A03: Injections

OWASP TOP 10 - Series

Presenter – Sanchay, Jalaj

@ null-meet, Airtel Centre, Gurgaon

19 November 2022

>_whoami (Jalaj)

- Tech Enthusiast
- Cyber security geek
- Ex-Suzuki, Ex-Dcm Hyundai (Mech.)
- Codes in C++ and Python. Exploring Go
- CTF Player
- CRAC research group : CVE Analysis and Cloud Security
- Learning about Cloud security and Malware analysis
- Bikes
- MMA

>_whoami (Sanchay)

With over 4-5 years of Practical experience in the field, I have good knowledge of Bug Hunting and PenTesting, especially System Hacking.

*Co-Founder of **HackersVilla CyberSecurity Pvt Ltd**, a security community and B2B VAPT Services Platform. Also working with Upgrad Campus as Subject Matter Expert.*

sanchayofficial

@sanchayofficial

sanchayofficial@gmail.com

SANCHAY SINGH

CYBERSECURITY EXPERT & TRAINER

A03: Injection

Injection slides down to the third position. 94% of the applications were tested for some form of injection with a max incidence rate of 19%, an average incidence rate of 3%, and 274k occurrences. Notable Common Weakness Enumerations (CWEs) included are CWE-79: Cross-site Scripting, CWE-89: SQL Injection, and CWE-73: External Control of File Name or Path.

These flaws occur because user controlled input is interpreted as actual commands or parameters by the application. Injection attacks depend on what technologies are being used and how exactly the input is interpreted by these technologies.

Common Weakness Enumerations

- *CWE-20 Improper Input Validation*
- *CWE-74 Improper Neutralization of Special Elements in Output Used by a Downstream Component ('Injection')*
- *CWE-75 Failure to Sanitize Special Elements into a Different Plane (Special Element Injection)*
- *CWE-77 Improper Neutralization of Special Elements used in a Command ('Command Injection')*
- *CWE-78 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')*
- *CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')*
- *CWE-80 Improper Neutralization of Script-Related HTML Tags in a Web Page (Basic XSS)*
- *CWE-83 Improper Neutralization of Script in Attributes in a Web Page*
- *CWE-87 Improper Neutralization of Alternate XSS Syntax*
- *CWE-88 Improper Neutralization of Argument Delimiters in a Command ('Argument Injection')*
- *CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')*
- *CWE-90 Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection')*
- *CWE-91 XML Injection (aka Blind XPath Injection)*
- *CWE-93 Improper Neutralization of CRLF Sequences ('CRLF Injection')*
- *CWE-94 Improper Control of Generation of Code ('Code Injection')*
- *CWE-95 Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection')*
- *CWE-96 Improper Neutralization of Directives in Statically Saved Code ('Static Code Injection')*
- *CWE-97 Improper Neutralization of Server-Side Includes (SSI) Within a Web Page*
- *CWE-98 Improper Control of Filename for Include/Require Statement in PHP Program ('PHP Remote File Inclusion')*
- *CWE-99 Improper Control of Resource Identifiers ('Resource Injection')*
- *CWE-100 Deprecated: Was catch-all for input validation issues*
- *CWE-113 Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting')*
- *CWE-116 Improper Encoding or Escaping of Output*
- *CWE-138 Improper Neutralization of Special Elements*
- *CWE-184 Incomplete List of Disallowed Inputs*
- *CWE-470 Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection')*
- *CWE-471 Modification of Assumed-Immutable Data (MAID)*
- *CWE-564 SQL Injection: Hibernate*
- *CWE-610 Externally Controlled Reference to a Resource in Another Sphere*
- *CWE-643 Improper Neutralization of Data within XPath Expressions ('XPath Injection')*
- *CWE-644 Improper Neutralization of HTTP Headers for Scripting Syntax*
- *CWE-652 Improper Neutralization of Data within XQuery Expressions ('XQuery Injection')*
- *CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')*

XSS INJECTIONS

Cross Site Scripting (XSS)

