International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 04 | April 2022 www.irjet.net p-ISSN: 2395-0072

OWASP Top 10 Web Attacks (2017) with Prevention Methods

Prasad Patole[1], Aditya Totade[2], Piyush Patil[3], Prof. Renuka Nagpure[4]

[1][2][3]*Student, Department of Information Technology, Atharva College of Engineering, Mumbai*
[4]*Professor, Department of Information Technology, Atharva College of Engineering*

Abstract - *In today's world as everyone is using some kind of device that connects to internet, it is very important to take care of internet security. It does not mean that you should dive deeper and learn all the aspects of cyber-security; but you should at least learn about cyber hygiene and practice them while you are connected to the internet. The internet is the ultimate source of knowledge but, it is also habitat of a small percentage of hackers (malicious actors to be specific). This paper will help you to understand on very basic level how the most performed attacks are carried out and what a developer can do to avoid and mitigate these attacks. According to OWASP (Open Web Application Security Project) these are the top 10 attacks that were the most critical security risks to web applications.*

Key Words: **Cyber-Hygiene, Cyber-Security, OWASP Top 10, Malicious Actors**

1. INTRODUCTION

In this day and age the usage and requirement of the internet is increasing day by day. There are a lot of reasons for this, maybe because of pandemic or maybe because of ease of access; whatever be the reason but internet is here to stay and it is a fact! As the use of internet is increasing the threats posed by malicious actors are also increasing. To tackle the situation we should learn basics of cyber-hygiene and cyber-security. There are a lot of web developers who may forget to take precautionary measures to mitigate the security risks which resulted into the cyber-attacks. The cyber-attacks are increasing as new technologies are emerging per year [1]. In this paper we demonstrated all the top 10 attacks and showed how the attacker may attack a particular site with a particular type of vulnerability. The actual steps may change but the general approach is usually stays same. It will show how a certain type of vulnerability van be leveraged to gain control of the web application.

1.1 MOTIVATION

In today's modern lifestyle everything you can imagine is digital and somehow connected to the internet from your watch to toilets! Yes, even toilets (thanks to IOT); Because of this ever growing need of internet the internet security is a big question; since people are using internet but do not know how to keep themselves safe on the internet. As an aspiring team web developers we thought that not a lot of developers take care of writing safe and secure code while developing web applications or any application in general. This was the sole reason for us to choose this path and since it is fairly a virgin territory and a lot of things are yet to be uncovered.

1.2 AIM and OBJECTIVE

The main aim of this project is to demonstrate all possible attacks from OWASP Top 10 and writing down all the findings of said demonstrations.

Following are the Objectives:

1. To demonstrate the attacks on the vulnerable virtual machines provided by tryhackme.com (since this is one of the only legal way to learn and demonstrate web attacks).

2. To construct and publish paper from the findings of demonstrations.

1.3 BASIC CONCEPT

In these demonstrations the basic concept is that we will perform the web attacks using the dedicated Linux distribution named "KALI Linux" which is a Linux distribution that comes with a lot of tools to perform Penetration Testing; there is another famous distribution named Parrot OS, we chose KALI since it is most up-to-date distribution. Using KALI Linux virtual machine and tryhackme.com (which provides users with vulnerable machines and even explanations for learning) we will demonstrate attacks and all the screenshots are taken from these demonstrations.

2. OWASP TOP 10

2.1 INJECTION

Injection attacks are one of the most performed and are one of the most dangerous attack according to OWASP. Injection attacks such as SQL, OS, NoSQL injection, occurs when a malicious data is sent to an interpreter as the part of command/query. When the interpreter processes the query it will trigger the malicious code and the attacker can get access to the database which contains sensitive data such as admin login, user login and may contain personal information.

2.1.1 DEMONSTRATION

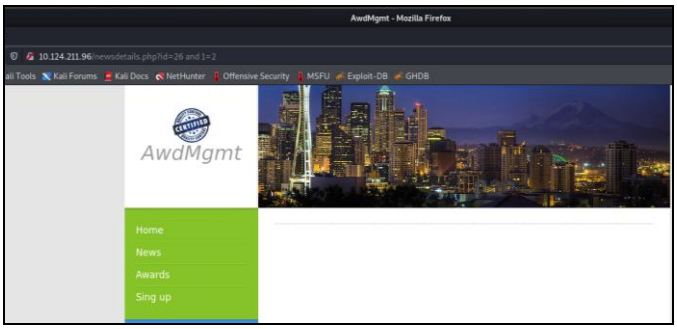

Fig. 1: SQL Injection Payload

As you can see in Fig.1 above while exploring we found a vulnerability in the news tab. The payload used in this case was – 'http://10.124.211.96/newsdetails.php?id=26 and 1=2'

