Security Auditing Report

Teleport - Features Testing Q3 2021

Advanced Access Workflows
Slack, Mattermost Plugins
Terraform Provider

Prepared for: Gravitational, Inc. DBA Teleport
Prepared by: Lorenzo Stella
Date: July 23rd, 2021

Table of Contents

Revision History

Version	Date	Description	Author
1	07/23/2021	First release of the final report	Lorenzo Stella
2	07/23/2021	Peer Review	Ben Caller
3	07/23/2021	Peer Review	Luca Carettoni
4	07/15/2022	Retesting	Lorenzo Stella

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 on some of the recent Teleport platform features. Teleport is a cloud-native SSH gateway for managing access to clusters of Linux servers via SSH or Kubernetes APIs.

The project commenced on 07/05/2021 and ended on 07/23/2021 requiring two (2) security researchers, for a total of fifteen (15) person/days. The project resulted in twelve (12) findings of which one (1) was rated with high severity.

In July 2022, Doyensec performed a retesting of the Teleport Features in scope and confirmed the effectiveness of the applied mitigations. **All issues with direct security impact have been addressed by Gravitational**.

The project consisted of a manual web application security assessment, source code review, and dynamic instrumentation of the target. Testing focused exclusively on the features listed in scope below.

Testing was conducted remotely from Doyensec EMEA and US offices.

Scope

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

- Identify misconfigurations and vulnerabilities in the new Teleport's Advanced Access Workflows feature and Slack+Mattermost plugins
- Identify misconfigurations in the Terraform provider
- Evaluate the overall security posture and feature design

Based on the information provided and through conversation with the Teleport team, Doyensec included in the scope the following components:

- Teleport Advanced Access Workflows
  - github.com/gravitational/teleport#5441
- Predicate Logic Library used by AAW
  - github.com/vulcand/predicate
- Teleport Plugins (Slack, Mattermost)
  - github.com/gravitational/teleport-plugins
- Teleport Terraform provider
  - github.com/gravitational/teleport-plugins/pull/197

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

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

- https://github.com/gravitational/teleport
  f475425bb88054e7f3944433236720fbd6b87e24
- https://github.com/vulcand/predicate
  8fbfb3ab0e94276b6b58bec378600829adc7a203
- https://github.com/gravitational/teleport-plugins
  b671c3dc3c458bc4ba1d553a3d96c475fab77cbf

Scoping Restrictions

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

Testing focused on Teleport's new Advanced Access Workflows feature and its Slack and Mattermost plugins. Additionally, Teleport included the "Terraform provider" in the scope for the engagement.

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 selected Teleport features.

While most of the issues are departures from best practices and low-severity flaws, Doyensec identified one (1) high severity and two (2) medium 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.

Major findings included an injection flaw that allows terminal escape sequences and new lines to be injected within approval request messages. The ability to use Markdown in approval messages was also reported as a potential area of concern since it can be easily abused to setup social engineering traps. A Denial of Service (DoS) vulnerability was discovered in the way HTTP request bodies are parsed by approval bots.

Considering the overall complexity of the targeted features, the security posture was found to be in line with industry best practices.

It is important to note that Doyensec performed a design review on all features included for testing, prior to their implementation.

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

- At the time of writing, Teleport's custom approval feature has been designed and implemented accepting the following risks. Future releases could attempt to mitigate such minor intrinsic risks:
  - Temporary, approved roles are also considered when evaluating the approval conditions for other roles
  - If the attacker has view permissions on the access requests, it is possible to leak the roles of the users approving a request
  - The request reason and reviewer traits are available at the same time inside of the threshold filter context, which allows the definition of predicates having arbitrary user-input, potentially leading to the disclosure of traits (e.g. declaring a predicate having `contains(reviewer.traits["foo"], request.reason)`)

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

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

Additionally, technical design documents were also provided to Doyensec.

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
- 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.

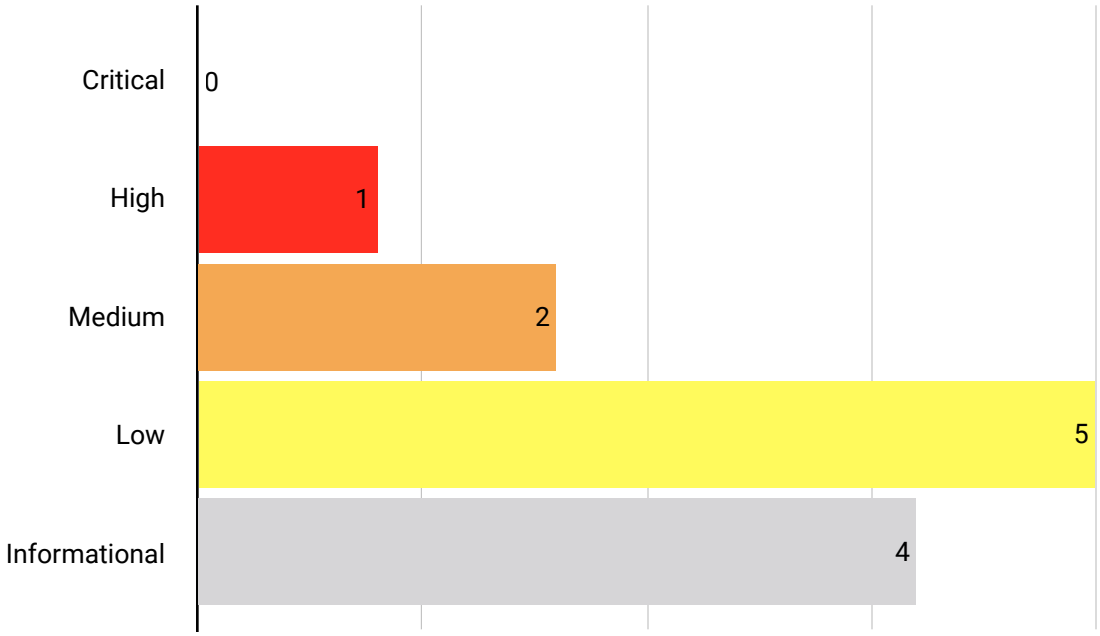
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
TEL-Q321-1	Injectable Markdown Syntax In Request Reason	Injection Flaws (SQL, XML, Command, Path, etc)	Medium	Closed
TEL-Q321-2	Email Validation Regular Expression Prone To Abuses	Insecure Design	Informational	Closed
TEL-Q321-3	Lack Of Active Session Invalidation Capabilities After Elevation Request	Insecure Design	Low	Closed
TEL-Q321-4	Missing Default Escalation Prevention Checks	Insecure Design	Informational	Risk Accepted
TEL-Q321-5	Injectable Terminal Escape Sequences And Newlines In Request Reason (TEL-Q420-11 regression)	Injection Flaws (SQL, XML, Command, Path, etc)	High	Closed
TEL-Q321-6	Incorrect Handling Of Large Request Bodies Leads to Bot Messages Being Discarded	Denial of Service (DoS)	Medium	Closed
TEL-Q321-7	Missing Audit Trail For Some Access Request Events	Insecure Design	Low	Closed
TEL-Q321-8	No Role Revocation After An Access Request Deletion	Insufficient Authentication and Session Management	Low	Closed
TEL-Q321-9	Missing Secret Management In Terraform Provider	Insufficient Cryptography	Informational	Closed

