SLIIT UNI
THE KNOWLEDGE UNIVERSITY

WEB SECURITY

Cybersecurity

JOURNAL BOOK
PRADHAP R it22337108
2nd Year 2nd semester

Day 01 2024/04/07

Today I started my journey to learn OWASP top 10 vulnerability. Destination of this journey I should learn about web application security project’s vulnerabilities.

Initially I started to figure out the OWASP Vulnerability.

I got the idea, there are 10 critical security risks for web applications. The attacker is using these methods to break the security.

- Injection
- Broken Authentication
- Sensitive Data Exposure
- XML External Entities (XXE)
- Broken Access Control
- Security Misconfiguration
- Cross-Site Scripting (XSS)
- Insecure Deserialization
- Using Components with Known Vulnerabilities
- Insufficient Logging and Monitoring

Understanding these vulnerabilities to identify the weak spots in a fortress, enabling us to fortify them effectively. Tomorrow, I aim to delve deeper into each vulnerability, gaining a more comprehensive understanding of the security of our web applications.

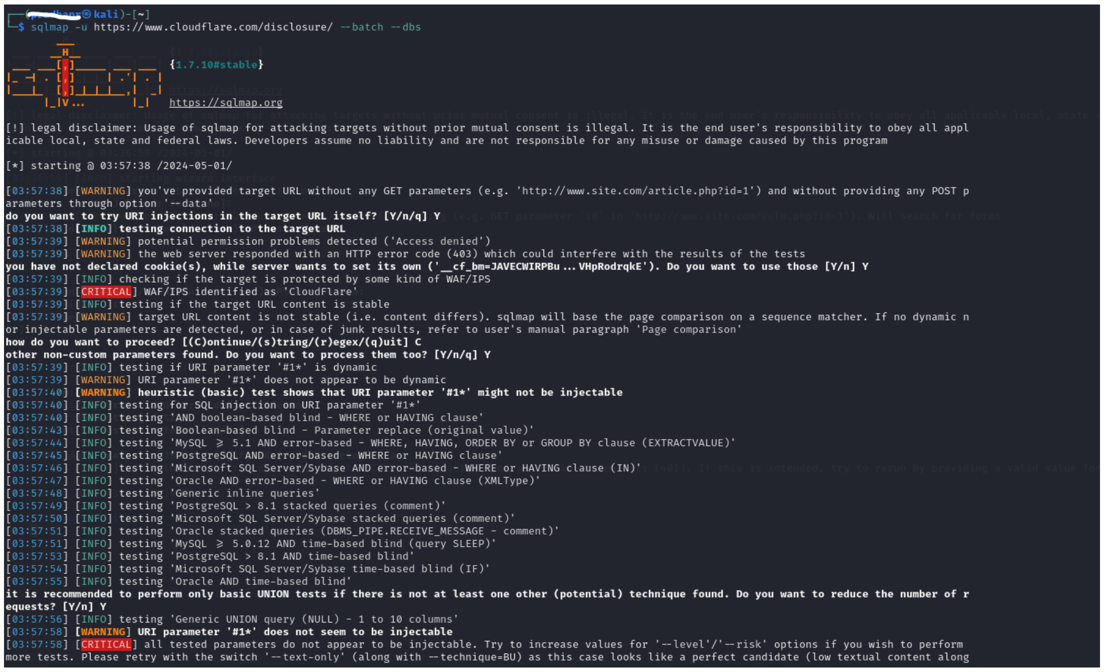
Day 02 2024/04/08

After gaining a basic understanding of the OWASP Top 10 vulnerabilities yesterday, today I decided to delve deeper into each one, starting with the first risk **Injection**.

SQL injection (SQLi) caught my attention, and I likened it to a hacker sneaking into a website's database by cleverly manipulating commands. This breach allows them to access, alter, or delete sensitive data, creating havoc.

To solidify my understanding, I engaged in practical exercises on the PortSwigger platform. These hands-on activities provided valuable insights into the mechanics of SQL injection, enhancing my knowledge and skills in combating this type of cyber threat.