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

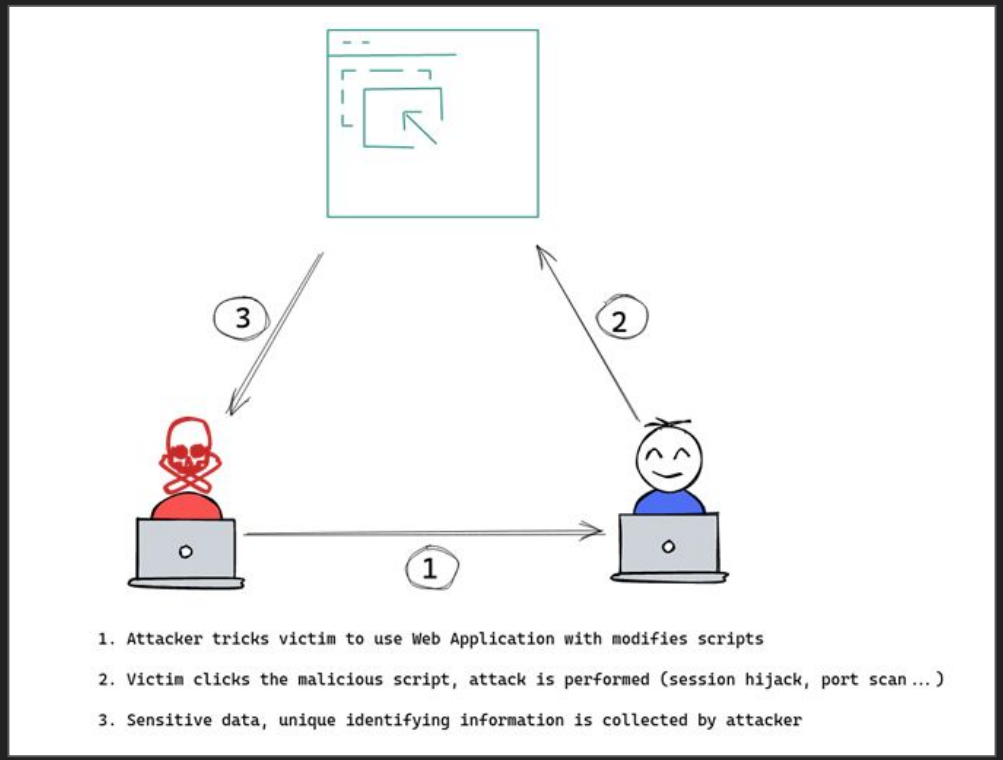
XSS is possible in javascript , CSS , Flash and VBScript

Cause : Unsanitized user input.

Types Of XSS

- **Reflected XSS,** works on browser
- **DOM Based XSS,** client side- writes to DOM
- **Stored XSS,** stores the script on server

Reflected XSS

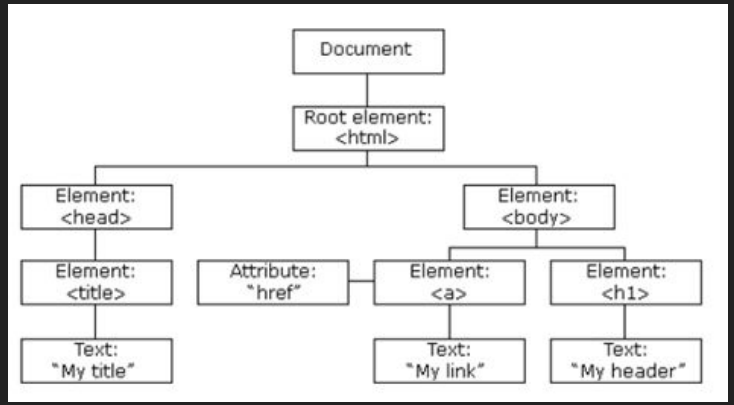


1. Attacker tricks victim to use Web Application with modifies scripts
2. Victim clicks the malicious script, attack is performed (session hijack, port scan ...)
3. Sensitive data, unique identifying information is collected by attacker

