OWASP TOP 10 VULNERABILITIES – 2017

LAB ENVIRONMENT CHEAT SHEET

A1-Injection Vulnerabilities

PHP Code Injection

Aim: Inject php code via the search bar.

Testing: Set the application insecure

Search a random string like “Biznet”
Send the payload
Set the application secure and go with the same process

Payload: phpinfo();

Solution: Don’t use the eval function unnecessarily.

S. Command Injection

Testing: Set the application insecure
Search the default nslookup query
Send the payload
Set the application secure and go with the same process

Payload: “www.biznetbilisim.com.tr; ls”

Solution: escapeshellcmd() best practise function
principle of least privilege

SQL Injection

Environment Setup:

```
CREATE USER 'newuser'@'localhost' IDENTIFIED BY ‘newpass’;
GRANT ALL PRIVILEGES ON * . * TO 'newuser'@'localhost';
FLUSH PRIVILEGES;
//exit and execute ‘service mysql restart’
//go to http://localhost/sqlInjection/install.php and install the db
```

Aim: Inject SQL statement to bypass the authentication

Test: Set the app insecure
Login as “admin:admin123” after a few pair of wrong credential.
Logout and send the payload
Set the app secure and go with the same process

Payload: “admin:dummypsass' or 1=1##”

Solution: 1-) prepared statements

2-) addslashes() filtering function

Iframe Injection

Aim: Inject a malicious resource into the system with the help of the default <iframe> tag

Testing: Set the application insecure
Go to ruroot.com using ?paramUrl= parameter and show that it’s vulnerable to injection
Set the application secure and try the same process

Solution: Whitelisting

Notes:

Insecure version only works with external resources if they have not implemented the same-origin policy.

A2 – Broken Authentication and Session Management

Captcha Replay Attacks

Environment Setup: apt-get install php-gd && service apache2 restart
Aim: Send the same true captcha value again and again to bypass authentication

Testing: Set the app insecure

Send a true captcha using burp (check the cookie value just in case) and
Show that this authentication can be repeated again and again.
Set the app secure with burp and show that it doesn't accept the same captcha value after the first one.

Solution: Accept only one captcha request with a session

Privilege Escalation

Aim: Use admin functions as a standard user

Testing:
1. Set the app insecure and fire burp

2. a. Login into the app as the standard user (simpleuser:123456).
   b. Refresh the page and when simple welcome page is shown send it to the repeater and name request as "simple login page default"
   c. Logout

3. a. Login into the app as admin (admin:biznet) in another page. Refresh the page and when admin welcome page is shown -> repeater
   b. Refresh the page and when admin welcome page is shown send it to the repeater and name request as "admin login page default"

4. a. Go to http://localhost/privesc/admin/deleteuser/delete.php by clicking the link on the screen and state that this is the page that we want to access with simple user
   b. Copy the request in another repeater page and name it as "NSA TOP SECRET"

5. a. Copy the first request and name it "HERE I COME NSA"

b. POST to /privesc/admin/deleteuser/delete.php and show the response that saying "non admin users are not welcomed here"

6. a. Compare the "NSA TOP SECRET" request and "HERE I COME NSA"
   b. Point the ?u= parameter in the cookie and change it from 20 to 1 in the first request.
   c. Show that we can access to this page by just changing the cookie value.

7. a. Open the "HERE I COME NSA" request
   b. Change the u parameter from 20 to 1 and security_level from 0 to 1 in the cookie
   c. Show that changing the u parameter in the cookie does not work anymore. It requires a successful admin login too.

Considerations:

Standard user has to know the link of the admin functional page

This attack is called vertical privilege escalation

Solution:

Do not rely on the cookie value, it can be modified!

Use both login flag and cookie value for double check

A3 – Sensitive Data Exposure

Nothing practical.

A4 – XML External Entities

Aim: Inject XML input to reach confidential data in the server.

Testing: Go to IP (I prepared another VM for this type of vulnerability.)

Set the app insecure a few times
Catch the request with burp and change the ?xml= parameter with the payload below
Set the app secure and show show that it doesn’t happen anymore.

Payload:

```
%3C%21DOCTYPE%20test%20%5B%3C%21ENTITY%20xxe%20SYSTEM%20%22file%3A%2f
%2f%2fetc%2fpasswd%22%3E%5D%3E%3Ctest%3E%26xxe%3B%3C%2ftest%3E
```

Solution: Disable external entities with libxml_disable_entity_loader(true);

A5 – IDOR

Env setup: Run the db script in the folder and restart the mysql service
Aim: Abuse the user id in the url and get access to another person’s credentials

Testing: Set the app insecure

Login with “idorUser:idorPass”
Click on the link and change the uid with 401, 402 etc.
Set the app secure and start over

Solution: Eliminate the bad code design. Don’t trust the user id only. Use both userid and login credentials for double check.