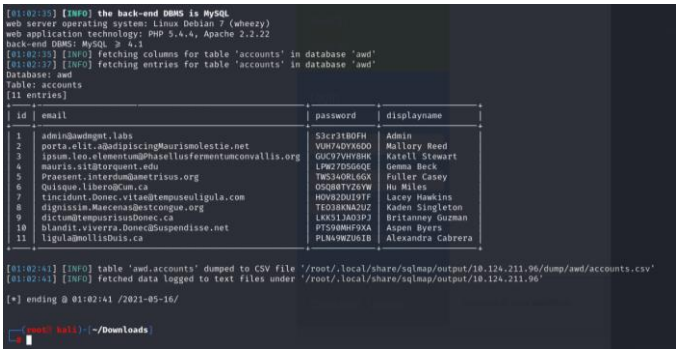

Fig. 2: Result of SQLMap

Using a tool named 'SQLMap' we were able to exploit this vulnerability and was able to gain the database information.

2.1.2 PREVENTION

1. Preventing SQL injection and Injection attacks in general is rather easy if you understand how this attack is executed; basically, to prevent injection attack you have to prevent it from triggering any malicious code or any code that is entered as an input.

2. Keeping the data separate from commands and queries. The preferred way is to use a safe API, it avoids the use of the interpreter entirely or provides a parameterized interface.

2.2 BROKEN AUTHENTICATION

Broken authentication is a vulnerability in any web app or any app which occur when functions related to authentication and session management are implemented incorrectly, allowing malicious actor to compromise passwords, keys(such as RSA keys), session tokens and gain access to accounts and profiles of victims.

2.2.1 DEMONSTRATION

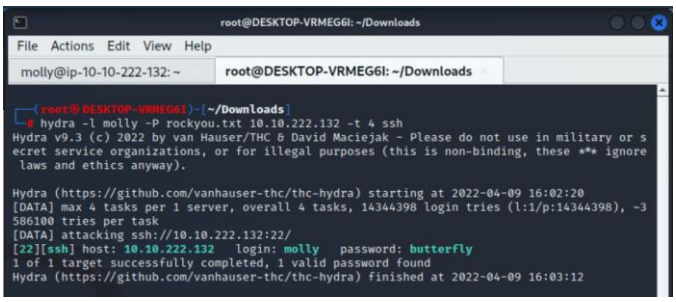

Fig. 3: Brute-force attack to gain the credentials

In above fig. 3 we performed a 'dictionary-based' brute-force attack using a password file named 'rockyou.txt' which was uploaded by hacker who hacked a company 'RockYou' back in 2009; since they stored their passwords in unencrypted form. The attacker got 32 million passwords, but the rockyou.txt used in this demo is provided by Kali and contains 12 Million passwords.

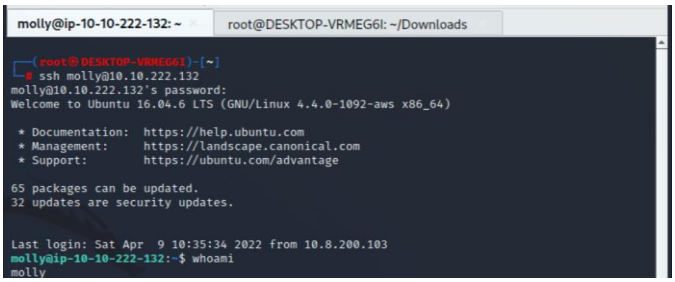

Fig. 4: Gaining Access with found credentials

In fig.4 you can see that we were able to gain the access to the victim's device using 'ssh'. The tool used was 'Hydra' we could have used other tools or other approach (e.g tools such as Gobuster, Burp Suit, etc.) but we have to choose the tools and approach according to the situation, since not all situations will be identical or it is highly unlikely.

2.2.2 PREVENTION

1. First of all stop using passwords; instead use 'passphrases' with numbers, letters and special symbols, e.g. 'Il1k3r41Nb0W$'.
2. Always change default credentials as the part of initial setup of any device especially routers and IOT devices (further discussed in 2.6).
3. Implementation of multi-factor authentication or at least two-factor authentication wherever possible.