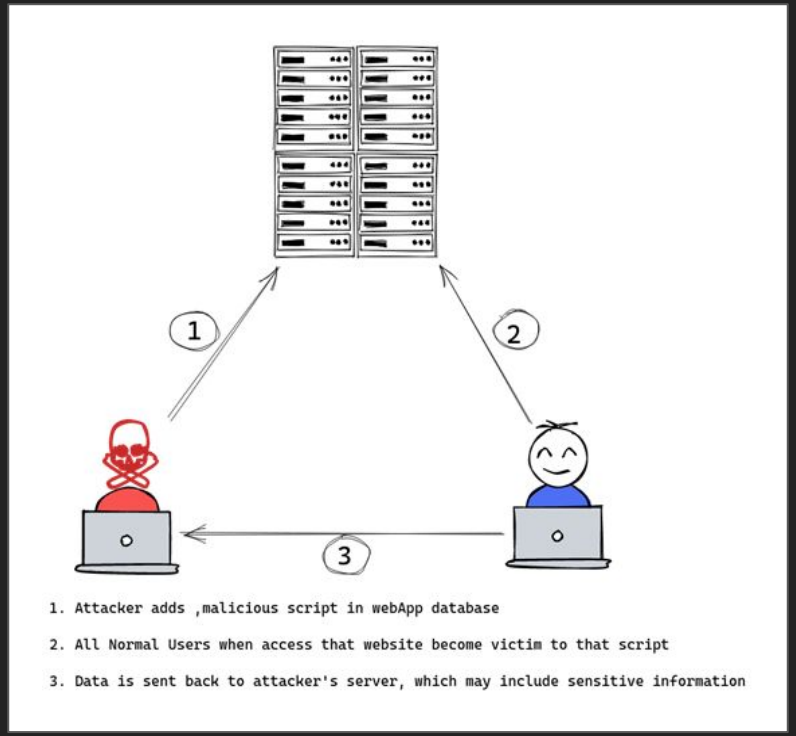
DOM-based XSS

The **Document Object Model (DOM)** is a programming interface that defines how to create, modify or erase elements in an HTML or XML document. DOM provides a standard API for the dynamic modification of the content, style, and structure of web page documents.

A **DOM model** represents each element as a node within a tree-like system, enabling easier programmatic access and management of the elements.



Stored XSS


1
2
3
1. Attacker adds ,malicious script in webApp database
2. All Normal Users when access that website become victim to that script
3. Data is sent back to attacker's server, which may include sensitive information

JS Payload Examples

```
<
script type = "text/javascript" >
    let l = ""; // Variable to store key-strokes in
document.onkeypress = function(e) { // Event to listen for key presses
        l += e.key; // If user types, log it to the l variable
        console.log(l); // update this line to post to your own server
    } <
    /script>
```

Figure 1 : Keylogger

```
<
script >
    for (let i = 0; i < 256; i++) { // This is looping from 0 to 255
        let ip = '192.168.0.' + i // Creates variable for forming IP
            // Creating an image element, if the resource can load, it logs to the
            /
            logs page.
        let code = '<img src="http://' + ip + '/favicon.ico"
        onload = "this.onerror=null; this.src=/log/' + ip + '" > '
        document.body.innerHTML += code // This is adding the image element to the
        webpage
    } <
    /script>
```

