

Security Auditing Report

Teleport Core & Cloud Testing Q3 2022

Prepared for: Gravitational, Inc. DBA Teleport
Prepared by: Norbert Szetei, Raul Mijan Titos
Date: 06/18/2023

© DOYENSEC

Table of Contents

Table of Contents	1
Revision History	2
Contacts	2
Executive Summary	3
Methodology	6
Project Findings	9
Appendix A - Vulnerability Classification	66
Appendix B - Remediation Checklist	67

Revision History

Version	Date	Description	Author
1	09/19/2022	First release of the final report	Norbert Szetei, Raul Mijan Titos
2	09/20/2022	Peer Review	Luca Carettoni
3	10/07/2022	Additional Cloud Configuration Review	Francesco Lacerenza
4	10/10/2022	Peer Review	Lorenzo Stella
5	05/22/2023	Retesting Update	Norbert Szetei, Raul Mijan Titos
6	06/12/2023	Additional Retesting	Luca Carettoni
7	06/18/2023	Peer Review	Anthony Trummer

Contacts

Company	Name	Email
Gravitational, Inc.	Reed Loden	reed@goteleport.com
Gravitational, Inc.	Russell Jones	rjones@goteleport.com
Gravitational, Inc.	Zac Bergquist	zac.bergquist@goteleport.com
Doyensec, LLC	Luca Carettoni	luca@doyensec.com
Doyensec, LLC	John Villamil	john@doyensec.com

Executive Summary

Overview

Gravitational, Inc. (DBA "Teleport") engaged Doyensec to perform a security assessment of the Teleport platform (Core and Cloud). Gravitational Teleport is a cloud-native gateway for managing access to clusters of servers, consolidating connectivity, authentication, authorization, and audit into a single platform.

The project commenced on 08/01/2022 and ended on 09/14/2022 requiring two (2) security researchers, for a total of forty (40) person/days. The project resulted in twenty (20) findings of which five (5) were rated as *medium* severity.

On 10/04/2022, Doyensec performed additional cloud configuration review. This deliverable represents the state of all discovered vulnerabilities as of 10/10/2022.

In May and June 2023, Doyensec performed two retest activities of the Teleport platform and confirmed the effectiveness of the applied mitigations. With the exception of 2 issues having informational and medium severities respectively, all vulnerabilities were mitigated in a timely manner by the Teleport Team.

This deliverable represents the state of all discovered vulnerabilities as of 06/12/2023. The retesting was performed using the latest version of the software available at the time of testing (Teleport commit ID `39cee50c` and Teleport Connect commit ID `993ac70`).

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

Testing was conducted remotely from Doyensec's EMEA and US offices.

Scope

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

- Identify misconfigurations and vulnerabilities in:
  - Teleport Community
  - Teleport Enterprise
  - Teleport Cloud
  - Teleport Terminal
  - Teleport Internal Dependencies

- Audit specific focus areas which have arisen due to new functionality since the last comprehensive audit in 2022, that now require a more detailed analysis

- Evaluate the overall security posture and adherence to best practices relative to industry peers

The testing took place in development and local environments using the latest version of the software at the time of testing. In detail, this activity was performed on the following releases:

- Teleport v.10.2.1
  - https://github.com/gravitational/teleport/releases/tag/v10.2.1

- Teleport Enterprise
  - https://github.com/gravitational/teleport.e
  - Commit ID 3724d90deda03309b29707791ca28dea5c332e3d

- Teleport Cloud
  - https://github.com/gravitational/cloud
  - Commit ID: b8335d365dac6345d5f6fc47f90ff8e1b4d1399b

- Teleport Cloud Terraform
  - https://github.com/gravitational/cloud-terraform

  - Commit ID: e626ac9a164a2178cc5d9c7b18a2bc46af8f6136

- Teleport Webapps
  - https://github.com/gravitational/webapps
  - Commit ID: bab46b0d3b757c4356a82edc50a11d0510d1bc39

- Teleport Webapps Enterprise
  - https://github.com/gravitational/webapps.e
  - Commit ID: e210aa8add01fe4753558b34d53e9c9273e69418

Scoping Restrictions

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

The following tasks drafted from the GitHub "Security" queue for Teleport v10 were not fully implemented before our assessment, hence they are considered out of scope:

- Require MFA tap on protocol level for all user-initiated administrative actions #9089
- Automatic upgrades #3121

A working RDP cloud instance was not provided. For this reason, we ran a local instance to perform RDP-related testing.

Findings Summary

Doyensec researchers discovered and reported twenty (20) vulnerabilities in the Teleport platform. While most of the issues were departures from best practices and low-severity flaws, Doyensec identified five (5) *medium* severity issues that can be leveraged to compromise the confidentiality, integrity, or availability of the solution.

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

During the assessment, we thoroughly analyzed all crashes reported during the Q12022 RDP audit (*Doyensec_Gravitational_TeleportRDP_Security_Audit_Q12022.pdf, part Appendix C - RDP Fuzzing*). This task led to the discovery of numerous issues in the forked rdp-rs third-party library or in the interaction of Teleport with this library.

Several findings are related to Denial of Service vulnerabilities against the RDP component (TEL-Q322-4, TEL-Q322-5, TEL-Q322-6, TEL-Q322-10).

Additionally, we identified another possible DoS that the attacker can trigger with a compromised Microsoft SQL Server. We also discovered several insecure design issues, two of them related to the audit logs (TEL-Q322-1, TEL-Q322-8) and MFA seed removal (TEL-Q322-7).

Moreover, we identified four weaknesses in the *Teleport Connect* desktop application. Using a hypothetical Cross-Site Scripting (XSS) vulnerability, an attacker could possibly leverage the medium severity issue (TEL-Q322-15) to gain Remote Code Execution (RCE). The remaining issues are related to insufficient hardening, security checks (TEL-Q322-12, TEL-Q322-13) and a slightly outdated ElectronJS version (TEL-Q322-14).

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

At the design level, Doyensec found the system to be well architected.

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

- Redesign the RDP client *Cgo* function calls to recover from the panic happening in the Rust code. Currently, all crashes caused by the malicious RDP server can crash the Teleport instance, which requires restarting the Teleport daemon

- On several occurrences, Teleport implements a feature without enforcing size checks (e.g., clipboard size for RDP, shared files length). If such behavior is desired, ensure that the daemon correctly handles excessive data and does not crash. Otherwise, set reasonable default limits and potentially allow users to further customize them

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 to standing against threats. Thus we recommend a *whitebox* approach combining dynamic fault injection with an in-depth study of the 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 vulnerability classes.

Setup Phase

Gravitational provided access to the online environment, source code repository and binaries for all components in scope. This included the staging cloud environment (platform.teleport.sh with access to Kubernetes and Sales Center), and staging AWS accounts.

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

Exploiting `TEL-Q322-4`, `TEL-Q322-5`, `TEL-Q322-6` and `TEL-Q322-10` involves proxying Windows Terminal Services and active interception (i.e., MITM) of the traffic. To simulate this, we confirmed the finding by modifying the RDP client to receive the server input from a file instead of using the legitimate Remote Desktop Services (RDS).

Even when the RDS modification can be, in some cases, more demonstrative, we did not perform this due to time constraints and it providing little additional value. For a sufficiently motivated attacker, we estimate this effort would take several days.

Tooling

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

During this engagement, we used the following tools:

- Burp Suite
- Protobuffer Decoder
- Nmap
- Gosec
- golangci-lint
- Visual Studio Code
- Electro.ng
- CloudMapper
- Aqua Cloudsploit
- 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 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.

Desktop and Server Applications

Doyensec has extensive experience finding flaws in thick clients, standalone binaries, and server daemons. We write customized tools to map out control flow and study an application's behavior and internals. Mapping out attack surface, whether local or remote, is paramount to a successful engagement. Doyensec also studies the application's ecosystem, looking for potential pitfalls and common misconceptions.

Fuzzing and instrumentation are important parts of the SDLC which we use to test an application's response to untrusted data. Doyensec has a mature, automated, and flexible fuzzing framework which will be customized to meet the needs of the target and detect its response to malicious input continuously over the testing period.

We deconstruct the application looking for privacy leaks and secrets. We understand the storage, transmission, and protection of user information is critical, along with the server-side handling of user provided data.

Electron Apps Testing

Doyensec was the first security company to publish a comprehensive security overview of the Electron framework during BlackHat USA 2017. Since then, we have reported dozens of vulnerabilities in the framework itself and many popular Electron-based applications.

Thanks to our research efforts, we have extensive experience in analyzing desktop runtime environments based on web technologies. During our testing effort, we will review security mechanisms that ensure isolation between sites, facilitate web security protections and prevent untrusted remote content to compromise the security of the host. We write custom tools to map out control flow and study an application's behavior and internals. Mapping out the attack surface, whether local or remote, is paramount to a successful engagement. Doyensec also studies the application's ecosystem, looking for potential pitfalls and common misconceptions.

Static analysis and instrumentation are important parts of the testing process we use to test an application's response to untrusted data. Doyensec combines manual security testing with a mature Electron.js security testing tool (https://github.com/doyensec/electronegativity) which will be customized to meet the needs of the target.

We take apart the application looking for privacy leaks and secrets. Storage, transmission, and protection of user information is critical.

Cloud Configuration Review

Cloud configuration reviews are centered around the evaluation of security measures that the client implements and are part of their responsibilities in the shared security model. In this realm, we identify assets, define boundaries and carefully review all applicable settings to uncover misconfigurations and departures from best practices.

By combining tooling (Nessus, ScoutSuite, G-Scout, Forseti Security and custom scripts) with manual review skills, Doyensec can perform in-depth reviews that will expose concrete risks and suggest actionable mitigations.

Our reviews can focus on the following areas:

- Governance
- Network Configuration and Management
- Asset Configuration and Management
- Logical Access Control
- Data Encryption
- Security Logging and Monitoring
- Security Incident Response
- Disaster Recovery

Examples of identifiable misconfigurations, weaknesses and departures from best practices include, but are not limited to:

- Sensitive services or databases exposed to the Internet
- Use of plain-text protocols
- Lack of network ACLs (Inbound)
- Lack of network ACLs (Outbound)
- Poor key management and rotation practices
- Lack of Multi-Factor Authentication
- Weak user policies
- Poorly maintained passwords and access keys
- World-readable buckets
- World-writable buckets
- Insecure network design and operational practices that could facilitate data exfiltration or lateral movement
- Missing network isolation or segregation layers

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-Q322-1	**Audit Log Events Bypass Via Exceeding Entry Size Limit**	**Insecure Design**	**Low**	**Closed**
Comment	The issue was addressed in https://github.com/gravitational/teleport/pull/19486 by implementing restrictions on the number of characters logged for the 2FA device name and user agent. Additional trimming logic for excessive strings was implemented.			
TEL-Q322-2	**Rdpclient Integer Overflow In decode_clipboard Function**	**Memory Corruption**	**Informational**	**Closed**
Comment	Static analysis confirmed that https://github.com/gravitational/teleport/pull/17175 correctly performs a size check on the clipboard content before indexing, effectively preventing integer overflow.			
TEL-Q322-3	**Rdpclient Missing Limit For Clipboard Message Size Leading To Terminated RDP Connection**	**Insecure Design**	**Low**	**Closed**
Comment	Static analysis revealed that the issue was remediated by enforcing the maximum message size for RDP messages (https://github.com/gravitational/teleport/pull/18131).			
TEL-Q322-4	**Rdpclient Denial Of Service Via Missing ScNet Or ScCore GCC Messages**	**Denial of Service (DoS)**	**Medium**	**Closed**
Comment	The fix incorporated checking the presence of `ScCore` and `ScNet` message types, implemented in https://github.com/citronneur/rdp-rs/commit/e7c7f9843d626e343b4de7a419af5982a3a7bd4b .			
TEL-Q322-5	**Rdpclient Denial Of Service Via Requesting ProtocolHybrid**	**Denial of Service (DoS)**	**Medium**	**Closed**