I expanded my practical experience by using **SQLMap on Kali Linux.** This powerful tool allowed me to further explore SQL injection vulnerabilities in web applications. By running scans and tests, I gained hands-on experience in identifying and exploiting SQL injection flaws, thereby strengthening my proficiency in defending against such attacks.



```
┌──([redacted]㉿kali)-[~]
└─$ sqlmap -u https://www.cloudflare.com/disclosure/ --batch --dbs

                          {1.7.10#stable}

                          https://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all appl
icable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting @ 03:57:38 /2024-05-01/

[03:57:38] [WARNING] you've provided target URL without any GET parameters (e.g. 'http://www.site.com/article.php?id=1') and without providing any POST p
arameters through option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q] Y
[03:57:38] [INFO] testing connection to the target URL
[03:57:39] [WARNING] potential permission problems detected ('Access denied')
[03:57:39] [WARNING] the web server responded with an HTTP error code (403) which could interfere with the results of the tests
you have not declared cookie(s), while server wants to set its own ('__cf_bm=JAVECWIRPBu...VHpRodrqkE'). Do you want to use those [Y/n] Y
[03:57:39] [INFO] checking if the target is protected by some kind of WAF/IPS
[03:57:39] [CRITICAL] WAF/IPS identified as 'CloudFlare'
[03:57:39] [INFO] testing if the target URL content is stable
[03:57:39] [WARNING] target URL content is not stable (i.e. content differs). sqlmap will base the page comparison on a sequence matcher. If no dynamic n
or injectable parameters are detected, or in case of junk results, refer to user's manual paragraph 'Page comparison'
how do you want to proceed? [(C)ontinue/(s)tring/(r)egex/(q)uit] C
other non-custom parameters found. Do you want to process them too? [Y/n/q] Y
[03:57:39] [INFO] testing if URI parameter '#1*' is dynamic
[03:57:39] [WARNING] URI parameter '#1*' does not appear to be dynamic
[03:57:40] [WARNING] heuristic (basic) test shows that URI parameter '#1*' might not be injectable
[03:57:40] [INFO] testing for SQL injection on URI parameter '#1*'
[03:57:40] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[03:57:43] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[03:57:44] [INFO] testing 'MySQL ≥ 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[03:57:45] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[03:57:46] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[03:57:47] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[03:57:48] [INFO] testing 'Generic inline queries'
[03:57:49] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[03:57:50] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[03:57:51] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[03:57:51] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[03:57:53] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[03:57:54] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[03:57:55] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of r
equests? [Y/n] Y
[03:57:56] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[03:57:58] [WARNING] URI parameter '#1*' does not seem to be injectable
[03:57:58] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform
more tests. Please retry with the switch '--text-only' (along with --technique=BU) as this case looks like a perfect candidate (low textual content along
```

Day 02 2024/04/10

Today was all about diving into cybersecurity. I found out that broken authentication is a big deal in keeping websites safe. Basically, hackers try to crack passwords or find usernames to sneak into accounts. It's like they're trying every key to find the right one. I read about it on OWASP's website.

Then, I explored OAuth, a cool framework for authorization, on PortSwigger's site. OAuth helps with secure logins, like when you sign in with Google or Facebook. It's like showing your ID to get into a club!

0Also, I revisited OAuth 2.0 and how it's used on social media platforms. It's fascinating to see how it all works behind the scenes. I even tackled some lab activities to practice.

Day 02 2024/04/12

Today, I decided to dive into understanding how sensitive data gets exposed. You know, like when personal info leaks out on social media? It got me thinking about how attackers target people's sensitive info. So, I realized that many attackers mess with encryption stuff, like public or private keys, and do sneaky middleman attacks to grab sensitive data.

I remember a few years back when some info wasn't encrypted, making it easy for attackers to snatch it up. But now, we've got super strong encryption methods, so it's crucial to encrypt our sensitive info to keep it safe. I picked up all this from browsing owasp.org.

So, after learning about sensitive data exposure, I decided to test my skills by doing some SQL injection labs on PortSwigger. It's been quite the journey today.

Day 02 2024/04/13

Today, I learned about something new called XML external entity (XXE) injections. It's a sneaky way hackers can mess with application servers, even messing up system files or other systems. Surprisingly, some websites still use XML code, leaving them open to these kinds of attacks.

