Tekton

Security Assessment

March 8, 2022

Prepared for:

Andrea Frittoli, Christie Wilson, Dibyo Mukherjee, Vincent Demeester

Linux Foundation

Prepared by: **Alex Useche and Shaun Mirani**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 80+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As such, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Table of Contents

Executive Summary

Engagement Overview

The Linux Foundation engaged Trail of Bits to review the security of its Tekton project. From February 22 to March 7, 2022, a team of two consultants conducted a security review of the client-provided source code, with four person-weeks of effort. Details of the project's timeline, test targets, and coverage are provided in subsequent sections of this report.

Project Scope

Our testing efforts were focused on the identification of flaws that could result in a compromise of confidentiality, integrity, or availability of the target system. We conducted this audit with access to the various Tekton code repositories and supporting documentation.

Summary of Findings

The audit uncovered one significant flaw that could impact system confidentiality, integrity, or availability. However, the majority of the findings are of lesser severity. A summary of the findings is provided below.

EXPOSURE ANALYSIS

Severity	*Count*
High	1
Medium	2
Low	4
Informational	6

CATEGORY BREAKDOWN

Category	*Count*
Timing	1
Data Validation	5
Denial of Service	3
Access Controls	1
Configuration	2
Documentation	1

Project Summary

Contact Information

The following managers were associated with this project:

Dan Guido, Account Manager dan@trailofbits.com	**Cara Pearson**, Project Manager cara.pearson@trailofbits.com

The following engineers were associated with this project:

Alex Useche, Senior Consultant alex.useche@trailofbits.com	**Shaun Mirani**, Consultant shaun.mirani@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
February 17, 2022	Pre-project kickoff call
February 28, 2022	Status update meeting #1
March 8, 2022	Delivery of report draft
March 18, 2022	Delivery of final report

Project Goals

The engagement was scoped to provide a security assessment of Tekton, with a focus on the Tekton Pipelines, Tekton Triggers, and Tekton Dashboard components. Specifically, we sought to answer the following non-exhaustive list of questions:

- Do the configurations provided for users generally follow best practices for security?
- Is there appropriate validation of file system operations such as handling symbolic links and setting file permissions?
- Are system secrets vulnerable to data exposure?
- Could an attacker perform log injection attacks against the application to trick operators into performing undesirable actions?
- Does the application properly handle errors?
- If the application is installed and configured based on official instructions, is it reasonably secure by default?
- Could attackers use malicious pipelines or triggers to perform container escape attacks and access the cluster?

Project Targets

The engagement involved a review and testing of the targets listed below.

Tekton Pipelines

Repository	https://github.com/tektoncd/pipelines/
Version	`99b8b196ea753af36befda8c0e0e1eaa9490ae68`
Type	Infrastructure
Platform	UNIX

Tekton Triggers

Repository	https://github.com/tektoncd/triggers/
Version	`99b8b196ea753af36befda8c0e0e1eaa9490ae68`
Type	Infrastructure
Platform	UNIX

Tekton Dashboard

Repository	https://github.com/tektoncd/dashboard/
Version	`bf3f51ac278d4ad49c7930a6abd8aeb0a3976440`
Type	Web application

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

- A review of controls protecting the system against denial of service revealed the use of insecure functions in a number of places in the codebase and a lack of rate-limiting controls (TOB-TKN-11).
- A review of secure-use concurrency revealed a minor issue related to the use of insecure functions for synchronization (TOB-TKN-1).
- An assessment of container security best practices revealed insecure network access controls between pods (TOB-TKN-9) and insufficient hardening of `TaskRun` containers (TOB-TKN-7).
- A review of the secret handling strategy did not reveal significant concerns.
- Investigations into the use of cryptography outside of TLS code paths did not reveal any issues.
- Fuzzing of the validation logic did not reveal any issues.
- A review of the project's adherence to web application security best practices uncovered a high-severity issue allowing the exfiltration of sensitive data from Tekton Dashboard (TOB-TKN-5).

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. During this project, we were unable to perform comprehensive testing of the following system elements, which may warrant further review:

- During the audit, we focused on the `pipelines` repository as requested by the Linux Foundation, and we reviewed the `triggers` and `dashboard` codebases in the last few days of the audit. No other repositories of the Tekton project were reviewed due to time limitations.
- The review of the `triggers` and `dashboard` repositories was less in-depth than the review of the `pipelines` repository. As a result, we did not review the JavaScript logic for Tekton Dashboard or the UI against concerns like cross-site scripting attacks.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	The use of time.After() in select statements can lead to memory leaks	Timing	Low
2	Risk of resource exhaustion due to the use of defer inside a loop	Denial of Service	Informational
3	Lack of access controls for Tekton Pipelines API	Access Controls	Informational
4	Insufficient validation of volumeMounts paths	Data Validation	Informational
5	Missing validation of Origin header in WebSocket upgrade requests	Data Validation	High
6	“Import resources" feature does not validate repository URL scheme	Data Validation	Informational
7	Insufficient security hardening of step containers	Configuration	Low
8	Tekton allows users to create privileged containers	Documentation	Medium
9	Insufficient default network access controls between pods	Configuration	Medium
10	“Import resources" feature does not validate repository path	Data Validation	Informational
11	Lack of rate-limiting controls	Denial of Service	Low
12	Lack of maximum request and response body constraint	Denial of Service	Informational

13	Nil dereferences in the trigger interceptor logic	Data Validation	**Low**

Detailed Findings

1. The use of time.After() in select statements can lead to memory leaks

Severity: **Low**	Difficulty: **High**
Type: Timing	Finding ID: TOB-TKN-1

Target:
- `pipeline/pkg/pipelinerunmetrics/metrics.go`
- `pipeline/pkg/taskrunmetrics/metrics.go`

Description

Calls to `time.After` in `for/select` statements can lead to memory leaks because the garbage collector does not clean up the underlying `Timer` object until the timer fires. A new timer, which requires resources, is initialized at each iteration of the `for` loop (and, hence, the `select` statement). As a result, many routines originating from the `time.After` call could lead to overconsumption of the memory.

```go
for {
        select {
        case <-ctx.Done():
                // When the context is cancelled, stop reporting.
                return

        case <-time.After(r.ReportingPeriod):
                // Every 30s surface a metric for the number of running pipelines.
                if err := r.RunningPipelineRuns(lister); err != nil {
                        logger.Warnf("Failed to log the metrics : %v", err)
                }
```

Figure 1.1: `tektoncd/pipeline/pkg/pipelinerunmetrics/metrics.go#L290-L300`

```go
for {
        select {
        case <-ctx.Done():
                // When the context is cancelled, stop reporting.
                return

        case <-time.After(r.ReportingPeriod):
                // Every 30s surface a metric for the number of running tasks.
                if err := r.RunningTaskRuns(lister); err != nil {
                        logger.Warnf("Failed to log the metrics : %v", err)
                }
        }
```

Figure 1.2: `pipeline/pkg/taskrunmetrics/metrics.go#L380-L391`

Exploit Scenario

An attacker finds a way to overuse a function, which leads to overconsumption of the memory and causes Tekton Pipelines to crash.

