source, "http://") || strings.HasPrefix(source, "https://") {
		for suffix := range Extractors {
			if strings.HasSuffix(source, suffix) {
				return true
			}
		}
	}
	return false
}
```

Figure 8.2: The extractor's extensions and the `isRemoteHTTPArchive` *function. (helm/pkg/plugin/installer/http_installer.go#L56-L60 and helm/pkg/plugin/installer/installer.go#L91-L101)*

The issue can be reproduced by forking the https://github.com/adamreese/helm-env repository, renaming it to end with `.tgz` or `.tar.gz`, and trying to install the forked Helm plugin via its https url. Figure 8.3 shows the output of such an install attempt.

```
$ ./helm plugin install https://github.com/disconnect3d/helm-env.tgz
Error: extracting files from archive: gzip: invalid header
```

Figure 8.3: An attempt to install the Helm plugin from the GitHub repository with its name ending in ".tgz."

Recommendation

Short term, either change the `NewForSource` function to allow for plugin names that end with archive extensions, or document the limitations for plugin URLs. Additionally, extend the error message to include more information about the issue to improve the Helm plugin installation experience for users.

Long term, add tests to ensure Helm plugins can be installed from VCS urls ending with archive extensions. This will prevent future logic changes from introducing regressions.

9. Plugin command name is not validated; can duplicate other plugin commands and Helm's top-level commands

Severity: Medium
Type: Data Validation
Target: `Helm plugin command`

Difficulty: High
Finding ID: TOB-HELM-009

Description
The `helm plugin install` and `update` commands do not validate the installed or updated plugin's command name specified in the plugin's `plugin.yaml` file's `name` key. As a result, a plugin installation or update can result in:

- Duplication of another plugin's or Helm's top level command name.
- Injection of special characters such as newlines into the plugin command name, which allows for spoofing the `helm --help` result display.

This Helm behavior is also contradictory to the restrictions listed in Helm's documentation, as shown in Figure 9.1.

> Restrictions on `name`:
>
> - `name` cannot duplicate one of the existing `helm` top-level commands.
> - `name` must be restricted to the characters ASCII a-z, A-Z, 0-9, `_` and `-`.

Figure 9.1: Helm plugin name restrictions from documentation. (https://helm.sh/docs/topics/plugins/)

Exploit Scenario
Eve releases a new version of her plugin and changes the plugin command. Alice updates Eve's plugin in her Helm installation, which ends up overlapping another plugin's command. Alice then executes another plugin whose command was overwritten, which ends up executing Eve's plugin.

Recommendation
Short term, add appropriate validation to the installed or updated Helm plugin's command name to prevent the duplication of other commands or the injection of unexpected characters into the plugin name. Additionally, update the "Installing a Plugin" documentation to describe the plugin command name restrictions.

Long term, add tests for plugin installation, and updates to ensure that plugin command names are validated properly. Plugin command names should be checked against unexpected values and overwriting of other Helm or plugin command names.

10. The `helm dependency list` command won't print correct dependency status for certain dependency names

Severity: Informational
Type: Data Validation
Target: `helm/pkg/action/dependency.go`

Difficulty: High
Finding ID: TOB-HELM-010

Description

The `helm dependency list` command lists dependencies for a given chart. If the chart consists of two dependencies `X` and `X-Y`, for which the "`.tgz`" archives exist in the chart's `charts/` directory, the dependency status printed for the `X` chart will incorrectly be "`too many matches.`" This might cause issues for users who check chart dependencies through the `helm dependency list` command.

This issue exists because the `dependencyStatus` function fetches the dependency chart archives with a "`<chartpath>/charts/<dependency-name>-*.tgz`" filepath glob:

```
func (d *Dependency) dependencyStatus(chartpath string, dep *chart.Dependency, parent
*chart.Chart) string {
    filename := fmt.Sprintf("%s-%s.tgz", dep.Name, "*")