XXE injections let attackers sneak in outside code, messing with how servers respond. It's kind of like a denial-of-service attack, where servers get overwhelmed and can't handle requests properly. This study showed me just how vulnerable websites can be, with important server info at risk.

One big thing I learned is that XXE attacks give hackers access to important files and systems behind websites. This means we've got to take this seriously in web development. Things like turning off certain features can help protect against XXE attacks.

To learn more, I tried some XXE injection lab sessions on portSwagger.org. But now, I'm taking a break for three days because of the festival season. I should spend time with my family. However, my commitment to learning remains steadfast, and I look forward to resuming my exploration on April 16th.

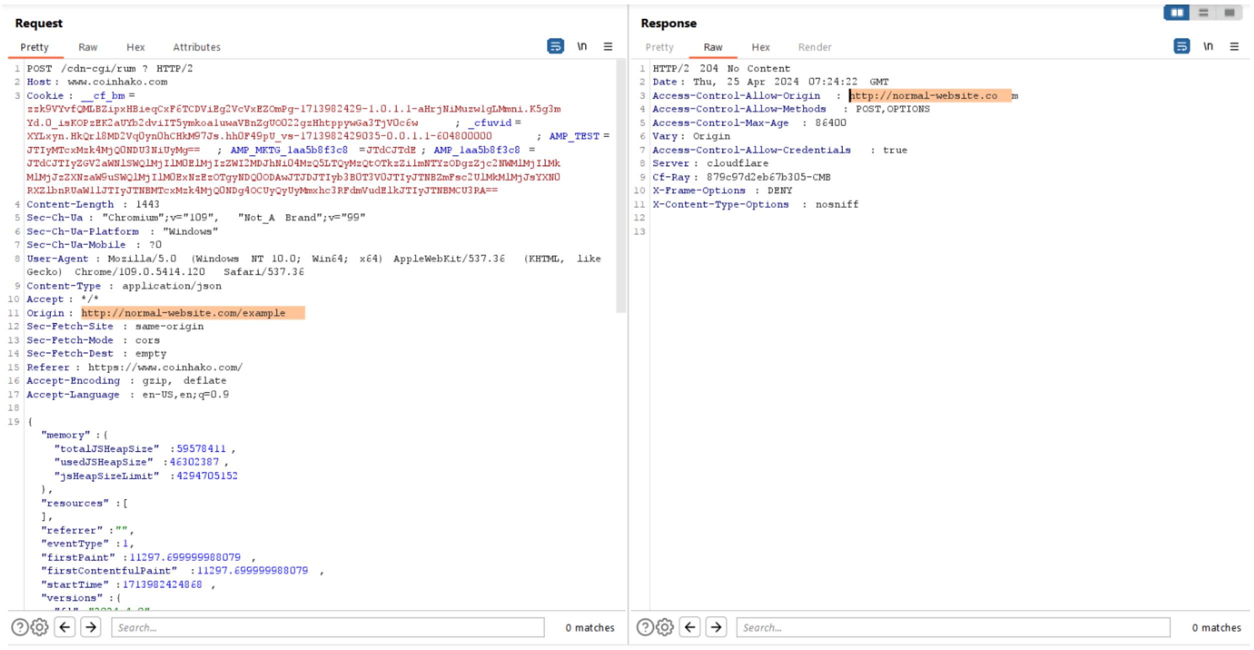
Day 02 2024/04/16

Today, first I reviewed what I did on my journey then I learned about Broken Access Control. Access control in web applications is crucial for determining who can do what. It involves confirming user identity (authentication), tracking user sessions, and deciding if a user can perform certain actions. Weak access controls are a major security risk. Designing them involves juggling technical, business, and legal aspects, which can lead to mistakes.based on Access control topic I did some lab practices and I got some idea about Access control.

Also, Today I started to find the bug in hackerone related from broken access control.

Through is host I understood about access control Origin. I found this issue in this website.and I studied about this issue based on this issue we can put Unauthorize web host into that origin because access control allow origin this allowing all web hosts so its one of the broken access controls iss so I submit the report to the hackerone.

This is the stuff I learned today.