Recommendations

Short term, consider refactoring the code that uses the `time.After` function in `for/select` loops using tickers. This will prevent memory leaks and crashes caused by memory exhaustion.

Long term, ensure that the `time.After` method is not used in `for/select` routines. Periodically use the Semgrep query to check for and detect similar patterns.

References

- Use with caution time.After Can cause memory leak (golang)
- Golang <-time.After() is not garbage collected before expiry

2. Risk of resource exhaustion due to the use of defer inside a loop

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-TKN-2

Targets:

- `pipeline/pkg/git/git.go:294`
- `triggers/pkg/sink/sink.go:469`

Description

The `ExecuteInterceptors` function runs all interceptors configured for a given trigger inside a loop. The `res.Body.Close()` function is deferred at the end of the loop. Calling `defer` inside of a loop could cause resource exhaustion conditions because the deferred function is called when the function exits, not at the end of each loop. As a result, resources from each `interceptor` object are accumulated until the end of the `for` statement. While this may not cause noticeable issues in the current state of the application, it is best to call `res.Body.Close()` at the end of each loop to prevent unforeseen issues.

```
func (r Sink) ExecuteInterceptors(trInt []*triggersv1.TriggerInterceptor, in
*http.Request, event []byte, log *zap.SugaredLogger, eventID string, triggerID
string, namespace string, extensions map[string]interface{}) ([]byte, http.Header,
*triggersv1.InterceptorResponse, error) {
        if len(trInt) == 0 {
                return event, in.Header, nil, nil
        }

        // (...)
        for _, i := range trInt {
                if i.Webhook != nil { // Old style interceptor
                          // (...)
                        defer res.Body.Close()
```

Figure 2.1: `triggers/pkg/sink/sink.go#L428-L469`

Recommendations

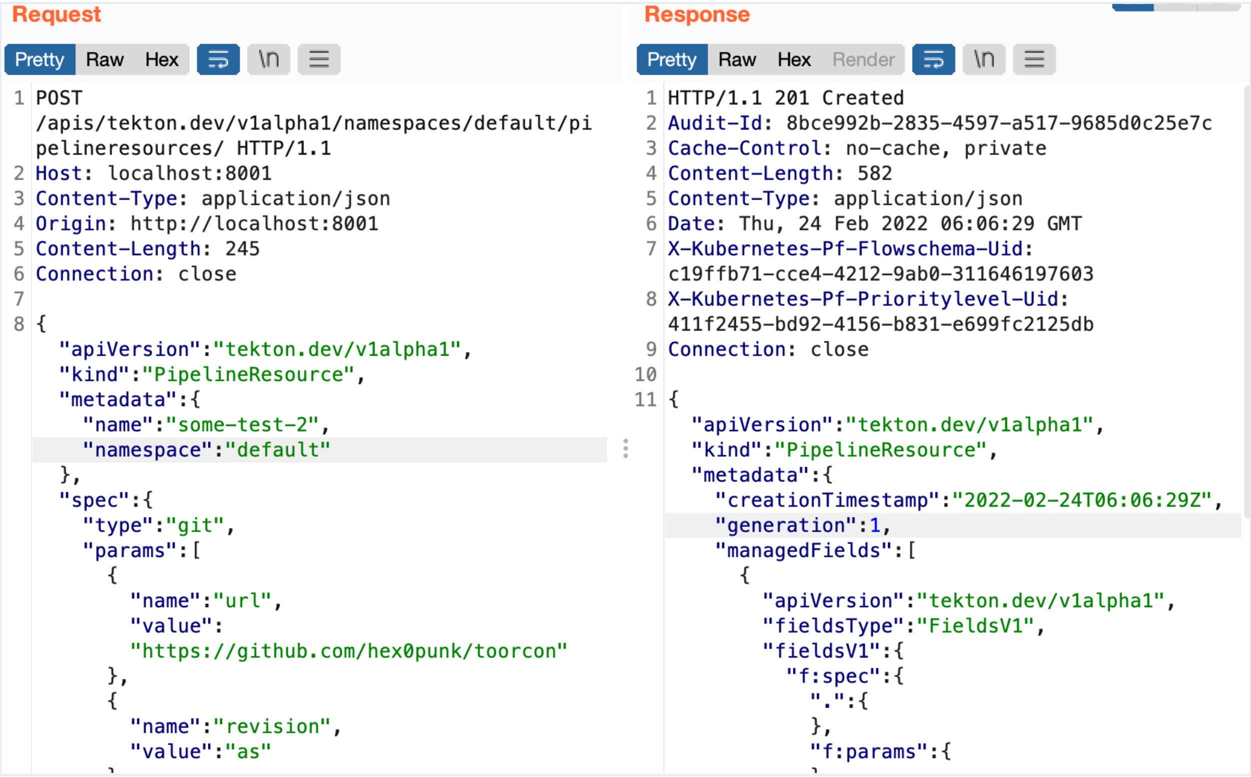
Short term, rather than deferring the call to `res.Body.Close()`, add a call to `res.Body.Close()` at the end of the loop.

3. Lack of access controls for Tekton Pipelines API

Severity: **Informational**	Difficulty: **Medium**
Type: Access Controls	Finding ID: TOB-TKN-3
Target: Pipelines API	

Description

The Tekton Pipelines extension uses an API to process requests for various tasks such as listing namespaces and creating `TaskRuns`. While Tekton provides documentation on enabling OAuth2 authentication, the API is unauthenticated by default. Should a Tekton operator expose the dashboard for other users to monitor their own deployments, every API method would be available to them, allowing them to perform tasks on namespaces that they do not have access to.



```
POST
/apis/tekton.dev/v1alpha1/namespaces/default/pi
pelineresources/ HTTP/1.1
Host: localhost:8001
Content-Type: application/json
Origin: http://localhost:8001
Content-Length: 245
Connection: close

{
  "apiVersion":"tekton.dev/v1alpha1",
  "kind":"PipelineResource",
  "metadata":{
    "name":"some-test-2",
    "namespace":"default"
  },
  "spec":{
    "type":"git",
    "params":[
      {
        "name":"url",
        "value":
        "https://github.com/hex0punk/toorcon"
      },
      {
        "name":"revision",
        "value":"as"
```

```
HTTP/1.1 201 Created
Audit-Id: 8bce992b-2835-4597-a517-9685d0c25e7c
Cache-Control: no-cache, private
Content-Length: 582
Content-Type: application/json
Date: Thu, 24 Feb 2022 06:06:29 GMT
X-Kubernetes-Pf-Flowschema-Uid:
c19ffb71-cce4-4212-9ab0-311646197603
X-Kubernetes-Pf-Prioritylevel-Uid:
411f2455-bd92-4156-b831-e699fc2125db
Connection: close

{
  "apiVersion":"tekton.dev/v1alpha1",
  "kind":"PipelineResource",
  "metadata":{
    "creationTimestamp":"2022-02-24T06:06:29Z",
    "generation":1,
    "managedFields":[
      {
        "apiVersion":"tekton.dev/v1alpha1",
        "fieldsType":"FieldsV1",
        "fieldsV1":{
          "f:spec":{
            ".":{
            },
            "f:params":{
```

