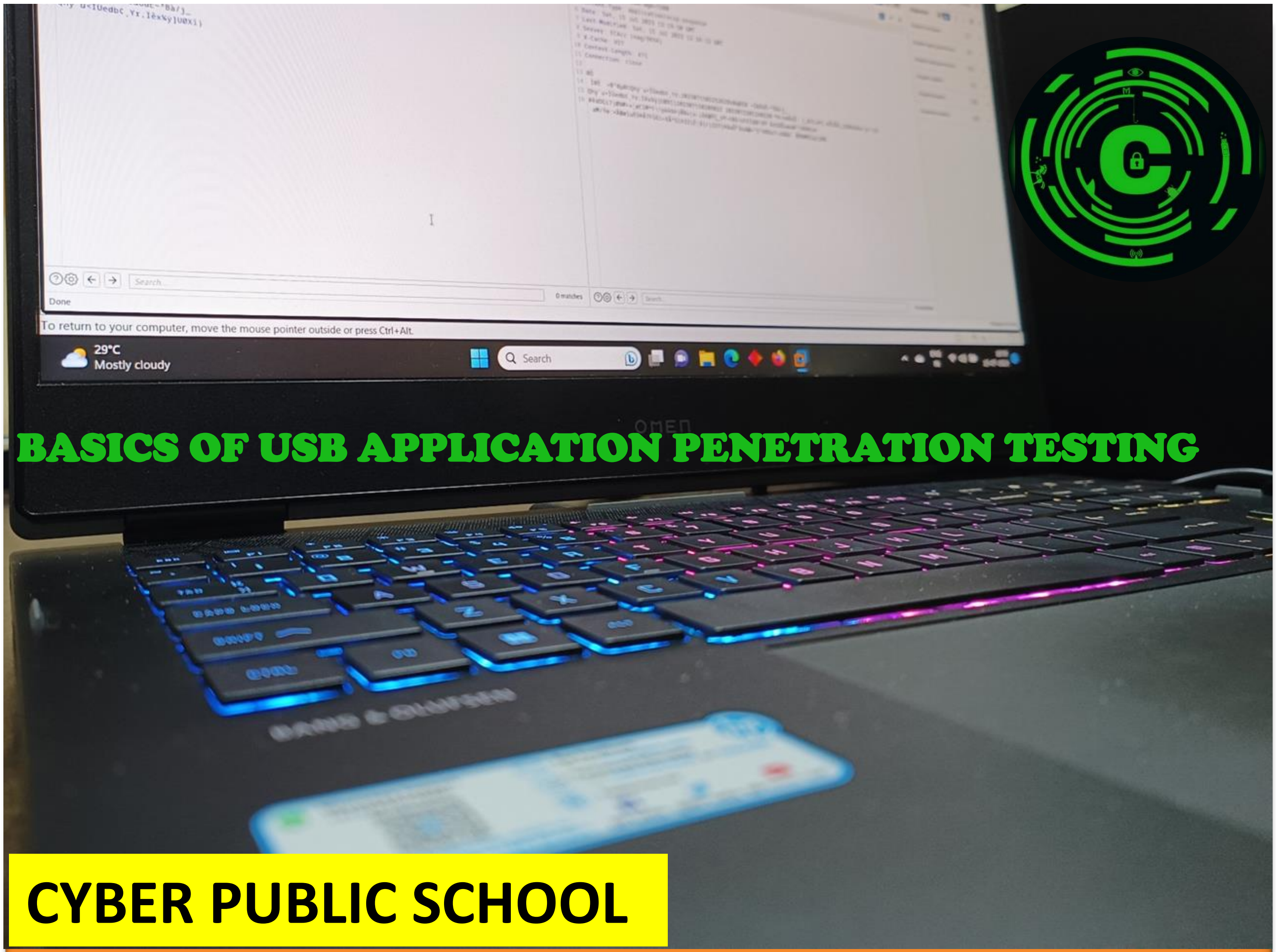

BASICS OF USB APPLICATION PENETRATION TESTING
To return to your computer, move the mouse pointer outside or press Ctrl+Alt.
29°C
Mostly cloudy