Request

```
POST /cdn-cgi/rum ? HTTP/2
Host : www.coinhako.com
Cookie : __cf_bm =
zzk9VYvfQMLBZipxHBieqCxF6TCDViEg2VcVxEZOmPg-1713982429-1.0.1.1-aHrjNiMuzwlgLMmni.K5g3m
Yd.0_isKOPzEK2aUYb2dviIT5ymkoaluwaVBnZgUO022gzHhtppywGa3TjVOc6w     ; _cfuvid =
XYLxyn.HkQr18MD2Vq0yn0hCHkM97Js.hh0F49pU_vs-1713982429035-0.0.1.1-604800000      ; AMP_TEST =
JTIyMTcxMzk4MjQ0NDU3NiUyMg==   ; AMP_MKTG_1aa5b8f3c8  =JTdCJTdE ; AMP_1aa5b8f3c8  =
JTdCJTIyZGV2aWNlSWQlMjIlM0ElMjIzZWI2MDJhNi04MzQ5LTQyMzQtOTkzZi1mNTYzODgzZjc2NWMlMjIlMk
MlMjJzZXNzaW9uSWQlMjIlM0ExNzEzOTgyNDQ0ODAwJTJDJTIyb3B0T3V0JTIyJTNBZmFsc2UlMkMlMjJsYXN0
RXZlbnRUaW1lJTIyJTNBMTcxMzk4MjQ0NDg4OCUyQyUyMmxhc3RFdmVudElkJTIyJTNBMCU3RA==
Content-Length : 1443
Sec-Ch-Ua : "Chromium";v="109",   "Not_A  Brand";v="99"
Sec-Ch-Ua-Platform  : "Windows"
Sec-Ch-Ua-Mobile  : ?0
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64)  AppleWebKit/537.36   (KHTML,  like
Gecko)  Chrome/109.0.5414.120   Safari/537.36
Content-Type : application/json
Accept : */*
Origin : http://normal-website.com/example
Sec-Fetch-Site  : same-origin
Sec-Fetch-Mode  : cors
Sec-Fetch-Dest  : empty
Referer : https://www.coinhako.com/
Accept-Encoding  : gzip,  deflate
Accept-Language  : en-US,en;q=0.9

{
  "memory" :{
    "totalJSHeapSize"  :59578411 ,
    "usedJSHeapSize"  :46302387 ,
    "jsHeapSizeLimit"  :4294705152
  },
  "resources" :[
  ],
  "referrer" :"",
  "eventType" :1,
  "firstPaint" :11297.699999988079  ,
  "firstContentfulPaint"  :11297.699999988079  ,
  "startTime" :1713982424868  ,
  "versions" :{
```

Response

```
HTTP/2  204  No  Content
Date :  Thu,  25  Apr  2024  07:24:22  GMT
Access-Control-Allow-Origin   :  http://normal-website.co   m
Access-Control-Allow-Methods   :  POST,OPTIONS
Access-Control-Max-Age   :  86400
Vary :  Origin
Access-Control-Allow-Credentials   :  true
Server :  cloudflare
Cf-Ray :  879c97d2eb67b305-CMB
X-Frame-Options  :  DENY
X-Content-Type-Options  :  nosniff
```

Day 02 2024/04/17

Today, I learned about Security Misconfiguration. The security Misconfiguration occurs when security settings in a web application are not implemented or configured properly. This could happen at any level of the application stack, including the web server, application server, database, framework, or custom code.