A6 – Security Misconfiguration

Denial Of Service Attacks

Env setup: chmod 777 tmp/ uploads/ images/

Aim: exhaust the server so that it cannot serve anymore.

Testing: Set the app insecure

Upload a sample photo (Desktop: suitup.png)
Catch the request with burp when sending the new resolution of the picture
Set w/h as 3333x3333 and replay the request a few times.
Show that there are many 43 mb sized copies of the same picture on the remote server (swap there)
Set the app secure and try the same request -> warning message
Reduce the resolution to 999x999 and successfully save the file
Try the same request and show that it doesn't accept with the same filename

Solution: Make the image upload for one session
Check the if the file exists
Limit the file resolution 1000x1000 (or sth you choose)

A7 – Cross Site Scripting

Testing: Set the app insecure
Send the first payload and show that it SEEMS that it has xss protection
Send the second payload and we can bypass the filters
Set the app secure
Send all the payloads and show that now it has a decent protection towards xss

Payloads:

```
<script>alert(1)</script>
<sCript>prompt(1)</sCript>
```

Solution: Blacklisting doesn't work, use more effective filters and make sure you did double check.

A8 – Insecure Serialization

Aim: Execute arbitrary command in the remote server using serialized php data

Testing: Set the app insecure
Catch the request and decode the data value in the cookie
Show the source code and create your own payload using phptester.net
Send the payload in the cookie and exploit the system
Set the app secure and show that the cookie syntax is changed to JSON format
Demonstrate that the vulnerability fixed

Payload:

```
O%3A8%3A%22Example2%22%3A1%3A%7Bs%3A14%3A%22%00Example2%00hook%22%3Bs

%3A19%3A%22system%28%22uname+-a%22%29%3B%22%3B%7D
```

Solution: Use json encoding/decoding instead of deserialization

A9 – Using Components of Known Vulnerabilities

Aim: Use the known struts2 vulnerability (CVE 5638) and implement remote code execution.

Testing: Scan the machine with nmap (nmap -sV 192.168.1.160)

Go to 8080 port of the machine and show that it might have struts vulnerability
Go to Downloads/struts and exploit the machine

Solution: Upgrade the struts framework

A10 – Insufficient Logging and Monitoring

Aim: Trick the server and change the client IP address

Testing: Open the app via entering 127.0.0.1 instead of localhost
Set the app insecure
Fire burp and catch the request when clicking on the title link of the app
Open the log folder and replay the request a few times so that I can show the logs
Open the the source code of the app and show the list
Use the list for fuzzing in burp (set secure first) and demonstrate that our guy is X-Forwarded-For
Add this header with a dummy IP address and replay the request a few times
Show that the log records that IP and we fooled the server.
Now, set the app secure and show that it doesn't log this time and produce alert.

Solution: x-forwarded-for headers can trick the server. For this reason, don't trust it. Use REMOTE_ADDR function for php

BONUS /EXTRAS

OTP Bypass

Aim: Bypass the 2 factor authentication

Testing: Go to otpbypass.php

Set the app insecure
Enter a wrong credential first
Login with "biznetuser:biznetpass"
Enter a wrong captcha and login again
Enter the right captcha and show that we want to achieve this page (login.php)
Enter a wrong credential and try to access the login.php ->fail
Enter the right credential and get access the login.php. ->success
Set the app secure
Enter a wrong credential and try to access to login.pgp -> fail
Enter the right credential and get access to login.php.-> faiil

Solution: Implement the two factor authentication mechanism right.

CSRF

Aim: Trick the user and force him to make malicious activity

Testing: Set the app insecure
Revert the total amount of money on the account
Copy the link and send it to the user
Demonstrate that the wanted activity can be done via this link.
Set the app secure now and try to force the user the same behavior.
Highlight that it doesn't work anymore because we used csrfToken and this token is valid for per request.

Solution: Use csrf token. I implemented it as request based but it can also be used as session based.

SSRF

Aim: Force the server to do malicious activity. Use it as a proxy server to pivot further attacks vectors.

Testing: Set the application insecure first
View the source code and go to the url
Start a web server on mac and send http request to that IP and verify
Set the app secure and verify that it doesn't happen anymore

Solution: Use whitelisting for external resources.

Path Traversal

Aim: Escape from the web folder and display system files

Testing: Set the app insecure

Click on the title
Change the ?page= parameter and go to /etc/passwd
Set the app secure now
Try the same thing -> fail

Solution: Don't let the user go outside the web folder
Set the privileges right for each and every file/directory