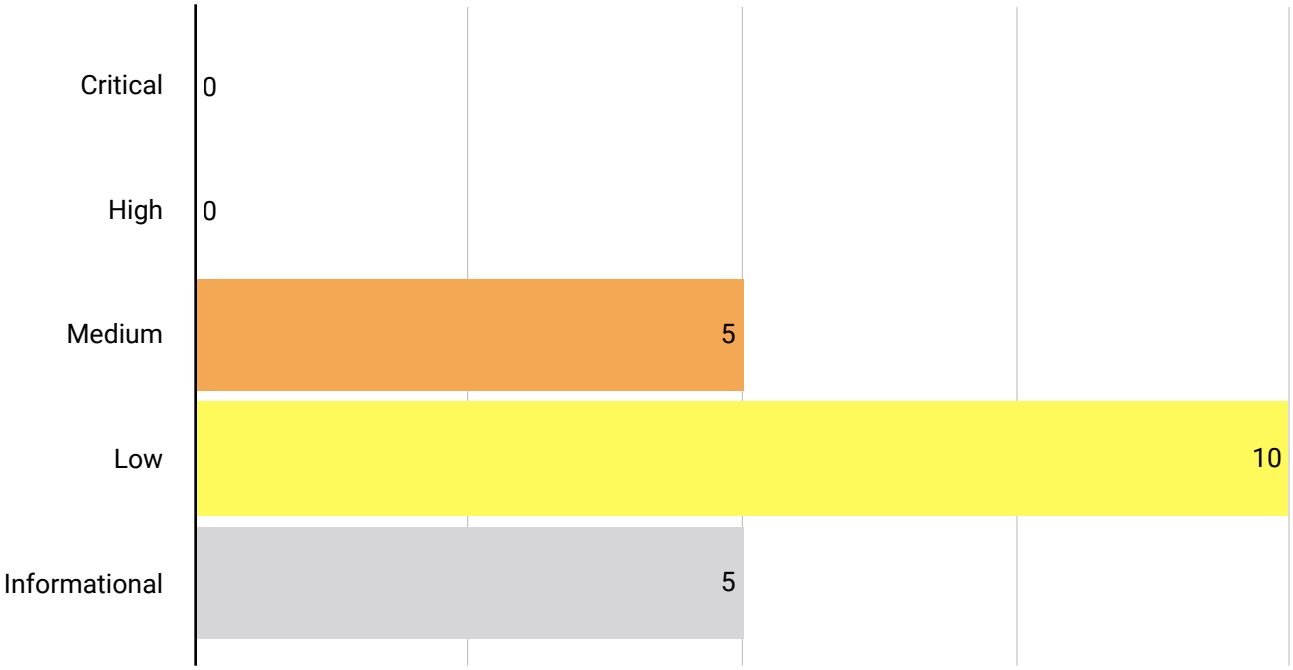
ID	Title	Vulnerability Class	Severity	Status
Comment	Static analysis confirmed the fix by correctly unwrapping the authentication protocol when requesting `ProtocolHybrid` protocol, included in https://github.com/citronneur/rdp-rs/commit/e7c7f9843d626e343b4de7a419af5982a3a7bd4 .			
TEL-Q322-6	**Rdpclient Denial Of Service Via read_synchronize_pdu**	**Denial of Service (DoS)**	**Medium**	**Closed**
Comment	Static analysis confirmed fixing the issue by using the `saturating_sub` function to mitigate the integer overflow in https://github.com/citronneur/rdp-rs/commit/e7c7f9843d626e343b4de7a419af5982a3a7bd4 .			
TEL-Q322-7	**Insecure TOTP MFA Seed Removal**	**Insecure Design**	**Low**	**Closed**
Comment	Static analysis confirmed that the QR code removal utilizes a secure removal function `utils.RemoveSecure` that overwrites the file three times with random data (https://github.com/gravitational/teleport/pull/18752)			
TEL-Q322-8	**Log Injection in Audit Events**	**Insecure Design**	**Low**	**Closed**
Comment	Dynamic analysis confirmed that interpreting special characters in audit logs is no longer possible.			
TEL-Q322-9	**Arbitrary Length Data Reading Via go-mssqldb**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	The issue was resolved by reading a fixed amount of data with a static size (https://github.com/gravitational/go-mssqldb/commit/0f00ae61fb60f7db4268cd73036452bb3d037b90) .			
TEL-Q322-10	**Missing Data Length Limitation For SharedDirectory Packets**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	As with TEL-Q322-3, the issue was remediated by enforcing the maximum message size for RDP messages (https://github.com/gravitational/teleport/commit/75832470467bf9408cd2db0ce66ce8dc4a978f90).			
TEL-Q322-11	**Missing Rate Limit For Proxy API Web Endpoints**	**Insecure Design**	**Low**	**Closed**
Comment	The issue was resolved by implementing rate limiting for unauthenticated routes (https://github.com/gravitational/teleport/pull/19593) .			
TEL-Q322-12	**Missing Protocol Check For openExternal URL**	**Insecure Design**	**Informational**	**Closed**
Comment	The issue was resolved by enforcing a protocol constraint in the `isUrlSafe` function. (https://github.com/gravitational/teleport/commit/d0283d14065286b8f80984845f0b5c3e027254eb).			
TEL-Q322-13	**Missing Hardening For BrowserWindow**	**Security Misconfiguration**	**Informational**	**Open *Risk Accepted***

ID	Title	Vulnerability Class	Severity	Status
Comment	Even though the `sandbox` option has been added to the `webPreferences` of the `createWindow` function, it has been set to `False`, which does not activate this security measure. (https://github.com/gravitational/webapps/blob/0fb98f4606f4d8b1854950f5b923afc7e42a662b/packages/teleterm/src/mainProcess/windowsManager.ts#L52) Gravitational's response: "This is an informational finding related to Teleport Connect not using process sandboxing for its browser window. One of the advantages of an Electron app over a regular web application is the ability to leverage not only browser APIs, but also those provided by Node.js. Teleport Connect makes heavy use of Node.js APIs for pseudo-terminals, logging, and gRPC, among other things. Enabling sandboxing while continuing to support these APIs would require a significant redesign of the application. Since Teleport Connect does not display any 3rd-party web content, and both Electron and React offer some built-in protections against XSS attacks, we do not believe these findings to be exploitable."			
TEL-Q322-14	**Outdated Electron Version In Use**	**Components With Known Vulnerabilities**	**Low**	**Closed**
Comment	The issue was resolved by updating the Electron version in use. (https://github.com/gravitational/webapps/blob/65e05e4a4aee1aef80f7748508711ed3d2f9fa9d/packages/teleterm/package.json#L44).			
TEL-Q322-15	**BrowserWindow Insecure Preload Leading to RCE**	**Insecure Design**	**Medium**	**Open *Risk Accepted***
Comment	The Electron application still makes use of `ptyServiceClient` (https://github.com/gravitational/webapps/blob/27c615b3ff6f317a85fac4aa28b8e73fa4aa0d28/packages/teleterm/src/preload.ts#L40). We've reviewed work in progress code that would reduce the risk introduced by this design. Gravitational's response: "The identified potential for arbitrary shell command execution is isolated to Teleport Connect and hinges on an attacker's ability to run arbitrary JavaScript within the application's browser window — a scenario not currently viable at the time of this report. Even though this is not currently exploitable, due to the potential risk, we will continue to strategize future improvements to reduce this risk."			
TEL-Q322-16	**Privilege Escalation Patterns In Multiple IAM Roles**	**Insecure Design**	**Medium**	**Closed**
Comment	The issue was resolved by removing the `AttachRolePolicy` and `PutRolePolicy` from the `tenant_operator` role.			
TEL-Q322-17	**ELB Configured as PlainText HTTP Listener**	**Cryptography – Missing**	**Informational**	**Closed**
Comment	The issue was resolved by disabling HTTP in `ELB` configuration.			
TEL-Q322-18	**Server-Side Encryption Not Enabled for AWS SNS Topics**	**Cryptography – Missing**	**Low**	**Closed**
Comment	Server-side encryption for AWS SNS topics is enabled on all production accounts.			

ID	Title	Vulnerability Class	Severity	Status
TEL-Q322-19	**Missing SSRF Protection for EC2 Instance Metadata**	**Security Misconfiguration**	**Low**	**Closed**
Comment	The issue was fixed as there are no `EC2` instances using IMDSv1.			
TEL-Q322-20	**Network ACLs Allow Unrestricted Access**	**Insecure Design**	**Informational**	**Closed**
Comment	Static analysis confirmed that the misconfigured Network ACL defined within the Terraform file `cloud/deploy/terraform/teleport-cloud.tf` was fixed at the time of testing.			

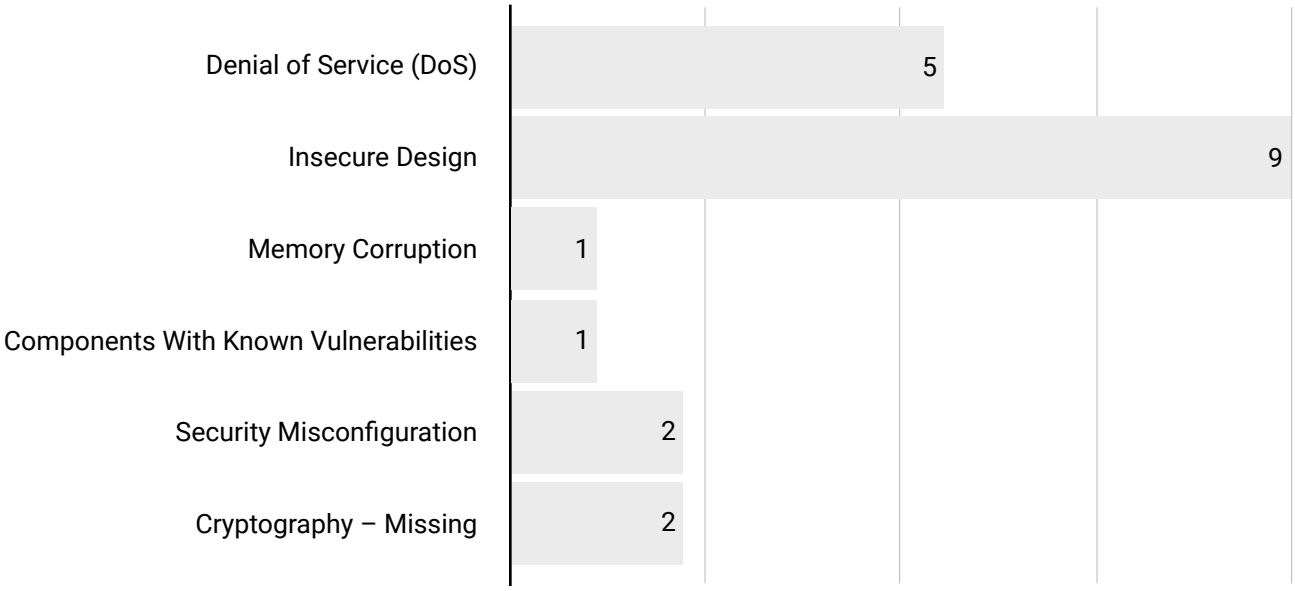
Findings per Severity

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-Q322-1 - Audit Log Events Bypass Via Exceeding Entry Size Limit	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	https://github.com/gravitational/teleport/blob/1f0b4a744b33843e1d670fd5f6073b7a7e3a2da9/lib/events/filelog.go#L171
Status	Closed

Description

Teleport implements an *Audit Events* feature for storing various logging events. Whenever a log event exceeds the maximum allowed size (`MaxScanTokenSize = 64 * 1024`), the entry is excluded from the audit entries.

Doyensec identified multiple parameters the user can supply to surpass the allowed maximum limit:

- The `User-Agent` HTTP request header during login when using Web UI
- The `deviceName`, entered while registering the Hardware Key for *Passwordless* feature

By inputting a large number of characters, it is possible to create `user.login` and `mfa.add` entries that are not audited by Teleport.