2.3 SENSITIVE DATA EXPOSURE

Often many web applications do not properly hide the sensitive data, such as database files, it a very rookie mistake

to make but still is a mistake so that it is a possibility. This mistake can result in data is visible to anyone that is accessing the public page which they should not be able to access in the first place.

2.3.1 DEMONSTRATION

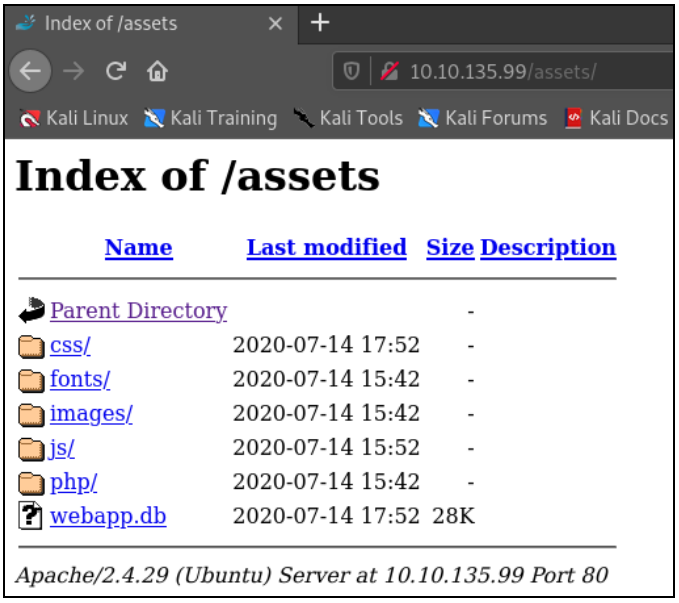

Fig. 5: Exposed Sensitive Data

In fig. 5 we can see a particular file named 'webapp.db' which is actual database file; which contains sensitive information. The folder is not harder to find we can easily find all the related files using tools such as gobuster and dirbuster which finds all types of files e.g. folders, .php, .js, .bak files etc.

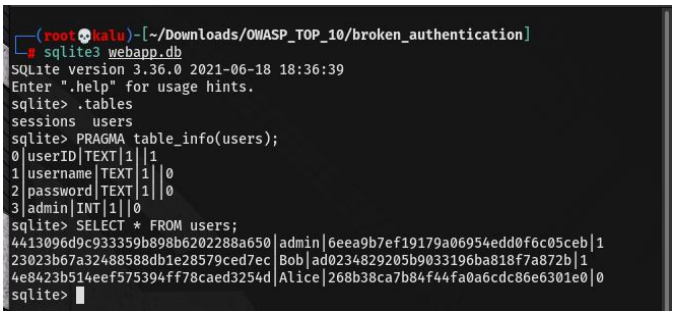

Fig. 6: contents of 'webapp.db' file

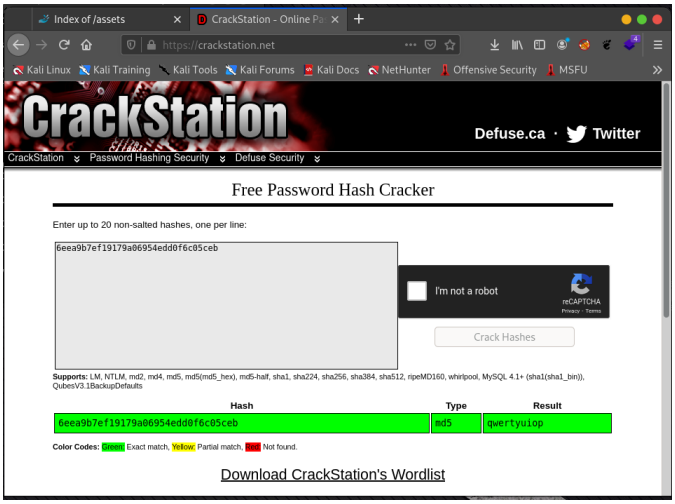

Fig. 7: decrypting the password

As demonstrated in fig. 6 we can read the content of file using sqlite3 and found that it contains credentials of users including 'admin' and using 'crackstation' (which is a useful tool to unhash the passwords) we can crack the password as shown in fig. 7 above.

2.3.2 PREVENTION

1. Use strong and up-to-date frameworks to implement the web app.
2. Always classify the data stored, processed and transmitted by an application; e.g. if you are not that advanced user then you can use third party hosts to host the web app; they have different containers to store files.
3. Apply controls as per classification.
4. Always use encryption methods to encrypt the data if you need to store them.
5. Always use SSL (secure socket layer) protection which encrypts the response and request so that, even if they can be intercepted but malicious actor cannot read.