    // (...)
    switch archives, err := filepath.Glob(filepath.Join(chartpath, "charts", filename)); {
    case err != nil:
        return "bad pattern"
    case len(archives) > 1:
        return "too many matches"
    // (...)
```

Figure 10.1: The `dependencyStatus` *function. (helm/pkg/action/dependency.go#L64-L65)*

Recommendations

Short term, fix the `dependencyStatus` function used by the `helm dependency list` command so it properly fetches the chart's dependencies statuses. Find archives that start with the dependency name specified in the `Chart.yaml` file, then check if the remaining name part is a version string. This will prevent the display of an unexpected "too many matches" status by the `helm dependency list` command for a chart's dependency with an "ok" status.

Long term, add tests to ensure the `helm dependency list` command works properly with dependencies that have specific names as described in the finding. This will prevent future logic changes from introducing regressions.

11. Path traversal through chart's dependency alias

Severity: Medium
Type: Data Validation
Target: `helm template rendering`

Difficulty: High
Finding ID: TOB-HELM-011

Description
The chart's dependency specification in the `Chart.yaml` file allows specification of an `alias` key so the dependency can later be referenced through its alias. However, it is possible to trigger a path traversal during template rendering with the `helm template <chart>` command through the `alias` field.

The path traversal happens in the `ChartFullPath` function (Figure 11.1), which is used in the `recAllTpls` function (Figure 11.2), and further used by the `helm template` command. While the `ChartFullPath` function uses the dependency chart's `Name` field, Figure 11.3 shows that when the `Alias` field exists, it overwrites the dependency's `Name` field so it is used instead later on.

Trail of Bits hasn't fully confirmed the impact of this vulnerability, but it seems this issue allows for adding files outside of the specified chart directory tree to be processed during rendering of a template.

Figure 11.4 shows the necessary steps to reproduce the issue.

```
// ChartFullPath returns the full path to this chart.
func (ch *Chart) ChartFullPath() string {
	if !ch.IsRoot() {
		return ch.Parent().ChartFullPath() + "/charts/" + ch.Name()
	}
	return ch.Name()
}
```

Figure 11.1: The `ChartFullPath` function. (helm/pkg/chart/chart.go#L111-L117) The `ch.Name()` implementation is in the same file. (helm/pkg/chart/chart.go#L70-L76)

```
func recAllTpls(c *chart.Chart, templates map[string]renderable, vals chartutil.Values) {
	// (...)
	for _, child := range c.Dependencies() {
		recAllTpls(child, templates, next)
	}

	newParentID := c.ChartFullPath()
	for _, t := range c.Templates {
		if !isTemplateValid(c, t.Name) {
			continue
		}
		templates[path.Join(newParentID, t.Name)] = renderable{
			tpl:      string(t.Data),
			vals:     next,
			basePath: path.Join(newParentID, "templates"),
		}
```

```
        }
}
```

Figure 11.2: The `recAllTpls` *function. (helm/pkg/engine/engine.go#L333-L369)*

```
                if dep.Alias != "" {
                        md.Name = dep.Alias
                }
```

Figure 11.3: The metadata's `Name` *field is overwritten by the* `Alias` *field in the* `getAliasDependency` *function. (helm/pkg/chartutil/dependencies.go#L109-L111) Similar code exists in the* `processDependencyEnabled` *function. (helm/pkg/chartutil/dependencies.go#L143-L145)*

```
$ ./helm create a
Creating a

$ ./helm pull stable/mariadb -d ./a/charts/
$ cat <<EOF >>a/Chart.yaml
heredoc> dependencies:
heredoc>   - name: mariadb
heredoc>     version: 7.3.14
heredoc>     alias: ../../traverse
heredoc> EOF

$ ./helm template a
Error: template: traverse/templates/tests.yaml:1:14: executing
"traverse/templates/tests.yaml" at <.Values.tests.enabled>: nil pointer evaluating interface
{}.enabled