Figure 2 : Port Scanner

JS Payload Examples

<script>alert(“Hello World”)</script>

xxs link

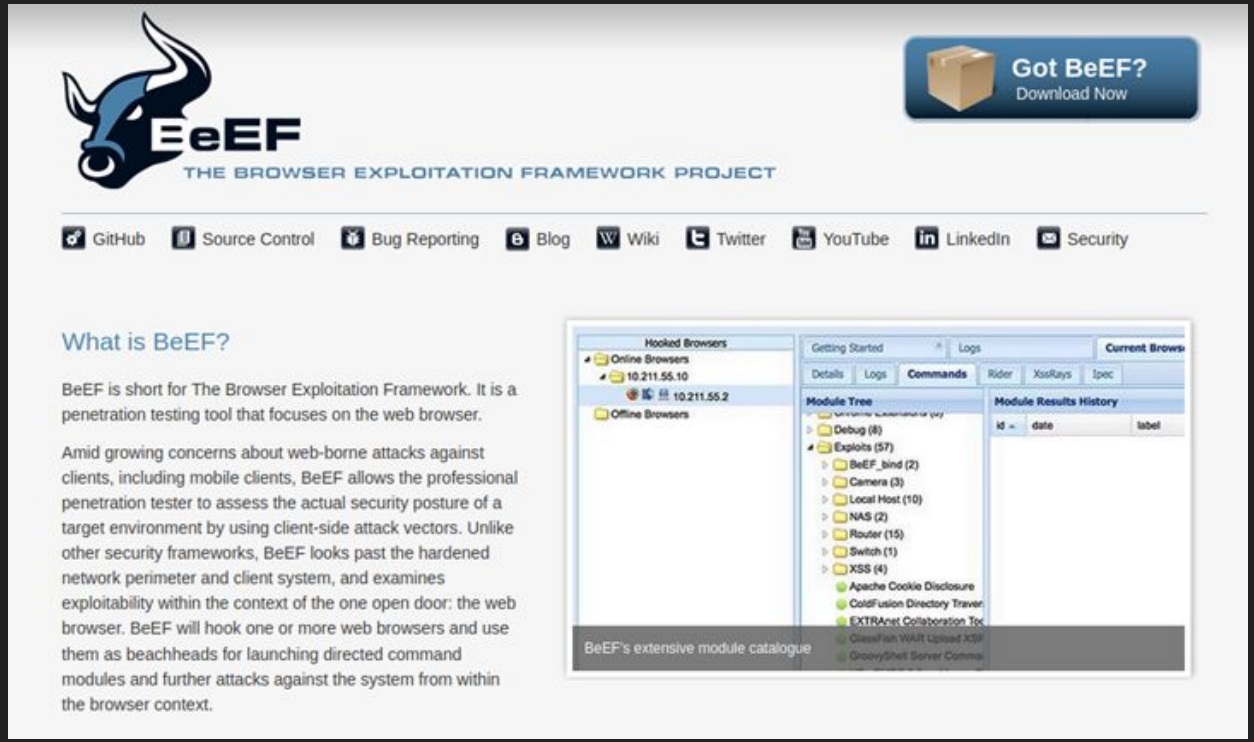
<<SCRIPT>alert("XSS");//<</SCRIPT>

<SCRIPT>document.write("XSS");</SCRIPT>

<script>alert(window.location.hostname)</script>

More payloads at:
xss-payload.com
github.com/payloadbox/xss-payload-list

Automate XSS hunting **beefproject.com**

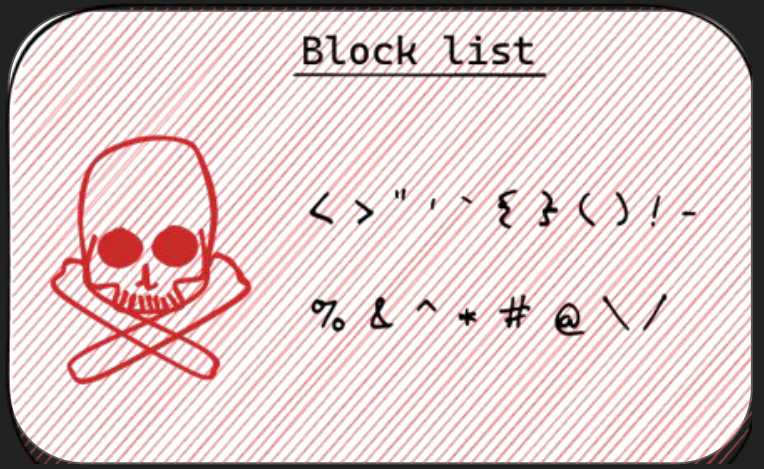


Prevention of XSS

“User Input cannot be trusted”

1. User Input Validation
2. User Input Transformation
3. Escaping Untrusted HTTP request data
4. Enable a Content Security Policy
5. Use a Framework!!

User Input Validation



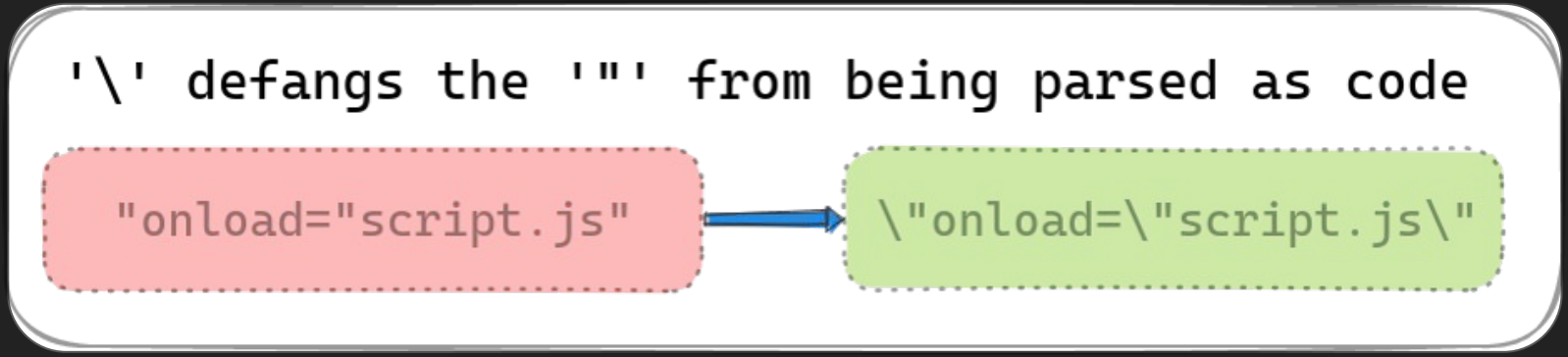
User Input Transformation

Django Framework Defaults

< is converted to <
> is converted to >
' (single quote) is converted to '
" (double quote) is converted to "
& is converted to &