Figure 3.1: Successful unauthenticated request

Exploit Scenario

An attacker discovers the endpoint exposing the Tekton Pipelines API and uses it to perform destructive tasks such as deleting `PipelineRuns`. Furthermore, the attacker can discover potentially sensitive information pertaining to deployments configured in Tekton.

Recommendations

Short term, add documentation on securing access to the API using Kubernetes security controls, including explicit documentation on the security implications of exposing access to the dashboard and, therefore, the API.

Long term, add an access control mechanism for controlling who can access the API and limiting access to namespaces as needed and/or possible.

4. Insufficient validation of volumeMounts paths

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-TKN-4
Target: Various	

Description

The Tekton Pipelines extension performs a number of validations against task steps whenever a task is submitted for Tekton to process. One such validation verifies that the path for a volume mount is not inside the `/tekton` directory. This directory is treated as a special directory by Tekton, as it is used for Tekton-specific functionality. However, the extension uses `strings.HasPrefix` to verify that `MountPath` does not contain the string "/tekton/" without first sanitizing it. As a result, it is possible to create volume mounts inside `/tekton` by using path traversal strings such as `/somedir/../tekton/newdir` in the `volumeMounts` variable of a task step definition.

```
for j, vm := range s.VolumeMounts {
        if strings.HasPrefix(vm.MountPath, "/tekton/") &&
                !strings.HasPrefix(vm.MountPath, "/tekton/home") {
                errs = errs.Also(apis.ErrGeneric(fmt.Sprintf("volumeMount cannot be
mounted under /tekton/ (volumeMount %q mounted at %q)", vm.Name, vm.MountPath),
"mountPath").ViaFieldIndex("volumeMounts", j))
        }
        if strings.HasPrefix(vm.Name, "tekton-internal-") {
                errs = errs.Also(apis.ErrGeneric(fmt.Sprintf(`volumeMount name %q
cannot start with "tekton-internal-"`, vm.Name),
"name").ViaFieldIndex("volumeMounts", j))
        }
}
```

Figure 4.1: `pipeline/pkg/apis/pipeline/v1beta1/task_validation.go#L218-L226`

The YAML file in the figure below was used to create a volume in the reserved `/tekton` directory.

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: vol-test
spec:
  taskSpec:
    steps:
    - image: docker
```

```
      name: client
      workingDir: /workspace
      script: |
          #!/usr/bin/env sh
          sleep 15m
      volumeMounts:
      - mountPath: /certs/client/../../../tekton/mytest
        name: empty-path
    volumes:
    - name: empty-path
      emptyDir: {}
```

Figure 4.2: Task run file used to create a volume mount inside an invalid location

The figure below demonstrates that the previous file successfully created the `mytest` directory inside of the `/tekton` directory by using a path traversal string.

```
$ kubectl exec -i -t vol-test -- /bin/sh
Defaulted container "step-client" out of: step-client, place-tools (init), step-init
(init), place-scripts (init)
/workspace # cd /tekton/
/tekton # ls
bin          creds        downward     home         mytest       results      run
scripts      steps        termination
```

Figure 4.3: Logging into the task pod container, we can now list the `mytest` directory inside of `/tekton`.

Recommendations

Short term, modify the code so that it converts the `mountPath` string into a file path and uses a function such as `filepath.Clean` to sanitize and canonicalize it before validating it.

5. Missing validation of Origin header in WebSocket upgrade requests

Severity: **High**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-TKN-5
Target: Dashboard, Kubernetes API	

Description

Tekton Dashboard uses the WebSocket protocol to provide real-time updates for `TaskRuns`, `PipelineRuns`, and other Tekton data. The endpoints responsible for upgrading the incoming HTTP request to a WebSocket request do not validate the `Origin` header to ensure that the request is coming from a trusted origin (i.e., the dashboard itself). As a result, arbitrary malicious web pages can connect to Tekton Dashboard and receive these real-time updates, which may include sensitive information, such as the log output of `TaskRuns` and `PipelineRuns`.

Exploit Scenario

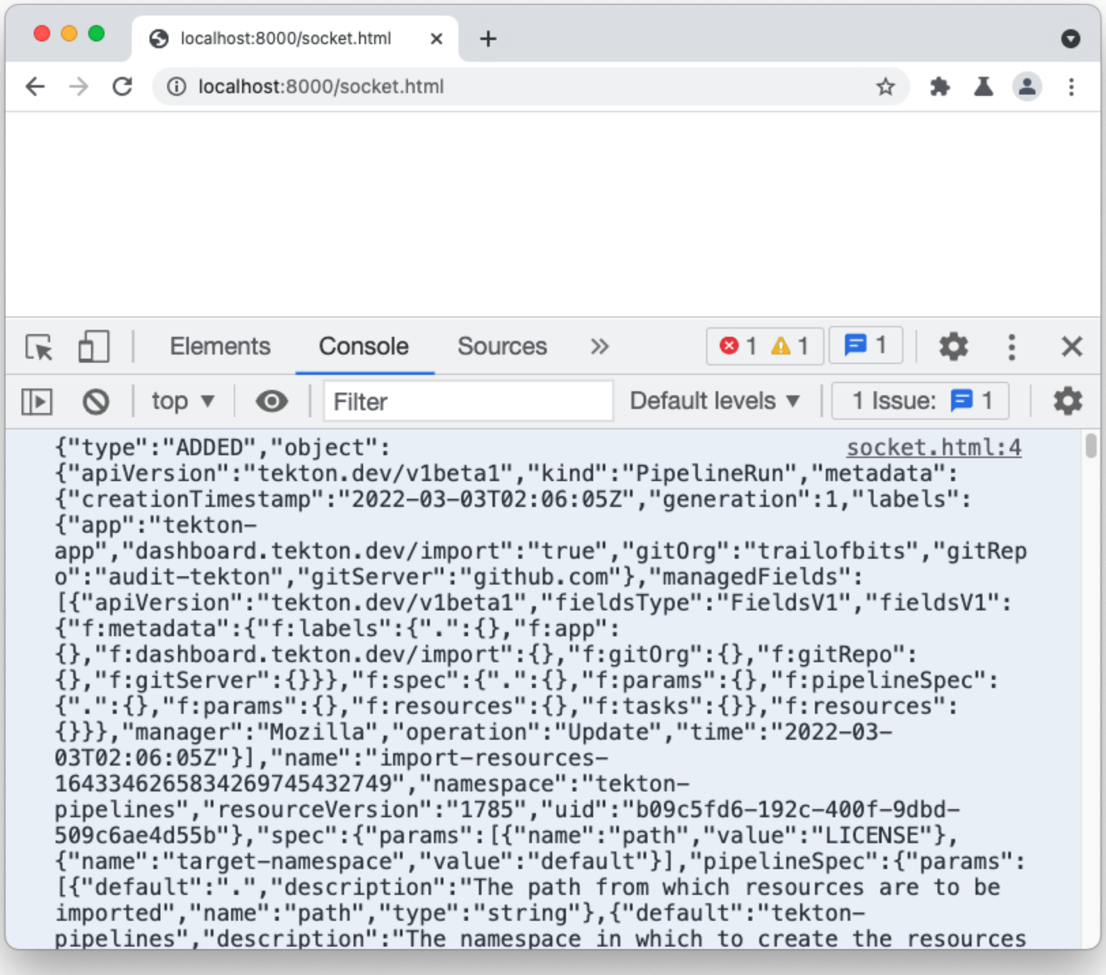
A user hosts Tekton Dashboard on a private address, such as one in a local area network or a virtual private network (VPN), without enabling application-layer authentication.

An attacker identifies the URL of the dashboard instance (e.g., http://192.168.3.130:9097) and hosts a web page with the following content:

```
<script>
var ws = new
WebSocket("ws://192.168.3.130:9097/apis/tekton.dev/v1beta1/namespaces/tekton-pipelines/pipelineruns/?watch=true&resourceVersion=1770");
ws.onmessage = function (event) {
  console.log(event.data);
}
</script>
```

Figure 5.1: A malicious web page that extracts Tekton Dashboard WebSocket updates

The attacker convinces the user to visit the web page. Upon loading it, the user’s browser successfully connects to the Tekton Dashboard WebSocket endpoint for monitoring `PipelineRuns` and logs received messages to the JavaScript console. As a result, the attacker’s untrusted web origin now has access to real-time updates from a dashboard instance on a private network that would otherwise be inaccessible outside of that network.



```
{"type":"ADDED","object":
{"apiVersion":"tekton.dev/v1beta1","kind":"PipelineRun","metadata":
{"creationTimestamp":"2022-03-03T02:06:05Z","generation":1,"labels":
{"app":"tekton-
app","dashboard.tekton.dev/import":"true","gitOrg":"trailofbits","gitRep
o":"audit-tekton","gitServer":"github.com"},"managedFields":
[{"apiVersion":"tekton.dev/v1beta1","fieldsType":"FieldsV1","fieldsV1":
{"f:metadata":{"f:labels":{".":{},"f:app":
{},"f:dashboard.tekton.dev/import":{},"f:gitOrg":{},"f:gitRepo":
{},"f:gitServer":{}}},"f:spec":{".":{},"f:params":{},"f:pipelineSpec":
{".":{},"f:params":{},"f:resources":{},"f:tasks":{}},"f:resources":
{}}},"manager":"Mozilla","operation":"Update","time":"2022-03-
03T02:06:05Z"}],"name":"import-resources-
1643346265834269745432749","namespace":"tekton-
pipelines","resourceVersion":"1785","uid":"b09c5fd6-192c-400f-9dbd-
509c6ae4d55b"},"spec":{"params":[{"name":"path","value":"LICENSE"},
{"name":"target-namespace","value":"default"}],"pipelineSpec":{"params":
[{"default":".","description":"The path from which resources are to be
imported","name":"path","type":"string"},{"default":"tekton-
pipelines","description":"The namespace in which to create the resources
```

Figure 5.2: The untrusted origin http://localhost:8080 has access to Tekton Dashboard WebSocket messages.

Recommendations

Short term, modify the code so that it verifies that the `Origin` header of WebSocket upgrade requests corresponds to the trusted origin on which Tekton Dashboard is served. For example, if the origin is not http://192.168.3.130:9097, Tekton Dashboard should reject the incoming request.

6. “Import resources” feature does not validate repository URL scheme

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-TKN-6
Target: Dashboard	

Description

Tekton Dashboard’s “import resources” feature relies on client-side checks to ensure that the repository URL adheres to the correct format. As the feature does not implement server-side validation, a malicious user can enter URLs with unintended schemes, such as `file://`, by sending a request directly to the Tekton Dashboard API:

```
POST /apis/tekton.dev/v1beta1/namespaces/tekton-pipelines/pipelineruns/ HTTP/1.1
Host: 192.168.3.130:9097
Content-Length: 1570
Accept: application/json
Tekton-Client: tektoncd/dashboard
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/95.0.4638.54 Safari/537.36
Content-Type: application/json
Origin: http://192.168.3.130:9097
Referer: http://192.168.3.130:9097/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

{"apiVersion":"tekton.dev/v1beta1","kind":"PipelineRun","metadata":{"name":"import-r
esources-1234","labels":{"gitServer":"github.com","gitOrg":"trailofbits","gitRepo":"
audit-tekton","app":"tekton-app","dashboard.tekton.dev/import":"true"}},"spec":{"pip
elineSpec":{"resources":[{"name":"git-source","type":"git"}],"params":[{"name":"path
","description":"The path from which resources are to be
imported","default":".","type":"string"},{"name":"target-namespace","description":"T
he namespace in which to create the resources being
imported","default":"tekton-pipelines","type":"string"}],"tasks":[{"name":"import-re
sources","taskSpec":{"resources":{"inputs":[{"name":"git-source","type":"git"}]},"pa
rams":[{"name":"path","description":"The path from which resources are to be
imported","default":".","type":"string"},{"name":"target-namespace","description":"T
he namespace in which to create the resources being
imported","default":"tekton-pipelines","type":"string"}],"steps":[{"name":"import","
image":"lachlanevenson/k8s-kubectl:latest","command":["kubectl"],"args":["apply","-f
","$(resources.inputs.git-source.path)/$(params.path)","-n","$(params.target-namespa
ce)"]}]},"params":[{"name":"path","value":"$(params.path)"},{"name":"target-namespac
e","value":"$(params.target-namespace)"}],"resources":{"inputs":[{"name":"git-source
","resource":"git-source"}]}}]},"resources":[{"name":"git-source","resourceSpec":{"t
ype":"git","params":[{"name":"url","value":"file:///etc/hostname"}]}}],"params":[{"n
ame":"path","value":""},{"name":"target-namespace","value":"default"}]}}
```

Figure 6.1: Request to import a repository from the local file system

The output from the associated `PipelineRun` shows that the system tried to import `/etc/hostname` and failed:

```
{"level":"error","ts":1646703297.3208265,"caller":"git/git.go:55","msg":"Error
running git [fetch --recurse-submodules=yes --depth=1 origin --update-head-ok
--force ]: exit status 128\nfatal: invalid gitfile format: /etc/hostname\nfatal:
Could not read from remote repository.\n\nPlease make sure you have the correct
access rights\nand the repository
exists.\n","stacktrace":"github.com/tektoncd/pipeline/pkg/git.run\n\tgithub.com/tekt
oncd/pipeline/pkg/git/git.go:55\ngithub.com/tektoncd/pipeline/pkg/git.Fetch\n\tgithu
b.com/tektoncd/pipeline/pkg/git/git.go:150\nmain.main\n\tgithub.com/tektoncd/pipelin
e/cmd/git-init/main.go:53\nruntime.main\n\truntime/proc.go:225"}

{"level":"fatal","ts":1646703297.3209455,"caller":"git-init/main.go:54","msg":"Error
fetching git repository: failed to fetch []: exit status
128","stacktrace":"main.main\n\tgithub.com/tektoncd/pipeline/cmd/git-init/main.go:54
\nruntime.main\n\truntime/proc.go:225"}
```

Figure 6.1: `PipelineRun` logs showing a failed import from the local file system

Recommendations

Short term, modify the code so that it verifies that the repository URL uses the `https://` scheme.

7. Insufficient security hardening of step containers	
Severity: **Low**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-TKN-7
Target: Pipelines	

Description

Containers used for running task and pipeline steps have excessive security context options enabled. This increases the attack surface of the system, and issues such as Linux kernel bugs may allow attackers to escape a container if they gain code execution within a Tekton container.

The figure below shows the security properties of a task container with the `docker` driver.

```
# cat /proc/self/status | egrep 'Name|Uid|Gid|Groups|Cap|NoNewPrivs|Seccomp'
Name:   cat
Uid:    0       0       0       0
Gid:    0       0       0       0
Groups:
CapInh: 00000000a80425fb
CapPrm: 00000000a80425fb
CapEff: 00000000a80425fb
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
NoNewPrivs:     0
Seccomp:        0
Seccomp_filters:        0
```

Figure 7.1: The security properties of one of the step containers

Exploit Scenario

Eve finds a bug that allows her to run arbitrary code on behalf of a confined process within a container, using it to gain more privileges in the container and then to attack the host.

Recommendations

Short term, drop default capabilities from containers and prevent processes from gaining additional privileges by setting the `--cap-drop=ALL` and `--security-opt=no-new-privileges:true` flags when starting containers.

Long term, review and implement the Kubernetes security recommendations in appendix C.

8. Tekton allows users to create privileged containers

Severity: **Medium**	Difficulty: **Medium**
Type: Documentation	Finding ID: TOB-TKN-8
Target: Pipelines	

Description

Tekton allows users to define `task` and `sidecar` objects with a privileged security context, which effectively grants task containers all capabilities. Tekton operators can use admission controllers to disallow users from using this option. However, information on this mitigation in the guidance documents for Tekton Pipelines is insufficient and should be made clear.

If an attacker gains code execution on any of these containers, the attacker could break out of it and gain full access to the host machine. We were not able to escape step containers running in privileged mode during the time allotted for this audit.

```yaml
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-secret-10
spec:
  serviceAccountName: build-bot
  taskSpec:
    steps:
    - name: secret
      securityContext:
        privileged: true
      image: ubuntu
      script: |
        #!/usr/bin/env bash
        sleep 20m
```

Figure 8.1: TaskRun definition with the privileged security context

```
root@build-push-secret-10-pod:/proc/fs# find -type f -maxdepth 5 -writable
find: warning: you have specified the global option -maxdepth after the argument
-type, but global options are not positional, i.e., -maxdepth affects tests
specified before it as well as those specified after it.  Please specify global
options before other arguments.
```

```
./xfs/xqm
./xfs/xqmstat
./cifs/Stats
./cifs/cifsFYI
./cifs/dfscache
./cifs/traceSMB
./cifs/DebugData
./cifs/open_files
./cifs/SecurityFlags
./cifs/LookupCacheEnabled
./cifs/LinuxExtensionsEnabled
./ext4/vda1/fc_info
./ext4/vda1/options
./ext4/vda1/mb_groups
./ext4/vda1/es_shrinker_info
./jbd2/vda1-8/info
./fscache/stats
```

Figure 8.2: With the privileged security context in figure 8.1, it is now possible to write to several files in `/proc/fs`, for example.

Exploit Scenario

A malicious developer runs a TaskRun with a privileged security context and obtains shell access to the container. Using one of various known exploits, he breaks out of the container and gains root access on the host.

Recommendations

Short term, create clear, easy-to-locate documentation warning operators about allowing developers and other users to define a privileged security context for step containers, and include guidance on how to restrict such a feature.

9. Insufficient default network access controls between pods

Severity: **Medium**	Difficulty: **High**
Type: Configuration	Finding ID: TOB-TKN-9
Target: Pipelines	

Description

By default, containers deployed as part of task steps do not have any egress or ingress network restrictions. As a result, containers could reach services exposed over the network from any task step container. For instance, in figure 9.2, a user logs into a container running a task step in the `developer-group` namespace and successfully makes a request to a service in a step container in the `qa-group` namespace.

```
root@build-push-secret-35-pod:/# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.17.0.17  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:ac:11:00:11  txqueuelen 0  (Ethernet)
        RX packets 21831  bytes 32563599 (32.5 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6465  bytes 362926 (362.9 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@build-push-secret-35-pod:/# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
172.17.0.16 - - [08/Mar/2022 01:03:50] "GET
/tekton/creds-secrets/basic-user-pass-canary/password HTTP/1.1" 200 -
172.17.0.16 - - [08/Mar/2022 01:04:05] "GET
/tekton/creds-secrets/basic-user-pass-canary/password HTTP/1.1" 200 -
```

Figure 9.1: Exposing a simple server in a step container in the `developer-group` namespace