ID	Title	Vulnerability Class	Severity	Status
TEL-Q321-10	API Path Injection In Mattermost Bot Calls Through Reviewer's Email	Injection Flaws (SQL, XML, Command, Path, etc)	Informational	Closed
TEL-Q321-11	OOM DoS Risk in Gitlab Webhook through ReadAll	Denial of Service (DoS)	Low	Closed
TEL-Q321-12	Insecure Comparison Of Gitlab Webhook Token	Insufficient Cryptography	Low	Closed

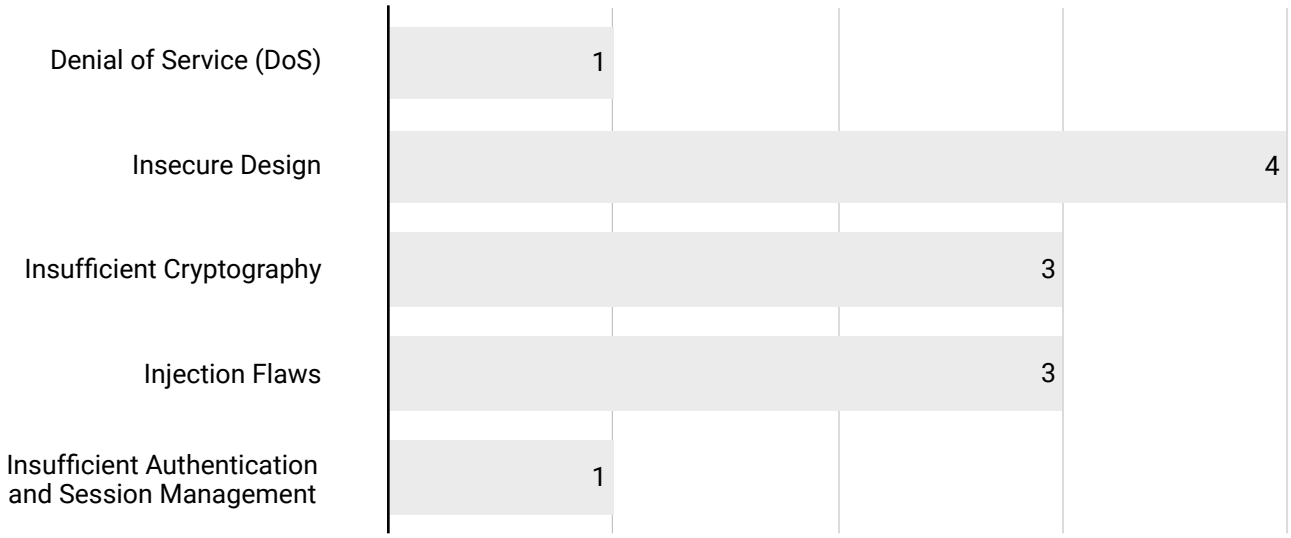
Findings per Severity

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-Q321-1. Injectable Markdown Syntax In Request Reason	
Severity	**Medium**
Vulnerability Class	Injection Flaws (SQL, XML, Command, Path, etc)
Component	teleport-plugins/access/slack/bot.go:255
Status	Closed

Description

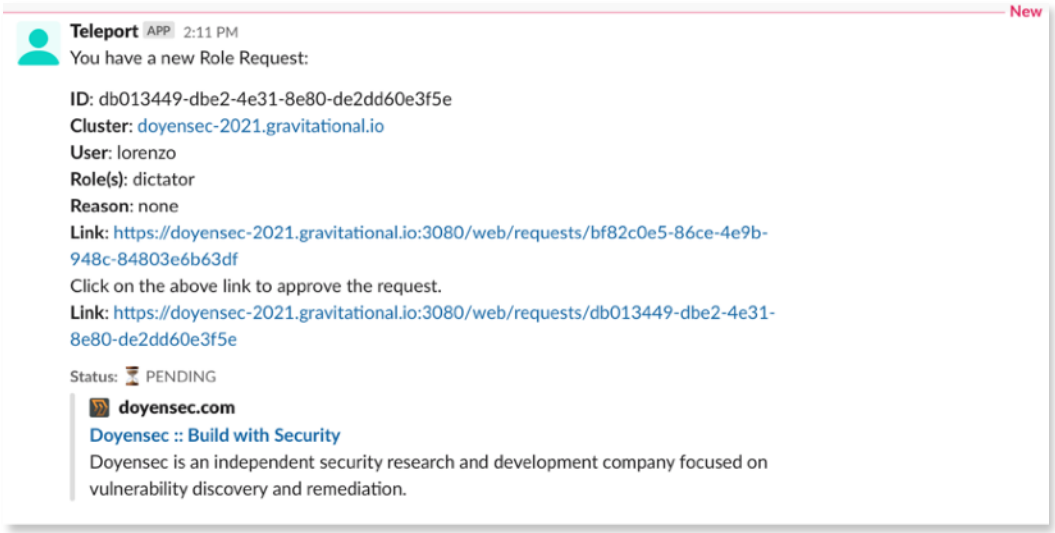
In Teleport, Access Requests submitted by users can also include a text message specifying the request reason. This message is then embedded in the body of the Slack or Mattermost messages sent by the respective Bots.

Doyensec found that it is possible for an attacker to inject valid Markdown syntax and manipulate the aspect and the content of the messages. This can be leveraged by an attacker to mount convincing phishing scams or to abuse operators carelessly trusting the Teleport Bot messages, leading to a different elevation request.

Reproduction Steps

Both the Slack and Mattermost Markdown flavors allow to forge links that are seemingly legitimate but pointing to arbitrary locations. In Mattermost, even inline images (https://docs.mattermost.com/help/messaging/formatting-text.html) are injectable. By way of example, injecting the following access request reason results in a spoofed link to https://doyensec.com:

```
none
*Link*: <https://doyensec.com/|https://doyensec-2021.gravitational.io:3080/web/
requests/bf82c0e5-86ce-4e9b-948c-84803e6b63df>
Click the above link to approve the request.
```



Impact

An attacker may successfully launch a phishing scam and potentially obtain sensitive information or escalate privileges.

Complexity

Complexity to craft the exploit is trivial; however, the attacker must be able to send access requests and mount a convincing attack using social engineering techniques.

Remediation

Completely sanitize the user-provided message placing the request reason inside a code block with tilde or back-tick characters, escaping any existing ones. Alternatively, allow only a restricted set of characters for basic formatting.