Reproduction Steps

Log into the application with a `User-Agent` field larger than 65536 bytes. The error that exceeds the maximum entry size is visible in the Teleport debug logs:

```
2022-08-16T08:35:09+02:00 WARN [AUDIT]     Got a event that exeeded max allowed
size. event_size:65906 event_type:user.login events/filelog.go:171
2022-08-16T08:35:09+02:00 ERRO             Failed to emit audit event. error:[
ERROR REPORT:
Original Error: trace.aggregate event size &#39;&#39; exceeds max entry size
&#39;&#39;
Stack Trace:
  src/teleport/lib/events/emitter.go:289 github.com/gravitational/teleport/lib/
events.(*MultiEmitter).EmitAuditEvent
```

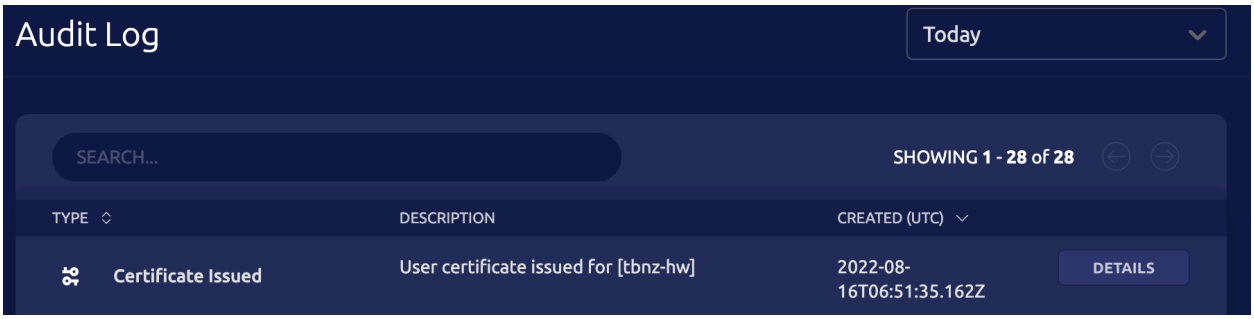
For reproducing the second issue, enable the *Passwordless* feature and invite a new user into your cluster. After setting the username and password, the next step is to assign a device name for the key.

Again, input at least 65536 characters (on macOS, paste the output from `python -c 'print("A"*(64*1024))'| pbcopy`) and observe in the log files the following exceptions:

```
2022-08-16T08:47:25+02:00            ERRO Failed to emit audit event. error:[
ERROR REPORT:
```

```
Original Error: trace.aggregate event size &#39;&#39; exceeds max entry size
&#39;&#39;
...
2022-08-16T08:47:25+02:00 [AUTH]      WARN Failed to emit add mfa device event.
error:[
ERROR REPORT:
Original Error: trace.aggregate event size &#39;&#39; exceeds max entry size
&#39;&#39;
...
```

After the user logs into the application, there is an entry with the issued certificate, while the auditing message about the login is missing:



Impact

Medium. Logins and MFA updates can be performed without appearing in audit trails.

Complexity

Low. Only basic web application testing skills are required. However, an attacker needs network access to the Teleport instance.

Remediation

Limit the maximum allowed size for the user agent and device name. Consider truncating *Audit Log* messages to prevent this error.

Resources

- Teleport, “Audit Events and Records" https://goteleport.com/docs/setup/reference/audit/

TEL-Q322-2 - Rdpclient Integer Overflow In decode_clipboard Function	
Severity	**Informational**
Vulnerability Class	Memory Corruption
Component	https://github.com/gravitational/teleport/blob/c018cd7deb9ec727dcf1592e9fdf4d5d74f38f1a/lib/srv/desktop/rdp/rdpclient/src/cliprdr.rs#L584
Status	Closed

Description

The Teleport RDP client implements a new feature for processing the clipboard buffer into UTF-8 format. Since the user-supplied data buffer can have arbitrary length, we identified an integer overflow when passing the data buffer of size one.

Even though the issue causes a crash of the whole Teleport cluster when using the debug build, on production, the logic that follows after overflowing the integer is not affected. For this reason, the issue is reported as “Informational” only.

Reproduction Steps

The highlighted line below shows where the overflow occurs.

```
fn decode_clipboard(mut data: Vec<u8>, format: ClipboardFormat) ->
RdpResult<Vec<u8>> {
    match format {
        ClipboardFormat::CF_TEXT | ClipboardFormat::CF_OEMTEXT => {
            if data.last().copied() == Some(b'\0') {
                data.pop();
            }

            Ok(data)
        }
        ClipboardFormat::CF_UNICODETEXT => {
            let mut data = data.as_slice();
            let clip = data.len() - 2;
            if data.len() >= 2 && data[clip..] == [0, 0] {
                data = &data[..clip];
            }
```

The `data.len()` returns a variable with `usize`, and if the length of the variable is 1, Rust performs two's complement wrapping and on the x64 architecture, the value becomes `2^64-1`.

We instrumented Teleport to confirm the finding and printed the value of the `clip` variable immediately after the assignment:

```
[2022-08-05T13:12:35Z DEBUG rdp_client::cliprdr] clip: 18446744073709551615
```

The following code triggers the overflow:

```c
#include <stdio.h>
#include <Windows.h>
// source: https://stackoverflow.com/questions/1264137/how-to-copy-string-to-clipboard-in-c
int main()
{
    const char* output = "";
    const size_t len = strlen(output) + 1;
    HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, len);
    memcpy(GlobalLock(hMem), output, len);

    GlobalUnlock(hMem);
    OpenClipboard(0);
    EmptyClipboard();
    SetClipboardData(CF_UNICODETEXT, hMem);
    CloseClipboard();
}
```

Impact

Currently, we discovered no impact when using the production build. For this reason, we assign Informational severity only.

However, the issue could be potentially harmful when developers extend the decoding logic or reuse it in a different component.

Complexity

To trigger the overflow, an attacker needs access to the RDP server via Teleport.

Remediation

Check the data buffer length before assigning the subtracted value to the clip variable.

Following the recommendations from *The Rust Programming Language* book, you can leverage several families of methods to prevent integer overflow:

- Wrap in all modes with the `wrapping_*` methods, such as `wrapping_add`
- Return the `None` value if there is overflow with the `checked_*` methods
- Return the value and a boolean indicating whether there was overflow with the `overflowing_*` methods
- Saturate at the value's minimum or maximum values with `saturating_*` methods

Resources

- GitHub Issue - Desktop Access: Clipboard support #9980
  https://github.com/gravitational/teleport/issues/9980

- S. Klabnik and C. Nichols, The Rust Programming Language
  https://doc.rust-lang.org/book/ch03-02-data-types.html#integer-overflow

TEL-Q322-3 - Rdpclient Missing Limit For Clipboard Message Size Leading To Terminated RDP Connection	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	https://github.com/gravitational/teleport/blob/1f0b4a744b33843e1d670fd5f6073b7a7e3a2da9/lib/srv/desktop/rdp/rdpclient/src/lib.rs#L570
Status	Closed

Description

The Teleport RDP client does not enforce any limit on the clipboard size. While this is the default behavior on Windows, and the clipboard size is limited only by the available memory and address space, an unhandled limit size passed to Teleport can cause unintended behavior.

Currently, it is enough if the Windows application sets the clipboard to around 10,000 characters and the RDP connection to the server is terminated.

Reproduction Steps

Compile the following C code on Windows:

```
#include <stdio.h>
#include <Windows.h>

int main(int argc, char * argv[]) {
    if (argc > 1) {
        const size_t len = atoi(argv[1]);

        char* output = malloc(len);
        memset(output, 0x43, len);
        HGLOBAL hMem = GlobalAlloc(GMEM_MOVEABLE, len);
        memcpy(GlobalLock(hMem), output, len);

        GlobalUnlock(hMem);
        OpenClipboard(0);
        EmptyClipboard();

        SetClipboardData(CF_UNICODETEXT, hMem);
        CloseClipboard();
    }
}
```

If the program runs with the first argument of at least 10,000, the RDP connection terminates, and the following error is triggered:

```
2022-08-08T11:30:53+02:00 DEBU [WINDOWS_D] Received 333329999 bytes of clipboard
data from Windows desktop client-ip:127.0.0.1 desktop-addr:192.168.1.155:3389
```

```
desktop-name:WIN-I37EEB9386R-playground3-sbx pid:8081.1 rdp-
addr:192.168.1.155:3389 rdpclient/client.go:489
2022-08-08T11:30:54+02:00 WARN [WEB] Error proxying a desktop protocol websocket
to windows_desktop_service desktop-name:WIN-I37EEB9386R-playground3-sbx error:[
ERROR REPORT:
Original Error: trace.aggregate websocket: close 1005 (no status), websocket:
close sent
Stack Trace:
/src/teleport/lib/web/desktop.go:332 github.com/gravitational/teleport/lib/
web.proxyWebsocketConn
/src/teleport/lib/web/desktop.go:182 github.com/gravitational/teleport/lib/web.
(*Handler).createDesktopConnection
/src/teleport/lib/web/desktop.go:61 github.com/gravitational/teleport/lib/web.
(*Handler).desktopConnectHandle
/src/teleport/lib/web/apiserver.go:2695 github.com/gravitational/teleport/lib/
web.(*Handler).WithClusterAuth.func1
[ .. SNIP ..]
[2022-08-08T09:30:54Z ERROR rdp_client] RDP read failed: TryError("failed to
handle remote copy")
```

Impact

Low. When using the Teleport RDP client, a potentially malicious application can prevent the user from accessing the Windows instance.

Complexity

High. An attacker needs access to the RDP server via Teleport and has to run a malicious application.

Remediation

Limit the number of characters the user can store inside the clipboard.

Alternatively, handle the excessive number of characters correctly and without causing a crash.

Resources

- GitHub Issue - Desktop Access: Clipboard support #9980
  https://github.com/gravitational/teleport/issues/9980

TEL-Q322-4 - Rdpclient Denial Of Service Via Missing ScNet Or ScCore GCC Messages	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	https://github.com/gravitational/rdp-rs/blob/004207e2edfbc6a57fd04daf6bcade93bcc3495e/src/core/gcc.rs#L410
Status	Closed

Description

Teleport RDP client uses an external library, named rdp-rs. The library processes several pieces of data passed from the RDP server to the client through a GCC layer (Generic Conference Control), including `ScNet` and `ScCore` message types.

During the RDP connection initialization, in the case where the server does not send at least one of the `channelIdArray` or `rdpVersion` fields, the library still queries for them. This results in the fatal crash of the Teleport cluster. Under such circumstances, an administrator would have to rerun it manually.

Reproduction Steps

The client reads the following data, but the library does not enforce any checks on whether the server sends all these fields:

```
// rdp-rs/blob/master/src/core/gcc.rs#L410

match MessageType::from(cast!(DataType::U16, header["type"])?) {
    MessageType::ScCore => {
        let mut server_core = server_core_data();
        server_core.read(&mut Cursor::new(buffer))?;
        result.insert(MessageType::ScCore, server_core);
    }
    MessageType::ScSecurity => {
        let mut server_security = server_security_data();
        server_security.read(&mut Cursor::new(buffer))?;
        result.insert(MessageType::ScSecurity, server_security);
    }
    MessageType::ScNet => {
        let mut server_net = server_network_data();
        server_net.read(&mut Cursor::new(buffer))?;
        result.insert(MessageType::ScNet, server_net);
    }
    _ => println!(
        "GCC: Unknown server block {:?}",
        cast!(DataType::U16, header["type"])?
    ),
}
```

Immediately after this code, there is a comment stating that all fields are important, however, there is no check if they are actually present.

```
// All section are important
Ok(ServerData {
    channel_ids: cast!(
        DataType::Trame,
        result[&MessageType::ScNet]["channelIdArray"]
    )?
    .into_iter()
    .map(|x| cast!(DataType::U16, x).unwrap())
    .collect(),
    rdp_version: Version::from(cast!(
        DataType::U32,
        result[&MessageType::ScCore]["rdpVersion"]
    )?),
})
```

If during the packet processing any of the highlighted messages are missing,

```
pub enum MessageType {
    //server -> client
    ScCore = 0x0C01,
    ScSecurity = 0x0C02,
    ScNet = 0x0C03,
    //client -> server
    CsCore = 0xC001,
    CsSecurity = 0xC002,
    CsNet = 0xC003,
    CsCluster = 0xC004,
    CsMonitor = 0xC005,
    Unknown = 0,
}
```

the application crashes with the following panic:

```
thread '<unnamed>' panicked at 'no entry found for key', lib/srv/desktop/rdp/
rdpclient/rdp-rs/src/core/gcc.rs:424:13
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
fatal runtime error: failed to initiate panic, error 5
SIGABRT: abort
PC=0x7ff8193e300e m=0 sigcode=0
signal arrived during cgo execution

goroutine 961 [syscall]:
runtime.cgocall(0x7b97a20, 0xc001365260)
    /usr/local/opt/go/libexec/src/runtime/cgocall.go:157 +0x5c fp=0xc001365238
sp=0xc001365200 pc=0x400553c
github.com/gravitational/teleport/lib/srv/desktop/rdp/
rdpclient._Cfunc_connect_rdp(0x1, 0x6000002085a0, 0x600000008410, 0x462,
0xc002deb680, 0x4a7, 0xc002f84000, 0x462, 0x2b9, 0x1, ...)
    _cgo_gotypes.go:224 +0x4e fp=0xc001365260 sp=0xc001365238 pc=0x743192e
```

Impact

Possibly high. An attacker, controlling the RDP server where the client connects, can cause a Denial of Service (DoS) of all components to terminate the Teleport daemon.