Use --debug flag to render out invalid YAML
```

Figure 11.4: Steps to reproduce the path traversal. The "`heredoc`*" is the prompt sign from multi-line input that ends with* `EOF`*. The red highlight shows that the engine tried to use the non-existent path.*

Exploit Scenario

Eve, who can only manipulate a chart's dependency alias, sets it so it traverses to another directory. Alice then renders the chart's templates and gets hit by Eve's attack.

Eve can use this bug to either include her own values into the rendered templates by preparing her own `yaml` files, or to leak data by traversing to another directory only readable by Alice. However, the latter scenario is unlikely, as it would require the data to be structured in such a way that the template render would not fail.

Recommendation

Short term, validate that the `alias` field parsed from the chart's dependency specification is in an expected format. Such validation should prevent path traversal during template rendering. For example, it should disallow special characters like newlines that could lead to keys and value injection in `yaml` files.

Long term, add more tests to ensure that all chart `yaml` files and all possible fields are validated properly against malicious data. This will help prevent various injection issues from appearing if the code changes.

12. Chart repository `index.yaml` file allows for duplicate entries

Severity: Medium
Type: Data Validation
Target: `chart repository index validation`

Difficulty: High
Finding ID: TOB-HELM-012

Description

The chart repository `index.yaml` file consists of entries parsed by the `loadIndex` function into a mapping (Figures 12.1-2), which can't contain duplicate keys. However, when the index file contains duplicate entries, the last entry is parsed. This is contrary to the `yaml 1.2` specification, which states that the mapping keys must be unique and duplicates should be treated as errors (Figure 12.3).

```
func loadIndex(data []byte) (*IndexFile, error) {
	i := &IndexFile{}
	if err := yaml.Unmarshal(data, i); err != nil {
		return i, err
	}
	i.SortEntries()
	if i.APIVersion == "" {
		return i, ErrNoAPIVersion
	}
	return i, nil
}
```

Figure 12.1: The `loadIndex` *function. (helm/pkg/repo/index.go#L279-L292)*

```
// IndexFile represents the index file in a chart repository
type IndexFile struct {
	APIVersion string                `json:"apiVersion"`
	Generated  time.Time             `json:"generated"`
	Entries    map[string]ChartVersions `json:"entries"`
	PublicKeys []string              `json:"publicKeys,omitempty"`
}
```

Figure 12.2: The `IndexFile` *structure. (helm/pkg/repo/index.go#L78-L84)*

> 1.3. Relation to JSON
> **JSON's RFC4627 requires that mappings keys merely "SHOULD" be unique, while YAML insists they "MUST" be. Technically, YAML therefore complies with the JSON spec, choosing to treat duplicates as an error.** In practice, since JSON is silent on the semantics of such duplicates, the only portable JSON files are those with unique keys, which are therefore valid YAML files.
>
> 3.2.1. Representation Graph
> (...) YAML supports two kinds of collection nodes: sequences and mappings. **Mapping nodes are somewhat tricky because their keys are unordered and must be unique.**
>
> 3.2.1.3. Node Comparison
> **Since YAML mappings require key uniqueness,** representations must include a mechanism for testing the equality of nodes. This is non-trivial since YAML allows various ways to format scalar content. For example, the integer eleven can be written as "0o13" (octal) or "0xB" (hexadecimal). **If both notations are used as keys in the same mapping, only a YAML processor**

```
which recognizes integer formats would correctly flag the duplicate key as an error.
```

Figure 12.3: Excerpts from `yaml 1.2` *specification. (*https://yaml.org/spec/1.2/spec.html*)*

Such activity can lead to unexpected behavior when entries are mistakenly duplicated, or to security issues if an attacker is able to inject special characters into a generated index file. Appendix D shows how to reproduce this issue.

Exploit Scenario
Alice hosts a chart repository server and generates the repository index file based on the uploaded data. Eve finds a bug in Alice's server that allows her to inject newlines into the generated index files, and she uses it to overwrite all other existing chart specifications to return a backdoored version of them.

Recommendation
Short term, validate that the chart repository `index.yaml` file does not contain duplicate entries. Use a library routine that allows for strict `yaml` decoding or verification. This will prevent security issues as well as bug-prone situations that would permit an entry to be written into the index twice and allow the last duplicated entry to be used by Helm.

13. Adding `helm` repository may overwrite another one without warning

Severity: Low
Type: Data Validation
Target: `helm repo add command`

Difficulty: High
Finding ID: TOB-HELM-013

Description
When the `helm repo add` command is executed with an already existing repository name, the command overwrites the existing repository without an error (Figure 13.1). This can lead to unexpected behavior or chart versioning issues if a user overwrites one repository with another that contains the same chart name.

