hackerone

Paradex Penetration Test

API & Web Application Penetration Test

March 25, 2025

Table of Contents

Executive Summary

Paradex engaged HackerOne to perform a HackerOne penetration test from February 25, 2025 to - March 12, 2025. The scope of testing included one (1) API and one (1) web application. This report reflects the state of security across applications and systems tested during this period.

During this timeframe, the pentest team identified three (3) vulnerabilities identified. Out of these, the testers assigned two (2) vulnerabilities assigned a Low- severity rating, and while one (1) vulnerability was marked with Informational and None severity.

The security assessment was conducted using a crowdsourced penetration testing methodology. From its community of over one million hackers, HackerOne curated a set of top-tier researchers to focus on identifying vulnerabilities in targets defined by Paradex to be in-scope during the agreed-upon testing window while abiding by the policies set forth by Paradex. The Approach section contains additional information related to the testing methodology used in this engagement.

This program assessment can be accessed via the HackerOne Portal.

Retest Note: Retesting was performed leading up to the writing of this, the final version of the report, on March 25th, 2025. The status of each vulnerability is documented in the Vulnerabilities by Severity section of this report.

Technical Summary

During the engagement, the pentesters found three (3) unique vulnerabilities across two (2) different Common Weakness Enumeration (CWE) and Common Attack Pattern (CAPEC) categories. The most common vulnerability type was Violation of Secure Design Principles with two (2) unique vulnerabilities. Reproduction steps for each vulnerability can be viewed and exported via the program inbox in the HackerOne Portal.

The pentest team conducted a gray box penetration test of Paradex platform to assess its risk posture and identify security issues that could negatively affect the data, systems, or reputation of Paradex. The primary focus was to evaluate the platform's readiness for open-source release.

The testers approached a hybrid model, mostly manual tests, with the help of some automated tools. The main goal was a holistic assessment of the security of the application's functionality, business logic, and vulnerabilities such as those cataloged in the OWASP Web and API Top 10. The assessment also included a review of security controls and requirements listed in the OWASP Application Security Verification Standard (ASVS).

The following list contains many of the main tests that the team performed on the backend API:

- Authenticated user testing for session and authentication issues
- Authorization testing for privilege escalation and access control issues
- Input injection tests (SQL injection, XSS, and others)
- Platform configuration and infrastructure tests
- OWASP Web and API Top 10 testing

The assessment team reported three (3) Low-severity findings, with the most significant being a low-risk username enumeration vulnerability in the username change feature. Testers also found a misconfiguration in the Content Security Policy implementation, which could leave the application more vulnerable to Cross-Site Scripting (XSS) attacks. Additionally, the team identified a minor information disclosure issue where the app exposed minimal stack trace details in WebSocket communications.

Overall, the Low-severity findings suggest that while Paradex has implemented strong and in-depth security measures, there are areas that require attention before open-sourcing the code. It is recommended that the development team address these issues as the platform moves towards open-source status.

Overall, while the applications exhibited some positive security controls, there are areas that require improvement to enhance the confidentiality, integrity, and availability of the assets.

Addressing the Highest Severity Vulnerability

The most severe vulnerability that testers reported during the engagement was a Username enumeration via the “username change feature” finding, which the team rated as Low-risk.

This vulnerability allowed the pentest team to determine whether a specific username existed in the system using the username change feature. By attempting to change their username to one that may already exist, an attacker can figure out whether that username is taken based on the system’s response, and use that information to build a list of valid usernames for further attacks.

To address this finding, implement the following:

- **Rate Limiting:** Implement a rate limiting mechanism on the username change feature to prevent automated enumeration attempts.
- **Generic Error Messages:** Return generic messages that do not reveal whether an account exists in the system.

See the following for more information:

- Username Enumeration and Guessable User Account

Addressing the Most Prevalent Vulnerability

The most common vulnerability reported during this engagement is related to information disclosure and improper error handling, as evidenced by the “Minor Stack Trace in WebSockets” finding.

These types of vulnerabilities can potentially reveal sensitive information about the application’s internal workings, which attackers could leverage for further exploitation. Stack traces, even if minor, can disclose implementation details that may aid attackers in understanding the application’s architecture or identifying potential weak points.