2.4 XML EXTERNAL ENTITIES (XEE)

Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks. [1]

2.4.1 DEMONSTRATION

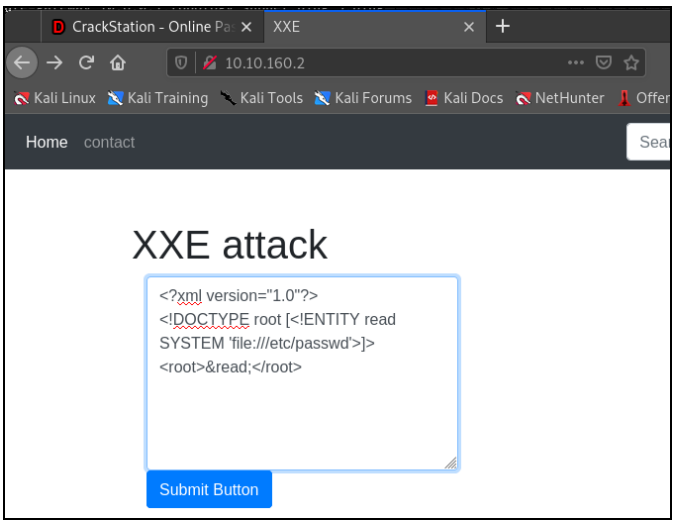

Fig. 8: XML Payload

As shown in fig. 8 by adding small XML code it actually triggers interpreter's XML framework and it processes it; which is absurd behavior.

Fig. 9: XML Payload Output

By processing the XML code snippet we were able to get the contents of '/etc/passwd' file which contains the information about the users of an operating system (in this case the site was hosted on Linux server). Exposing sensitive information of users, their privileges and other data as illustrated in fig. 9 above.

2.4.2 PREVENTION

1. This XXE often occurs when the data formats used are not that known to the developer and because of its complexity using it in unnecessary places may result in this anomaly/absurd behavior.
2. That's why whenever possible, using less complex data formats such as JSON is a much easy and secure option.
3. Also avoid serialization of sensitive data.

2.5 BROKEN ACCESS CONTROL

In many web applications generally have certain 'Access Control' some boundaries between admin, public user and developer. Basically, this vulnerability arises when restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can leverage this vulnerability to access unauthorized functionality or data also can access other user's accounts, change access rights, etc.

2.5.1 DEMONSTRATION

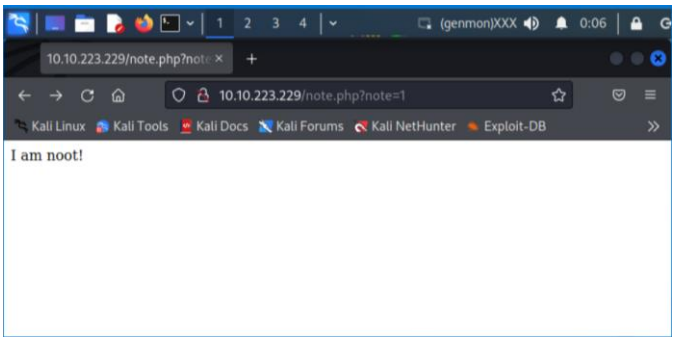

Fig. 10: Normal landing page for authenticated user

Broken access control may seem like a small flaw but actually can be a very dangerous security threat. Let's take a simple example; you are accessing your online net banking portal, after logging in you see the URL it shows something like 'user=2268' at the end of the URL. You thought that let's just change 'user=2268' to 'user=2070' (as shown in fig.11) and after changing the URL you reload the page and you see that now you have logged in as someone else having full control over that user account. You have just found a security vulnerability.

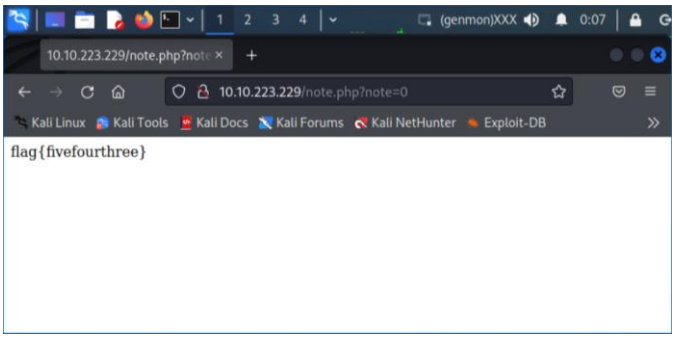

Fig. 11: User accessing unauthorized page

Now, you can see how dangerous it can be if the access control is not there.