```
$ ./helm repo add my http://localhost:5000/
"my" has been added to your repositories

$ ./helm repo list
NAME   URL
stable https://kubernetes-charts.storage.googleapis.com/
my     http://localhost:5000/

$ ./helm repo add my http://localhost:8000/
"my" has been added to your repositories

$ ./helm repo list
NAME   URL
stable https://kubernetes-charts.storage.googleapis.com/
my     http://localhost:8000/
```

Figure 13.1: Overwriting one existing repository with another.

Exploit Scenario
Alice, who has a "release" and "staging" repository added to Helm, updates the "staging" repository url by using the `helm repo add` command and mistakenly names it "release." Alice then installs charts from the staging and release repository, not noticing that the installed charts are not the intended ones. The installation of the untested version causes problems in Alice's production environment.

Recommendation
Short term, validate that a repository already exists in Helm and error out that the user should first remove it if they want to overwrite it during the `helm repo add` command. Also, consider adding a special command line flag to trigger the overwriting of an existing repository.

Long term, add tests to ensure the `helm repo add` command errors out when an already existing repository is added.

14. Directories created via `os.MkdirAll` are not checked for permissions

Severity: Low
Type: Access Controls
Target: `multiple locations`

Difficulty: High
Finding ID: TOB-HELM-014

Description
Helm uses the `os.MkdirAll` function to create certain directory paths with specific access permissions (`0755`). This function does not perform any permission checks when a given directory path already exists. This allows an attacker to create a directory with broad permissions that Helm will use later, thus allowing the attacker to tamper with the files used by Helm.

Exploit Scenario
Eve has unprivileged access to the machine where Alice uses Helm. Eve watches the commands executed by Alice and introduces new directories/paths with `0777` permissions before Helm does so. Eve can then delete and forge files in that directory to change the result of further commands executed by Alice.

Recommendation
Short term, when using utilities such as `os.MkdirAll`, check all directories in the path and validate their owner and permissions before performing operations on them. This will help prevent sensitive information from being written to a pre-existing attacker-controlled path.

Long term, enumerate files and directories for their expected permissions overall, and build validation to ensure appropriate permissions are applied before creation and upon use. Ideally, this validation should be centrally defined and used throughout the application as a whole.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for

	client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses in order to exploit this issue

B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability or user experience and may prevent the introduction of vulnerabilities in the future.

- `internal/monocular/search.go#L132-L134`: Return and check the error result from JSON decoding. This ensures that a failure to search due to invalid JSON is propagated up to the user for easier debugging.
- `cmd/helm/list.go#L94-L99`: Instead of returning `nil`, return the error from `EncodeJSON` and `EncodeYAML`.
- `pkg/storage/driver/util.go#L41-L48`: `defer w.Close()` after creation of the writer in case an error causes a return before the writer is closed in a linear fashion.
- Document the "memory" storage backend in the "Storage backends" documentation page.
- `pkg/kube/wait.go#L191-L192`: Either simplify the expression to `if s.Spec.ClusterIP == ""` or, if both conditions are intended, change the `&&` to `||`.

C. The `ValidName` Regex Denial-of-Service Timing

This appendix shows the timing of the regular expression denial-of-service vulnerability described in TOB-HELM-002. Figure C.1 shows the timing code, and Figure C.2 shows the output of running it.

```
package main

import (
    "time"
    "log"
    "regexp"
    "strings"
)

var ValidName =
regexp.MustCompile(`^[a-z0-9]([-a-z0-9]*[a-z0-9])?(\.[a-z0-9]([-a-z0-9]*[a-z0-9])?)*$`)

func measure(input string) {
    start := time.Now()
    result := ValidName.MatchString(input)
    elapsed := time.Since(start)
    log.Printf("Match took %s for length %d, matched: %v", elapsed, len(input), result)
}

func main() {
    for i := 1; i<=1000000000; i *= 10 {
        a := strings.Repeat("a", i) + "."
        measure(a)
    }
}
```

Figure C.1: Code used to confirm the regular expression denial-of-service issue with the `ValidName` *regex.*