To address this finding, implement the following:

- **Validation and Sanitization:** Implement proper input validation and error handling mechanisms.
- **Generic Error Messages:** Regarding stack traces, sanitize all error messages properly before sending them to the client. Replace detailed technical information with generic error messages in production environments.

See the following for more information:

- Insecure Design
- Secure Design Principles

Vulnerabilities by Severity

The following table documents each identified vulnerability:

Report ID	Vulnerability	Severity	CVSS	Reference	Status
#3030175	Username enumeration via username change feature	Low	3.7	CWE-402	**Fixed**
#3030547	Minor Stack Trace in WebSockets	Low	3.7	CWE-657	**Fixed**
#3030542	Limitations in Content-Security-Policy: No directive mitigating XSS Attacks	None	0.0	CWE-657	**Open**

Table: Vulnerabilities by severity

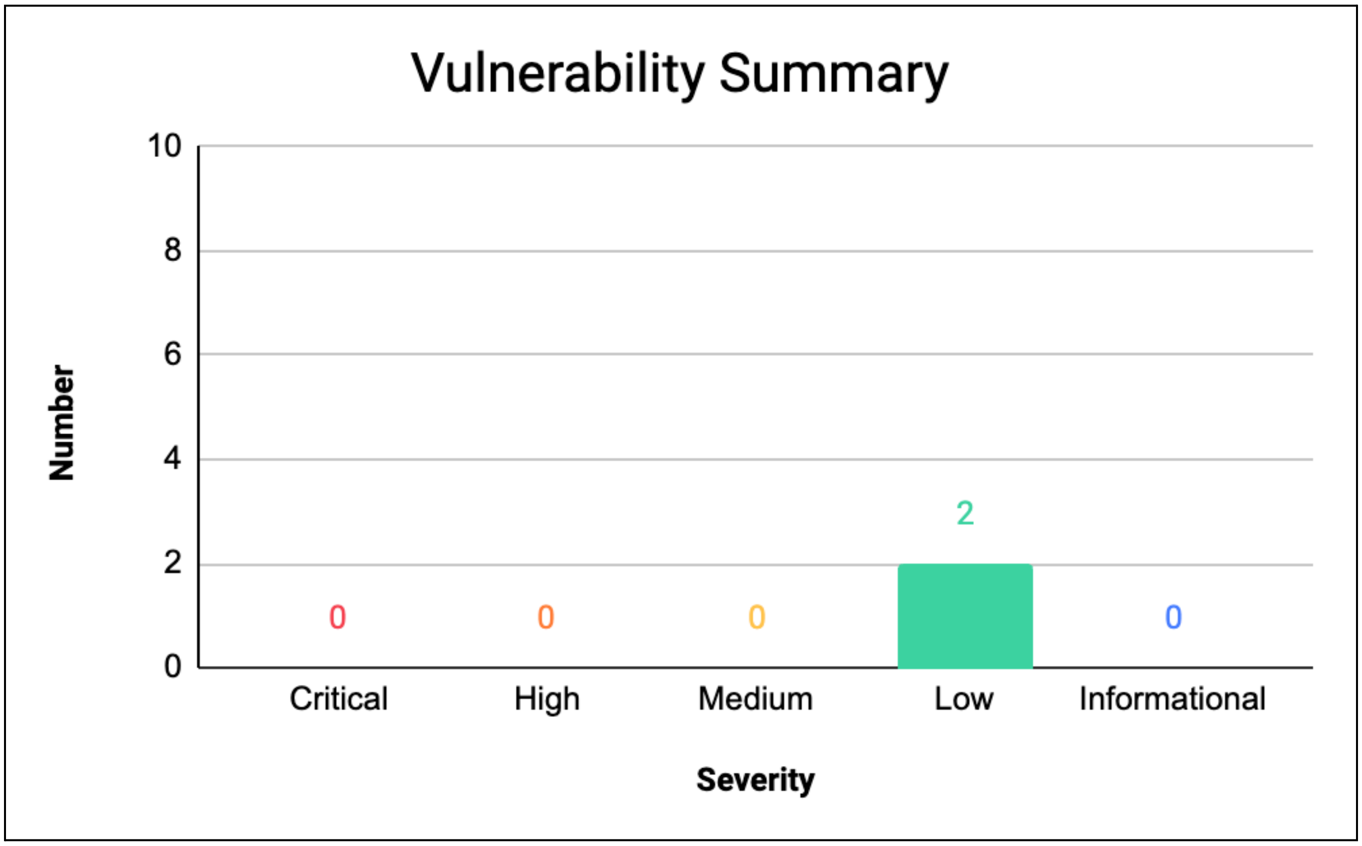
Vulnerabilities by Asset

Vulnerabilities were found in the following assets:

Asset	Critical	High	Medium	Low	None	Total
https://app.testnet.paradex.trade	-	-	-	2	1	**3**
Total	**0**	**0**	**0**	**2**	**1**	**3**

Table: Severity of findings by asset

The following graph presents the number of identified vulnerabilities and their respective severity ratings:



Graph: Number of vulnerabilities per severity

Vulnerabilities by Category

The following table shows the number of individual findings and their distribution of severity:

Category	Critical	High	Medium	Low	None	Total
Violation of Secure Design Principles	-	-	-	1	1	**2**
Transmission of Private Resources into a New Sphere ('Resource Leak')	-	-	-	1	-	**1**

Table: Severity of finding(s) by weakness (CWE)

A breakdown of severities and their mapping to Common Vulnerability Scoring System (CVSS) scores is available in the Vulnerability Classification & Severity section of this document.

Vulnerability Details

#3030175 Username enumeration via username change feature

Affected Asset

- `https://app.testnet.paradex.trade`

Severity: Low (3.7)

Description

Username enumeration is the process of determining whether a certain username is valid in the system. This can occur on “login” or “change password” screens, if the application returns a message that says something like “this username does not exist” when it receives an invalid response. An attacker could use this behavior to compile a list of valid usernames for further attacks. If the app also lacks sufficient rate limiting, an attacker could automate this attack.

The testers found that the application allowed users to onboard using an username, which the user can later change in the User Interface (UI). While reviewing HTTP requests, the team noticed the `is_username_private` parameter, which indicated that some usernames might be private. Furthermore, the back end did not seem to implement any rate limiting on the changing username feature, which could allow an attacker to use a list of usernames and find out which of them are valid.

While changing the username is a required feature, missing rate limitation mechanisms and the existence of private usernames might allow an attacker using a list of usernames to find out which of them are valid in order to discover valid platform accounts.

Recommendation

To address this finding, implement the following:

- **Rate Limiting:** Implement a rate limiting mechanism to mitigate username enumeration.
- **Generic Error Messages:** Return generic messages that do not reveal whether an account exists in the system.

References

See the following for more information:

- Account Enumeration and Guessable User Account

Impact

If not addressed, this finding could lead to the following:

- **List of Targets:** An attacker could automate guessing multiple usernames and then compile a list of valid targets to use in brute-force, password spray, phishing, or other attacks against the identified users.

#3030547 Minor Stack Trace in WebSockets

Affected Asset

- `https://app.testnet.paradex.trade`

Severity: Low (3.7)

Description

Missing security design principles occur when an organization does not follow best practices around secure design and Secure Development Lifecycle (SDLC) in a way that results in the accidental introduction of security vulnerabilities. These vulnerabilities could also occur when controls around change management are operating as ineffective. The lack thereof or failure in the design or implementation of secure design and SDLC best practice principles will result in the accidental introduction of security vulnerabilities.

During the assessment, the pentesters identified a minor stack trace revealed in some WebSocket responses, which could provide an attacker with insights into the underlying technology stack, code structure, or error-handling mechanisms. While this does not directly lead to exploitation, it increases the attack surface.

Recommendation

To address this finding, implement the following:

- **Disable Stack Traces in Production:** Ensure that detailed error messages and stack traces are not exposed to end-users. Use generic error responses instead.
- **Implement Proper Error Handling:** Use structured error handling to catch unexpected exceptions and return user-friendly messages without exposing internal code details.
- **Sanitize Responses:** Ensure that error messages do not leak internal file paths, function names, or sensitive information.

References

See the following for more information:

- OWASP: Secure by Design Principles
- OWASP: A04:2021 - Insecure Design

Impact

If not addressed, this finding could lead to the following:

- **Reconnaissance:** An attacker could exploit this to identify the underlying framework, programming language, and server structure, use disclosed file paths and function names to craft targeted exploits, and improve their understanding of the application's internal workings for further attacks.