2.5.2 PREVENTION

1. Access control is only effective only if enforced in trusted server-side code API, where cannot modify access control. [1]
2. Only exception of public resources deny all requests by default. Like in above example the change in URL should not yield any result rather it should throw an error.

3. There should be consistency in the implementation of access control mechanisms (like only implement once and re-use them throughout the application to ensure consistency).

2.6 SECURITY MISCONFIGURATION

Security misconfiguration is the most commonly seen vulnerability as it can happen very often. This is a result of insecure default configurations. Most common example is IOT devices that come with default credentials such as 'user/password' or 'admin/admin123', etc. It is very important to change these credentials as soon as possible since, IOT devices are everywhere and can give a lot of sensitive information away to any malicious actor who is skillful enough to exploit these vulnerabilities.

2.6.1 DEMONSTRATION

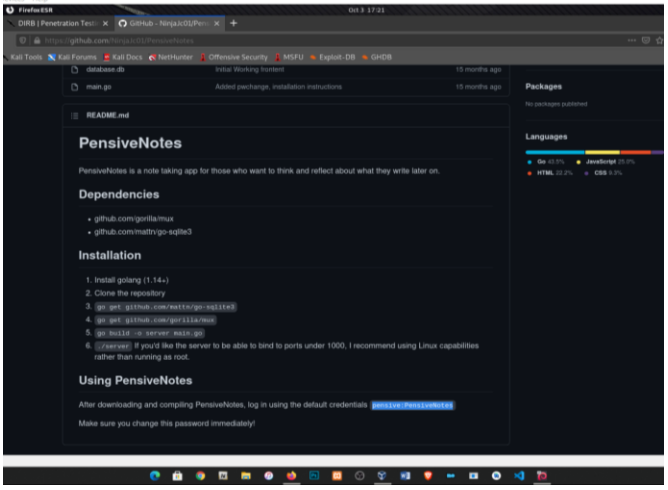

Fig. 12: Default Credentials

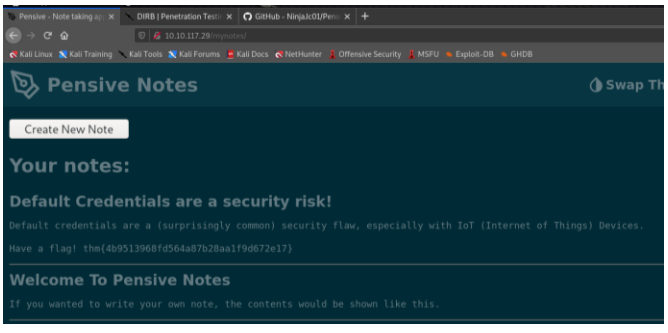

Fig. 13: Gaining access using default credentials

In the demo we performed we were easily able to find default credentials of the web application as shown in fig. 12 the highlighted area highlights the default credentials and we were able to gain access to the web application as illustrated in fig.13, it may look harmless but is actually a very dangerous security risk.

2.6.2 PREVENTION

1. Always check for updates not just on the OS level but also the frameworks/library used, e.g. updating PHP version if you used PHP as back-end language.
2. Always change default credentials as soon as possible and preferably use strong username and 'passphrase' combination.
3. Install only the used features and framework, rather than installing unused features/libraries as it may give you hard time while updating all the components.

2.7 CROSS SITE SCRIPTING (XSS)

XSS attacks occur when an application includes untrusted data without validating, especially when data is given by user using a browser API that can create HTML or JS.

XSS allows attackers to execute scripts on victim's browser which can hijack user sessions, or redirect the user to malicious sites. It may look similar XEE vulnerability, but it differs from it in terms of origin. XSS vulnerability arises from not handling the user-supplied data correctly.

2.7.1 DEMONSTRATION

There are various different forms of XSS named Reflected XSS, Stored XSS, and DOM XSS. In this demo we have shown how stored XSS works, since it is often considered a high or critical risk.