```
root@build-push-secret-35-pod:/# curl
```

```
172.17.0.17:8000/tekton/creds-secrets/basic-user-pass-canary/password
mySUPERsecretPassword
```

Figure 9.2: Reaching the service exposed in figure 9.1 from another container in the `qa-group` namespace

Exploit Scenario

An attacker launches a malicious task container that reaches a service exposed via a sidecar container and performs unauthorized actions against the service.

Recommendations

Short term, enforce ingress and egress restrictions to allow only resources that need to speak to each other to do so. Leverage allowlists instead of denylists to ensure that only expected components can establish these connections.

Long term, ensure the use of appropriate methods of isolation to prevent lateral movement.

10. “Import resources" feature does not validate repository path

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-TKN-10
Target: Dashboard	

Description

When importing a resource in Tekton Dashboard, a malicious user can specify a `path` value such as `../../../../../passwd` to traverse outside of the cloned repository and cause the system to access unintended files.

While we did not find a way to exploit this issue to read arbitrary files, we observed that, for certain files containing invalid YAML document separators, partial file contents were output in the `PipelineRun` logs. For instance, attempting to access `../../../../../bin/cat` resulted in the following error:

```
error: error parsing /workspace/git-source/../../../../../bin/cat: invalid Yaml
document separator: %s ping statistics ---
```

Figure 10.1: Logs revealing partial contents of `/bin/cat`

Recommendations

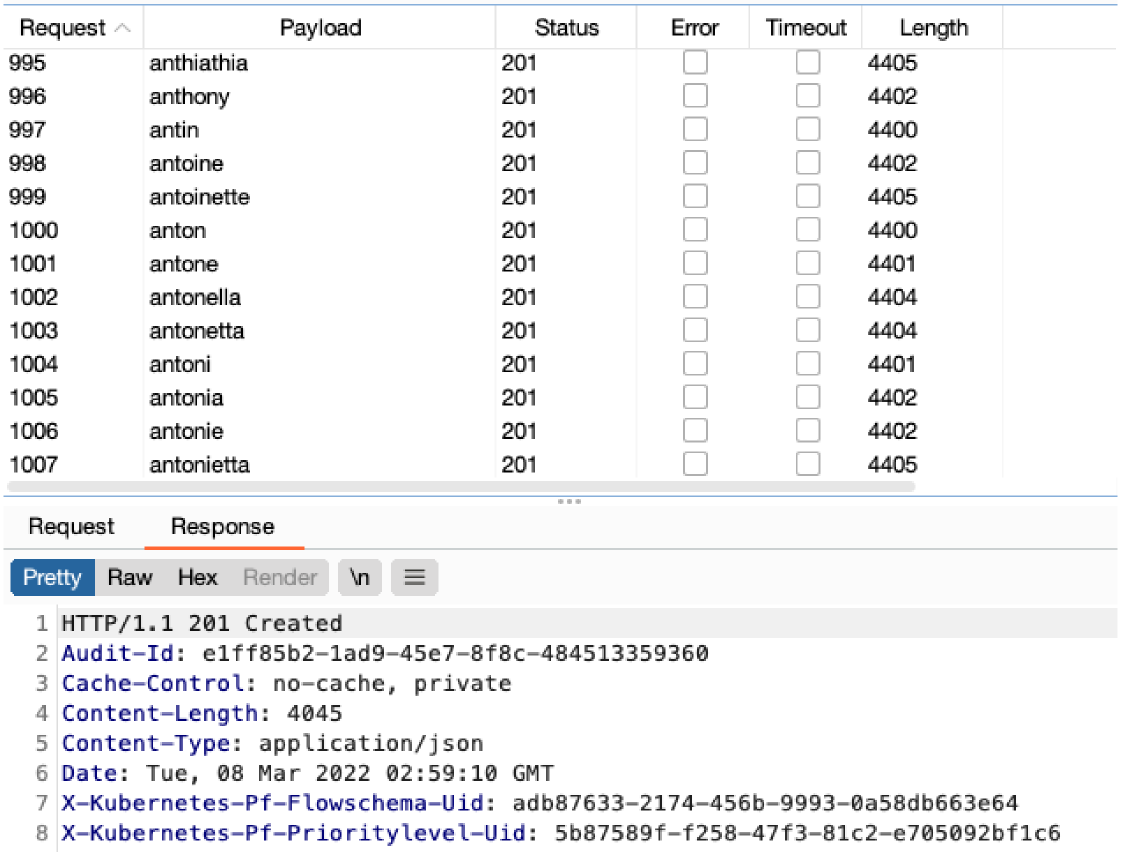
Short term, add a check to verify that the repository path does not point to locations outside of `/workspace/git-source`.

11. Lack of rate-limiting controls

Severity: **Low**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-TKN-11
Target: Dashboard	

Description

Tekton Dashboard does not enforce rate limiting of HTTP requests. As a result, we were able to issue over a thousand requests in just over a minute.



Request	Payload	Status	Error	Timeout	Length
995	anthiathia	201	□	□	4405
996	anthony	201	□	□	4402
997	antin	201	□	□	4400
998	antoine	201	□	□	4402
999	antoinette	201	□	□	4405
1000	anton	201	□	□	4400
1001	antone	201	□	□	4401
1002	antonella	201	□	□	4404
1003	antonetta	201	□	□	4404
1004	antoni	201	□	□	4401
1005	antonia	201	□	□	4402
1006	antonie	201	□	□	4402
1007	antonietta	201	□	□	4405

```
HTTP/1.1 201 Created
Audit-Id: e1ff85b2-1ad9-45e7-8f8c-484513359360
Cache-Control: no-cache, private
Content-Length: 4045
Content-Type: application/json
Date: Tue, 08 Mar 2022 02:59:10 GMT
X-Kubernetes-Pf-Flowschema-Uid: adb87633-2174-456b-9993-0a58db663e64
X-Kubernetes-Pf-Prioritylevel-Uid: 5b87589f-f258-47f3-81c2-e705092bf1c6
```

Figure 11.1: We sent over a thousand requests to Tekton Dashboard without being rate limited.

Processing requests sent at such a high rate can consume an inordinate amount of resources, increasing the risk of denial-of-service attacks through excessive resource consumption. In particular, we were able to create hundreds of running “import resources” pods that were able to consume nearly all the host’s memory in the span of a minute.

Exploit Scenario

An attacker floods a Tekton Dashboard instance with HTTP requests that execute pipelines, leading to a denial-of-service condition.

Recommendations

Short term, implement rate limiting on all API endpoints.

Long term, run stress tests to ensure that the rate limiting enforced by Tekton Dashboard is robust.

12. Lack of maximum request and response body constraint

Severity: **Informational**	Difficulty: **High**
Type: Denial of Service	Finding ID: TOB-TKN-12
Target: Various APIs	

Description

The `ioutil.ReadAll` function reads from source until an error or an end-of-file (EOF) condition occurs, at which point it returns the data that it read. This function is used in different files of the Tekton Triggers and Tekton Pipelines codebases to read requests and responses. There is no limit on the maximum size of request and response bodies, so using `ioutil.ReadAll` to parse requests and responses could cause a denial of service (due to insufficient memory). A denial of service could also occur if an exhaustive resource is loaded multiple times. This method is used in the following locations of the codebase:

File	Project
`pkg/remote/oci/resolver.go:L211`	Pipelines
`pkg/sink/sink.go:147,465`	Triggers
`pkg/interceptors/webhook/webhook.go:77`	Triggers
`pkg/interceptors/interceptors.go:176`	Triggers
`pkg/sink/validate_payload.go:29`	Triggers
`cmd/binding-eval/cmd/root.go:141`	Triggers
`cmd/triggerrun/cmd/root.go:182`	Triggers

Recommendations

Short term, place a limit on the maximum size of request and response bodies. For example, this limit can be implemented by using the `io.LimitReader` function.

Long term, place limits on request and response bodies globally in other places within the application to prevent denial-of-service attacks.

13. Nil dereferences in the trigger interceptor logic

Severity: **Low**	Difficulty: **Medium**
Type: Data Validation	Finding ID: TOB-TKN-13

Target:

- `triggers/pkg/interceptors/github/github.go:85`
- `triggers/pkg/interceptors/bitbucket/bitbucket.go:79`
- `triggers/pkg/interceptors/gitlab/gitlab.go:78`
- `triggers/pkg/interceptors/cel/cel.go:128`

Description

The `Process` functions, which are responsible for executing the various triggers for the `git`, `gitlab`, `bitbucket`, and `cel` interceptors, do not properly validate request objects, leading to `nil` dereference panics when requests are submitted without a `Context` object.