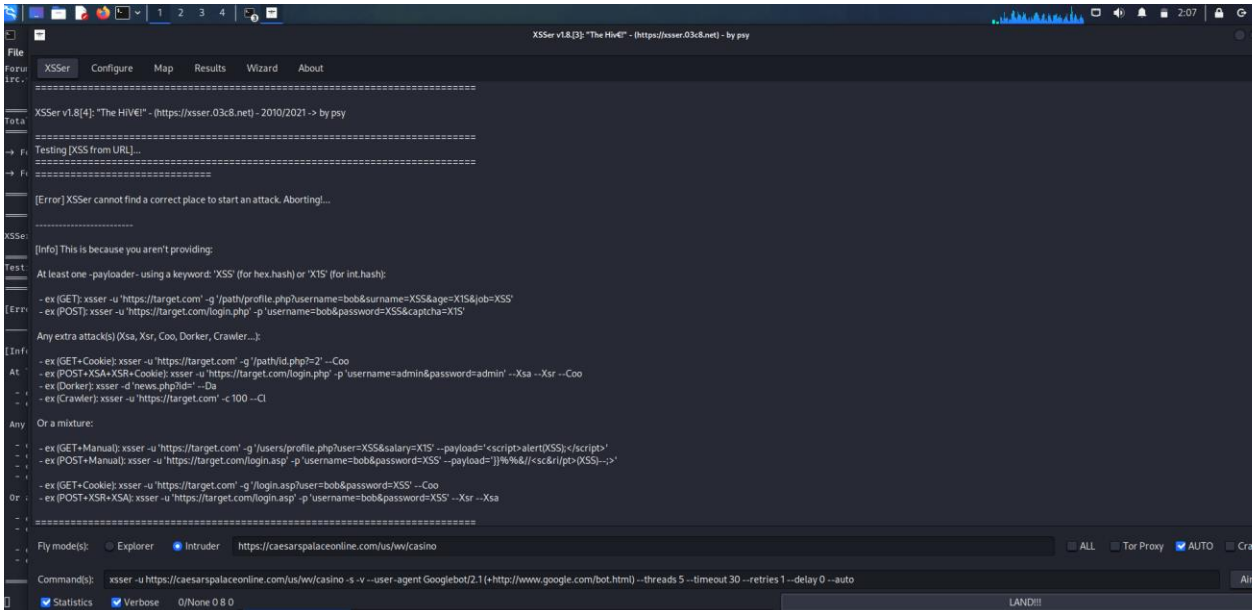
Nowadays XML External Entities (XXE) is main reason for security misconfiguration, so I tried understanding through the XML lab activities.XML injection are allowing the external code inject into the host request its directly deal with the server so attacker can retrieve the server data files from server.

After this stuff happened, I tried to find the bug in hackerone from what I'm studied. Unlucky I could not find the bugs but I got more knowledge from A Tool **SQLMap scan**.

Day 02 2024/04/18

Today, I learned about Cross-Site Scripting (XSS). Cross-Site Scripting (XSS) is a serious problem for web sites and the people who use them. It's one of the top risks according to OWASP, a group that focuses on web security. XSS happens when attackers find holes in a website's security. They use these holes to sneak harmful code into the website. Then, when other people use the Website, the code runs on their computers without them knowing.

Through the Portswigger I tried to do some lab practices to gain the knowledge about XSS.And also I got the new tool called **xsser.**


XSSer v1.8.[3]: "The Hiv€!" - (https://xsser.03c8.net) - by psy
File
XSSer
Configure
Map
Results
Wizard
About
2:07

```
===========================================================================

XSSer v1.8[4]: "The HiV€!" - (https://xsser.03c8.net) - 2010/2021 -> by psy

===========================================================================
Testing [XSS from URL]...
===========================================================================
==============================

[Error] XSSer cannot find a correct place to start an attack. Aborting!...

------------------------

[Info] This is because you aren't providing:

 At least one -payloader- using a keyword: 'XSS' (for hex.hash) or 'X1S' (for int.hash):

  - ex (GET): xsser -u 'https://target.com' -g '/path/profile.php?username=bob&surname=XSS&age=X1S&job=XSS'
  - ex (POST): xsser -u 'https://target.com/login.php' -p 'username=bob&password=XSS&captcha=X1S'

 Any extra attack(s) (Xsa, Xsr, Coo, Dorker, Crawler...):

  - ex (GET+Cookie): xsser -u 'https://target.com' -g '/path/id.php?=2' --Coo
  - ex (POST+XSA+XSR+Cookie): xsser -u 'https://target.com/login.php' -p 'username=admin&password=admin' --Xsa --Xsr --Coo
  - ex (Dorker): xsser -d 'news.php?id=' --Da
  - ex (Crawler): xsser -u 'https://target.com' -c 100 --Cl

 Or a mixture:

  - ex (GET+Manual): xsser -u 'https://target.com' -g '/users/profile.php?user=XSS&salary=X1S' --payload='<script>alert(XSS);</script>'
  - ex (POST+Manual): xsser -u 'https://target.com/login.asp' -p 'username=bob&password=XSS' --payload='}}%%&//<sc&ri/pt>(XSS)--;>'

  - ex (GET+Cookie): xsser -u 'https://target.com' -g '/login.asp?user=bob&password=XSS' --Coo
  - ex (POST+XSR+XSA): xsser -u 'https://target.com/login.asp' -p 'username=bob&password=XSS' --Xsr --Xsa

===========================================================================
```