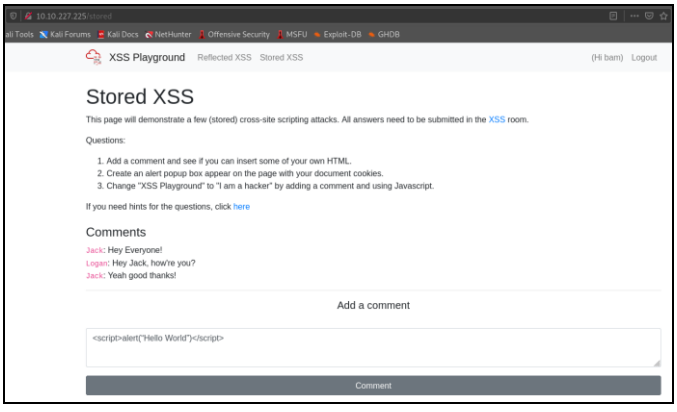

Fig. 14: Stored XSS

Triggering Stored XSS is quite easy, only thing that has to be done is put HTML/JS code in the input field. As we can see in fig. 14, we given a simple Alert tag in JavaScript as input and it processed and given result shown in fig. 15 in this case it was a simple alert tag but attacker will not use such simple tag. Attacker can pose threat to the very balance of application with this vulnerability present in the web application or they may retrieve any trivial information such as OS of the victim's device using 'navigator.userAgent property'

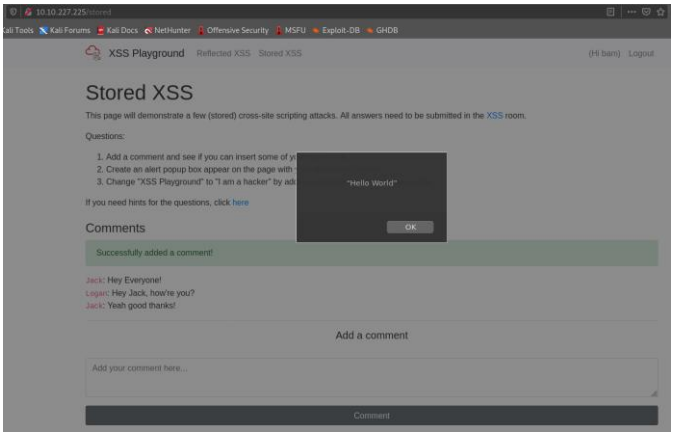

Fig. 15: Output of XSS attack

2.7.2 PREVENTION

In basic terms, avoiding XSS requires separation of untrusted data (user-supplied) data from active browser content. This can achieved by:

1. Escaping untrusted HTTP request data based on context in the HTML output (such as body, attribute, CSS, JS, etc.). It will resolve Reflected and Stored XSS vulnerabilities. [1]
2. Using frameworks that are designed to escape/avoid XSS by default; such as Ruby on Rails, and most common React JS. [1]
3. Even if you do not want to use these frameworks; learning about framework's limitations and data handling will help you to avoid the vulnerabilities, e.g. in the above demo the input should have thrown an error instead of triggering the script.

2.8 INSECURE DESERIALIZATION

Applications and APIs is vulnerable if they de-serialize hostile or tampered objects supplied by an attacker. This can result in RCE (Remote Code Execution) if not, it can result in other attacks including injection attacks, and privilege escalation attacks.

2.8.1 DEMONSTRATION

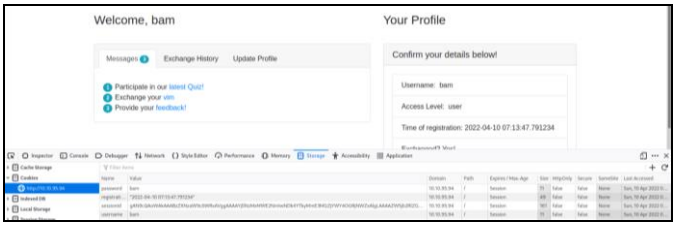

Fig. 16: Cookies

Above figure (fig. 16) shows exactly how attacker can tamper with the sensitive data and overwrite the values to gain admin privileges; just by changing the value (in this case user type). These cookies should have been hidden from any public interface.

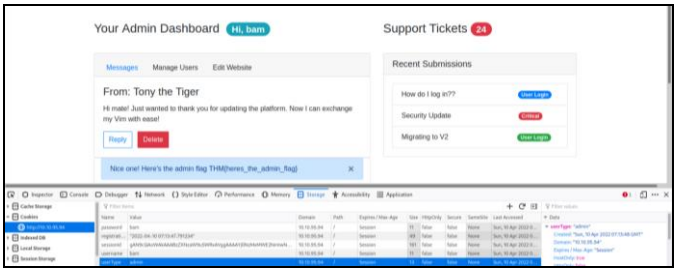

Fig. 17: Gaining Admin Privileges

Anyone can gain access to the admin page which is equivalent to gaining root access of an Operating System and is equally dangerous. But since, first attacker should know all these vulnerabilities and how to leverage this vulnerability and it is quite hard thing to pick that's why probability of this attack occurring is quite less; hence this vulnerability has exploitability rating of 1 out of 3 (which is severe).