TEL-Q321-2. Email Validation Regular Expression Prone To Abuses	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	teleport-plugins/lib/email.go
Status	Closed

Description

Doyensec inspected the source code and found that the access request endpoint uses a Regular Expression (RegEx). The expression is used to validate the user-provided reviewer field, usually containing an email address (`teleport-plugins/access/slack/app.go:361`). This validation of the email address format is too lax and as a result the check unexpectedly succeeds when validating incorrect email address values.

In the Teleport web application, the risk of this scenario is present in `teleport-plugins/lib/email.go:5`:

```
var emailRegex = regexp.MustCompile("^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$")
```

While RFC 5322 supports a variety of email address formats, the risk profile of Teleport should lead to a layered approach rather than relying on a single filter or defense based on a regular expression.
This was shown later in TEL-Q321-10, where the allowed presence of special character in the local part of the email address was weaponized. Additionally, the Regular Expression is using grouping with repetition, but since Golang's regex standard package[1] is not using a Nondeterministic Finite Automaton (NFA)[2] to navigate the string but an RE2 implementation, there's a guarantee of a linear-time performance and graceful failing: the memory available to the parser, the compiler and the execution engines is limited.

Reproduction Steps

The issue was validated using a dedicated regex security module (*ReScue*[3]) that detects potentially catastrophic exponential-time regular expressions and creates DoS payloads. Using the generated payload:

- `?@0`
- `'/**/OR/**/1=1/**/--/**/@a.a`
- `%00%2a@a.a`
- `+%/`?/*|~6@0.0.0.0`
- `iS%+$qOSS/YT1v3s#kjo~7mAQHlPWZrSQ54Zim0t.Tg=$'2jGUZ^Q25w^SHUqaHp@KczuQdjtH-5BMMvnhjNvK7oPG`

[1] https://golang.org/pkg/regexp/

[2] http://en.wikipedia.org/wiki/Nondeterministic_finite_state_machine

[3] https://github.com/2bdenny/ReScue

```
FuWjvMLjIbYNE4epqkAV9U9Uv07AKzARJlpkrlObqMJaNDE0f8Ru$976cNCxevgl9QYB3IO6f6Q7haFB
AKpyVkb83uaxfJsImWD77c2R9IBbanvLZZcRhvHFp-96CPLJQ0wdbOgknAybMPV5hSAkPupCO1wSc"6p
NPNkfURHkutSbmx9kiplHiT8Joxwo9okDRs44nLegAhppCxN10zrpoH755U81BOkEEkCxnHkc1gX3LSN
r6OgAqzyTKU3hfysg870VTa5uWD7WHYvlPPqvEx4UiUeU0nyFkt65Zl7ThPvb7c7IUtNktng68o1TskY
MCaT0g5AHz2KlWBQcFZlOn4jP2FygpwIm5RakqmRxoLOdsCERxEfjya5AzUz5DMrr6RxGOukaDWxSuTU
QLng5K48LOnxOVYQkKmXI6tpcMHCNxCsOzgA1ZYb89]MGe&0qw8DSIWgVvsunay6rkAq3MS70GL6UrLX
0gian2~NLez0AOjF1ekx8zbXVxx75shAoawKv18tsNfJTvK3FsOGmwiMkWR6Ws1.U`hRCrWqlbRhvQkH
7fnAqhePm{9rqHcwZ5hOvnA1j0AKWZJEK1MxTL2ce-
XjTzIgLDazitsHYlymSR2FNq8YgS1ayg13VYvML4Y00`zRruWvPYDfazKiFubMAc1QWOxdcqFyoYMWzE
CTjrXXx-a7OerG1PuSCBZSuNLG!
WfoNuIwbCV0C8p9fbH2WtzOjIlIb4eYOdfKE2NWcYtA%3.a6GZ2VRHnxXx1gak1y04aQOib|Q&p>a
```

Impact

Medium. An attacker may use this issue to mount more complex attack leveraging a weak validation.

Complexity

Medium. An attacker would need to craft a payload for the RegEx-validated string in order to exploit the issue.

Remediation

Instead of updating the RegEx for the email validation, we recommend using a dedicated validation library, which supports correct email validation.

Email validation is a fairly challenging process that we do not recommend conducting with manual regular expressions.

Resources

- "Input Validation Cheat Sheet; Email Address Validation", OWASP Cheat Sheet https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html#email-address-validation
- "Safe email validation", Information Security Stack Exchange https://security.stackexchange.com/questions/116116/safe-email-validation
- Codacy Blog, “Best Practices for Regular Expressions” https://www.codacy.com/blog/best-practices-for-regular-expressions/

TEL-Q321-3. Lack Of Active Session Invalidation Capabilities After Elevation Request	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Teleport Advanced Access Workflows
Status	Closed

Description

When a Teleport user requests additional roles, the workflow API makes it easy to dynamically approve or deny such requests. After an elevation is approved, there's no immediate way to immediately and proactively terminate it. This design is fundamental to ensure that access privileges are handled appropriately in case a request is wrongfully or maliciously approved or if a deprovisioning process to immediately cut all granted access should be executed.

Reproduction Steps

In order to reproduce the issue:

1. Submit a new access request with a user (`/web/requests/new`)
2. Approve the access request
3. Observe that there's no way for the reviewers to terminate the elevation before the expiration period passes

Impact

Medium. A user could maintain access until the expiration period, even if the role is mistakenly or maliciously approved. A privileged reviewer should be able to remove the role assignment at any time.

Complexity

High. An attacker would still require a request to be approved. Scenarios that could require an administrator to revoke an elevated role for a user include compromised accounts, access request phishing, and other insider threats.

Remediation

To mitigate the risk, implement a session revocation feature for role elevations, initiated through the web or the console.

Resources

- "Session Management Cheat Sheet", OWASP CheatSheets Series https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html

TEL-Q321-4. Missing Default Escalation Prevention Checks	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	N/A
Status	Risk Accepted

Description

The management of access control requests is a complex and dynamic problem that involves business, organizational, and legal constraints and a technical implementation. Access control design decisions have to be made by humans, not technology, and the potential for errors is high. In the same way, Teleport roles' policies are user-defined and can therefore be inherently insecure on creation, allowing role elevations that can be abused to alter and promote attacker-controlled resources. This can happen if after the role elevation the attacker can then:

- *create* or *update* their assigned roles
- *create* or *update* their user resources

The risk of vertical privilege escalation should be notified to the user, for instance by issuing a warning during the request approval.

A secure by default design involves highlighting this risk to the reviewers before a potentially dangerous role approval or when defining the `request.roles` elevations for a role.

The *Resource Access Requests*[4,5] feature introduced in Teleport 10 partially mitigates this issue, making vertical privilege escalation less likely to occur.