Fly mode(s): Explorer [✓ Intruder] https://caesarspalaceonline.com/us/wv/casino ALL Tor Proxy [✓ AUTO] Cra

Command(s): xsser -u https://caesarspalaceonline.com/us/wv/casino -s -v --user-agent Googlebot/2.1 (+http://www.google.com/bot.html) --threads 5 --timeout 30 --retries 1 --delay 0 --auto

[✓ Statistics] [✓ Verbose] 0/None 0 8 0 LAND!!!

Day 02 2024/04/19

Today, I learned about Insecure deserialization. Insecure deserialization like when developers create software, they often need to store or send complex data structures. To make this easier, they use a process called serialization, which basically turns these complicated data structures into a simpler format that can be easily managed.

Now, the opposite of serialization is deserialization. This is when the simplified data is converted back into its original, complex form. Here's where the trouble starts: if a program isn't strong, it might just accept this deserialized data without making sure it's safe. It's like blindly opening a package without checking what's inside first.

And that's where insecure deserialization comes into play. If a bad actor messes with this serialized data before it's deserialized, they could sneak in some nasty stuff, like unauthorized commands or access to sensitive info. It's like someone tampered with your package and put something harmful in it.

After got an idea about Insecure deserialization, Through the PicCTF I just tried to do some flag finding based on cookie.

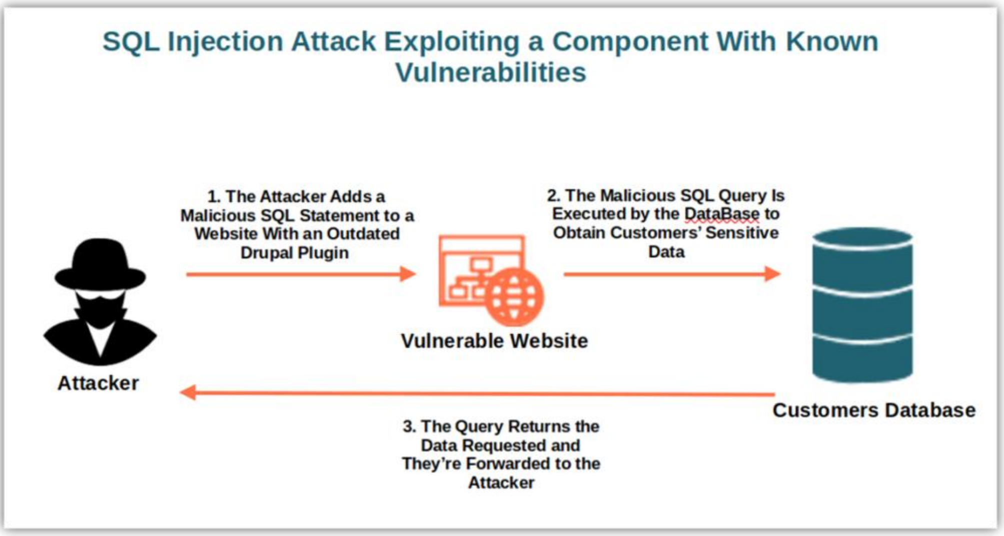
Day 02 2024/04/21

Today, I learned about “Using Components with Known Vulnerabilities”. Those components might have flaws or weaknesses, just like how a tool might be a bit worn out or a material might be faulty. Now, if i use these flawed components in my works without checking them thoroughly, it's like building something without realizing there's a weak spot.