CYBER PUBLIC SCHOOL

WEB APPLICATION PENETRATION TESTING METHODOLOGY

https://cyberpublicschool.com/

Table of Contents

Introduction:

The dramatic rise of web applications in the last decade has revolutionized every aspect of our lives. Today, web applications drive many businesses across the globe. It is needless to mention about their presence in social networking, banking, entertainment, insurance, health, automobiles and many other domains. We all enter a lot of sensitive personal information into these applications every day. From the user's perspective, they take it granted that these applications are secure. For instance, when a user has his private conversations with his friends in Facebook, he never wonders if those messages can be seen by a third person. Users take it granted that Facebook maintains them securely. Similarly, from the corporate perspective, web applications are crucial since they drive their core businesses. Any damage to them is going to affect their business directly. In order to win the user's trust and to make sure that business is not hit, it is imperative that these web applications are maintained in a secure manner.

In the late 90's, the attacks on companies were mostly network based. Therefore, a hacker had to gain access first to a system in a network and from there had to grab control of a server within that network, perform his attack and then finally clear the footprints. But, as network security improved and robust control came into picture over the years, the attacks slowly moved towards the web application end. In addition, with the advent of dynamic pages and several other new technologies, the client increasingly started interacting with server. This opened the gates of opportunity for hackers who no longer had to take the trouble of getting access to network, erasing the footprints etc. In web application hacking, with a well-crafted payload, an attacker can download all the details present in a database. The recent hacks on Sony, Adobe, JP Morgan Chase, eBay and several other companies ring alarm bells to prepare for the worst. Hence securing web applications is the need of the hour and *Penetration testing of web applications* is one way of achieving it. In the remaining part of this course, we are going to see what penetration testing is all about, and the methodology that needs to be in place to secure web applications effectively.

Security in SDLC:

One of the main problems with security of web applications today is the lack of a coordinated and well-planned program that lasts throughout the SDLC. Security needs to be incorporated in every phase of SDLC (what is now being called as 'secure SDLC') to make sure that the final product is robust and secure. The section below lists important tasks that need to be done in each phase to ensure security in all phases.

Activities	Core	Security
Requirements	1. Functional requirements 2. Nonfunctional requirements 3. Technical requirements	Security Objectives
Architecture and Design	1. Design guidelines 2. Design Review	Threat Modeling, Architecture risk analysis
Development	1. Unit tests 2. Code Review	Static Analysis
Testing	1. Integrated testing 2. System testing	Security testing
Deployment	1. Deployment review	Penetration testing
Maintenance		Repeat

As seen above, penetration testing is performed once the application is deployed in some environment. Penetration testing focuses on finding out the security loopholes by actually performing the attacks. Testing the application at this stage ensures that any issues that were missed during other phases could be identified once the application is deployed. The sections below discuss penetration testing in detail.

Penetration Testing Methodology: Case Study

Penetration testing is a process that tries to identify the security loopholes present in the application by actually performing the attack. The pen tester attacks the application assuming the role of a hacker, identifies vulnerabilities present in the application, and then reports them to the application owner. There is no definitive procedure for performing penetration testing on web applications due to wide range of technologies that are now being used and also due to the fact that each application is very much different from the other. While penetration testing is definitely one of the primary testing programs to secure web applications, it is necessary but not sufficient.

Consider a fictional bank BestBank who has their services available online through a web application. BestBank now wants to assess the security posture of their web application and outsources this to a security firm to conduct penetration testing. We will now analyze various phases involved in conducting penetration testing on the BestBank application. Penetration testing methodology explained here lists out below phases:

- Information Gathering
- Vulnerability Testing
- Risk Assessment
- Reporting

Information Gathering

Prior to the start of pen test; the tester needs to gather information such as URL, valid credentials, roles, valid test data from the BestBank company. Information gathering phase is concerned about gathering as much information about the application as possible. This includes understanding the server & technology, application entry points, technologies used, understanding application structure and so on. Below are some of the steps that can be followed to gather as much information as possible about the web application:

- <u>Understand application functionality & structure:</u>
  Understanding the functionalities present within the application is crucial to identify the vulnerabilities associated with them. Make a list of crucial functionalities that an attacker might be interested in. For example account balance of a user; transferring funds to third party, updating account details are all highly sensitive. Identifying these is very important. Its only when you identify the functionalities you will be able to figure out applicable threats.

