İNTİGRİTİ

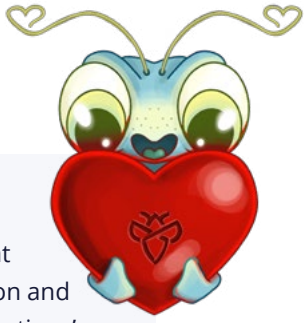
Bug Bounty

Starter Kit

Everything you need to go from
zero to your first valid bug report!

intigriti.com

Table of contents



About Intigriti

Intigriti is a community-driven bug bounty platform that empowers ethical hackers worldwide to earn recognition and rewards by uncovering security flaws in leading organizations' digital assets. Through flexible programs, transparent triage, and hacker-focused events, Intigriti actively nurtures talent and gives security researchers the tools, trust, and opportunities they need to thrive.

Do you have what it takes to hack global companies? Sign up now!

go.intigriti.com/signup

Foreword

Every bug starts with a hunch. Something feels off. A response time that is too long. A parameter that should not be there. A flow that almost makes sense, but not quite. You pull on the thread, and the whole thing unravels.

That is the work. That is why we do it.

I started Intigriti because I believed Europe had the talent to compete with anyone in the world. A decade later, the proof is on the leaderboards, in the payouts, and in the CVEs. Hackers built this industry. Not vendors. Not frameworks. Hackers.

People ask me if AI changes that. It does, and it does not. AI is fast. It is tireless. It will happily generate a thousand payloads while you sleep. But it does not get the itch. It does not stare at a request for twenty minutes because something feels wrong. It does not chain a forgotten subdomain to a misconfigured bucket to a full account takeover. That is still you. That will stay you.

At Intigriti the community comes first. Always. Researchers are not a cost line. They are the reason the platform exists. Every decision we make answers one question: does this make the work better for the people doing it.

If you are opening this book, you are early enough. The scope is bigger than it has ever been. The tooling is sharper. The community around you is more generous than the one I started in. There is still a seat at the table, and your name is not yet on it.

Pick a target. Read the scope. Trust the itch.

The rest is yours to find.

STIJN JANS

FOUNDER & CEO

İNTİGRİTİ

"Your bounty bag is awaiting you"

What is bug bounty

Bug bounty programs invite ethical hackers to find security vulnerabilities in software applications and systems in exchange for monetary rewards.

Unlike traditional pentests, which are scheduled periodically and have limited-time engagements, or automated scanners that produce enormous output with few meaningful results, **bug bounty hunters** and **security researchers** like you provide **continuous**, **real-world testing** that uncovers critical vulnerabilities the preceding two methods fail to find.

This book was curated from the **hundreds of hacking resources** we published over the past few years to help more researchers submit more valid security reports. It is designed for aspiring bug bounty hunters seeking guidance on discovering their first vulnerability, but it can also be used by seasoned hackers who enjoy delving into more web hacking content.

Let's dive in!

What it takes to excel in bug bounty hunting

To get good at bug bounty, you just have to stay curious and keep learning. Try new techniques, step outside your comfort zone, and don't be afraid to fail a lot. Sometimes that means exploring completely different attack surfaces, like hardware hacking or game security. The more you explore and stick with it, the more things start to click

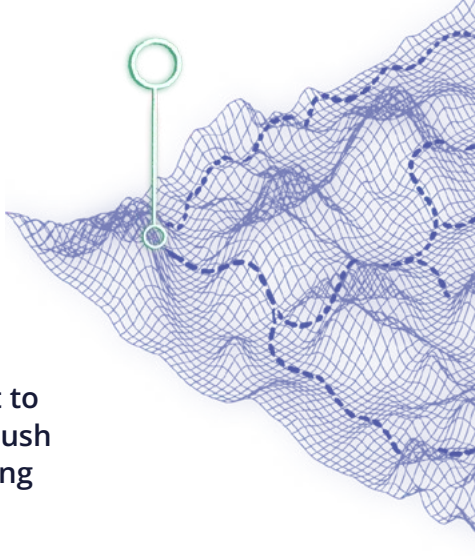
hg_real

BELGIUM'S #1 ALL-TIME HACKER ON INTIGRITI

Follow us!