Complexity

High. The attacker has to set up a malicious RDP server and send the malformed packet back to the client.

Remediation

Terminate the connection when the server does not send the `ScCore` **or** `ScNet` **messages.** Alternatively, ensure that the Teleport application can recover from the unhandled exceptions.

TEL-Q322-5 - Rdpclient Denial Of Service Via Requesting ProtocolHybrid	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	https://github.com/gravitational/rdp-rs/blob/004207e2edfbc6a57fd04daf6bcade93bcc3495e/src/core/x224.rs#L233
Status	Closed

Description

When the RDP client calls the connect function, it leaves the `authentication_protocol` parameter in the X.224 Connection Request PDU set to `None`.

Later, the connect function calls `Self::read_connection_confirm` and expects that the server provides the `ProtocolSSL` protocol value. However, if the server decides to request the `ProtocolHybrid` instead, the `authentication_protocol` with the value of `None` is unwrapped, resulting in a crash.

Similarly to `TEL-Q322-4`, the administrator has to rerun the Teleport service manually.

Reproduction Steps

The `x224::Client::connect` sets the fourth parameter as `None`, which is used as the `authentication_protocol` parameter.

```
// lib/srv/desktop/rdp/rdpclient/src/lib.rs
let x224 = x224::Client::connect(tpkt::Client::new(tcp), protocols, false, None,
false, false)?;

// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/x224.rs
    pub fn connect(
        mut tpkt: tpkt::Client<S>,
        security_protocols: u32,
        check_certificate: bool,
        authentication_protocol: Option<&mut dyn AuthenticationProtocol>,
        restricted_admin_mode: bool,
        blank_creds: bool,
    ) -> RdpResult<Client<S>> {
```

Later in the same `connect` function, the server has to respond with the *X.224 Connection Confirm PDU* with one of the enum values:

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/x224.rs
#[repr(u32)]
#[derive(Copy, Clone, Debug, TryFromPrimitive)]
pub enum Protocols {
    /// Basic RDP security
    /// Not supported by rdp-rs
    ProtocolRDP = 0x00,
    /// Secure Socket Layer
```

```
        ProtocolSSL = 0x01,
        /// Network Level Authentication over SSL
        ProtocolHybrid = 0x02,
        /// NLA + SSL + Quick respond
        ProtocolHybridEx = 0x08,
    }
```

Depending on the server's response, the RDP client could unwrap the `None` value:

```
        match Self::read_connection_confirm(&mut tpkt)? {
            Protocols::ProtocolHybrid => Ok(Client::new(
                tpkt.start_nla(
                    check_certificate,
                    authentication_protocol.unwrap(),
                    restricted_admin_mode || blank_creds,
                )?,
                Protocols::ProtocolHybrid,
            )),
            Protocols::ProtocolSSL => Ok(Client::new(
                tpkt.start_ssl(check_certificate)?,
                Protocols::ProtocolSSL,
            )),
```

The Teleport server crashes with the following panic:

```
$ thread '<unnamed>' panicked at 'called `Option::unwrap()` on a `None` value',
src/teleport/lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/x224.rs:241:21
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
fatal runtime error: failed to initiate panic, error 5
SIGABRT: abort
PC=0x7ff8193e300e m=16 sigcode=0
signal arrived during cgo execution
goroutine 4887 [syscall]:
runtime.cgocall(0x7b984a0, 0xc00234b260)
/usr/local/opt/go/libexec/src/runtime/cgocall.go:157 +0x5c fp=0xc00234b238
sp=0xc00234b200 pc=0x4005fbc
github.com/gravitational/teleport/lib/srv/desktop/rdp/
rdpclient._Cfunc_connect_rdp(0x3, 0x600000204560, 0x6000000142b0, 0x463,
0xc000a24900, 0x4a7, 0xc001766500, 0x462, 0x2b9, 0x1, ...)
```

Impact

Possibly high. An attacker controlling the RDP server where the client connects can cause the denial of service of all components to terminate the Teleport daemon.

Complexity

High. The attacker has to set up a malicious RDP server and send the malformed packet back to the client.

Remediation

Initialize the `authentication_protocol` **property or remove the support for the** `ProtocolHybrid`**.**

Alternatively, ensure that the Teleport application can recover from the unhandled exceptions.

Resources

- Microsoft, “4.1.2 Server X.224 Connection Confirm PDU” https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-rdpbcgr/49095420-c6ef-4256-a262-3800e1e233a7

TEL-Q322-6 - Rdpclient Denial Of Service Via read_synchronize_pdu	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	https://github.com/gravitational/rdp-rs/blob/888b15477d9aaceb83b2afcc73de040a07781d28/src/core/global.rs#L142
Status	Closed

Description

During the `Global` channel read, the server sends the synchronize PDU packet to the client. If the server specifies the `PdutypeDatapdu` PDU type, the PDU message includes the `uncompressedLength` field. A specially crafted server's response can leverage an integer overflow during the client's parsing, causing an allocation of the vector with the size *18446744073709551615*, hence crashing the Teleport service.

Similarly to `TEL-Q322-4`, the administrator has to rerun the Teleport service manually.

Reproduction Steps

First, the client reads the payload on the `Global` channel, where we assume the client's state is already at `SynchronizePDU`.

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/global.rs

ClientState::SynchronizePDU => {
   if self.read_synchronize_pdu(&mut try_let!(tpkt::Payload::Raw, payload)?)? {
        // next state is control cooperate
        self.state = ClientState::ControlCooperate;
    }
    Ok(())
}
```

The implementation calls `PDU::from_stream(stream)`:

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/global.rs

    /// Read server synchronize pdu
    /// This sent from server to client
    ///
    /// This function return true if it read the expected PDU
    fn read_synchronize_pdu(&mut self, stream: &mut dyn Read) -> RdpResult<bool>
{
        let pdu = PDU::from_stream(stream)?;
        if pdu.pdu_type != PDUType::PdutypeDatapdu {
            return Ok(false);
        }
        if DataPDU::from_pdu(&pdu)?.pdu_type != PDUType2::Pdutype2Synchronize {
            return Ok(false);
        }
        Ok(true)
```

```
        }
```

`PDU::from_stream(stream)` builds the PDU structure from the reading stream:

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/global.rs

    /// Build a PDU structure from reading stream
    pub fn from_stream(stream: &mut dyn Read) -> RdpResult<Self> {
        let mut header = share_control_header(None, None, None);
        header.read(stream)?;
        PDU::from_control(&header)
    }
```

First, the header is created using `share_control_header` with PDU payload format. After that, it fills the header from the stream and calls `from_control`, which we revisit next.

To leverage the finding, we have to set the PDU type as `PDUType::PdutypeDatapdu` (`0x17 = 23`).

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/global.rs

    /// Build a PDU data directly from a control message
    pub fn from_control(control: &Component) -> RdpResult<Self> {
        let pdu_type = cast!(DataType::U16, control["pduType"])?;
        let mut pdu = match PDUType::try_from(pdu_type)? {
            PDUType::PdutypeDemandactivepdu => ts_demand_active_pdu(),
            PDUType::PdutypeDatapdu => share_data_header(None, None, None),
            PDUType::PdutypeConfirmactivepdu => ts_confirm_active_pdu(None, None,
None),
            PDUType::PdutypeDeactivateallpdu => ts_deactivate_all_pdu(),
            _ => {
                return Err(Error::RdpError(RdpError::new(
                    RdpErrorKind::NotImplemented,
                    "GLOBAL: PDU not implemented",
                )))
            }
        };
        pdu.message.read(&mut Cursor::new(cast!(
            DataType::Slice,
            control["pduMessage"]
        )?))?;
        Ok(pdu)
    }
```

Then, the header would be created using `share_data_header`, with this format:

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/core/global.rs

/// All Data PDU share the same layout
fn share_data_header(
    share_id: Option<u32>,
    pdu_type_2: Option<PDUType2>,
    message: Option<Vec<u8>>,
) -> PDU {
    let default_message = message.unwrap_or(vec![]);
    PDU {
        pdu_type: PDUType::PdutypeDatapdu,
        message: component![
            "shareId" => U32::LE(share_id.unwrap_or(0)),
            "pad1" => 0 as u8,
            "streamId" => 1 as u8,
```

```
                "uncompressedLength" =>
DynOption::new(U16::LE(default_message.length() as u16 + 18), | size |
MessageOption::Size("payload".to_string(), size.inner() as usize - 18)),
                "pduType2" => pdu_type_2.unwrap_or(PDUType2::Pdutype2ArcStatusPdu) as
u8,
                "compressedType" => 0 as u8,
                "compressedLength" => U16::LE(0),
                "payload" => default_message
            ],
        }
    }
```

The important part here is the `uncompressedLength` value, that could overwrite the payload size attribute via `size.inner() as usize - 18`, causing integer overflow. This value on 64 bit platform could be as big as `0xffffffffffffffff = 18446744073709551615L`.

Inside `from_control`, `pdu.message.read` reads the `pduMessage` data:

```
// lib/srv/desktop/rdp/rdpclient/rdp-rs/src/model/data.rs

impl Message for Component {
...
    fn read(&mut self, reader: &mut dyn Read) -> RdpResult<()> {
        let mut filtering_key = HashSet::new();
        let mut dynamic_size = HashMap::new();
        for (name, value) in self.into_iter() {
            // ignore filtering keys
            if filtering_key.contains(name) {
                continue;
            }

            if dynamic_size.contains_key(name) {
                let mut local = vec![0; dynamic_size[name]];
                reader.read_exact(&mut local)?;
                value.read(&mut Cursor::new(local))?;
            } else {
                value.read(reader)?;
            }

            match value.options() {
                MessageOption::SkipField(field) => {
                    filtering_key.insert(field);
                }
                MessageOption::Size(field, size) => {
                    dynamic_size.insert(field, size);
                }
                MessageOption::None => (),
            }
        }
        Ok(())
    }
```

Here the `MessageOption` enum is used, allowing the overwriting of the parent's attribute in the tree.

```
/// Allow to a son to inform parent of something special
///
/// IN tree type a son can control parser of a parent node
/// by providing some type depend fields
///
/// This is control by the options function of Message Trait
pub enum MessageOption {
```

```
            /// You ask to skip a field
            /// during reading operation
            SkipField(String),
            /// You ask to limit the size of reading buffer
            /// for a particular field
            Size(String, usize),
            /// Non option
            None,
        }
```

Therefore, `uncompressedLength` can overflow the payload size when the payload size equals zero.

We dynamically confirmed that this causes a crash of the Teleport daemon when the `vec` object is created.

```
thread 'main' panicked at 'capacity overflow', library/alloc/src/raw_vec.rs:518:5
```

Impact

Possibly high. An attacker controlling the RDP server where the client connects can cause the denial of service of all components to terminate the Teleport daemon.

Complexity

High. The attacker has to set up a malicious RDP server and send the malformed packet back to the client.

Remediation

Ensure the uncompressed length for the PDUType::PdutypeDatapdu object cannot overflow and limit its size. Alternatively, ensure that the Teleport application can recover from the unhandled exceptions.

TEL-Q322-7 - Insecure TOTP MFA Seed Removal	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Teleport client tool (tsh)
Status	Closed

Description

The `tsh` client allows adding the 2FA TOTP authenticator via the command line. In that case, the picture containing a secret seed is stored on the file system inside the `tmp` directory. After the user scans the seed QR and confirms the code, the file is removed.