```
$ go run main.go
2020/07/30 13:23:57 Match took 16µs  for length 2, matched: false
2020/07/30 13:23:57 Match took 2µs   for length 11, matched: false
2020/07/30 13:23:57 Match took 8µs   for length 101, matched: false
2020/07/30 13:23:57 Match took 98µs  for length 1001, matched: false
2020/07/30 13:23:57 Match took 1ms   for length 10001, matched: false
2020/07/30 13:23:57 Match took 9ms   for length 100001, matched: false
2020/07/30 13:23:58 Match took 86ms  for length 1000001, matched: false
2020/07/30 13:23:58 Match took 816ms for length 10000001, matched: false
2020/07/30 13:24:07 Match took 8.1s  for length 100000001, matched: false
2020/07/30 13:25:32 Match took 1m26s for length 1000000001, matched: false
```

Figure C.2: Output of the code from Figure C.1 tested on `Go 1.13.9 darwin/amd64` *build. Reformatted for easier readability (removed the timing fraction part and added alignment).*

D. How to Reproduce Finding 12 (Duplicate Entries in Chart Repository)

To reproduce Finding TOB-HELM-012:

1. Create the `index.yaml` file from Figure E.1 and update the urls and digest (if needed).
2. In the same directory, host a simple http server, e.g., with the `python3 -m http.server` command.
3. In another console, add the hosted server as a chart repository, update the cache, and install the "`my/mytest`" chart, as shown in Figure E.2.

The installation will install the last duplicated entry, which in this case is a `mariadb` chart.

```
apiVersion: v1
entries:
  mytest:
  - apiVersion: v1
    appVersion: 1.1.1
    created: "2020-08-09T19:25:06.92284+02:00"
    digest: 666967b217ed498eadb4b025e3bfe382a12ae2ff9c442e8abc11d8f27b78194c
    name: mytest
    urls:
    - https://kubernetes-charts.storage.googleapis.com/mysql-1.6.6.tgz
    version: 1.1.1
  mytest:
  - apiVersion: v1
    appVersion: 1.1.0
    created: "2020-11-03T19:25:06.92284+02:00"
    digest: 8f91980656568074178e2c02ad808b1db124f244ceb19d3850776c7feac80184
    name: mytest
    urls:
    - https://kubernetes-charts.storage.googleapis.com/mariadb-7.3.14.tgz
    version: 1.1.0
generated: "2020-08-09T19:25:06.916876+02:00"
```

Figure E.1: Example chart repository `index.yaml` *file.*

```
$ ./helm repo add my http://localhost:8000/
$ ./helm repo update
$ ./helm install --generate-name my/mytest
```

Figure E.2: Adding the local chart repository to Helm and installing a chart from it.

E. CNCF Requirements Criteria Review

This appendix lists general improvements based upon best practices for Free/Libre and Open Source Software (FLOSS) projects that could be applied to the Helm project. Introducing these changes will help accommodate the CNCF project graduation criteria.

Detail the contribution process in the repository's README. While there is a "Community, discussion, contribution, and support" section, it doesn't link to the CONTRIBUTING file or to the helm/community repository that details the contribution process.

Consider updating the GitHub issues and pull request templates to the new flow. This will make it easier to keep a consistent format for submitted issues, feature requests, and pull requests.

Link the "Helm Security Process and Policy" document in the CONTRIBUTING document. This document is currently only referenced in the SECURITY file, and the CONTRIBUTING document does not describe the whole process, e.g., that the report can be encrypted.

Force the use of TLS and check the downloaded chart's digests. As pointed out during Helm threat modeling, Helm does not enforce TLS connections or check the downloaded charts hashes, especially when they are downloaded through an insecure HTTP connection. The project should enforce TLS and check the downloaded archive's hash to prevent man-in-the-middle (MITM) attacks. Another flag can be added so users can work with an insecure connection if needed.

Add more tests and track test coverage in pull requests. Increase project test coverage (Figure F.1) and introduce a component to the project's CI system to track changes in code coverage as the project matures.