Reproduction Steps

In order to reproduce the issue in the *doyensec-2021* testing environment:

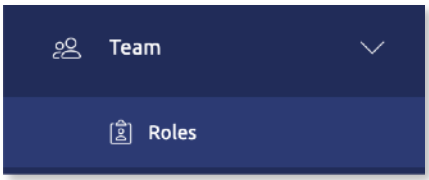
1. Change the *Dictator* privileges to also grant:

```
kind: role
metadata:
  id: 1625816982826230463
  name: dictator
spec:
  allow:
    rules:
    - resources:
```

[4] https://github.com/gravitational/teleport/issues/10887

[5] https://goteleport.com/docs/enterprise/workflow/resource-requests/?scope=enterprise

```
- role
verbs:
- list
- create
- read
- update
- delete
```



2. While normally Bob has the *Populist* role assigned, Bob requests *Dictator* privileges

3. After assuming the *Dictator* role, Bob can edit their default *Populist* role and maintain the privileges indefinitely. A similar result can be achieved if the elevated *Dictator* role had permissions over *user* resources.

Impact

Medium. An insider threat could abuse loosely written roles resources together with the access request workflow to escalate their privileges indefinitely.

Complexity

High. An attacker would still require their request to be approved. Sufficient understanding of Teleport users/roles design is required.

Remediation

Prevent or warn about potential escalations, either on role creation/edit or on elevation approval. Remediations against this type of attack vectors might involve UX and product changes.

TEL-Q321-5. Injectable Terminal Escape Sequences And Newlines In Request Reason	
Severity	**High**
Vulnerability Class	Injection Flaws (SQL, XML, Command, Path, etc)
Component	• e/lib/web/access_request.go:29 • lib/auth/auth.go:1602
Status	Closed

Description

As previously reported in TEL-Q420-11, the Approval Workflow API suffered from a CLI Content spoofing issue through the request reason field. This could allow an attacker to inject arbitrary content, spoofing multiple subsequent lines in the CLI table, in the context of a social engineering or phishing attack. The issue was fixed by integrating a footnote mechanism, truncating long request strings.

While analyzing the web-based access requests, we found that terminal escape sequences (ANSI/VT) and new lines are allowed and respected in the reasons text content when submitted via the web panel. A terminal escape sequence is a special sequence of characters that is printed (like any other text). However, if the terminal understands the sequence, it won't display the character-sequence, but will perform some action. Besides changing the color of the text, making it bold, or making the cursor blink, they can also:

- Move the cursor in any direction or to any position
- Delete or spoof an arbitrary text
- Perform various screen manipulations
- Re-map keys on the keyboard
- ..or perform other tasks depending on the terminal emulator

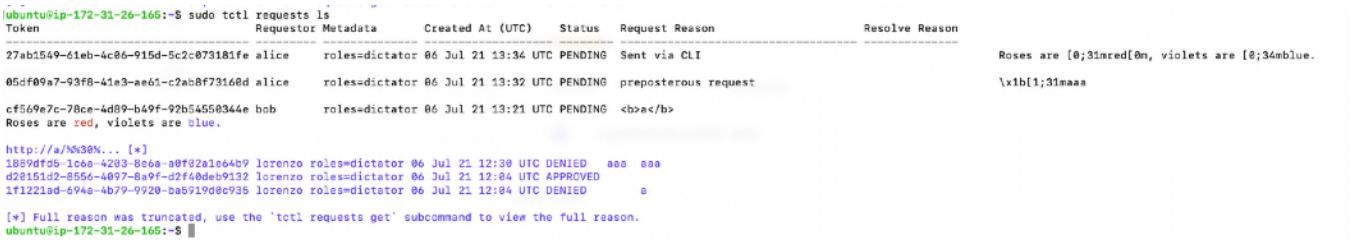
Since these sequences can adversely change the appearance on the terminal, the risk of an attacker abusing this to trick an operator into approving their access requests is very high[6].

Reproduction Steps

The following screenshot contains two requests. The first was submitted via CLI using the command, while the second was submitted via web UI:

```
%  tsh login --proxy doyensec-2021.gravitational.io:3080 --user alice --request-
roles dictator --request-reason "Sent via CLI                Roses are
[0;31mred[0m, violets are [0;34mblue. "
```

[6] https://www.infosecmatter.com/terminal-escape-injection/



Impact

In a successful attack scenario, an attacker could inject arbitrary terminal sequences and interfere with the console output, possibly forging a credible line on the elevation requests table. This could be used to change the appearance of the request UUID before the copy-paste and approval.

Complexity

Whether such output can be exploited depends on the terminal program, and what that terminal does depends on the escape codes that are being sent. Because of this, the complexity of crafting the exploit can vary. However, the payload must be carefully crafted to work on the victim's terminal. The attacker should also own a valid account and the elevation request should also be allowed by the account's assigned role.

Remediation

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

Resources

- "Can “cat-ing” a file be a potential security risk?", StackExchange Security
  https://security.stackexchange.com/questions/56307/can-cat-ing-a-file-be-a-potential-security-risk
- "A Blast From the Past: Executing Code in Terminal Emulators via Escape Sequences", Dejan Lukan
  https://www.proteansec.com/linux/blast-past-executing-code-terminal-emulators-via-escape-sequences/
- "Content Spoofing", OWASP Community Guides
  https://owasp.org/www-community/attacks/Content_Spoofing

TEL-Q321-6. Incorrect Handling Of Large Request Bodies Leads to Bot Messages Being Discarded	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	• teleport-plugins/access/slack/bot.go:120 • teleport-plugins/access/mattermost/bot.go:212
Status	Closed

Description

One of the many objectives of Teleport Access Request bots in organizations is to provide visibility into what's happening on the assets owned by the company. Using this feature, organizations can perform continuous access monitoring to the appropriate team, helping in detecting intrusions or highlighting any suspicious activity. An attacker in the middle of a privilege escalation attack will consequently be interested in suppressing bot messages flagging their malicious activity. During the course of the engagement, Doyensec discovered a technique to suppress elevation requests messages for both Mattermost and Slack bots.

Because of a discrepancy between the maximum allowed request body of the Bot server versus the amount allowed by the Slack or Mattermost APIs, an attacker can use the controlled "Reason" field to craft a request with a large number of bytes. For Slack bots, any "Reason" having more than 5.000 characters will result in a rejection from the Slack API. For Mattermost, any "Reason" above 10.000 will also throw an error, preventing the message from being sent.