The removal is using `os.Remove()` without overwriting the file's content. If an attacker has raw access to the filesystem or local access to the computer used to import the seed, they can relatively easily recover the removed secret.

Reproduction Steps

Issue the following command to add a TOTP MFA:

```
build/tsh mfa add --insecure
```

Observe the removal procedure in the source code:

```
// tool/tsh/mfa.go
    log.Debugf("Opened QR code via %q", imageViewer)
    return func() {
        if err := os.Remove(imageFile.Name()); err != nil {
            log.WithError(err).Debugf("Failed to clean up temporary QR code file
%q", imageFile.Name())
        }
        if err := cmd.Process.Kill(); err != nil {
            log.WithError(err).Debug("Failed to stop the QR code image viewer")
        }
    }, nil
```

Impact

High. In case of physical access to the computer used to add MFA, an attacker can recover the 2FA seed and take over the victim's account.

Complexity

High. Physical access to the victim's computer is needed. Alternatively, the seed can be recovered using root access, where an attacker can access the raw disk device using standard off-the-shelf tools.

Remediation

Repeatedly overwrite the seed file content to make it harder to recover the data. For this purpose, there are several Golang libraries, such as shred or go-wiper. Note that Doyensec have not reviewed these implementations.

Resources

- GitHub lu4p/shred
  https://github.com/lu4p/shred
- GitHub 0x9ef/go-wiper
  https://github.com/0x9ef/go-wiper

TEL-Q322-8 - Log Injection in Audit Events	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	https://github.com/gravitational/teleport/blob/master/lib/events/emitter.go
Status	Closed

Description

While reviewing the web application source code, Doyensec discovered a Log Injection vulnerability affecting the *Teleport Audit Events*, where it is possible to add crafted content to the audit debug logs.

This specific log injection attack type is related to an attacker being able to inject arbitrary text into some parameters that are displayed as regular proxy logs.

Reproduction Steps

Follow the example below to craft a fake log entry in the Teleport Database service.

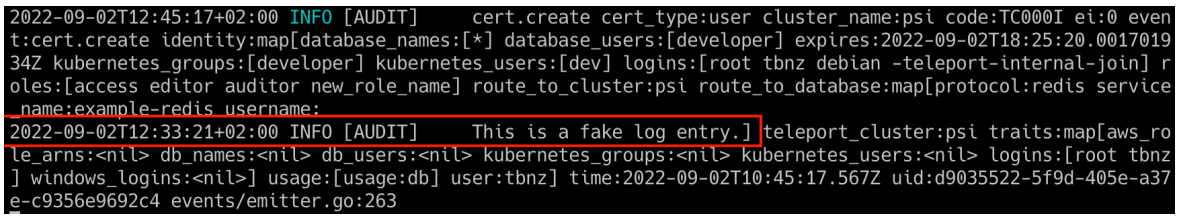
1. Create a new Database service with the following command:

```
teleport db start –token={token} –auth-server={auth-server} --name=test
–protocol=redis –uri={uri}
```

2. Log in and connect to the database:

```
tsh db login --db-user="$(python3 -c 'print("\n2022-09-02T12:33:21+02:00 INFO
[AUDIT]     This is a fake log entry.")')" example-redis
```

3. Observe that the newline character was interpreted, creating a new entry visually almost indistinguishable from a legit one.



Impact

Low. An attacker can insert malicious data and false entries into the logs and ultimately corrupt the logs to cover the tracks of a hacking attempt.

Complexity

Low. Injecting the new entries into the audit events is trivial.

Remediation

Remove special characters such as newlines or tabs from the log entries.

Attackers can also inject an excessive number of spaces, but they need to obtain information such as the terminal used or the screen size.

Resources

- OWASP , “Log Injection”
  https://owasp.org/www-community/attacks/Log_Injection

TEL-Q322-9 - Arbitrary Length Data Reading Via go-mssqldb	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	go-mssqldb third-party library https://github.com/gravitational/teleport/blob/5be324c89d7ca56b374fe45dbbfcb2aa853548b1/lib/srv/db/sqlserver/protocol/rpc_request.go#L112
Status	Closed

Description

The application depends on the go-mssqldb client for connecting to Microsoft SQL Server DBMS. For the variable-sized data not known in advance, the SQL protocol supports *Partially Length-prefixed Bytes*:

> *Unlike fixed or variable byte stream formats, Partially length-prefixed bytes (PARTLENTYPE), introduced in TDS 7.2, do not require the full data length to be specified before the actual data is streamed out. Thus, it is ideal for those applications where the data length is not known upfront (that is, xml serialization).*

In specific scenarios, the server can propose to allocate 2^64 bytes, resulting in the crash of the Teleport daemon.

Reproduction Steps

The client RPC Request packet is read using the third line of the output:

```
tds := mssql.NewTdsBuffer(data[int(r.Size())-r.Len():], r.Len())
ti := mssql.ReadTypeInfo(tds)
val := ti.Reader(&ti, tds)
```

In case we pass the variable length data type using `ReadTypeInfo`, the `readVarLen` function is invoked:

```
// go-mssqldb/types.go

func readTypeInfo(r *tdsBuffer) (res typeInfo) {
..
 default: // all others are VARLENTYPE
        readVarLen(&res, r)
 }
 return
}
```

The `readVarLen` function supports `typeXml`, where it sets the `ti.Reader` pointer to `readPLPType`:

```
func readVarLen(ti *typeInfo, r *tdsBuffer) {
```

```
..
 case typeXml:
        ti.XmlInfo.SchemaPresent = r.byte()
        if ti.XmlInfo.SchemaPresent != 0 {
               // dbname
               ti.XmlInfo.DBName = r.BVarChar()
               // owning schema
               ti.XmlInfo.OwningSchema = r.BVarChar()
               // xml schema collection
               ti.XmlInfo.XmlSchemaCollection = r.UsVarChar()
        }
        ti.Reader = readPLPType
```

Finally, `readPLPType` reads the buffer size using `r.uint64()` and uses the size value for creating a new buffer. See the highlighted lines:

```
// partially length prefixed stream
// http://msdn.microsoft.com/en-us/library/dd340469.aspx
func readPLPType(ti *typeInfo, r *tdsBuffer) interface{} {
 size := r.uint64()
 var buf *bytes.Buffer
 switch size {
 case _PLP_NULL:
        // null
        return nil
 case _UNKNOWN_PLP_LEN:
        // size unknown
        buf = bytes.NewBuffer(make([]byte, 0, 1000))
 default:
        buf = bytes.NewBuffer(make([]byte, 0, size))
```

The last remaining thing is to verify that `r.uint64()` does not enforce any size check validation, which is indeed true:

```
// go-mssqldb/buf.go

func (r *tdsBuffer) uint64() uint64 {
 var buf [8]byte
 r.ReadFull(buf[:])
 return binary.LittleEndian.Uint64(buf[:])
}
```

Since the server can ask the client to allocate up to *18446744073709551616* bytes, the findings lead to a denial of service via memory exhaustion.

Impact

Possibly high. Due to the insecure design, an attacker controlling the MS SQL server, where the client connects, can cause a Denial of Service which would ultimately lead to the crash of the Teleport daemon.

Complexity

High. The attacker has to set up a malicious MS SQL server and send the malformed packet back to the client.

Remediation

The “go-mssqldb” library would need to enforce the maximum size of the data read from the client. This would require a change in the upstream version or forking and patching.

Resources

- Microsoft, “Data Type Dependent Data Streams" https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-tds/3f983fde-0509-485a-8c40-a9fa6679a828

TEL-Q322-10 - Missing Data Length Limitation For SharedDirectory Packets	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	Windows Desktop Directory Sharing https://github.com/gravitational/teleport/blob/master/lib/srv/desktop/tdp/proto.go
Status	Closed

Description

The Teleport Desktop Access protocol (TDP) supports several message types when communicating with the remote server. Two message types required for the directory sharing allow reading a `uint32` length from a socket. Since there is no length validation, this allows allocating 4GB of memory and potential DoS attacks.

Note that the official documentation clearly states that there is no limit for the file resources, but the recommended size is 32 MB to avoid impacting network bandwidth and system resources.

Reproduction Steps

The highlighted variables below allow the handling of 4GB of data.

```
// lib/srv/desktop/tdp/proto.go

func decodeSharedDirectoryReadResponse(in peekReader)
(SharedDirectoryReadResponse, error) {
  // ..
 var completionID, errorCode, readDataLength uint32
  // ..
 err = binary.Read(in, binary.BigEndian, &readDataLength)
 if err != nil {
        return SharedDirectoryReadResponse{}, trace.Wrap(err)
 }
 readData := make([]byte, int(readDataLength))
 if _, err := io.ReadFull(in, readData); err != nil {
        return SharedDirectoryReadResponse{}, trace.Wrap(err)
 }
  // ..
}

func decodeSharedDirectoryWriteRequest(in peekReader)
(SharedDirectoryWriteRequest, error) {
  // ..
  var completionID, directoryID, writeDataLength uint32
  // ..
  err = binary.Read(in, binary.BigEndian, &writeDataLength)
 if err != nil {
        return SharedDirectoryWriteRequest{}, trace.Wrap(err)
```

```
        }

        writeData := make([]byte, int(writeDataLength))
        if _, err := io.ReadFull(in, writeData); err != nil {
                return SharedDirectoryWriteRequest{}, trace.Wrap(err)
        }
        // ..
}
```

Use the following test case with the packet length 0xffffffff to reproduce the finding:

```
$ cat testdata/fuzz/FuzzProtoDecode/
848daf932448abc3253016d34663535cf4878a1ab58de4c56fe06536e3310baf
go test fuzz v1
[]byte("\x14\x14\x14\x14\xad\xd9\x05\xef\xf9\xff\xff\xff\xff\x14\x1400")

$ go test -run=FuzzProtoDecode/
848daf932448abc3253016d34663535cf4878a1ab58de4c56fe06536e3310baf -test.memprofile
mem.out
```

With go profiler, display the top10 memory allocations to confirm the issue:

```
$ go tool pprof mem.out
(pprof) top
Showing nodes accounting for 4GB, 99.74% of 4.01GB total
Dropped 73 nodes (cum <= 0.02GB)
Showing top 10 nodes out of 12
      flat  flat%   sum%        cum   cum%
       4GB 99.74% 99.74%        4GB 99.74%  github.com/gravitational/teleport/
lib/srv/desktop/tdp.decodeSharedDirectoryReadResponse
         0     0% 99.74%        4GB 99.74%  github.com/gravitational/teleport/
lib/srv/desktop/tdp.Decode
         0     0% 99.74%        4GB 99.74%  github.com/gravitational/teleport/
lib/srv/desktop/tdp.FuzzProtoDecode.func1
         0     0% 99.74%        4GB 99.74%  github.com/gravitational/teleport/
lib/srv/desktop/tdp.FuzzProtoDecode.func1.1
         0     0% 99.74%        4GB 99.74%  github.com/gravitational/teleport/
lib/srv/desktop/tdp.decode
         0     0% 99.74%        4GB 99.74%  github.com/stretchr/testify/
assert.NotPanics
         0     0% 99.74%        4GB 99.74%  github.com/stretchr/testify/
assert.didPanic
         0     0% 99.74%        4GB 99.74%  github.com/stretchr/testify/
require.NotPanics
         0     0% 99.74%        4GB 99.74%  reflect.Value.Call
         0     0% 99.74%        4GB 99.74%  reflect.Value.call
```

Impact

An attacker can consume the system resources of the Teleport instance and potentially prevent other users from accessing it.

Complexity

High. An attacker needs privileged access to the RDP server to affect the traffic.

Remediation

Since 4GB surpasses the recommended value by two orders of magnitude, **we recommend enforcing the file size limit or making the limit configurable.**

Similar protection is already implemented inside the `decodeString` function.

```
// lib/srv/desktop/tdp/proto.go
func decodeString(r io.Reader, maxLen uint32) (string, error) {
 var length uint32
 if err := binary.Read(r, binary.BigEndian, &length); err != nil {
        return "", trace.Wrap(err)
 }

 if length > maxLen {
        return "", trace.BadParameter("TDP string length exceeds allowable
limit")
 }
```

Resources

- Gravitational, Teleport Desktop Access Directory Sharing
  https://goteleport.com/docs/desktop-access/directory-sharing/#file-size-restrictions

TEL-Q322-11 - Missing Rate Limit For Proxy API Web Endpoints	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Teleport Core
Status	Closed

Description

The Teleport Proxy web API does not implement rate-limiting, including completely anonymous login activities. While the maintainers were already aware of the issue since 2020, the attack surface has significantly increased by extending the application functionalities and adding more endpoints.