- <u>Map and enumerate all available paths:</u>
  Enumerating the application and its attack surface is an important step before trying out the actual attacks. Try to map out every area within the application that should be investigated.

  Example: Enumerating all the links can be done by using automated tools such as crawlers. For example, tools such as Burp Suite Spider starts with a URL and crawls the entire site. This gives an idea about how big the application is. Also by navigating through various functionalities, identify how sensitive data in the application is segregated and how it is mapped to different user roles.

- <u>Identify application entry & exit points:</u>
  Look for functional entry and exit points of the application where the application takes some sensitive information and then provides access.

  Example: Consider the below GET request that is captured when purchasing an item from an online shopping application.

```
GET /shopping/Product.asp?CustID=100&ITEM=mobile&PRICE=112

Host: example
```

Cookie: SESSIONID=dsfafdjnj324n324234q242qn432j4n23j4nj34jnjn

...

In the above case, note down all the parameters of the request such as CUSTOMERID, ITEM, PRICE and the Cookie. These are considered entry points to the application that can be tampered (explained in later sections).

- <u>Fingerprint Web server:</u>
  Understanding the web server and its underlying version would help us to narrow down to those attacks, which are specific to that version. Fingerprinting the web server can be done in several ways – by checking response headers, through error messages etc., which are going to be covered in detail in the later sections.

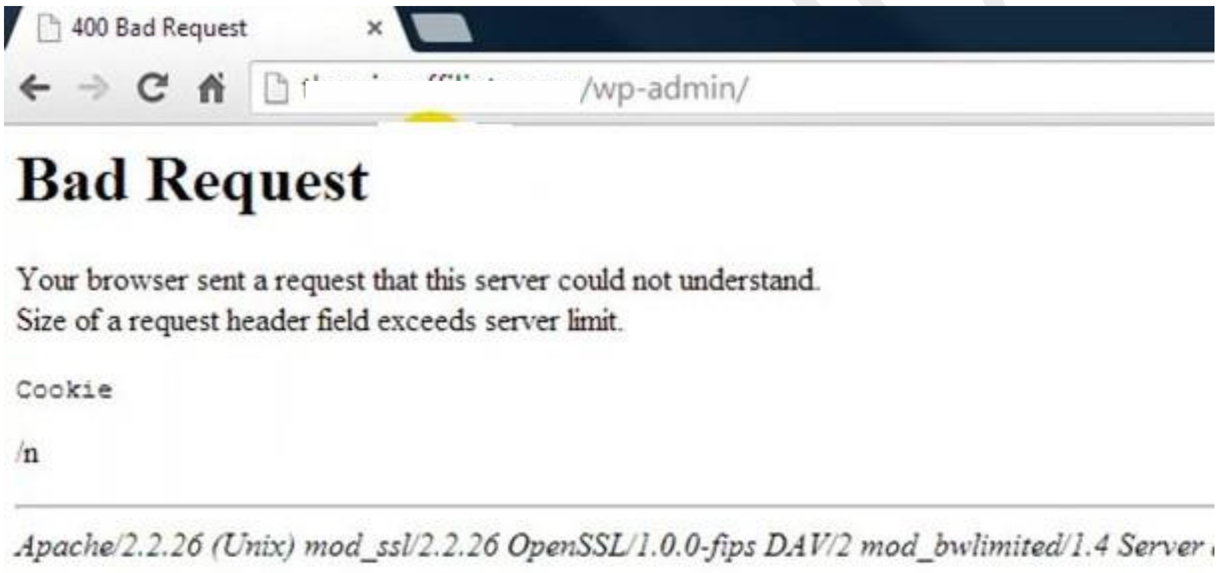
  Example: Consider the below response which is captured using any of the proxy tools. As revealed by the 'Server' header, the application is running Apache 1.3.3 version of the UNIX operating system.

```
HTTP/1.1 200 OK
Date: Mon, 16 Jun 2003 02:53:29 GMT
Server: Apache/1.3.3 (Unix)  (Red Hat/Linux)
Last-Modified: Wed, 07 Oct 1998 11:18:14 GMT
ETag: "1813-49b-361b4df6"
Accept-Ranges: bytes
```