Escaping Untrusted HTTP request data

Content like Http body, attribute, JavaScript, CSS, or URL



Enable Content Security Policy

- To enable CSP, you need to configure your web server to return the Content-Security-Policy HTTP header. (X-Content-Security-Policy)

- Alternatively, the <meta> element can be used to configure a policy, for example:

```
<meta
  http-equiv="Content-Security-Policy"
  content="default-src 'self'; img-src https://*; child-src 'none';" />
```

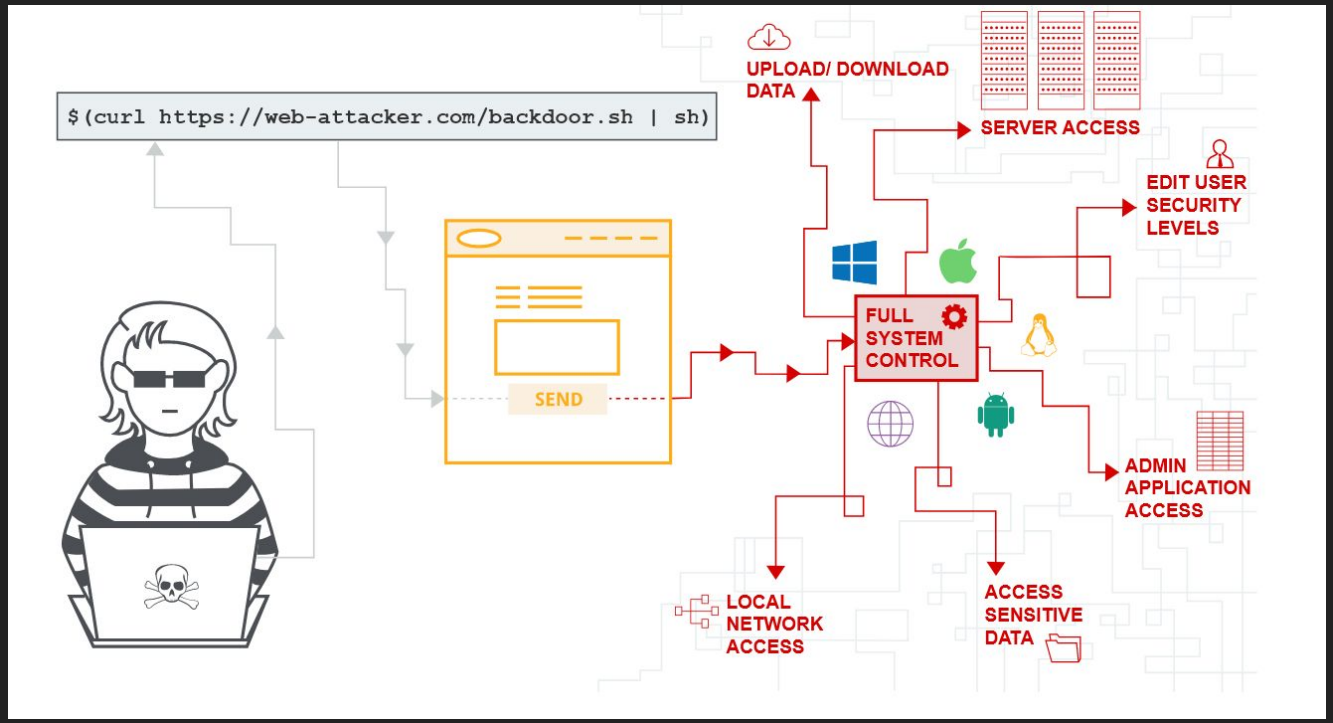
Further Reading Resources on XSS

COMMAND INJECTIONS

Command Injection

OS command injection *(also known as shell injection)* is a web security vulnerability that allows an attacker to execute arbitrary operating system (OS) commands on the server that is running an application, and typically fully compromise the application and all its data.