#3030542 Limitations in Content-Security-Policy: No directive mitigating XSS Attacks

Affected Asset

- `https://app.testnet.paradex.trade`

Severity: None (0.0)

Description

The Content Security Policy (CSP) header is a security feature designed to mitigate Cross-Site Scripting (XSS) and other injection attacks by restricting the sources from which content such as scripts, styles, and media can be loaded. A well-configured CSP significantly reduces the risk of malicious scripts executing in a web application.

The team determined that the CSP policy for the target application was as follows:

```
content-security-policy: frame-ancestors 'self' <a
href="https://www.paradex.trade">https://www.paradex.trade</a> <a
href="https://*.paradigm.trade;">https://*.paradigm.trade;</a>
```

This policy only defined `frame-ancestors`, which prevented the website from being embedded in unauthorized iframes to mitigate Clickjacking attacks. However, it did not include key directives that help mitigate XSS and other client-side attacks, such as:

- `script-src`: Controls which sources JavaScript can be executed from.
- `style-src`: Defines allowed sources for CSS to prevent style-based injection attacks.
- `default-src`: A fallback directive that restricts content loading from unauthorized domains.
- `base-uri`: Prevents attackers from overriding the base URL for links.

Recommendation

To address this finding, implement the following:

- **CSP Directives:** Configure the CSP header to include directives that help mitigate XSS and other client-side attacks.
    - `script-src`: Controls which sources JavaScript can be executed from.
    - `style-src`: Defines allowed sources for CSS to prevent style-based injection attacks.
    - `default-src`: A fallback directive that restricts content loading from unauthorized domains.
    - `base-uri`: Prevents attackers from overriding the base URL for links.

References

See the following for more information:

- OWASP: Secure by Design Principles
- OWASP: A04:2021 - Insecure Design

Impact

If not addressed, this finding could lead to the following:

- **XSS or Injection Attacks:** The current CSP configuration helps prevent clickjacking but does not mitigate XSS or other injection attacks. Without proper directives, an attacker may inject and execute unauthorized scripts, leading to minor data leaks, UI manipulation, or small-scale phishing attempts. Strengthening the CSP will reduce these risks.

Appendix

Statement of Coverage

The following assets formed the scope of the assessment:

Assets
https://app.testnet.paradex.trade

Table: In-scope assets

Summary of Assets

Testers found that the web application at `https://app.testnet.paradex.trade` demonstrated some minor security weaknesses, including a username enumeration, information disclosure, and misconfigurations in the Content Security Policy. While the team rated these issues as Low-severity, an attacker could exploit them to gather information or launch more sophisticated attacks, which indicates some room for improvement in hardening of the application's security controls.

All applicable test cases and respective vulnerabilities were identified against each OWASP category, and are available on the HackerOne Portal.

HackerOne Security Checklists

HackerOne API Security Checklist

Check Name	Description
Broken Authentication	Authentication mechanisms are often implemented incorrectly, allowing attackers to compromise authentication tokens or to exploit implementation flaws to assume other user's identities temporarily or permanently. Compromising a system's ability to identify the client/user, compromises API security overall.
Broken Function Level Authorization	Complex access control policies with different hierarchies, groups, and roles, and an unclear separation between administrative and regular functions, tend to lead to authorization flaws. By exploiting these issues, attackers gain access to other users' resources and/or administrative functions.
Broken Object Level Authorization	APIs tend to expose endpoints that handle object identifiers, creating a wide attack surface of Object Level Access Control issues. Object level authorization checks should be considered in every function that accesses a data source using an ID from the user.
Broken Object Property Level Authorization	This is a combination of Excessive Data Exposure and Mass Assignment, focusing on the root cause: the lack of or improper authorization validation at the object property level. This leads to information exposure or manipulation by unauthorized parties.
Improper Inventory Management	APIs tend to expose more endpoints than traditional web applications, making proper and updated documentation highly important. A proper inventory of hosts and deployed API versions also are important to mitigate issues such as deprecated API versions and exposed debug endpoints.