@intigriti
@hackwithintigriti

Bug bounty fundamentals



Before you can hack a target, you need to know what you're working with. Reconnaissance is the process of gathering information about your target to map out its entire attack surface. Most rush straight into vulnerability hunting, missing the majority of the attack surface.

By conducting thorough reconnaissance, you can discover forgotten subdomains, exposed development environments, leaked credentials in GitHub repositories, hidden API endpoints, and misconfigured services that other researchers overlook.

The more thorough your reconnaissance phase, the larger your attack surface becomes, and the higher your chances of finding critical vulnerabilities that others missed!

Crafting your own methodology

Exploiting security vulnerabilities is just one part of the game. Understanding where they're present and how to test for them is where your methodology comes into play. As a bug bounty hunter, you're always competing with hundreds of other ethical hackers on the same scope. That's why you'll need to differentiate yourself with your own, uniquely crafted methodology.

Learn how to craft one yourself through the link below!

go.intigriti.com/crafting-methodology

Common pitfalls to avoid

Hacking can be overwhelming, especially if you're just getting started. Below are 4 of the most common pitfalls we have listed, which we recommend you avoid.

1. **Read the scope**

   Before hacking any target, always read the program's scope policy. The information provided in a program's testing policy will help you avoid invalid submissions.

2. **Pick the target that matches your skill set**

   Select a program with an interesting scope that matches your skill set. Select programs with mobile apps in scope if you're more proficient in mobile testing. The same applies if you're more proficient in testing targets built with a specific technology.

3. **Avoid full reliance on automated tooling**

   Try to understand the full process first before picking up any tools. 'Spraying and praying' with automated tooling often floods programs with invalid submissions rather than the quality findings that pay out big rewards.

4. **Avoid sending in poor reports**

   Avoid submitting reports that are incomplete. Learn how to write compelling vulnerability reports that get triaged faster. Poorly written reports may cause confusion for the triage and respective security team, which you ultimately want to avoid.

Always adhere to your scope policy!

Before hacking any target, we always recommend that you carefully read the program's scope policy. Understanding what's in scope, what's prohibited, how tests should be carried out (e.g., following rate limits, using your Intigriti.me email alias, program-specific request headers, etc.), and how to report your findings properly can mean the difference between a new payout registered in your name and a rejected report.

Tooling

Besides your web browser, you'll be working with all sorts of tooling when hacking web and mobile applications. In this section, we've listed some of the most important tools you'll need, including their use cases and a few tips to help you get started!

Proxy interceptor

A proxy interceptor sits between your browser (or mobile device) and the target application, capturing every request and response as they are transmitted. You'll notice that most of your work will be performed within your proxy interceptor tool. **Burp Suite, Caido, and OWASP ZAP** are the 3 most recognized proxy interceptors.

Mobile emulator

If you're targeting mobile apps, you'll need an emulator to test applications in runtime. Depending on whether you're targeting iOS or Android apps, both platforms have tooling that can simplify your job:

- **Android Studio** and **Apple Xcode**, the official IDEs from Google and Apple, both ship with built-in emulators for their respective platforms, which can help you when testing mobile apps.
- **Frida** is a dynamic instrumentation toolkit allowing you to hook into running processes. It can help you with essential tasks, such as bypassing SSL pinning and root/jailbreak detection.

Receive a free Burp Suite Professional

Intigriti has teamed up with PortSwigger! Hit 400+ valid reputation points in a single quarter on Intigriti and earn a free 6-month Burp Suite Pro license through our PortSwigger collaboration. Terms and conditions apply.

go.intigriti.com/burp-pro

Intro to reconnaissance

Asset mapping

1

The first phase involves identifying all IP ranges, ASNs, and root domains owned by your target organization. Query ARIN, RIPE, or APNIC with your target's organization name to search for netblocks, then cross-reference with tools like Amass or bgp.he.net to discover autonomous system numbers (ASNs) and the full scope of infrastructure under the target's control.

Subdomain enumeration

2

The next phase involves mapping out all subdomains from the root domains identified in phase one. Combine passive sources, such as Google search results and certificate transparency logs, with active brute-forcing using wordlists. Subdomains frequently point to staging environments, internal tooling, and legacy applications that receive significantly less security attention than the primary domain or assets.