Reproduction Steps

To reproduce the attack, it is possible to either:

- Instrument a non-transparent proxy between the Teleport web service and the User-Agent through Burp Suite or Fiddler
- Intercept a valid web access request
- Use the Intruder tool or a script to craft the request payload including 5.000 characters in the request reason value

Or adapt the following *curl* command, populating it with valid session tokens:

```
curl -i -s -k -X $'POST' \
    -H $'Host: doyensec-2021.gravitational.io:3080' -H $'Authorization: Bearer
c552d06e819347a1c2a5999a7a5cc90b5c4acf8a17fd1120c5c7375d0393adbc' -H $'Content-
Type: application/json; charset=utf-8' -H $'Origin: https://
doyensec-2021.gravitational.io:3080' -H $'Connection: close' -H $'Content-Length:
5081' \
    --data-binary $'{\"reason\":\"A*5000\",\"roles\":[\"dictator\"],
\"suggestedReviewers\":[\"lorenzo@doyensec.com\"]}' \
```

```
$'https://doyensec-2021.gravitational.io:3080/v1/enterprise/accessrequest'
```

This will fire the following request:

```
POST /v1/enterprise/accessrequest HTTP/1.1
Host: doyensec-2021.gravitational.io:3080
Authorization: Bearer
c552d06e819347a1c2a5999a7a5cc90b5c4acf8a17fd1120c5c7375d0393adbc
Content-Type: application/json; charset=utf-8
Connection: close
Content-Length: 5081

{
    "reason": "AAAAAAAAAA...*5.000",
    "roles": [
        "dictator"
    ],
    "suggestedReviewers": [
        "lorenzo@doyensec.com"
    ]
}
```

The service journal will then report the following exception messages:

Slack

```
User Message: invalid_blocks, invalid_blocks]
request_id:7a91edb7-27a3-405c-894c-0db5623b29b9 request_op:put request_state:PENDING slack/
app.go:195
ERRO   Failed to process request error:[
ERROR REPORT:
Original Error: trace.aggregate invalid_blocks, invalid_blocks
Stack Trace:
        /go/src/github.com/gravitational/teleport-plugins/access/slack/
bot.go:140 main.Bot.Broadcast
        /go/src/github.com/gravitational/teleport-plugins/access/slack/app.go:274 main.
(*App).broadcastMessages
        /go/src/github.com/gravitational/teleport-plugins/access/slack/app.go:231 main.
(*App).onPendingRequest
        /go/src/github.com/gravitational/teleport-plugins/access/slack/app.go:184 main.
(*App).onWatcherEvent
        /go/src/github.com/gravitational/teleport-plugins/lib/watcherjob/
watcherjob.go:228 github.com/gravitational/teleport-plugins/lib/
watcherjob.job.eventFuncHandler.func1
        /go/src/github.com/gravitational/teleport-plugins/lib/process.go:195 github.com/
gravitational/teleport-plugins/lib.jobFunc.DoJob
        /go/src/github.com/gravitational/teleport-plugins/lib/process.go:83 github.com/
gravitational/teleport-plugins/lib.NewProcess.func2.1
        /usr/local/go/src/runtime/asm_amd64.s:1371 runtime.goexit
User Message: invalid_blocks, invalid_blocks] request_id:7579013b-7a27-478f-873d-
cc9464f6497c request_op:put request_state:PENDING slack/app.go:195
```

Mattermost

```
ERRO   Failed to process request error:[
ERROR REPORT:
```

```
Original Error: trace.aggregate api error status_code=400, message=&#34;Invalid
message.&#34;
Stack Trace:
        /go/src/github.com/gravitational/teleport-plugins/access/mattermost/
bot.go:212 main.Bot.Broadcast
        /go/src/github.com/gravitational/teleport-plugins/access/mattermost/
app.go:274 main.(*App).broadcastMessages
        /go/src/github.com/gravitational/teleport-plugins/access/mattermost/
app.go:232 main.(*App).onPendingRequest
        /go/src/github.com/gravitational/teleport-plugins/access/mattermost/
app.go:185 main.(*App).onWatcherEvent
        /go/src/github.com/gravitational/teleport-plugins/lib/watcherjob/
watcherjob.go:228 github.com/gravitational/teleport-plugins/lib/
watcherjob.job.eventFuncHandler.func1
        /go/src/github.com/gravitational/teleport-plugins/lib/process.go:195 github.com/
gravitational/teleport-plugins/lib.jobFunc.DoJob
        /go/src/github.com/gravitational/teleport-plugins/lib/process.go:83 github.com/
gravitational/teleport-plugins/lib.NewProcess.func2.1
        /usr/local/go/src/runtime/asm_amd64.s:1371 runtime.goexit
User Message: api error status_code=400, message=&#34;Invalid message.&#34;]
request_id:a8a78041-f207-4f73-9a44-f0e5a6f233d1 request_op:put request_state:PENDING
mattermost/app.go:196
```

Impact

In a successful attack scenario, an attacker could prevent bot messages from being fired and carry out a privilege escalation attack without alerting the team on the assigned channel, increasing the chances of a successful attack.

Complexity

The attacker should own a valid session and the elevation request should also be allowed by the account's assigned role.

Remediation

Limit the length of the request reason and enforce a limit for the whole message sent to the IM services APIs.

TEL-Q321-7. Missing Audit Trail For Some Access Request Events	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	teleport/api/types/events.go
Status	Closed

Description

The exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. A compromised user, an insider threat, or access control failures can all be detected and recorded in case of an extensive logging and monitoring mechanism, allowing for fast and active responses from the security team. By design, auditable events related to access requests are only reporting basic approval events.

During our features review, we identify a potentially dangerous lack of visibility into several actions, such as:

- When a user assumes the role after approval or resumes her own static roles (only some entries related to the cert generation are present in the service journal of the *teleport.service*);
- When an access request is deleted.

Reproduction Steps

N/A

Impact

Medium. An attacker could attempt more verbose attacks without the risk of being logged. Allowing for vulnerability probing or snooping to continue without logs can raise the likelihood of successful exploitation. If the audit log or the resources versioning is not detailed and extensive, it could delay forensic analysis performed by the CSIRT and the security team's remediation actions. See TEL-Q3-8 for an example of a malicious action that could go unnoticed.

Complexity

High. Any attacker's minimal activity would probably be logged anyway. Log data may be missing, modified, forged, or replayed.

Remediation

The level and content of security monitoring, alerting, and reporting needs to be carefully evaluated and should be proportional to the information security risks for a PAM solution like Teleport. **Because of the high-risk profile of the Access Workflows feature, improve the existing audit trail extending the number of different event types that are ingested.**

Resources

- "Logging Cheat Sheet", OWASP Cheat Sheet Series
  https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html

TEL-Q321-8. No Role Revocation After An Access Request Deletion	
Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	teleport/lib/services/local/dynamic_access.go
Status	Closed

Description

Teleport roles having the delete permission over access requests can force the deletion of all the pending or processed past access requests. This is handled by the `DeleteAccessRequest` function on `teleport/lib/services/local/dynamic_access.go:249`:

```
func (s *DynamicAccessService) DeleteAccessRequest(ctx context.Context, name
string) error {
 err := s.Delete(ctx, accessRequestKey(name))
 if err != nil {
        if trace.IsNotFound(err) {
                return trace.NotFound("cannot delete access request %q (not
found)", name)
        }
        return trace.Wrap(err)
 }
 return nil
}
```

Given that no other cascading actions are performed by the function, whenever an access request is approved and then deleted, the promoted user will always retain the elevated role up until its expiration period. A malicious user could delete their elevation requests while retaining access to their promoted roles. Considering that no traces of the access request deletion are shown in the audit log (see `TEL-Q321-7`) and that it is possible for a user to also suppress IM bot messages (see `TEL-Q321-6`), the current design increases the effectiveness of privilege escalation attacks, leaving little to no trails.