Security Misconfiguration	APIs and the systems supporting them typically contain complex configurations, meant to make the APIs more customizable. Software and DevOps engineers can miss these configurations, or don't follow security best practices when it comes to configuration, opening the door for different types of attacks.
Server Side Request Forgery	Server-Side Request Forgery (SSRF) flaws can occur when an API is fetching a remote resource without validating the user-supplied URI. This enables an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall or a VPN.
Unrestricted Access to Sensitive Business Flows	APIs vulnerable to this risk expose a business flow, such as buying a ticket, or posting a comment, without compensating for how the functionality could harm the business if used excessively in an automated manner. This doesn't necessarily come from implementation bugs.
Unrestricted Resource Consumption	Satisfying API requests requires resources such as network bandwidth, CPU, memory, and storage. Other resources such as emails, SMS, phone calls, or biometrics validation are made available by service providers via API integrations, and paid for per request. Successful attacks can lead to Denial of Service or an increase of operational costs.
Unsafe Consumption of APIs	Developers tend to trust data received from third-party APIs more than user input, and so tend to adopt weaker security standards. In order to compromise APIs, attackers go after integrated third-party services instead of trying to compromise the target API directly.

HackerOne Web Security Checklist

Check Name	Description
Broken Access Control	Broken access control occurs when an application lacks thorough permission or role checks. These missing checks typically lead to unauthorized information disclosure, modification or destruction of data, or performing a business function outside of the typical limits of the user.
Cryptographic Failures	Sensitive data exposure relates to issues regarding insecure or missing cryptographic protocols. This may affect data transmission, storage, or access.
Identification and Authentication Failures	Identification and Authentication Failures occur when an application lacks controls around the authentication process or session management. This may lead to an attacker being able to compromise an individual account or even a system-wide authentication bypass.
Injection	Injection flaws occur when unvalidated or unfiltered user-supplied data is used directly by an application. Injection vulnerabilities are often found in SQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries.
Insecure Design	Insecure Design relates to flaws in threat modeling, secure design patterns and principles, and reference architectures. This is a broad category representing different weaknesses, expressed as "missing or ineffective control design."
Security Logging and Monitoring Failures	Security Logging and Monitoring Failures occur when there are missing or inadequate mechanisms to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected.
Security Misconfiguration	Security misconfiguration occurs when application development or deployment does not follow security best practices. Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage.

Server-Side Request Forgery (SSRF)	SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network Access Control List (ACL).
Software and Data Integrity Failures	Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and Content Delivery Networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise. Lastly, many applications now include auto-update functionality, where updates are downloaded without sufficient integrity verification and applied to the previously trusted application.
Vulnerable and Outdated Components	Vulnerable and Outdated Components occur when application developers or deployers fail to keep third-party libraries or tools up-to-date. This can result in cases where known vulnerabilities and their exploits are able to apply outside of their original context.

Vulnerability Classification & 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.

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):

Severity	CVSS Rating
Critical	9.0 - 10.0
High	7.0 - 8.9
Medium	4.0 - 6.9
Low	0.1 - 3.9

Note: Reports marked as **None** are informational in nature and are considered best practices. They have a CVSS score of 0.0.

More information about CWE can be found on MITRE's website: Common Weakness Enumeration

More information about CVSS can be found on the Forum for Incident Response and Security Teams' (FIRST) website: Common Vulnerability Scoring System SIG

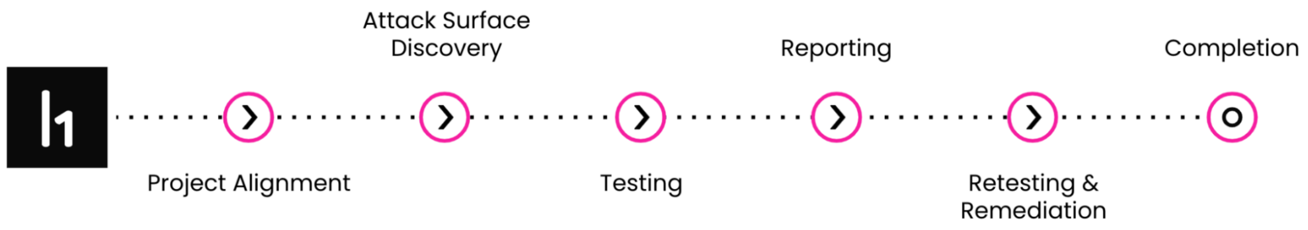
Approach

A HackerOne pentest engagement follows a series of methodologies, checklists, and guidelines to ensure a balance between consistent customer experience, coverage of testing, and depth of testing. This engagement included testing to evaluate the assets for vulnerabilities included in the SANS Top 25 and OWASP Top 10. Using this combination of best practices and proprietary experience HackerOne is confident that its penetration tests provide a thorough level of security assurance and an unbiased assessment of the state of security for its customers.