Live host probing

3

Filter the raw subdomain list down to hosts that actively resolve. Tools like massdns and dnsx can be used to resolve hosts at scale. This step prevents you from wasting time scanning hosts that no longer respond, plus provides you with a more accurate view of your attack surface.

Network port scanning

4

After filtering for live hosts, it's time to run a port scan and map out live services. Tools like Nmap and Masscan can help scan network ports at scale. Non-standard ports regularly expose administrative panels, database interfaces, and internal APIs that are not intended to be publicly accessible. Pay particular attention to ports commonly used by development stacks, such as 3000, 8080, 8443, and 9200.

Technology fingerprinting

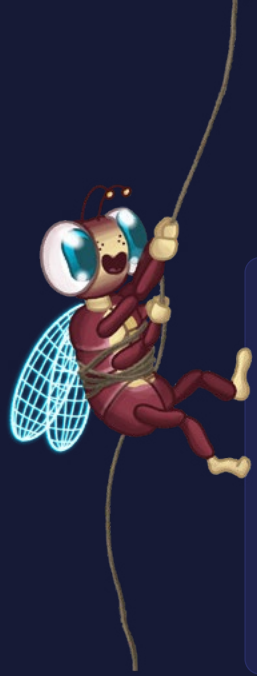
5

Next, it's time to fingerprint technologies. Each live host (and open port) is running various services, including web servers and frameworks, file-sharing services, and more. In bug bounty, we're specifically interested in mapping technologies used by each live domain, as some are vulnerable to CVEs or common security misconfigurations. There are several tools that can help map out based on the HTTP response header(s) and web structure.

Content discovery

6

Once you've mapped out the utilized technologies, it's time to perform some content discovery. Each individual host serves files, runs on a specific web framework, or hosts a custom panel or application. With the help of a (custom) wordlist and the use of a bruteforcing tool like FFuf or Feroxbuster, we can identify unlinked files and folders. These results often reference back to exposed panels, configuration files, undocumented API endpoints, and sometimes even backup archives.



Parameter discovery

7

Parameters can be vulnerable to various security vulnerabilities, such as injection attacks. Our next step after content discovery is to enumerate unlinked query and body parameters. With the help of tools like ParamMiner and Arjun, we can quickly find forgotten parameters that are a frequent source of injection and IDOR vulnerabilities.

Third-party service mapping

8

Your target organization utilizes various third-party tools and services, including Atlassian Jira, AWS S3, Jenkins, Freshworks, and similar platforms. These environments are often deployed but receive less security attention compared to the main asset, making it possible for security misconfigurations to arise. Tools like Misconfig-Mapper can help identify common security misconfigurations in third-party services at scale.

Exploiting SQL injection vulnerabilities

SQL injection (SQLi) is a vulnerability that allows attackers to manipulate the queries an application makes to its database.

A SQL query is a structured command that instructs a relational database to read, modify, or delete data based on the conditions provided. SQLi vulnerabilities arise whenever unsafe, user-controllable input is directly concatenated into such a query without proper sanitization.

EXAMPLE: READING ALL PRIVATE NOTES

Identifying SQL injections

The first step is to map out all areas in the application where your input may be evaluated against a database. Think search bars, filter parameters, sorting options, and any endpoint that accepts an identifier or reference to a resource.

Once you've listed your potential injection points, probe each one by submitting special characters that are likely to interfere with the query's syntax, such as single and double quotes (', "), comment sequences (--, #), and logical operators (OR, AND).

Look for

If the application returns a database error, behaves differently based on your input, or takes noticeably longer to respond, you may have found an injection point worth investigating further.

Exploiting classic SQL injections

Most web applications today support authentication, and your target's login form is often one of the first places worth testing for SQL injection.