2.8.2 PREVENTION

The basic thing to do is not to accept serialized objects from untrusted sources or to use serialization mediums that only permit primitive data types. If this is not possible then consider one or more of the following:

1. Isolating and running code that de-serializes in low privilege environments when possible. In simple terms avoid using high privilege environments to run code whenever possible.
2. Restricting or monitoring the incoming and outgoing network traffic from containers/servers that de-serialize.
3. Implementing integrity checks such as digital signatures on any serialized objects to prevent hostile object creation or data tampering. [1]

2.9 USING COMPONENTS WITH KNOWN VULNERABILITIES

Components used such as libraries, frameworks, and other modules run with same privileges as the application. If even one of the component (not the actual application itself) is vulnerable and is exploited, it may result in data loss or server hijack; which will affect the web application/API. Using such components or not updating them as soon as possible may damage/lower the overall security of the application. Following demonstration will show how attacker may leverage security vulnerability just by searching on the internet. Prime and latest example of this vulnerability is recent 'log4j vulnerability' which was first discovered in 9th December 2021 in a Computer Game named 'Minecraft' 'JAVA edition' to be specific.

2.9.1 DEMONSTRATION

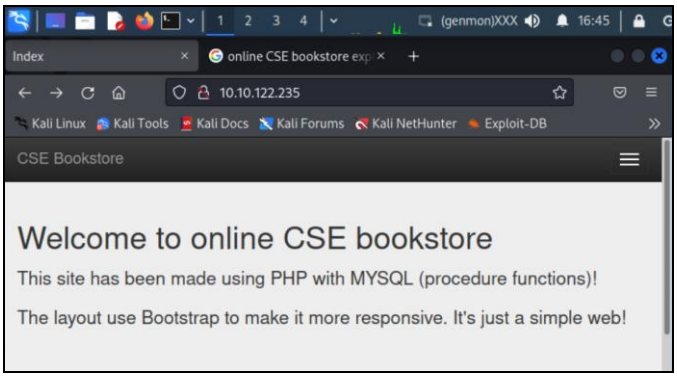

Fig. 18: Vulnerable Application

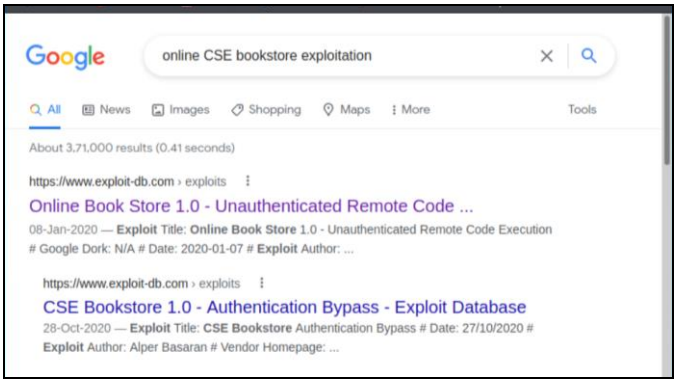

Fig. 19: Searching for known exploit

Fig. 20: Downloading the exploit

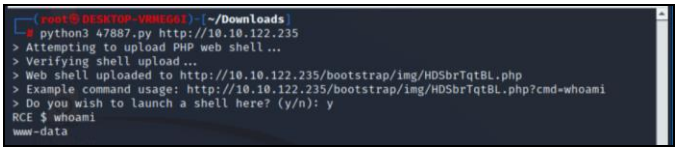

Fig. 21: Execution of exploit script

As the name suggests; this vulnerability is easiest to find but, is massive hit or miss. As illustrated in fig. 19 luckily we found working and unpatched exploit using internet, especially on website named 'exploit-db'. This site consists of all the known vulnerabilities and if they are patched or not. After finding the exploit (fig. 20), it is as simple as running simple python script (fig. 21). It may require some slight modifications to work.

2.9.2 PREVENTION

To prevent this vulnerability there are two important steps to perform; detecting the vulnerable dependencies and patching them or if possible removing them as soon as possible.