```
$ GO111MODULE=on go test -run . ./... -coverprofile cover.out
ok      helm.sh/helm/v3/cmd/helm 9.090s  coverage: 66.7% of statements
ok      helm.sh/helm/v3/cmd/helm/require 0.335s  coverage: 100.0% of statements
ok      helm.sh/helm/v3/cmd/helm/search  0.566s  coverage: 96.1% of statements
ok      helm.sh/helm/v3/internal/experimental/registry   2.654s  coverage: 80.1% of statements
ok      helm.sh/helm/v3/internal/fileutil        0.494s  coverage: 58.3% of statements
ok      helm.sh/helm/v3/internal/ignore  0.348s  coverage: 86.1% of statements
ok      helm.sh/helm/v3/internal/monocular       0.603s  coverage: 74.1% of statements
ok      helm.sh/helm/v3/internal/resolver        1.139s  coverage: 81.2% of statements
ok      helm.sh/helm/v3/internal/sympath 0.619s  coverage: 71.1% of statements
?       helm.sh/helm/v3/internal/test    [no test files]
?       helm.sh/helm/v3/internal/test/ensure     [no test files]
ok      helm.sh/helm/v3/internal/third_party/dep/fs      0.645s  coverage: 45.1% of statements
?       helm.sh/helm/v3/internal/third_party/k8s.io/kubernetes/deployment/util    [no test files]
ok      helm.sh/helm/v3/internal/tlsutil 0.459s  coverage: 74.3% of statements
```

```
ok      helm.sh/helm/v3/internal/urlutil 0.299s  coverage: 91.7% of statements
?       helm.sh/helm/v3/internal/version [no test files]
ok      helm.sh/helm/v3/pkg/action       1.398s  coverage: 52.4% of statements
ok      helm.sh/helm/v3/pkg/chart        0.223s  coverage: 78.6% of statements
ok      helm.sh/helm/v3/pkg/chart/loader 0.269s  coverage: 81.5% of statements
ok      helm.sh/helm/v3/pkg/chartutil    0.768s  coverage: 80.3% of statements
ok      helm.sh/helm/v3/pkg/cli  0.584s  coverage: 83.3% of statements
?       helm.sh/helm/v3/pkg/cli/output   [no test files]
ok      helm.sh/helm/v3/pkg/cli/values   0.941s  coverage: 17.1% of statements
ok      helm.sh/helm/v3/pkg/downloader   1.004s  coverage: 68.6% of statements
ok      helm.sh/helm/v3/pkg/engine       1.002s  coverage: 69.5% of statements
ok      helm.sh/helm/v3/pkg/gates        1.063s  coverage: 100.0% of statements
ok      helm.sh/helm/v3/pkg/getter       1.457s  coverage: 87.8% of statements
ok      helm.sh/helm/v3/pkg/helmpath     0.223s  coverage: 68.2% of statements
?       helm.sh/helm/v3/pkg/helmpath/xdg [no test files]
ok      helm.sh/helm/v3/pkg/kube 1.407s  coverage: 28.4% of statements
?       helm.sh/helm/v3/pkg/kube/fake    [no test files]
ok      helm.sh/helm/v3/pkg/lint 1.212s  coverage: 100.0% of statements
ok      helm.sh/helm/v3/pkg/lint/rules   1.318s  coverage: 85.0% of statements
ok      helm.sh/helm/v3/pkg/lint/support 0.699s  coverage: 100.0% of statements
ok      helm.sh/helm/v3/pkg/plugin       1.343s  coverage: 87.5% of statements
?       helm.sh/helm/v3/pkg/plugin/cache [no test files]
ok      helm.sh/helm/v3/pkg/plugin/installer     8.803s  coverage: 80.6% of statements
ok      helm.sh/helm/v3/pkg/postrender   2.125s  coverage: 82.6% of statements
ok      helm.sh/helm/v3/pkg/provenance   1.472s  coverage: 71.7% of statements
?       helm.sh/helm/v3/pkg/release      [no test files]
ok      helm.sh/helm/v3/pkg/releaseutil  0.932s  coverage: 93.1% of statements
ok      helm.sh/helm/v3/pkg/repo 0.627s  coverage: 82.1% of statements
ok      helm.sh/helm/v3/pkg/repo/repotest        0.413s  coverage: 52.8% of statements
ok      helm.sh/helm/v3/pkg/storage      0.389s  coverage: 80.5% of statements
ok      helm.sh/helm/v3/pkg/storage/driver       0.489s  coverage: 75.5% of statements
ok      helm.sh/helm/v3/pkg/strvals      0.208s  coverage: 92.3% of statements
ok      helm.sh/helm/v3/pkg/time 0.210s  coverage: 38.5% of statements
```

Figure F.1: Helm's test coverage.