Reproduction Steps

1. First, provide a test user with a role allowing `access_request` > `delete`:

```
kind: role
metadata:
  id: 1234567891234567891
  name: test
spec:
  allow:
    rules:
    - resources:
      - access_request
      verbs:
      - list
```

```
        - create
        - read
        - update
        - delete
    deny: {}
      options:
        cert_format: standard
        enhanced_recording:
        - command
        - network
        forward_agent: true
        max_session_ttl: 30h0m0s
        port_forwarding: true
    version: v3
```

2. Then, request and approve an access request.
3. After assuming the role, delete the access request.
4. Observe that it is still possible to use the elevated role.

Impact

Medium. An attacker can more easily carry out privilege escalation attacks without the risk of being discovered. In some scenarios, combining this vulnerability with other design pitfalls (e.g. `TEL-Q321-6` and `TEL-Q321-7`) could result in a higher likelihood of successful exploitation.

Complexity

High. The attacker should own a valid session for a user with *delete* permissions on access requests in the first place.

Remediation

On access request deletion, immediately demote to their default static roles all active users having an elevated role connected to the deleted access request.

TEL-Q321-9. Missing Secret Management In Terraform Provider	
Severity	**Informational**
Vulnerability Class	Insufficient Cryptography
Component	teleport-plugins/terraform/tfschema/types_terraform.go
Status	Closed

Description

Teleport's Terraform Provider can be used to initialize and configure several resources of a Teleport installation for Cloud, Enterprise, and Open Source editions. Some of these resources can also contain secrets such as:

- `teleport_oidc_connector`, specifying a `client_secret`, used to authenticate the client
- `teleport_github_connector`, specifying a `client_secret`, the Github OAuth app client secret
- `teleport_trusted_cluster`, specifying the `token` used as the authorization token provided by another cluster needed by this cluster to join.
- `teleport_saml_connector`, specifying a `private_key`, the PEM encoded x509 private key used for signing authentication requests or decrypting SAML assertions.

Storing secrets in plain text files is a bad practice which could lead among there things to users hard-coding the credentials directly in their Terraform code and check it into version control. Other threats include local malicious processes having read access to the file.

An implementation for sensitive values exists[7], but it is meant only to help prevent their display in normal CLI usage and logs. It does not prevent the storage of the values in the `terraform.tfstate` state file[8], and so will be visible to anyone who is able to access the state data. As advised in the terraform documentation[9]:

> *"Marking variables as sensitive is not sufficient to secure them. You should use secrets management tools and secure your state in addition to marking variables as sensitive."*

Reproduction Steps

Currently, the example Terraform code included in `teleport-plugins/terraform/example/main.tf` is declaring all the resources providing cleartext inline secrets:

```
resource "teleport_github_connector" "github" {
```

[7] https://www.hashicorp.com/blog/terraform-0-14-adds-the-ability-to-redact-sensitive-values-in-console-output

[8] https://blog.gruntwork.io/how-to-manage-terraform-state-28f5697e68fa

[9] https://learn.hashicorp.com/tutorials/terraform/sensitive-variables?in=terraform/configuration-language

```
  metadata {
     name = "test"
     labels = {
       test = "yes"
     }
  }
  spec {
    client_id = "client"
    client_secret = "value"
    teams_to_logins {
       organization = "gravitational"
       team = "em"
       logins = ["terraform"]
    }
  }
}

resource "teleport_trusted_cluster" "cluster" {
  metadata {
    name = "primary"
    labels = {
      test = "yes"
    }
  }
  spec {
    enabled = false
    role_map {
      remote = "test"
      local = ["admin"]
    }
    proxy_address = "localhost:3080"
    token = "salami"
  }
}

resource "teleport_oidc_connector" "oidc" {
  metadata {
     name = "test"
     labels = {
       test = "yes"
     }
  }
  spec {
    client_id = "client"
    client_secret = "value"
    claims_to_roles {
      claim = "test"
      roles = ["terraform"]
    }
  }
}
```

Impact

The use of a hard-coded password tremendously increases the possibility of secret leaks. This exposes organizations using Teleport to a series of threats: a rogue employee with access to this information can use it to break into the system, or an attacker with an arbitrary file read vulnerability may access the Terraform code and escalate access to the provisioned Teleport infrastructure.

Complexity

High. An attacker will need to access Teleport's Terraform code in order to retrieve the secrets. Depending on how the Infrastructure code is stored, the complexity may vary.

Remediation

Warn users in the repository and in the documentation about the risk of credentials disclosure when writing their configurations. Edit the example Terraform file in the *teleport-plugins* repository not to use inline cleartext secrets.

While Terraform officially does not have plans for improved handling of secrets, multiple proposals for the handling of sensitive values, in general, are being tracked in #9556[10] and #516[11]. Nonetheless, multiple workarounds exist[12]:

- Declaring variables for the secrets relying on Terraform's native support for reading environment variables.
- Defining secrets as variables in a separate `.tfvars` file. The secret variables can then be sourced from a secret `.tfvars` file that is added to the `.gitignore` and typically contains all the secrets for the Terraform project.
- Using a dedicated Terraform provider that lets you inject secrets into the Terraform code simply by referencing a path (e.g. using commercial solutions such as *HashiCorp Vault*[13], *SecretHub*[14], *AWS/GCP Secrets Manager*).

	Environment Variables	Encrypted Files (e.g., KMS, SOPS)	Secret Stores (e.g., Vault, AWS Secrets Manager)
Secrets are encrypted	✅	✅	✅
All secrets management is defined as code	❌	✅	✅
Audit log for encryption keys	❌	✅	✅
Audit log for secrets	❌	❌	✅
Secrets solution is easy to integrate with apps	❌	❌	✅
Secrets can be rotated or revoked	❌	❌	✅
Secrets are versioned with the code, reducing config errors	❌	✅	❌
Test friendly	✅	❌	❌
Cost	0	💰	💰💰💰
Ease of use	👍👍	👍	👍👍👍

[10] https://github.com/hashicorp/terraform/issues/9556

[11] https://github.com/hashicorp/terraform/issues/516

[12] https://blog.gruntwork.io/a-comprehensive-guide-to-managing-secrets-in-your-terraform-code-1d586955ace1

[13] https://www.vaultproject.io/docs/secrets/kv

[14] https://github.com/secrethub/terraform-provider-secrethub

TEL-Q321-10. API Path Injection In Mattermost Bot Calls Through Reviewer's Email

Severity	Informational
Vulnerability Class	Injection Flaws (SQL, XML, Command, Path, etc)
Component	teleport-plugins/access/mattermost/bot.go:277
Status	Closed