Overview

HackerOne develops these tools using industry best practices such as OSSTMM, OWASP, NIST, PTES, and ISSAF, as well as proprietary knowledge gained through HackerOne's platform that services hundreds of ongoing or timeboxed engagements and a community of over one million hackers. Using this combination of best practices and proprietary experience, HackerOne is confident that its penetration tests provide a thorough level of security assurance and an unbiased assessment of the state of security for its customers. This section covers the engagement experience and approach.

Engagement Phases



Project Alignment

HackerOne leverages experts from several internal teams to support customers and understand the goals and expectations for the pentest engagement. HackerOne then works with our community to select highly talented and qualified pentesters that best fit the individual customer's needs and technologies. HackerOne works with the customer to define Rules of Engagement for the testing activities where applicable and establish lines of communication for all stakeholders to ensure that risks to the in-scope assets are minimized. The outcome of this phase is a fully aligned team, from customer to hackers, to ensure the testing engagement launches with the utmost chance of success.

Attack Surface Discovery

The selected pentesters for the engagement begin their testing efforts by conducting initial discovery efforts, including tasks such as ensuring hosts are alive and stable, understanding all possible functionalities, and identifying access levels that exist on the in-scope assets. Further, the pentesters will begin active testing activities to understand the state of perimeter defenses and inventory the most likely attack vectors for the environment. They will also look at core functionality to begin to identify initial weaknesses in the system. During this phase findings may certainly be discovered, however, the true intent of this phase is for pentesters to familiarize themselves with the environment and conduct initial research towards the customer's in-scope assets. The outcome of this phase is that the Pentest Team is familiar with the assets and environment that they are conducting testing against/within and to spot likely attack vectors, and gain a deeper understanding towards the state of security for the assets/environment being tested.

Testing

In this phase, HackerOne empowers the Pentest Team with both high-level coverage requirements to ensure breadth of coverage and detailed testing guides to ensure depth of coverage towards the assets in-scope for the engagement. Pentesters also launch their most aggressive attacks toward the in-scope assets in an effort to ensure the most thorough level of security assurance for the customer. The outcome of this phase is to gain an appropriate level of understanding towards the security assurance for all assets engaged.

Reporting

During the Reporting phase HackerOne collaborates with the Pentest Team and the customer to ensure 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 assets, and while they may be customized, they can not 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.

Retesting & Remediation

Paradigm engaged HackerOne to retest the findings made during the assessment to ensure vulnerabilities were patched properly. Where possible, retests are conducted by the original finder to ensure the vulnerability is mitigated correctly.

Testing Team

HackerOne's Clear program plays a crucial role in enhancing our pentest business and Bug Bounty programs, significantly advancing HackerOne's trust revolution. Clear is essential for sourcing, ensuring engagement with verified and trustworthy professionals.

Further information on the program is available via the HackerOne Clear page.

Testing Lead

The lead, Ionut Popescu (@nytro), is the primary contact for the pentesting team. It is their responsibility to ensure smooth running of testing efforts, that coverage across the assets is provided, that any draft reports are reviewed, and to maintain communication with the client.

Testing Support

The following pentesters supported the lead:

- Yash Sodha - @yashrs

Technical Engagement Manager

Manjesh S is the Technical Engagement Manager for this assessment and is responsible for orchestration and quality assurance.

Tooling

During the assessment, the team utilized a combination of open-source and commercial tools to thoroughly evaluate the target attack surface. The primary tools that testers employed were:

- **Burp Suite Pro:** This web application security testing suite used for intercepting and manipulating HTTP/S traffic, enabling thorough analysis and exploitation of identified vulnerabilities.
- **SQLmap:** This open-source penetration testing tool, designed for automating the process of detecting and exploiting SQL injection flaws, allowed testers to identify and validate SQL-related vulnerabilities.

These tools, along with manual techniques and industry best practices, formed the foundation of the comprehensive assessment approach, ensuring a thorough evaluation of the target attack surface.

Restrictions

For this pentesting engagement, the team did not encounter any specific restrictions, and testers were able to use self-registered accounts.

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* --------------------