An attacker can easily spam the *Audit Logs* by sending several thousands of characters potentially arbitrary number of times. Moreover, even when the application only supports a restricted alphabet in the parameters, the user-supplied values are still reflected in the error message.

Reproduction Steps

We identified the following endpoints to spam the Audit Logs without authentication:

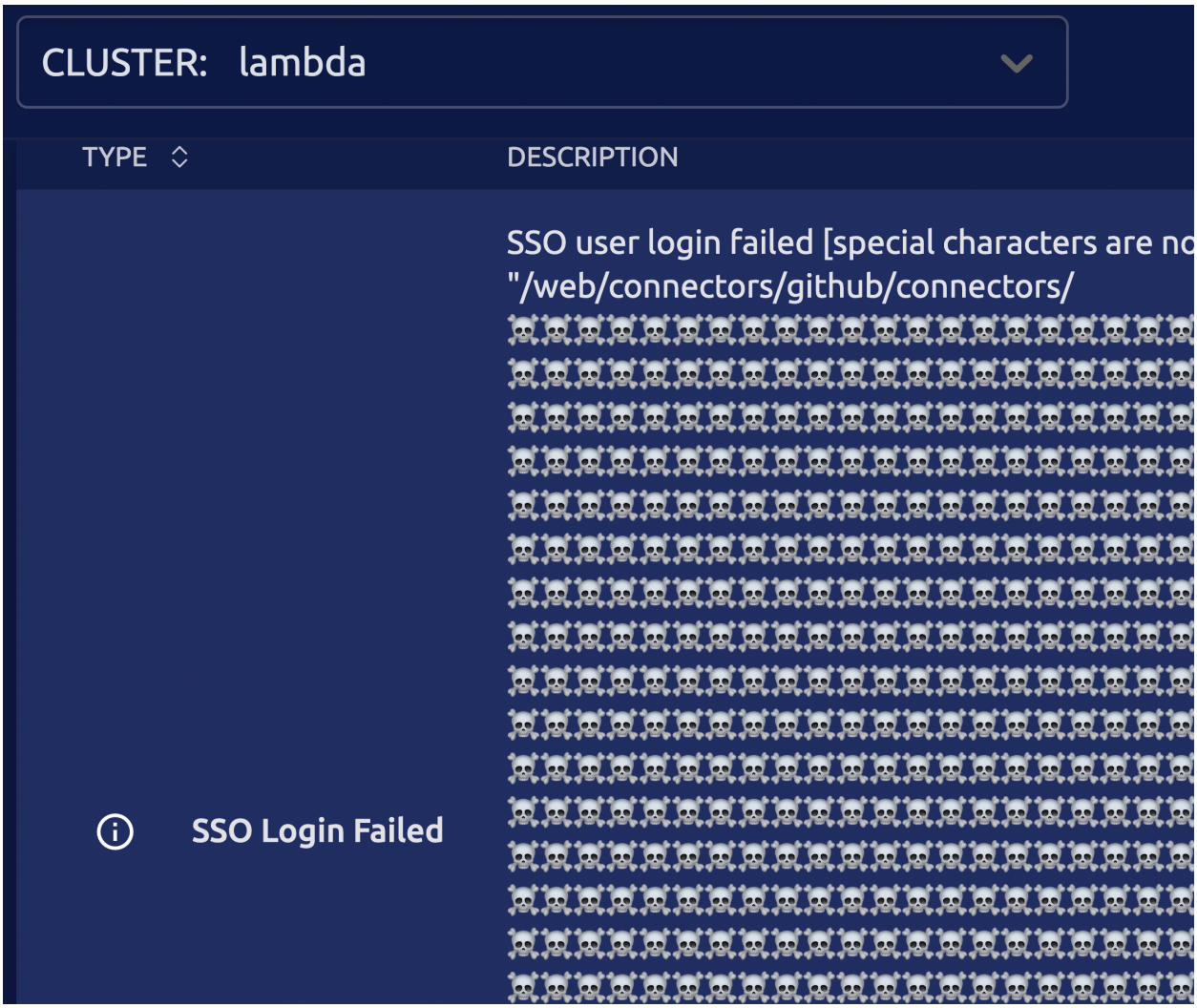
- `/webapi/oidc/login/console`
- `/webapi/saml/login/console`
- `/webapi/github/login/console`
- `/webapi/sessions`

To reproduce the finding, repetitively issue the following HTTP request in which the `connector_id` parameter can be adjusted to contain several thousands of characters:

```
POST /v1/webapi/github/login/console HTTP/1.1
Host: lambda:3080
Content-Type: application/json; charset=utf-8
Accept: application/json
Content-Length: 12078

{
"redirect_url":"https://testing",
"public_key":"AAA=",
"connector_id":"\u2620\u2620\u2620"}
```

The resulting *Audit Log* window after we submitted 2000 Unicode `\u2620` characters:



Impact

Medium. An attacker can easily spam and overflow the Audit Logs, making it challenging or impossible to find legitimate entries.

Complexity

Medium. Basic web application skills are required to identify and exploit the vulnerability.

Remediation

Implement rate limiting for all Proxy API Web Endpoints. At the very least, protect the endpoints available without authentication, since these generate *Audit Logs*.

Resources

- GitHub Issue - Rate limit failed login requests. #4554
  https://github.com/gravitational/teleport/issues/4554

- GitHub Issue - Rate-limit proxy web API endpoints #4330
  https://github.com/gravitational/teleport/issues/4330

TEL-Q322-12 - Missing Protocol Check For openExternal URL	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	(Teleterm) https://github.com/gravitational/webapps/blob/27c615b3ff6f317a85fac4aa28b8e73fa4aa0d28/packages/teleterm/src/main.ts#L120
Status	Closed

Description

When the renderer process in Teleport Connect creates a new window, the `isUrlSafe` function verifies whether the URL is secure to open. The current implementation checks the *host* and *pathname* before it passes the input to the ElectronJS `openExternal` function[1].

However, the code accepts arbitrary protocols registered by the operating system allowing the execution of third-party applications. This insecure design increases the potential attack surface available to an attacker.

The highlighted code shows where the host and path check is applied, but the protocol check is missing.

```
// packages/teleterm/src/main.ts
  app.on('web-contents-created', (_, contents) => {
    // ..

    contents.setWindowOpenHandler(details => {
      const url = new URL(details.url);

      function isUrlSafe(): boolean {
        if (url.host === 'goteleport.com') {
          return true;
        }
        if (
          url.host === 'github.com' &&
          url.pathname.startsWith('/gravitational/')
        ) {
          return true;
        }
      }

      // Open links to documentation and GitHub issues in the external browser.
      // They need to have `target` set to `_blank`.
      if (isUrlSafe()) {
        shell.openExternal(url.toString());
      } else {
      return { action: 'deny' };
  // ..
  });
```

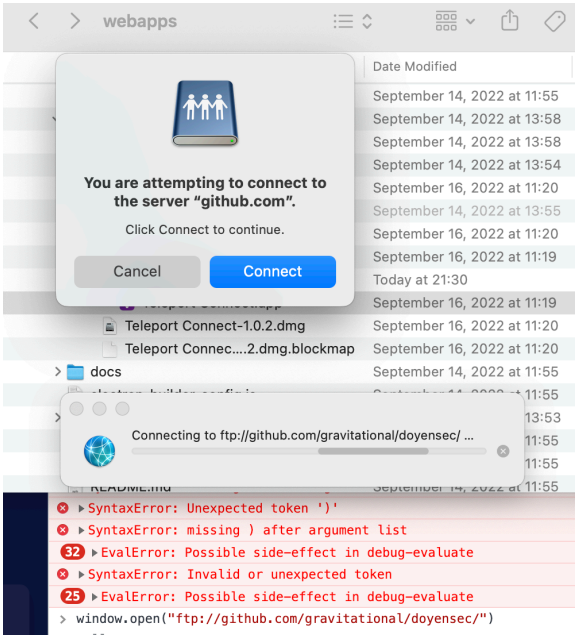
[1] https://www.electronjs.org/docs/latest/api/shell#shellopenexternalurl-options

Reproduction Steps

To dynamically reproduce this issue, open the Teleport Connect application and click on the “Toggle Developers Tools”.

Execute the following code and verify the execution of the default macOS FTP client:

```
window.open("ftp://github.com/gravitational/doyensec/")
```



Impact

An attacker can use various application protocol handlers (e.g., `http://`, `ftp://`, `jar://`, `skype://`). Protocols supporting plaintext communication, such as HTTP, are vulnerable to MITM attacks. Since all currently whitelisted hosts deploy *HTTP Strict Transport Security*, it mitigates MITM, protocol downgrade attacks, and cookie stealing.

Complexity

An attacker has to leverage a different finding (e.g., Cross Site Scripting) to call the `window.open` function, or change the `document.location`. An attacker can use various application protocol handlers (e.g., `http://`, `ftp://`, `jar://`, `skype://`), however there are limited whitelisted hosts (namely goteleport.com and github.com). Due to this limitation, we consider the issue as Informational only.

Remediation

Only permit protocols that the application is intended to open. For instance, we recommend allowing only HTTPS by parsing the user-supplied input with Node’s `URL()` and verifying the `protocol` property.

Resources

- Benjamin Altpeter, “The dangers of Electron's shell.openExternal()—many paths to remote code execution”
  https://benjamin-altpeter.de/shell-openexternal-dangers/

- OpenJS Foundation, "Do not use shell.openExternal with untrusted content"
  https://www.electronjs.org/docs/latest/tutorial/security#15-do-not-use-shellopenexternal-with-untrusted-content

TEL-Q322-13 - Missing Hardening For BrowserWindow	
Severity	**Informational**
Vulnerability Class	Security Misconfiguration
Component	(Teleterm) https://github.com/gravitational/webapps/blob/master/packages/teleterm/src/mainProcess/mainProcess.ts
Status	Open (Risk Accepted)

Description

In Electron.js, the `webPreferences` object of `BrowserWindow`[2] controls its web page's features. When working with Electron, it is crucial to understand the security settings on which every `BrowserWindow` is instantiated play a critical role in its security. While many security flags are enabled by default as new Electron versions are released, some security features may not be natively enabled, or their interaction could lead to unexpected dangerous behaviors under certain circumstances.

Because of this, we strongly advise to explicitly change the following `webPreferences` options for all the *Teleport Terminal* windows:

- **sandbox** to `true`
  This option creates a browser window with a sandboxed renderer. When the sandbox is enabled, the renderers can only make changes to the system by delegating tasks to the main process via IPC, accessing the node APIs in this way. The only exception is the preload script, which has access to a subset of the Electron renderer API. Another consequence of this flag is that sandboxed renderers won't modify any of the default JavaScript APIs. Consequently, some APIs, such as `window.open`, will work as they do in Chromium (i.e., they do not return a `BrowserWindowProxy`).

Reproduction Steps

Currently the `createWindow` function in `webapps/packages/teleterm/src/mainProcess/mainProcess.ts` instantiates a new `BrowserWindow` with the following settings:

```
createWindow(): void {
  const windowState = this.getWindowState();
  const window = new BrowserWindow({
    x: windowState.x,
    y: windowState.y,
    width: windowState.width,
    height: windowState.height,
    backgroundColor: theme.colors.primary.darker,
    minWidth: 400,
    minHeight: 300,
```

[2] https://www.electronjs.org/docs/api/browser-window#new-browserwindowoptions

```
      autoHideMenuBar: true,
      title: 'Teleport Connect Preview',
      webPreferences: {
        devTools: this.settings.dev,
        webgl: false,
        enableWebSQL: false,
        safeDialogs: true,
        contextIsolation: true,
        nodeIntegration: false,
        preload: path.join(__dirname, 'preload.js'),
      },
    });
```

As can be easily observed, the `sandbox` option is not enabled.

Impact

An attacker could leverage the lack of `webPreferences` security features to carry out attacks more easily.