Every time you try to sign in to your account, the application's backend code will query the database to find any matching records. In this instance, it will match and return our user account.

```
MYSQL
SELECT * FROM USERS WHERE USERNAME = 'intigriti' AND PASSWORD = 'hunter2';
```

Suppose now, instead of signing in with the username 'intigriti', we decide to break out of the syntax and manipulate the query in a way to disregard the password field:

```
POST /api/auth/signin HTTP/2
Host: www.example.com
User-Agent: Mozilla/5.0 (Macintosh;
Intel Mac OS X 10.15; rv:150.0)
Gecko/20100101 Firefox/150.0
Content-Type: application/x-www-form-
urlencoded

username=intigriti'+OR+1=1;+--
+&password=xyz
```

```
HTTP/2 302 Found
Content-Type: text/html; charset=utf-8
Location: /account/home
Server: nginx
X-Powered-By: Express
Set-Cookie: auth.token=eyJ...; Path=/;
Secure; SameSite=Strict
```

This is a classic example of a SQL injection. By injecting a single quote to break out of the string context, appending **OR 1=1** to force the condition to always evaluate as true, and using **--** to comment out the remainder of the query, the password check is never evaluated , and the application grants us access as the first matching user in the database.

Explore SQL injections

SQL injections are one of the most fun and impactful vulnerability classes you'll encounter in bug bounty. This is just the tip of the iceberg, if you want to learn more advanced detection and exploitation techniques, check out our in-depth resource on exploiting SQL injection vulnerabilities.

go.intigriti.com/exploiting-sqli

Exploiting Cross-Site Scripting vulnerabilities

Cross-site scripting (XSS) is a client-side injection vulnerability that allows attackers to inject malicious scripts into web pages viewed by other users.

It occurs when user-supplied input is reflected back in the HTTP response without proper encoding or sanitization, causing the victim's browser to execute the attacker's script.

Depending on where the vulnerability lies, a successful XSS attack can allow an attacker to steal session cookies, capture keystrokes, or perform actions on the victim's behalf.

Identifying XSS vulnerabilities

1. **Reflection**

   Inject a unique test string, such as **intigriti1337**, into input fields, URL parameters, or headers, and search for this string in the HTTP response to map all reflection points where your input appears.

2. **Injection**

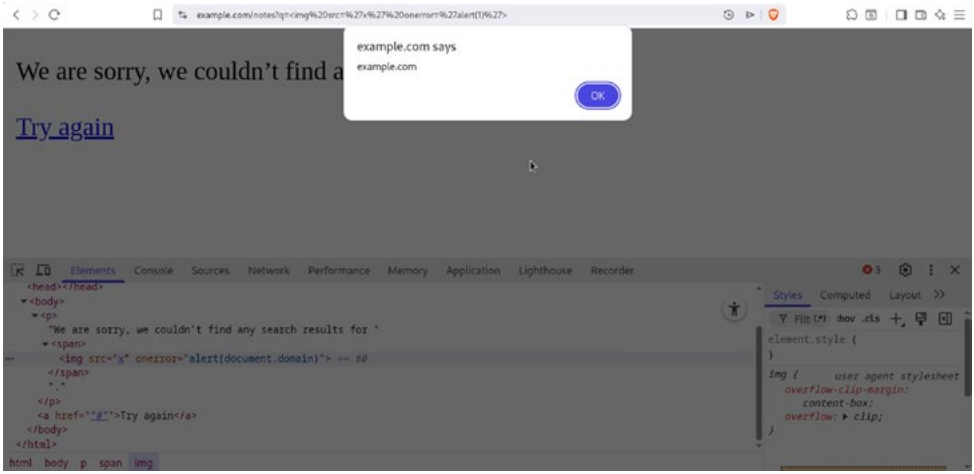
   Found a reflection? Test if you can break out of the current context by injecting simple HTML tags. If special characters are unencoded/unescaped, XSS is likely possible; if they are encoded/escaped, exploitation is usually blocked.

3. **Payload**

   Craft a proof-of-concept based on the context where input is reflected and any existing filters. Try to execute a simple JavaScript function to prove the presence of XSS such as **alert()** or **print()**.

Exploiting reflected XSS

The note-taking app from earlier features a search bar that reflects your query directly in the page, making it a good candidate for XSS testing.



Suppose we inject our payload via the search query parameter. We can observe in the returned HTML response that it is reflected without any encoding. The browser will render the HTML response, treat our injected image element as valid markup, and execute the event handler. This is a simple example of a reflected XSS vulnerability.