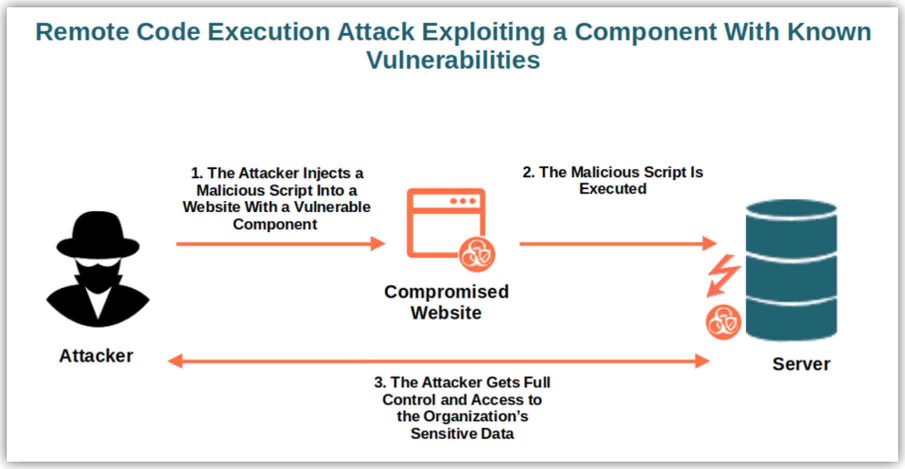
And that's where the risk lies. If attackers know about these weaknesses in the components I’m using, they could exploit them to get into my system or cause damage. It's like someone finding out about the weak spot in my website or web apps and using it to break in or mess things up.

I learned about 4 dangers of using components with known vulnerabilities. Those are Code Injection, code execution, buffer overflow and Cross-site Scripting.

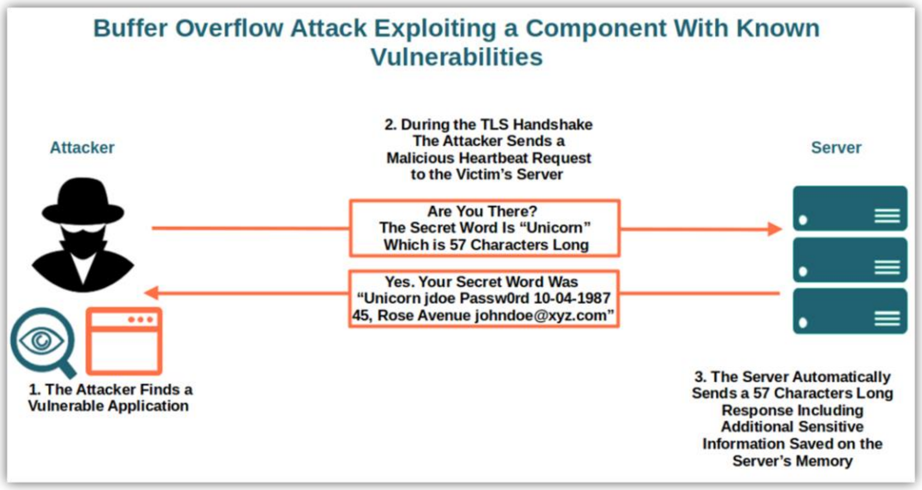
1. <u>Code Injection</u>


SQL Injection Attack Exploiting a Component With Known Vulnerabilities
1. The Attacker Adds a Malicious SQL Statement to a Website With an Outdated Drupal Plugin
2. The Malicious SQL Query Is Executed by the DataBase to Obtain Customers' Sensitive Data
Attacker
Vulnerable Website
Customers Database
3. The Query Returns the Data Requested and They're Forwarded to the Attacker