Complexity

An attacker must be able to run arbitrary JavaScript code on the renderer isolated world in the first place. Since `contextIsolation` is set, this option provides JavaScript context isolation for preload scripts, as implemented in Chrome Content Scripts. Therefore, different JavaScript contexts between renderers, preload scripts, and JavaScript contexts between renderers and Electron's framework code are present, preventing trivial exploitations.

Remediation

Enable `sandbox` **when creating the main** `BrowserWindow` **used by Teleterm.** Enabling sandbox requires a redesign of the current logging service and potentially other components.

Resources

- OpenJS Foundation, "BrowserWindow webPreferences" https://www.electronjs.org/docs/latest/api/browser-window#new-browserwindowoptions
- OpenJS Foundation, "Process Sandboxing" https://www.electronjs.org/docs/latest/tutorial/sandbox

TEL-Q322-14 - Outdated Electron Version In Use	
Severity	**Low**
Vulnerability Class	Components With Known Vulnerabilities
Component	(Teleterm) https://github.com/gravitational/webapps/blob/27c615b3ff6f317a85fac4aa28b8e73fa4aa0d28/packages/teleterm/package.json#L43
Status	Closed

Description

An application built with an older version of Electron, Chromium, or Node.js is an easier target than an application that uses more recent versions of those components. Generally speaking, security issues and exploits for older versions of Chromium and Node.js are more widely available. When developing an Electron-based application, the latest available version of Electron should always be used when possible, even whenever a new major version is released.

Several security issues have been fixed in versions of Electron.js, succeeding the one shipped with Teleport Terminal for Desktop (currently on Electron.js v19.0.0). Some of them consist of Chromium back-ported fixes. Because Electron applications are web applications running in the Chromium engine, they may be vulnerable through some of the same attack vectors as the Chromium browser. Because of this, Electron security releases currently follow Chromium's security releases.

For example, the following security fixes were fixed in version 20.1.2:

- Security: backported fix for CVE-2022-3038. #35549
- Security: backported fix for CVE-2022-3040. #35553
- Security: backported fix for CVE-2022-3041. #35557
- Security: backported fix for CVE-2022-3075. #35544

These bugs remain unfixed and their risk should be taken into account. A recent Black Hat USA 2022 presentation highlighted the risk of running an outdated Electron due to n-days Chromium vulnerabilities.

Impact

An attacker may abuse the aforementioned Chromium security issues to evade the isolation and achieve remote code execution.

Complexity

Medium to High, an attacker needs to be able to run arbitrary JavaScript code in the context of the Electron application to leverage the vulnerabilities mentioned above.

Remediation

Update to the latest stable Electron.js version (v. 20.0.0) to benefit from the latest security fixes.

TEL-Q322-15 - BrowserWindow Insecure Preload Leading to RCE	
Severity	**Medium**
Vulnerability Class	Insecure Design
Component	(Teleterm) https://github.com/gravitational/webapps/blob/27c615b3ff6f317a85fac4aa28b8e73fa4aa0d28/packages/teleterm/src/preload.ts#L38
Status	Open (Risk Accepted)

Description

In *Electron.js*, the `preload` attribute of the `webPreferences`[3] controls the definition of preload scripts. Preload scripts contain code that executes in a renderer process before its web content begins loading. These scripts run within the renderer context, but are granted more privileges by having access to *Node.js* APIs. This powerful mechanism can introduce functionalities that can be abused by untrusted code running in the context of the renderer (e.g., XSS).

In *Teleport Connect*, the preload exposes access to a multitude of capabilities:

```
async function getElectronGlobals(): Promise<ElectronGlobals> {
  const [addresses, credentials] = await Promise.all([
    mainProcessClient.getResolvedChildProcessAddresses(),
    getClientCredentials(runtimeSettings),
  ]);
  const tshClient = createTshClient(addresses.tsh, credentials.tsh);
  const ptyServiceClient = createPtyService(
    addresses.shared,
    credentials.shared,
    runtimeSettings
  );

  return {
    mainProcessClient,
    tshClient,
    ptyServiceClient,
    loggerService,
  };
}
```

These services and clients are used throughout the application to implement Teleterm's functionalities, hence they are directly exposed to the renderer. One of these capabilities (namely `ptyServiceClient`) can be leveraged to compromise the user's workstation as it allows code execution. In detail, the `ptyServiceClient.createPtyProcess` allows invoking a specific command (`pty.shell`) which is executed by the user's shell.

[3] https://www.electronjs.org/docs/api/browser-window#new-browserwindowoptions

Reproduction Steps

Please follow the next steps to reproduce the finding:

1. Open the *Teleport Connect* application
2. Click on "*Toggle Developer Tools*" to simulate a Cross-Site Scripting vulnerability
3. Execute the following JavaScript payload:

```
var cmd = await ele.ptyServiceClient.createPtyProcess({
  "uri": "/docs/18IUn71Df-iBdgMLySCzH",
  "title": "/Users/ikki/doyensec-2022.teleport.sh",
  "kind": "pty.shell",
  "initCommand": "touch /tmp/doyensec",
  "rootClusterId": "doyensec-2022.teleport.sh",
  "status": "connected",
  "cwd": "/Users/ikki",
  "proxyHost": "doyensec-2022.teleport.sh:443",
  "clusterName": "doyensec-2022.teleport.sh"
})
cmd.process.start()
```

Verify that the `touch /tmp/doyensec` command has been successfully executed on the local machine.

Impact

A malicious attacker with the ability to execute JavaScript (e.g., XSS) can compromise the victim's host.

Complexity

This issue requires arbitrary JavaScript execution (e.g., XSS) within the *Teleport Connect* desktop application context.

Remediation

This specific issue affecting `ptyServiceClient` can be mitigated by removing the dangerous `pty.shell` command type. The application is already whitelisting commands in order to limit the execution to `tsh` supported commands.

We believe that other objects and functions exposed by the preload could also lead to vulnerabilities. Hence, we suggest reviewing and performing hardening on all exposed functions. For instance, the `mainProcessClient.fileStorage` functions allow read/write operations on the application's `app_state.json` file. Given the limited time of the engagement, we did not explore this issue further.

Resources

- Luca Carettoni, "Preloading Insecurity In Your Electron"
  https://doyensec.com/resources/Asia-19-Carettoni-Preloading-Insecurity-In-Your-Electron.pdf

TEL-Q322-16 - Privilege Escalation Patterns In Multiple IAM Roles	
Severity	**Medium**
Vulnerability Class	Insecure Design
Component	AWS IAM Roles: tenant_operator cloudteam-dev-role cloudteam-stage-role cloudteam-prod-role
Status	Closed

Description

Privilege escalation is a common attack to obtain more permissions within a compromised system or infrastructure. Attackers can find and exploit presumably "limited" permissions to escalate their privileges.

The `AdministratorAccess` permission policy provides full administrator privileges in AWS:

```
"Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        }
    ]
```

Any entity assigned with the above statement has the permissions to perform any action on every resource.

During our engagement, we discovered multiple roles with a non-hardened set of permissions allowing them to potentially obtain full admin permissions.

Misconfigured users:

- Infrastructure roles : `tenant_operator`
- Special roles : `cloudteam-dev-role`, `cloudteam-stage-role` and `cloudteam-prod-role`

Tenant_operator privilege escalation pattern

The tenant operator policy is defined at `cloud/deploy/terraform/teleport-cloud.tf:119`

```
data "aws_iam_policy_document" "tenant_operator" {
  statement {
    # Due to it's nature, tenant-operator requires full access for
    # many different (and potentially dangerous) actions.
    #tfsec:ignore:aws-iam-no-policy-wildcards
```

```
    resources = ["*"]
    actions = [
      …[REDACTED]…
      "iam:AttachRolePolicy",
      "iam:PutRolePolicy",
      …[REDACTED]…
    ]
  }
  statement {
    actions = ["sts:AssumeRole"]
    resources = [
      "arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/tenant-*",
    ]
  }
}
```

The `iam:AttachRolePolicy` permission allows attaching an IAM policy (e.g., `AdministratorAccess`) to a role. Secondly, the `iam:PutRolePolicy` permission allows the creation of an inline policy for a role.

Both of the cited permissions are granted without limitations on the attachable policies. The missing limitation could be combined with the `sts:AssumeRole` on any tenant role to attach `AdministratorAccess` to a tenant and assume its full administrator role.

Special roles privilege escalation patterns

As stated in `cloud/deploy/terraform/README.md:13`

```
All operations in AWS need to go through special roles. Those are `cloudteam-dev-role`, `cloudteam-stage-role`, and `cloudteam-prod-role` respectively.
```

The three super users are not defined in the terraform files, their privileges were observed by using the testing credentials.

In particular, while analyzing `cloudteam-stage-role`, it was found to have multiple permissions that allow becoming full admin (e.g., with `AdministratorAccess` policy).

- `iam:PutUserPolicy`
  An attacker can create an inline policy for a user they have access to and grant them administrator access.

- `iam:AttachUserPolicy`
  An attacker can attach an IAM policy to an IAM user they have access to.

- `iam:PutGroupPolicy`
  An attacker can create an inline policy for a group they are part of.

- `iam:AttachGroupPolicy`
  An attacker can attach an IAM policy to a group they are a member of.

As a result, a potential attacker who compromises the reported roles will be able to execute any possible action in the targeted environment by exploiting the escalation patterns.

Reproduction Steps

An attacker with access as a `tenant_operator` could perform the following commands to escalate her privileges to a full administrator:

1. Use the `AttachRolePolicy` permission to attach `AdministratorAccess` to a tenant role:

   ```
   aws iam attach-role-policy --role-name "arn:aws:iam::ACCOUND_ID:role/tenant-TENANTNAME" --policy-arn “arn:aws:iam::aws:policy/AdministratorAccess”
   ```

2. Use the `AssumeRole` permission to obtain a session with the administrator tenant:

   ```
   aws sts assume-role --role-arn "arn:aws:iam::ACCOUND_ID:role/tenant-TENANTNAME" --role-session-name privesc-Session
   ```

3. Perform an action permitted to tenant operators

Impact

High. An attacker with AWS IAM knowledge could easily exploit the escalation patterns to obtain the highest privileges in the infrastructure.

Complexity

High. As per the current implementation, the attacker is more likely to be an internal threat actor. Engineers operating on the internal infrastructure may use temporary access to a machine with `tenant_operator` permissions to obtain full administrator access in the environment.

Nevertheless, even if it is less probable, external actors may compromise internal credentials in multiple ways (e.g., `TEL-Q121-7 SSRF` exposing tenant’s role credentials to users).

Remediation

We recommend applying condition blocks to limit potentially dangerous IAM permissions and align the vulnerable roles to the least privilege principle.

In AWS IAM, a condition block allows specifying conditions for policies’ effects. The condition is built with expressions using different operators (e.g., equal, less than). They are used to match the condition keys and values in the policy against keys and values in the request context.

The following example shows a policy statement limiting the `AttachRolePolicy` to a specific role:

```
{
            "Sid": "rul2",
            "Effect": "Allow",
            "Action": "iam:AttachRolePolicy",
            "Resource": [
```

```
            "arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/tenant-*"
             ],
             "Condition": {
                 "StringEquals": {
                 "iam:PolicyARN": "arn:aws:iam::aws:policy/AmazonGlacierReadOnlyAccess"
                 }
             }
        }
```

The basic example will deny any attach role policy operation involving a policy different from `AmazonGlacierReadOnlyAccess`. Consequently, trying to attach `AdministratorAccess` policy will generate an access denied error.

Condition blocks should be used to limit the reported IAM permissions to the required policies.

Resources

- Amazon Web Services, “IAM JSON policy elements: Condition”
  https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_condition.html

- Amazon Web Services, “IAM JSON policy elements: Condition operators”
  https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_condition_operators.html

TEL-Q322-17 - ELB Configured as PlainText HTTP Listener	
Severity	**Informational**
Vulnerability Class	Cryptography – Missing
Component	AWS ELB
Status	Closed

Description

Elastic Load Balancing (ELB) automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, IP addresses, and Lambda functions. Elastic Load Balancing supports three types of load balancers: Application Load Balancers, Network Load Balancers, and Classic Load Balancers.

ELB can be configured to support TLS connections, and disable HTTP. This feature enables traffic encryption between your load balancer and the clients that initiate HTTPS sessions, and for connections between your load balancer and the EC2 instances.