Look out for common pitfalls!

Before submitting, make sure your XSS is actually exploitable. Self-XSS, in which the payload only executes in your own browser and requires you to perform the attack against yourself, is generally not accepted as a valid finding. The same applies to XSS in responses served with non-executable content types such as application/json or text/plain, browsers won't render any HTML in these contexts, making the vulnerability unexploitable in practice.

go.intigriti.com/exploiting-rxss

Exploiting Broken Access Control vulnerabilities

Broken access control (BAC) vulnerabilities arise when an application fails to properly enforce authorization rules, allowing users to access resources or perform actions beyond their intended privileges.

Unlike other vulnerability classes that rely on insufficient input sanitization, BAC stems from missing or flawed server-side validation checks. When exploited, these flaws can result in unauthorized data access, privilege escalation, or even full account takeover.

Exploiting BAC vulnerabilities

Consider the following API endpoint that returns your note based on a unique note ID:

```
GET /api/notes/1234 HTTP/2
Host: www.example.com
User-Agent: Mozilla/5.0...
Cookie: auth.token=eyJ...
```

```
HTTP/2 200 OK
Content-Type: application/json
Server: nginx
Content-Length: 70

{“id”: 1234, “title”: “IDOR test”,
“content”: “...” }
```

By simply changing the note ID to a different value, we can retrieve a different user’s private note, since the API fails to validate whether the private note actually belongs to us. This is an example of an **Insecure Direct Object Reference (IDOR)**, one of the most common and straightforward types of broken access control flaws.

```
GET /api/notes/1337 HTTP/2
Host: www.example.com
User-Agent: Mozilla/5.0...
Cookie: auth.token=eyJ...
```

```
HTTP/2 200 OK
Content-Type: application/json
Server: nginx
Content-Length: 71

{“id”: 1337, “title”: “My secret bug
bounty notes”, “content”: “...” }
```

Identifying BAC vulnerabilities

1. **Discovery**

   Start with mapping out all the application routes, API endpoints, and parameters that lead to functionalities and features your target application offers.

2. **Context**

   Next, understand what you're supposed to access with your current account privileges. Document which endpoints, features, and resources are available to your role or user level.

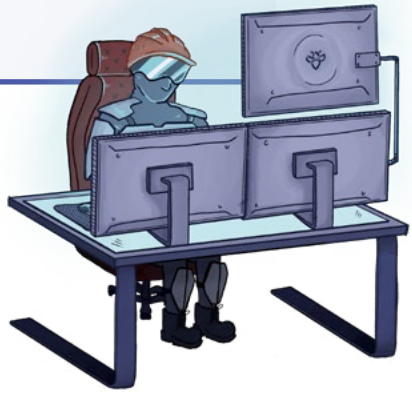
3. **Test**

   Finally, try to access resources you shouldn't be able to access. This includes data from other users as well as higher-privilege functionality. If the server does not properly deny access, you may have discovered a broken authorization flaw.

Always test responsibly!

Only test for broken access control vulnerabilities using test accounts you own or are permitted to use. **Never access, modify, or delete real users' data to prove a finding.** Create two separate test accounts and use them to validate access control issues safely. Also keep in mind that a 200 OK response alone does not always confirm the presence of a BAC vulnerability, always ensure you verify your findings before submitting your report.

go.intigriti.com/exploiting-bac



Writing compelling vulnerability reports

A well-written vulnerability report can be just as important as finding the bug itself.

A clear submission shortens the time to triage, leads to faster payouts, and avoids unnecessary back-and-forth with the triage team. We've lined up 5 tips below to help you write more compelling vulnerability reports:

1. **Title**

   Write a clear, descriptive title that specifies the vulnerability type, the affected component, and the impact. Avoid generic titles.

2. **Description**

   Describe the vulnerability, its location, the possible root cause, and any attack requirements. Keep it concise and avoid lengthy AI-generated paragraphs that add no value.

3. **Proof of concept**

   Always provide a validated, working PoC that demonstrates real exploitability in the target's intended configuration. Never submit unverified or AI-generated payloads.