```go
func (w *Interceptor) Process(ctx context.Context, r *triggersv1.InterceptorRequest)
*triggersv1.InterceptorResponse {
	headers := interceptors.Canonical(r.Header)
	// (...)
	// Next validate secrets
	if p.SecretRef != nil {
		// Check the secret to see if it is empty
		if p.SecretRef.SecretKey == "" {
			return interceptors.Fail(codes.FailedPrecondition, "github
interceptor secretRef.secretKey is empty")
		}
		// (...)
		ns, _ := triggersv1.ParseTriggerID(r.Context.TriggerID)
```

Figure 13.1: ***`triggers/pkg/interceptors/github/github.go#L48-L85`***

We tested the panic by forwarding the Tekton Triggers webhook server to localhost and sending HTTP requests to the GitHub endpoint. The Go HTTP server recovers from the panic.

```
curl -i -s -k -X $'POST' \
    -H $'Host: 127.0.0.1:1934' -H $'Content-Length: 178' \
    --data-binary
$'{\x0d\x0a\"header\":{\x0d\x0a\"X-Hub-Signature\":[\x0d\x0a\x09\"sig\"\x0d\x0a],\x0
d\x0a\"X-GitHub-Event\":[\x0d\x0a\"evil\"\x0d\x0a]\x0d\x0a},\x0d\x0a\"interceptor_pa
rams\": {\x0d\x0a\x09\"secretRef\":
{\x0d\x0a\x09\x09\"secretKey\":\"key\",\x0d\x0a\x09\x09\"secretName\":\"name\"\x0d\x
0a\x09}\x0d\x0a}\x0d\x0a}' \
    $'http://127.0.0.1:1934/github'
```

Figure 13.2: The `curl` request that causes a panic

```
2022/03/08 05:34:13 http: panic serving 127.0.0.1:49304: runtime error: invalid
memory address or nil pointer dereference
goroutine 33372 [running]:
net/http.(*conn).serve.func1(0xc0001bf0e0)
	net/http/server.go:1824 +0x153
panic(0x1c25340, 0x30d6060)
	runtime/panic.go:971 +0x499
github.com/tektoncd/triggers/pkg/interceptors/github.(*Interceptor).Process(0xc00000
d248, 0x216fec8, 0xc0003d5020, 0xc0002b7b60, 0xc0000a7978)
	github.com/tektoncd/triggers/pkg/interceptors/github/github.go:85 +0x1f5
github.com/tektoncd/triggers/pkg/interceptors/server.(*Server).ExecuteInterceptor(0x
c000491490, 0xc000280200, 0x0, 0x0, 0x0, 0x0, 0x0)
	github.com/tektoncd/triggers/pkg/interceptors/server/server.go:128 +0x5df
github.com/tektoncd/triggers/pkg/interceptors/server.(*Server).ServeHTTP(0xc00049149
0, 0x2166dc0, 0xc0000d42a0, 0xc000280200)
	github.com/tektoncd/triggers/pkg/interceptors/server/server.go:57 +0x4d
net/http.(*ServeMux).ServeHTTP(0xc00042d000, 0x2166dc0, 0xc0000d42a0, 0xc000280200)
	net/http/server.go:2448 +0x1ad
net/http.serverHandler.ServeHTTP(0xc0000d4000, 0x2166dc0, 0xc0000d42a0,
0xc000280200)
	net/http/server.go:2887 +0xa3
net/http.(*conn).serve(0xc0001bf0e0, 0x216ff00, 0xc00042d200)
	net/http/server.go:1952 +0x8cd
created by net/http.(*Server).Serve
	net/http/server.go:3013 +0x39b
```

Figure 13.3: Panic trace

Exploit Scenario

As the codebase continues to grow, a new mechanism is added to call one of the `Process` functions without relying on HTTP requests (for instance, via a custom RPC client implementation). An attacker uses this mechanism to create a new interceptor. He calls the `Process` function with an invalid object, causing a panic that crashes the Tekton Triggers webhook server.

Recommendations

Short term, add checks to verify that request `Context` objects are not `nil` before dereferencing them.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	Description
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Running GCatch

This appendix explains how to use GCatch, a tool that automatically detects concurrency bugs in Go. It also includes relevant output generated by GCatch when it is run over Tekton (figure B.1). We omitted from the figure any output pertaining to packages in which no issues were detected and to packages that did not compile. Additionally, we replaced the prefix of the package paths (`$TKNPIPELINES`) with "`$TKNPIPELINES`" in the figure.

To run GCatch over the Tekton project, take the following steps:

1. Clone the GCatch project as a Go package. For example, if your Go root directory were `~/go`, you would clone the repository to the following package: `~/go/src/github.com/system-pclub/GCatch`.

2. Go to the `GCatch/GCatch` directory and run `Installz3.sh` and `install.sh`.

3. Install the project in the Go root directory and enter the project directory (`~/go/src/github.com/tekton/pipelines`).

4. Run GCatch by using the following command:

   `GCatch -path="$(pwd)" -include=github.com/tektoncd/$REPO -checker=BMOC:unlock:double:conflict:structfield:fatal -r -compile-error`.