Command Injection



Prevention of Command Injections

“Never call out to OS commands from application-layer code”

- Validating against a whitelist of permitted values.
- Validating that the input is a number.
- Validating that the input contains only alphanumeric characters, no other syntax or whitespace.

SQL INJECTIONS

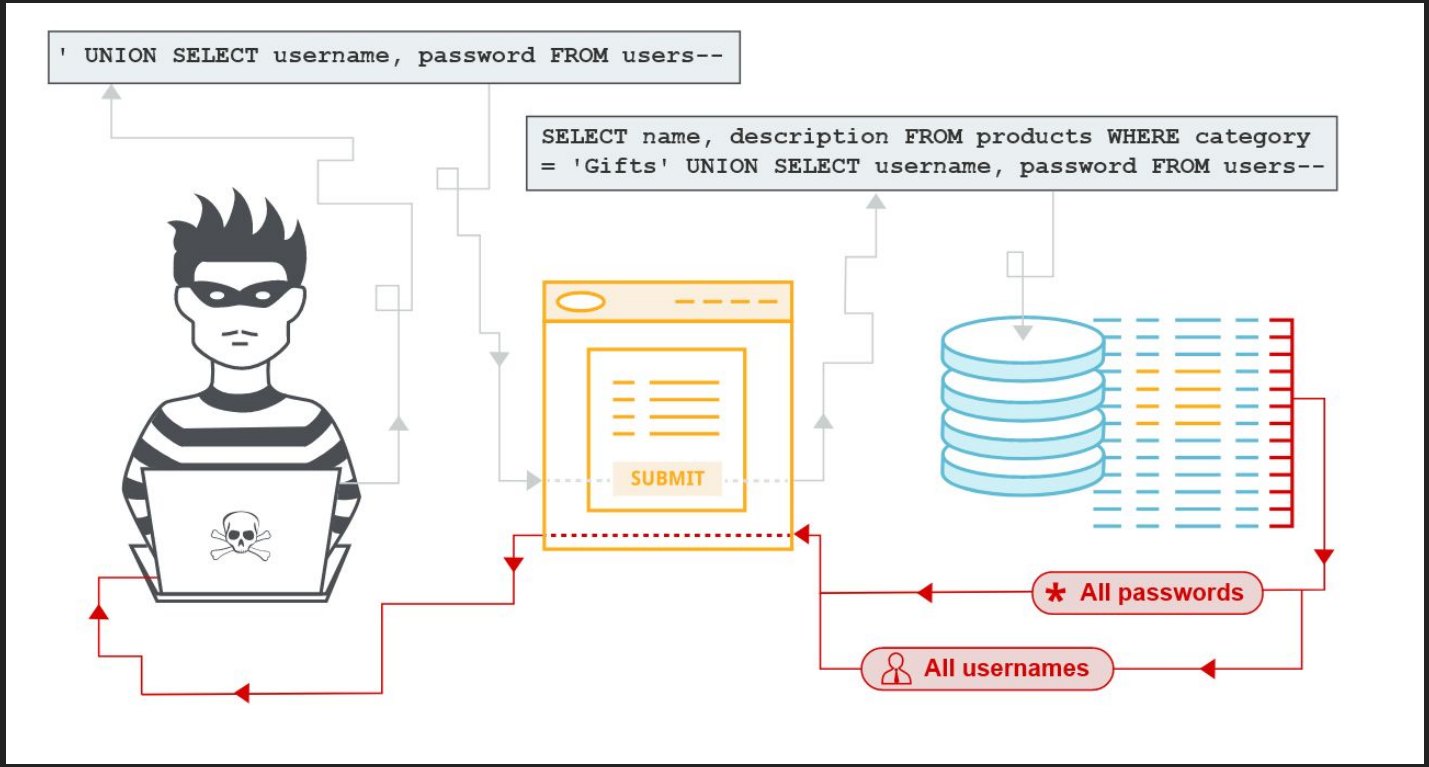
SQL Injections

SQL injection is a code injection technique that **might destroy your database.** SQL injection is one of the most common web hacking techniques. SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL Injections can be possible in PHP, ASP or related framework.

Cause : Unsanitized user input.

SQL Injections



Prevention of SQL Injections

“User Input cannot be trusted”

1. User Input Validation
2. User Input Transformation
3. Use a Framework!!

Thank You