4. **Steps to reproduce**

   Write clear, step-by-step instructions as if the triager has never seen the target before. One line per step, prerequisites upfront, and validate that every step works before submitting.

5. **Impact**

   State a realistic impact with specific consequences. Focus on what you can demonstrably prove, not theoretical possibilities.

IDOR on /api/v1/users/{id}/profile allows unauthorized access to user PII 1

Description 2

The /api/v1/users/{id}/profile endpoint fails to verify that the authenticated user matches the requested {id} parameter. By changing the user ID to another user's ID, an attacker can read that user's full profile, including name, email, phone number, and home address.

Proof of concept

```
GET /api/v1/users/1337/profile HTTP/2
Host: app.example.com
Authorization: Bearer eyJ...
---
HTTP/2 200 OK
Content-Type: application/json
...
{"id":1337,"name":"Intigriti", "email":"hunter2@example.
com", "phone":"+1-555-5555", "address":"1337 Hackers Ave"}
```

3

Steps to reproduce 4

1. Log in as user A (attacker) at app.example.com/login.
2. Navigate to your own profile and intercept the request to /api/v1/users/{your_id}/profile.
3. Change the user ID in the path to another user's ID (e.g., 1337).
4. Observe the full profile data of user 1337 in the response.

Impact 5

Any authenticated user can read the full personal profile (name, email, phone, home address) of any other user on the platform by iterating over user IDs.

Don't inflate your severity rating

Always assign an accurate severity for your submission. Overestimating it forces triagers to incorrectly prioritize your report, leading to reassessment and ultimately delaying the entire triage process.

go.intigriti.com/writing-reports

About Intigriti

Intigriti is a crowd-sourced security provider that helps organizations identify and address vulnerabilities through its global community of ethical hackers. With a community of 150,000+ security researchers, Intigriti connects hackers with bug bounty and vulnerability disclosure opportunities where their skills can help create real-world security impact. Whether you're an experienced hunter or submitting your first report, Intigriti offers a platform to put your skills to work.

Hacker spotlight: Marc-Oliver Munz (c1phy)

RESEARCHER
c1phy

Bug bounty hunting has completely changed the way I look at technology and security. It gave me the opportunity to continuously improve my technical skills, stay up to date with real-world attack techniques, and connect with talented people from all over the world. It also helped me grow professionally in my role and opened doors to opportunities I never expected when I started.

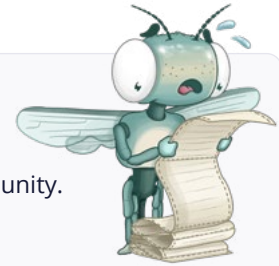
C1PHY

ETHICAL HACKER, AND INTIGRITI HACKER AMBASSADOR FROM GERMANY

Dive deeper into web hacking!

The web app hacking community continuously evolves. It requires a huge commitment to stay on top of the latest trends, vulnerabilities, attack vectors, and tools. We publish free, high-quality resources frequently to help you keep learning, keep your skill set sharp, and grow as a security researcher.

Visit our Hacking blog



Visit our Hacking blog for in-depth articles, cheat sheets, and vulnerability breakdowns curated by us for the community.

go.intigriti.com/hacking-blog

Follow us on Twitter/X

We post valuable security content daily, from vulnerability breakdowns to curated resources and community highlights.

Follow us @INTIGRITI to stay posted!

go.intigriti.com/twitter

Join us on Discord

With close to 15,000 members, our Discord is the place to find like-minded researchers, start collaborating, and stay up to date with everything happening in the Intigriti community.

go.intigriti.com/discord

İNTİGRİTİ

Bug bounty hunting is one of the few careers where curiosity and persistence are your most valuable assets. Whether you're a complete beginner or looking to sharpen your skills, the opportunity to find real vulnerabilities in real applications and get paid for it has never been more accessible.

This starter kit covers everything you need to go from zero to your first valid bug report. From reconnaissance and selecting your target to exploiting SQL injection, XSS, and broken access control vulnerabilities, each step is designed to be practical and immediately applicable. By the end, you'll also learn how to write compelling vulnerability reports that get triaged faster. The only thing left to do is to dive in!