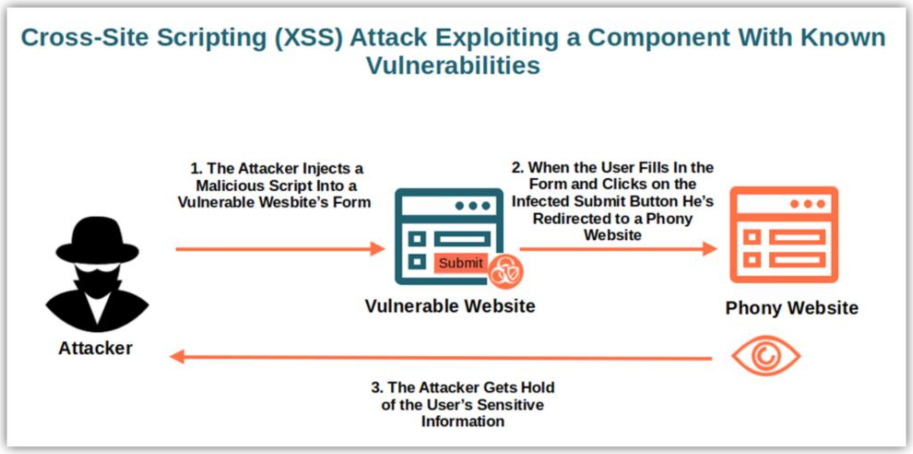
2. Code Execution


Remote Code Execution Attack Exploiting a Component With Known Vulnerabilities
1. The Attacker Injects a Malicious Script Into a Website With a Vulnerable Component
2. The Malicious Script Is Executed
Compromised Website
Attacker
Server
3. The Attacker Gets Full Control and Access to the Organization's Sensitive Data

3. Buffer Overflow


Buffer Overflow Attack Exploiting a Component With Known Vulnerabilities
Attacker
2. During the TLS Handshake The Attacker Sends a Malicious Heartbeat Request to the Victim's Server
Server
Are You There? The Secret Word Is "Unicorn" Which is 57 Characters Long
Yes. Your Secret Word Was "Unicorn jdoe Passw0rd 10-04-1987 45, Rose Avenue johndoe@xyz.com"
1. The Attacker Finds a Vulnerable Application
3. The Server Automatically Sends a 57 Characters Long Response Including Additional Sensitive Information Saved on the Server's Memory

4. Cross Site Scripting (XXS)


Cross-Site Scripting (XSS) Attack Exploiting a Component With Known Vulnerabilities
1. The Attacker Injects a Malicious Script Into a Vulnerable Wesbite's Form
2. When the User Fills In the Form and Clicks on the Infected Submit Button He's Redirected to a Phony Website
Submit
Vulnerable Website
Phony Website
Attacker
3. The Attacker Gets Hold of the User's Sensitive Information

Day 02 2024/04/22

Today, I learned about Insufficient Logging and Monitoring. When my software systems don't keep proper logs of important events, like user logins, access attempts, or changes to sensitive data, it's like running that store without any way to track what's happening. And if there's a security breach or some suspicious activity, I won't have any clues to figure out what went wrong or who's responsible.

Now, let's add monitoring to the find suspicious. Monitoring is like having a security guard who keeps an eye on everything happening in my system. With proper monitoring in place, I can spot unusual behavior, detect potential threats, and respond quickly to any security incidents.

Conclusion

As I completed this journal documenting my journey to learn about the OWASP Top 10 Vulnerabilities and improve my understanding of web application security, I was filled with a sense of accomplishment and growth. Over the past few days, I've explored each of the OWASP Top 10 Vulnerabilities and explored their nuances, impacts, and mitigation strategies.

My journey began with an overview of the OWASP Top 10 Vulnerabilities, giving me a road map to navigate the issues of web application security. Each day, I delved deeper into specific vulnerabilities such as injection, broken authentication, sensitive data exposure, XML External Entities (XXE), broken access control, security misconfiguration, cross-site scripting (XSS), insecure deployment, using Known vulnerabilities and adequate logging and monitoring.

Through hands-on exercises, labs, and exploring real-world scenarios, I gained valuable insight and practical skills in identifying, exploiting, and mitigating security vulnerabilities in web applications. From simulating SQL injection attacks to understanding the importance of access control mechanisms, from exploring the risks of insecure deserialization to recognizing the importance of proper logging and monitoring, each day presented new challenges and opportunities for learning.

As I reflect on this journey, I am reminded of the critical importance of web application security in today's digital landscape. With cyber threats evolving rapidly and attackers constantly seeking to exploit vulnerabilities, the need for robust security measures has never been greater. By equipping myself with knowledge and skills in web application security, I am better prepared to contribute to the development of secure and resilient web applications, protecting data, privacy and user trust.

While this journal marks the end of a certain learning period, my commitment to continuous learning and improvement remains unwavering. As I move forward into the future, I take with me the lessons and experiences learned during this journey, knowing that they will be a solid foundation for my ongoing growth and development in the cybersecurity industry.

Grateful for the shared experiences and knowledge gained, I look forward to the next chapter in my journey to excellence in web application security.

End Of Journal