1. To detect the vulnerable dependencies of both client-side and server-side, tools like DependencyCheck, retire.js (especially for unused JS dependencies).
2. Remove unused dependencies, unnecessary features, components, files, and documentation.
3. Patch the bugs and vulnerabilities.

2.10 INSUFFICIENT LOGGING AND MONITORING

This vulnerability poses threat in damage-control phase. As the name suggests; insufficient logging may result in more loss of data and capital in general. Logging is a method of storing all the actions performed in periodic and sometimes chronological (timely) manner. It helps in many ways for maintenance and most importantly, gaining back the access from attacker after being attacked. It plays important role in Disaster recovery, because logs are the first things that professionals should be referring to. That's why insufficient logging may cost more than what is already been damaged.

2.10.1 DEMONSTRATON

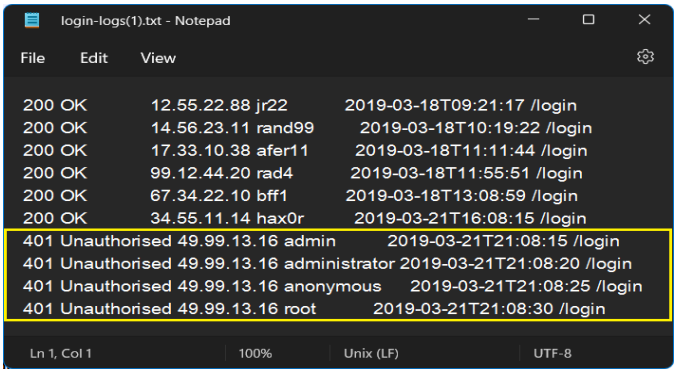

Fig. 22: Example of log

The figure provided above (fig. 22) is self-explanatory that what kind of content a log file may contain. In this case if observed carefully, we can see a series unauthorized access to the secure infrastructure. Well, this is an example of well-

functioning log file. Lack of useful and important logs and insufficient user context may pass as an insufficient logging vulnerability.

2.10.2 PREVENTION

According to the risk of data stored or processed by application following can be done to prevent this vulnerability:

1. Ensuring that all login, access control failures, and server-side input validation failures (e.g. failed authentication attempts) should be logged with sufficient user context to identify any suspicious or malicious activity.
2. Ensuring that logs are generated in a format that can be easily consumed by a centralized log management solutions.

3. CONCLUSION

As mentioned before in the introduction cyber-hygiene is most important thing to follow whenever having online presence and for various developers, taking care of application security should be on top of their priority list. These attacks were the only top 10 most critical vulnerabilities, there are more attacks than these.

On very rare occasions, there is chance of attackers may use combination of more than two or three attacks.

One of the recent example of this incident was data breach at EA (Electronic Arts) game development and publishing company which is infamous for such titles as Battlefield (1, 2, 3, 4, 5 and 2047) and FIFA. In short what happened was that in June of 2021; first, the hackers purchased stolen cookies linked to EA's Slack channel for $10, and logged in as the compromised employee, and message the IT department, telling them that they lost their phone and needed help with multifactor authentication to log into the corporate network. They stole roughly 750 – 800 GB of source code, luckily none of the user data was breached/leaked.

IT professionals and any users of internet should stay vigilant. In this cyberpunk world the data exposed to the outer world through internet may pose danger to anyone's personal presence.

This can be avoided with good internet usage habits/cyber-hygiene.

ACKNOWLEDGEMENT

A big thanks to tryhackme.com for providing us and over 1million (currently in April 2022) learners a platform to learn and understand different aspects of cybersecurity with very interactive manner.

OWASP.org for providing all the theoretical information of the attacks and vulnerabilities.

REFERENCES

[1] OWASP – owasp.org.

[2] Hasan Alsobhi, Reem Alshareef "SQL Injection Countermeasures Methods" IEEE, 2020.

[3] Limei Ma, Yijun Gao, Dongmei Zhao, Chen Zhao, "Research on SQL Injection Attack and Prevention Technology Based on Web", IEEE, 2019.

[4] Arvind Goutam, Vijay Tiwari, "Vulnerability Assessment and Penetration Testing to Enhance the Security of Web Application" IEEE, 2019.

[5] Ajjarapu Kusuma Priyanka, Siddemsetty Sai Smruthi, "Web Application Vulnerabilities: Exploitation and Prevention" IEEE, 2020.

[6] Murat Aydos, Ilker Kara "Detection and Analysis of Attacks against Web Services by the SQL Injection Method" IEEE, 2019.