Description

While testing the Teleport bots for the Advanced Access Workflow, Doyensec found a code path at risk of Split API Injections. The root cause of the vulnerability is that the Mattermost bot service, unlike the Slack one, performs HTTP requests to the Mattermost API directly by building the path components of the URL, employing a string interpolation of the parameters. One of the code paths for these API requests (`LookupDirectChannel`) also includes a user-provided string: the reviewer's email address specified by the requester.

```
// LookupDirectChannel fetches user's direct message channel id by email.
func (b Bot) LookupDirectChannel(ctx context.Context, email string) (string,
error) {
 resp, err := b.client.NewRequest().
        SetContext(context.WithValue(ctx, etagCacheCtxKey{},
getUserByEmail{email: email})).
        SetPathParams(map[string]string{"email": email}).
        SetResult(&User{}).
        Get("api/v4/users/email/{email}")
 if err != nil {
        return "", trace.Wrap(err)
 }
 user, err := userResult(resp)
 if err != nil {
        return "", trace.Wrap(err)
 }

 me, err := b.GetMe(ctx)
 if err != nil {
        return "", trace.Wrap(err)
 }

 resp, err = b.client.NewRequest().
        SetContext(ctx).
        SetBody([]string{me.ID, user.ID}).
        SetResult(&Channel{}).
        Post("api/v4/channels/direct")
 if err != nil {
        return "", trace.Wrap(err)
 }
 channel, err := channelResult(resp)
 if err != nil {
        return "", trace.Wrap(err)
 }

 return channel.ID, nil
```

```
}
```

The Mattermost bot uses the Resty HTTP client dependency (github.com/go-resty/resty/v2) to perform such requests. While the library actually URL-encode the URL fragments in its middleware (github.com/go-resty/resty/v2/middleware.go):

```
func parseRequestURL(c *Client, r *Request) error {
 // GitHub #103 Path Params
 if len(r.pathParams) > 0 {
        for p, v := range r.pathParams {
               r.URL = strings.Replace(r.URL, "{"+p+"}", url.PathEscape(v), -1)
        }
 }
 if len(c.pathParams) > 0 {
        for p, v := range c.pathParams {
               r.URL = strings.Replace(r.URL, "{"+p+"}", url.PathEscape(v), -1)
        }
 }
 ...
}
```

The Mattermost API normalizes any URL-encoded slash (`/`) characters in the request path (`%2F`), allowing for directory traversal sequences (`../`) to be injected:

```
GET /api/v4/users/email/..%2F..%2F..%2F..%2Faaa@doyensec.com HTTP/1.1
Host: gravitational-doyensec-21.cloud.mattermost.com


HTTP/1.1 301 Moved Permanently
Date: Thu, 15 Jul 2021 20:22:14 GMT
Content-Length: 0
Connection: close
Location: /aaa@doyensec.com
X-Ratelimit-Limit: 101
```

However, due to the very limited control over the parameters, the use of the idempotent GET verb, and the fact that the payload must be a valid email address (see TEL-Q321-2), it is extremely difficult to exploit the issue. At the current state, no state-changing, abusable endpoints are exposed by the Mattermost API however the risk of abuses leveraging this design is non-negligible.

Reproduction Steps

It is possible to infer from the *teleport-mattermost* service journal the outcome of the request with a successful path traversal:

- `../../../../aaa@doyensec.com` should redirect (301) to a 404 error Location (Mattermost API returns "*Invalid or missing user_id in request body*");

```
teleport-mattermost[19312]: WARN   No channel to post request_id:7042853d-
f0aa-48eb-af05-109b454b9610 request_op:put request_state:PENDING mattermost/
app.go:236
teleport-mattermost[19312]: WARN   Failed to lookup direct channel info: "Invalid
or missing user_id in request body." mm_user_email:../../../../aaa@doyensec.com
```

```
request_id:6de270d6-8c1b-4353-8075-9658c0a9c1e8 request_op:put
request_state:PENDING mattermost/app.go:352
```

- `../../..@doyensec.com` should redirect (301) to a 200 Location (the root `/`);

```
Jul 15 20:01:59 ip-172-31-26-165 teleport-mattermost[19312]: WARN    No channel
to post request_id:6de270d6-8c1b-4353-8075-9658c0a9c1e8 request_op:put
request_state:PENDING mattermost/app.go:236
Jul 15 20:02:32 ip-172-31-26-165 teleport-mattermost[19312]: WARN    Failed to
lookup direct channel info: "" mm_user_email:../../..@doyensec.com
request_id:c00de998-5f83-4d14-b35f-3306d07db1a8 request_op:put
request_state:PENDING mattermost/app.go:352
```

- `aaa@doyensec.com` is the control sample for the test.

```
Jul 15 19:58:36 ip-172-31-26-165 teleport-mattermost[19312]: WARN    Failed to
lookup direct channel info: "Unable to find the user."
mm_user_email:aaa@doyensec.com request_id:1192e1c5-8fef-4918-a071-34edc06b7409
request_op:put request_state:PENDING mattermost/app.go:352
Jul 15 19:58:36 ip-172-31-26-165 teleport-mattermost[19312]: WARN    No channel
to post request_id:1192e1c5-8fef-4918-a071-34edc06b7409 request_op:put
request_state:PENDING mattermost/app.go:236
```

Impact

Medium. An attacker may forge authenticated requests meant for the Mattermost API. This vulnerability is marked as *Informational* because Doyensec researchers could not find API endpoints in which user input was unconstrained enough to manipulate the API request in a significant way. Additionally, the fact that a valid email must be provided hinders the chances of a meaningful attack. Nonetheless, the current design is dangerous and very prone to mistakes that can introduce vulnerabilities.

Complexity

Low. If a vulnerability deriving from this pattern was to be introduced, exploitation would require basic web security skills, crafting a valid path, and reaching a sensitive Mattermost API endpoint.

Remediation

Normally, URL-encoding the user-provided values before embedding them in the request path is enough. However, in this instance, the Mattermost API normalization must be prevented. Because of this, a simpler approach can be implemented by forbidding or stripping URL control characters in any passed variable (%, &, /, . ., ?, =, #). This can also be accomplished by exclusively allowing a restricted set of characters for the users' email addresses.

TEL-Q321-11. OOM DoS Risk in Gitlab Webhook through ReadAll	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	teleport-plugins/access/gitlab/ webhook_server.go:61
Status	Closed

Description

Similarly to the already reported TEL-Q420-15[15] ("Unauthenticated OOM DoS in ReadJSON"), the Gitlab webhook server is using the `ReadAll` function from `ioutil` to unmarshal the request body in its `processWebhook` function:

```
func (s *WebhookServer) processWebhook(rw http.ResponseWriter, r *http.Request,
ps httprouter.Params) {
 // TODO: figure out timeout
 ctx, cancel := context.WithTimeout(r.Context(), time.Second*5)
 defer cancel()

 httpRequestID := fmt.Sprintf("%v-%v", time.Now().Unix(),
atomic.AddUint64(&s.counter, 1))
 ctx, log := logger.WithField(ctx, "gitlab_http_id", httpRequestID)

 if contentType := r.Header.Get("Content-Type"); contentType != "application/
json" {
        log.Errorf(`Invalid "Content-Type" header %q`, contentType)
        http.Error(rw, "", http.StatusBadRequest)
        return
 }
 if r.Header.Get("X-Gitlab-Token") != s.secret {
        log.Error(`Invalid webhook secret provided`)
        http.Error(rw, "", http.StatusUnauthorized)
        return
 }

 body, err := ioutil.ReadAll(r.Body)
 if err != nil {
        log.WithError(err).Error("Failed to read webhook payload")
        http.Error(rw, "", http.StatusInternalServerError)
        return
 }

 var event interface{}
 switch eventType := r.Header.Get("X-Gitlab-Event"); eventType {
 case "Issue Hook":
        var issueEvent IssueEvent
        if err = json.Unmarshal(body, &issueEvent); err != nil {
               log.WithError(err).Error("Failed to parse webhook payload")
               http.Error(rw, "", http.StatusBadRequest)
               return
        }
```