During our AWS configuration review, the following ELBs were configured to accept incoming plain-text HTTP connections:

- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/af04a93a042c44917b4cc218b1af5c92`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/ab5cac5a2d6a142d99e6805750a1493d`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a84245e4ec95148178793d8f1fd2049b`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/ad31448cc3fba407ebfaa1458f46e01d`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a8af0b965c7a646a6968aa152ec1e36a`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a94e55459fd9747a085734f152df1c7c`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a4105876385aa495f8bb7b2d5b040574`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a7c877f35b43c413eaf72b13861932b2`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a7413805b49ec4f76875529b4f3f027d`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a917eefacc0424cc0878784a140e6e13`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/aaad5db4e8f924155a7f2ebc4206f73e`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a481dd1173f234cc3b660ee04a75fb6e`
- `arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a3058447153204158bc8cc2a31858cbf`

- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/ac10e425e62db4bc29e8b75cb7cea04a
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a1648ae8b02ae4482bff664b0b7c21b4
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a664664f70ab8479ebae1af2a413aee7
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/michael-no-ga
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a07c4285462ea46a48dc15cbaf69e01c
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/ac7c5b1296af94d269671fa8e0f63fe6
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a0e4d93e5bd74426d84a55f95932df99
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/ab20ea035eac64144b382050c7d8df38
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/ab62712baf49445799a837254aa63b71
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a1da43df1cdfe418ab188af16108e787
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a723a0708df24446799f464eb5c3604d
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a162de997b3ee4b2697c286b285039c2
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/afe60bfa29745481fb94603593db26f7
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/adec91d7c84214ca98f270c5f23e0510
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/aa3a12c17dbfc4fc88999d5d4675be3a
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a68b4f286e068485da01f04938d5e71f
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a3d7d7d77d38e4df4a70a0398066e388
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a28f34b2f988f4d30bfec4a7338ed3b2
- arn:aws:elasticloadbalancing:us-west-2:599519581022:loadbalancer/a9666e3788f7c4edd81a967a31f42e41

Reproduction Steps

This issue can be verified using the following AWS CLI command:

```
$ aws elb describe-load-balancers –load-balancer-name <load balancer name>
```

The `ListenerDescriptions` section can be reviewed for identifying insecure listeners having `“Protocol": “HTTP”`.

Impact

None. This plain-text HTTP ELB is set up for redirects to `443/tcp` only.

Complexity

The exploitation of such misconfigurations requires an attacker with privileged network access (e.g., MiTM).

Remediation

Ensure ELBs are configured to only accept connections on HTTPS ports.

Standard HTTP connections will be blocked. This should only be done if the client application is configured to query HTTPS directly and not rely on a redirect from HTTP. No changes are required if the ELB is used for redirects only.

Resources

- Amazon Web Services, “Elastic Load Balancing > Classic Load Balancers: Security Policies” http://docs.aws.amazon.com/ElasticLoadBalancing/latest/DeveloperGuide/elb-security-policy-options.html

TEL-Q322-18 - Server-Side Encryption Not Enabled for AWS SNS Topics	
Severity	**Low**
Vulnerability Class	Cryptography – Missing
Component	AWS SNS Topics
Status	Closed

Description

Amazon's SNS Server-Side Encryption (SSE) feature protects the content of the published messages within SNS topics, making it ideal for security-sensitive applications with strict encryption compliance and regulatory requirements.

We identified that SNS does not leverage AWS's server-side encryption feature.

Reproduction Steps

The topic's server-side encryption can be verified using the following AWS CLI commands:

```
$ aws sns list-topics --output table  --query 'Topics[]'

 [
    {
        "TopicArn": "arn:aws:sns:ANY_REGION:ACCOUNT_ID:aws-controltower-
SecurityNotifications"
    }
]

$ aws sns get-topic-attributes --topic-arn arn:aws:sns:ANY_REGION:ACCOUNT_ID:aws-
controltower-SecurityNotifications  --query 'Attributes.KmsMasterKeyId'

null
```

Since the `get-topic-attributes` command returns `null`, the server-side encryption is not enabled for the specific SNS topic.

Impact

High. The missing encryption will cause the plain-text transmission of sensitive data delivered as messages to subscribers.

Complexity

N/A - The finding describes a missing hardening.

Remediation

Enable server-side encryption for AWS SNS topics.

A guide to set up SSE on SNS topics with AWS KMS is present in the resources below.

Resources

- Amazon Web Services, “Enabling server-side encryption (SSE) for an Amazon SNS topic” https://docs.aws.amazon.com/sns/latest/dg/sns-dg.pdf#sns-enable-encryption-for-topic

TEL-Q322-19 - Missing SSRF Protection for EC2 Instance Metadata	
Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	EC2 IMDS Endpoint
Status	Closed

Description

In EC2, AWS provides a unique feature in the REST interface available at the following endpoint:

`http://169.254.169.254/`

This IP provides internal access to configuration and authentication information on all EC2 instances.

This feature has been commonly abused during Server-Side Request Forgery attacks. Server Side Request Forgery (SSRF) vulnerabilities let an attacker send crafted requests from a vulnerable web application. Attackers can use SSRF attacks to target internal systems that are behind firewalls and are not accessible from the external network. In this particular case, an attacker may leverage SSRF to access services available through the metadata server (`169.254.169.254`) of the exploited EC2 instance.

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

During our AWS configuration review, **the following EC2 instances were found to be using the old version:**

- `arn:aws:ec2:us-east-1:599519581022:instance/i-0eb7cdad1a9fe6315`
- `arn:aws:ec2:us-east-1:599519581022:instance/i-0ccf4a71c0e2160b9`
- `arn:aws:ec2:us-east-1:599519581022:instance/i-0b9bafc8aa960e390`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-03af8eb9e8367f5fa`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-019f8bc3865dfacd8`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0b4d37ce7831e34db`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0dcf009a870e42f82`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0cbd7553153b8ab88`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-04a29036f2ab28f2d`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-05f37710470807930`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-000a18baff7c3db21`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0c68e94d3cad98ecf`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0e282c772ec1744cf`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-09ac6790e318d6f6b`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0a8ecf1df343dd24e`
- `arn:aws:ec2:us-west-2:599519581022:instance/i-0f6410b40073e5d71`
- `arn:aws:ec2:eu-central-1:599519581022:instance/i-04993689c4b62b537`
- `arn:aws:ec2:eu-central-1:599519581022:instance/i-00dc9327f8c13960f`
- `arn:aws:ec2:eu-central-1:599519581022:instance/i-0fb0c9cdb2557167d`

- arn:aws:ec2:ap-southeast-1:599519581022:instance/i-04b38141d1e2f2d87
- arn:aws:ec2:ap-southeast-1:599519581022:instance/i-0a6bea3c1b9705442
- arn:aws:ec2:ap-southeast-1:599519581022:instance/i-0d38e9ff07ccb2ba0

Reproduction Steps

To verify that IMDSv1 is still in use, log in to a vulnerable EC2 instance and execute:

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

If the HTTP response returns a 200 OK attaching the instance's data, IMDSv1 is indeed in use. Otherwise, the HTTP request will fail since no authentication token is provided.

Impact

High. Usage of IMDSv1 leaves the instance unprotected against attacks such as SSRF. It can lead to information disclosure and potentially full system compromise.

Complexity

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

Remediation

Transition to "Instance Metadata Service Version 2" in order to protect Teleport Cloud against AWS metadata SSRF.

Resources

- Amazon Web Services, "Configure the instance metadata options" https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html
- Colm MacCarthaigh, AWS Security Blog, "Add defense in depth against open firewalls, reverse proxies, and SSRF vulnerabilities with enhancements to the EC2 Instance Metadata Service" https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/

TEL-Q322-20 - Network ACLs Allow Unrestricted Access	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	AWS Network ACL
Status	Closed

Description

AWS allows the configuration of high-level Network Access Control Lists that can be associated directly with VPC subnets.

Currently defined Network ACLs within the VPC Infrastructure allow unrestricted access for both inbound and outbound traffic.

Moreover, an egress security group rule defined at `cloud/deploy/terraform/teleport-cloud.tf:811` allows traffic to `0.0.0.0/0`.

```
resource "aws_security_group" "sales_center_db" {
  count       = local.is_dev ? 0 : 1
  name        = "sales-center-db"
  description = "Created by RDS management console"

  vpc_id = local.vpc_id

  ingress {
    from_port       = 5432
    to_port         = 5432
    protocol        = "tcp"
    security_groups = [var.default_security_group]
  }

  egress {
    from_port = 443
    to_port   = 443
    protocol  = "tcp"

      # TG: I can't find a way to restrict output to just our S3 Bucket, some
ideas:
    #
    # Specify only AWS IP ranges but data could be sent to another AWS customer
          #  IP  ranges  https://docs.aws.amazon.com/general/latest/gr/aws-ip-
ranges.html
    #
    # Use a VPC for the S3 buckets and limit egress to that VPC
    # https://aws.amazon.com/blogs/aws/new-vpc-endpoint-for-amazon-s3/
    cidr_blocks = ["0.0.0.0/0"]
```

```
        }
    }
```

As stated in the comments, traffic limitations should be applied with a properly configured VPC.

Reproduction Steps

In order to verify this misconfiguration, leverage the AWS Command Line Interface (CLI):

```
$ aws ec2 describe-network-acls
```

Among the results, the following rules should be present :

```
    "Entries": [
                {
                    "CidrBlock": "0.0.0.0/0",
                    "Egress": true,
                    "Protocol": "-1",
                    "RuleAction": "allow",
                    "RuleNumber": 100
                },
                {
                    "CidrBlock": "0.0.0.0/0",
                    "Egress": true,
                    "Protocol": "-1",
                    "RuleAction": "deny",
                    "RuleNumber": 32767
                },
                {
                    "CidrBlock": "0.0.0.0/0",
                    "Egress": false,
                    "Protocol": "-1",
                    "RuleAction": "allow",
                    "RuleNumber": 100
                },
                {
                    "CidrBlock": "0.0.0.0/0",
                    "Egress": false,
                    "Protocol": "-1",
                    "RuleAction": "deny",
                    "RuleNumber": 32767
                }
            ], …
```

The orange highlighted parts show the presence of Ingress and Egress rules with a small `RuleNumber` that allows traffic. As they have a small `RuleNumber`, they prevent the default Deny from being matched.

Impact

High. An instance deployed by an adversarial IAM user would not be restricted.

Complexity

High. The attacker needs IAM credentials with EC2 instances creation permission.

Remediation

Apply firewall rules between VPC subnets by using AWS EC2 ACL.

Resources

- Amazon Web Services, "Control traffic to subnets using network ACLs" https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html

Appendix A - Vulnerability Classification

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

☐	Limit the maximum allowed size for the user agent and device name.
☐	Check the data buffer length before assigning the subtracted value to the clip variable.
☐	Limit the number of characters the user can store inside the clipboard.
☐	Terminate the connection when the server does not send the ScCore or ScNet messages.
☐	Initialize the authentication_protocol property or remove the support for the ProtocolHybrid.
☐	Ensure the uncompressed length for the PDUType::PdutypeDatapdu object cannot overflow and limit its size.
☐	Repeatedly overwrite the seed file content to make it harder to recover the data.
☐	Remove special characters such as newlines or tabs from the log entries.
☐	Enforce in go-mssqldb library the maximum size of the data read from the client.
☐	Enforce the file size limit or make the limit configurable.
☐	Implement the rate limit for Proxy API Web Endpoints.
☐	Only permits the protocols that the application is intended to open. We recommend allowing only HTTPS.
☐	Enable sandbox when creating the main BrowserWindow used by Teleterm.
☐	Update to the latest stable Electron.js version (v. 20.0.0) to benefit from the latest security fixes.
☐	Remove the dangerous pty.shell command type.
☐	We recommend applying condition blocks to limit potentially dangerous IAM permissions and align the vulnerable roles to the least privilege principle.

☐	Ensure ELBs are configured to only accept connections on HTTPS ports.
☐	Enable server-side encryption for AWS SNS topics.
☐	Transition to "Instance Metadata Service Version 2" in order to protect Teleport Cloud against AWS metadata SSRF.
☐	Apply firewall rules between VPC subnets by using AWS EC2 ACL.

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.