```
----------Bug[1]----------
       Type: Double Lock   Reason: A Mutex/RWMutex is locked twice. (Note: even
double RWMutex.RLock() can produce deadlock bug)

Call Chain (with FN Pointer):
CloudEvents (at $TKNPIPELINES/pkg/taskrunmetrics/metrics.go: 473) -> Record (at
$TKNPIPELINES/vendor/knative.dev/pkg/metrics/record.go: 30) -> record (at
$TKNPIPELINES/vendor/knative.dev/pkg/metrics/config.go: 116) -> optionForResource
(at $TKNPIPELINES/vendor/knative.dev/pkg/metrics/resource_view.go: 288) -> Do (at
/usr/local/go/src/sync/once.go: 59) -> doSlow (at /usr/local/go/src/sync/once.go:
68) -> NewRecorder$1 (at $TKNPIPELINES/pkg/taskrunmetrics/metrics.go: 122) ->
viewRegister
       Location of the 2 lock operations:
       File: $TKNPIPELINES/pkg/taskrunmetrics/metrics.go:433
       File: $TKNPIPELINES/pkg/taskrunmetrics/metrics.go:133
----------Bug[2]----------
       Type: Double Lock   Reason: A Mutex/RWMutex is locked twice. (Note: even
double RWMutex.RLock() can produce deadlock bug)
```

```
Call Chain (with FN Pointer):
DurationAndCount (at $TKNPIPELINES/pkg/taskrunmetrics/metrics.go: 324) -> Record (at
$TKNPIPELINES/vendor/knative.dev/pkg/metrics/record.go: 30) -> record (at
$TKNPIPELINES/vendor/knative.dev/pkg/metrics/config.go: 116) -> optionForResource
(at $TKNPIPELINES/vendor/knative.dev/pkg/metrics/resource_view.go: 288) -> Do (at
/usr/local/go/src/sync/once.go: 59) -> doSlow (at /usr/local/go/src/sync/once.go:
68) -> NewRecorder$1 (at $TKNPIPELINES/pkg/taskrunmetrics/metrics.go: 122) ->
viewRegister
       Location of the 2 lock operations:
       File: $TKNPIPELINES/pkg/taskrunmetrics/metrics.go:293
       File: $TKNPIPELINES/pkg/taskrunmetrics/metrics.go:133
----------Bug[3]----------
       Type: Double Lock   Reason: A Mutex/RWMutex is locked twice. (Note: even
double RWMutex.RLock() can produce deadlock bug)

Call Chain (with FN Pointer):
RecordPodLatency (at $TKNPIPELINES/pkg/taskrunmetrics/metrics.go: 425) -> Record (at
$TKNPIPELINES/vendor/knative.dev/pkg/metrics/record.go: 30) -> record (at
$TKNPIPELINES/vendor/knative.dev/pkg/metrics/config.go: 116) -> optionForResource
(at $TKNPIPELINES/vendor/knative.dev/pkg/metrics/resource_view.go: 288) -> Do (at
/usr/local/go/src/sync/once.go: 59) -> doSlow (at /usr/local/go/src/sync/once.go:
68) -> NewRecorder$1 (at $TKNPIPELINES/pkg/taskrunmetrics/metrics.go: 122) ->
viewRegister
       Location of the 2 lock operations:
       File: $TKNPIPELINES/pkg/taskrunmetrics/metrics.go:398
       File: $TKNPIPELINES/pkg/taskrunmetrics/metrics.go:133
```

Figure B.1: GCatch results for Tekton Pipelines

C. Hardening Containers Run via Kubernetes

This appendix provides context for the hardening of containers spawned by Kubernetes. Please note our definitions of the following terms:

- Container: This is the isolated "environment" created by Linux features such as namespaces, cgroups, Linux capabilities, and AppArmor and secure computing (seccomp) profiles. We are specifically concerned with Docker containers since the tested environment uses Docker as its container engine.
- Host: This is the unconfined environment on the machine running a container (e.g., a process run in global Linux namespaces).

Root Inside Container

User namespaces allow for the remapping of user and group IDs between a host and a container; unless namespaces are used, the root user inside the container will be the root user in the host. In a default configuration of Docker containers, the container features limit the actions that the root user can take. However, if a process does not need to be run as root, it is best to run it with another user.

To run a container with another user, use the USER Dockerfile instructions. In Kubernetes, one can specify the user ID (UID) and various group IDs (GIDs) (e.g., a primary GID, a file system–related GID, and those for supplemental groups) using the `runAsUser`, `runAsGroup`, `fsGroup`, and `supplementalGroups` attributes of a `securityContext` field of a pod or other objects used to spawn containers.

Dropping Linux Capabilities

Linux capabilities split the privileged actions that a root user's process can perform. Docker drops most Linux capabilities for security purposes but leaves others enabled for convenience. We recommend dropping all Linux capabilities and then enabling only those necessary for the application to function properly.

Linux capabilities can be dropped in Docker via the `--cap-drop=all` flag and in Kubernetes by specifying `capabilities`, `drop`, and `--all` in the `securityContext` key of the deployment's container configuration. Then, to restore necessary capabilities, use the `--cap-add=<cap>` flag in a `docker run` or specify them in `capabilities`, and use `add` in the `securityContext` field in the Kubernetes object manifest.

NoNewPrivs Flag

The `NoNewPrivs` flag prevents additional privileges for a process or its children from being assigned. For example, it prevents a UID/GID from gaining capabilities or privileges by executing `setuid` binaries.

The `NoNewPrivs` flag can be enabled in a `docker run` via the `--security-opt=no-new-privileges` flag. In a Kubernetes deployment, specify `allowPrivilegeEscalation: false` in the `securityContext` field to enable it.

Seccomp Policies

A seccomp policy limits the available system calls and their arguments. Normally, using seccomp requires a call to a `prctl` syscall with a special structure, but Docker simplifies the process and allows a seccomp policy to be specified as a JSON file. Using the default Docker profile is a good start for implementing a specific policy. Seccomp is disabled by default in Kubernetes.

The seccomp policy can be specified with a `--security-opt seccomp=<filepath>` flag in Docker. In Kubernetes, the seccomp policy can be set either by using a `seccompProfile` key in the `securityContext` field of a pod (in Kubernetes v1.19 or later) or by using the `container.seccomp.security.alpha.kubernetes.io/<container_name>:<profile_ref>` annotation (in pre-v1.19 versions). The Kubernetes documentation includes examples of both methods of setting a specific seccomp policy.

Linux Security Module (AppArmor)

The Linux Security Module (LSM) is a mechanism that allows kernel developers to hook various kernel calls. AppArmor is an LSM used by default in Docker. Another popular LSM is SELinux, but since it is more difficult to set up, it is not discussed here.

AppArmor limits what a process can do and which resources a process can interact with. Docker uses its default AppArmor profile, which is generated from this template. When Docker is used as a container engine in Kubernetes, the same profile is often used by default, depending on the Kubernetes cluster configuration. One can override the AppArmor profile in Kubernetes with the following annotation (which is further described here):

```
container.apparmor.security.beta.kubernetes.io/<container_name>:
<profile_ref>
```