[15] https://goteleport.com/pdf/teleport-audit-q4-2020.pdf

```
                event = issueEvent
        default:
                log.Warningf(`Received unsupported hook %q`, eventType)
                rw.WriteHeader(http.StatusNoContent)
                return
        }

        if err := s.onWebhook(ctx, Webhook{Event: event}); err != nil {
                log.WithError(err).Error("Failed to process webhook")
                log.Debugf("%v", trace.DebugReport(err))
                var code int
                switch {
                case lib.IsCanceled(err) || lib.IsDeadline(err):
                        code = http.StatusServiceUnavailable
                default:
                        code = http.StatusInternalServerError
                }
                http.Error(rw, "", code)
                return
        }

        rw.WriteHeader(http.StatusNoContent)
}
```

When attempting to read the request body `r.Body`, the function will read in the incoming JSON and attempt to deserialize the webhook data. This results in the entire JSON being loaded into memory from a remote network request. An attacker could abuse this implementation to load large chunks of content into the server's memory, causing an Out-Of-Memory (OOM) error condition and the consequent forceful restart of the webhook service.

Note that depending on the network speed of the attacker, the 5 second timeout may prevent similar attacks to some degree.

Reproduction Steps

Source code finding only. A weaponized example of this can be reproduced adapting the reproduction steps highlighted in `TEL-Q420-15`.

Impact

Depending on the attacker's network speed and on the supervisor's restart policy set up for the service, the Gitlab webhook server could crash or be killed, leading to a considerable downtime of the service in case of a prolonged attack.

Complexity

Medium. An attacker needs to find a way to bypass the token authentication first (either stealing the token or leaking it using `TEL-Q321-12`) and issue a very large request body.

Remediation

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

remote endpoints exposed to untrusted parties.

An example of a secure implementation from `golang.org/x/crypto/acme/acme.go` is shown below:

```
 func (c *Client) responseCert(ctx context.Context, res *http.Response, bundle
bool) ([][]byte, error) {
 b, err := ioutil.ReadAll(io.LimitReader(res.Body, maxCertSize+1))
 if err != nil {
        return nil, fmt.Errorf("acme: response stream: %v", err)
 }
 if len(b) > maxCertSize {
        return nil, errors.New("acme: certificate is too big")
 }
...
}
```

Resources

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

TEL-Q321-12. Insecure Comparison Of Gitlab Webhook Token	
Severity	**Low**
Vulnerability Class	Insufficient Cryptography
Component	teleport-plugins/access/gitlab/ webhook_server.go:74
Status	Closed

Description

While inspecting the use of `ReadAll` across the scoped sections of the *teleport-plugins* repository, Doyensec incidentally found that the Gitlab webhook function (`processWebhook`) is performing an insecure comparison.

Insecure comparison, or byte-by-byte comparison, fails and returns as soon as it encounters two bytes that do not match. Timing oracles leak information to an attacker, facilitating byte-by-byte brute-forcing of data such as usernames, passwords, and others.

For further reference, there is research on measuring nanosecond long timing differences over the internet in timing attack scenarios such as the one described above[16].

Reproduction Steps

The `processWebhook` function executes the following code to do the actual comparison:

```
func (s *WebhookServer) processWebhook(rw http.ResponseWriter, r *http.Request,
ps httprouter.Params) {
 // TODO: figure out timeout
 ctx, cancel := context.WithTimeout(r.Context(), time.Second*5)
 defer cancel()

 ...

 if r.Header.Get("X-Gitlab-Token") != s.secret {
        log.Error(`Invalid webhook secret provided`)
        http.Error(rw, "", http.StatusUnauthorized)
        return
 }

 ...
}
```

Impact

High. Cryptographically insecure string comparisons are oracles for malicious actors. This opens a vector to brute force the webhook secret value.

[16] https://codahale.com/a-lesson-in-timing-attacks/

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 provided webhook secret.

A built-in way of doing constant time string comparison in Go is by using the `ConstantTimeCompare`[17] function of the `crypto/subtle`[18] 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".

```
if subtle.ConstantTimeCompare(r.Header.Get("X-Gitlab-Token"), s.secret) {
      log.Error(`Invalid webhook secret provided`)
      http.Error(rw, "", http.StatusUnauthorized)
      return
}
```

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

Resources

- Coda Hale, “A Lesson In Timing Attacks”
  https://codahale.com/a-lesson-in-timing-attacks/

- Morgan, Timothy D. & Jason W., ”Web Timing Attacks Made Practical"
  https://www.blackhat.com/docs/us-15/materials/us-15-Morgan-Web-Timing-Attacks-Made-Practical-wp.pdf

[17] http://golang.org/pkg/crypto/subtle/#ConstantTimeCompare

[18] http://golang.org/pkg/crypto/subtle/

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)
	Insecure Deserialization

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.

☑	Completely sanitize the user-provided message placing the request reason inside a code block with tilde or back-tick characters, escaping any existing ones. Alternatively allow only a restricted set of characters for basic formatting.
☑	Instead of updating the RegEx for the email validation, we recommend using a dedicated validation library, which supports correct email validation.
☑	To mitigate the risk, implement a session revocation feature for role elevations, initiated through the web or the console.
☑	Prevent or warn about potential escalations, either on role creations/edit or on elevation approval.
☑	Limit the length and the charset of the requesters' reasons (e.g. `/^[\w @\/()]{0,60}$/`).
☑	Limit the length of the request reason and enforce a limit for the whole message sent to the IM services APIs.
☑	Because of the high-risk profile of the Access Workflows feature, improve the existing audit trail extending the number of different event types ingested.
☑	On access request deletion, immediately demote to their default static roles all active users having an elevated role connected to the deleted access request.
☑	Warn users in the repository and in the documentation to take care when writing their configurations to avoid unnecessary credential disclosure. Edit the example Terraform file in the *teleport-plugins* repository to not use inline cleartext secrets.
☑	Normally URL-encoding all the user-provided values before embedding them in the request path is enough. However, in this instance, the Mattermost API normalization must be prevented
☑	Avoid loading arbitrary data into memory regardless of the size. Limit the size of a valid JSON tree and return an error closing the connection when it consumes a substantial amount of memory, especially for remote endpoints exposed to untrusted parties.
☑	Perform a constant-time comparison on the provided webhook secret.

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 or two days. Please reach out if you'd like more information on our retesting process.