- <u>Analyze error codes:</u>
  Many applications reveal information, which is not intended to be viewed by the user. This information can be valuable in launching further attacks. Analyzing those error messages can be very helpful in certain circumstances.

Example:

Many times, applications respond with a 400 Bad response error message when unexpected request formats are sent. For example consider the below screenshot which shows an application revealing information about its server version, Operating system, OpenSSL version etc. in its 400 response.



Bad Request

Your browser sent a request that this server could not understand.
Size of a request header field exceeds server limit.

```
Cookie
```

/n

Apache/2.2.26 (Unix) mod_ssl/2.2.26 OpenSSL/1.0.0-fips DAV/2 mod_bwlimited/1.4 Server

- <u>Taking advantage of Google hacking techniques:</u>
  Google hacking techniques (taking aid of Google search engine techniques) can be leveraged to look out for those pages that are not commonly accessed by the users.

  Example: For instance, if you want to look for those URL's in a web site (www.bestbank.com) which contains the word 'admin' just type "*inurl:admin site:www.bestbank.com*" in Google to see respective pages.

Once the information-gathering phase is complete, a pen tester needs to have basic knowledge about the functionality of the application, its structure, available paths, underlying web server and the technology, entry and exit points etc. With this information in hand, we can move to next section, which is Vulnerability Testing.

Vulnerability Testing

Vulnerability testing is the phase, which involves analysis of the application for security weaknesses, technical flaws, or vulnerabilities. This would require penetration testing skill set. The security issues identified in this phase will have to be drafted and presented to BestBank along with the remediation solutions. Depending on the application type, there are different areas that need to be tested. Below is a comprehensive list of areas that are to be tested for a given web application.

- Authentication Testing
- Authorization Testing
- Session Management Testing
- Input Validation Testing
- Cryptography
- Configuration and Deployment Management Testing
- Identity Management Testing
- Error Handling
- Business Logic Testing
- Client Side Testing

There is no defined method to test the application for the above issues in any particular order. It usually depends on the tester's approach. While it is beyond the scope of this document to dig deep into each of these areas, we will nevertheless look at some of the most common vulnerabilities that are present in the web applications today. OWASP, an open source project, releases the list of

TOP 10 critical web application security flaws, which helps us, understand the kind of vulnerabilities that are identified in this phase. Below is the list of OWASP top 10 vulnerabilities.

OWASP TOP 10 Vulnerabilities:

- Injection
- Broken Authentication and Session Management
- Cross-Site Scripting (XSS)
- Insecure Direct Object References
- Security Misconfiguration
- Sensitive Data Exposure
- Missing Function Level Access Control
- Cross-Site Request Forgery (CSRF)
- Using Components with Known Vulnerabilities
- Unvalidated Redirects and Forwards

We will go through each of them in detail in the below sections:

Injection Flaws

Injection flaws occur when an application takes in some untrusted data and sends it to the interpreter. Injection flaws are one of the most common and dangerous issues found in many web applications. Depending on the underlying interpreter, there are different injection flaws such as SQL injection, Xpath Injection, LDAP injection, OS command injection and so on. We will examine more about SQL injection from here on, as it is the most common vulnerability associated with web applications.

Example:

1. Consider a BestBank user who tries to view his account balance. When the user enters the following URL:
   http://www.bestbank.com/users/AccountBalance.asp?uid=123

   The corresponding SQL query is executed would be:

   SELECT name, balance FROM userinfo WHERE userid = 123

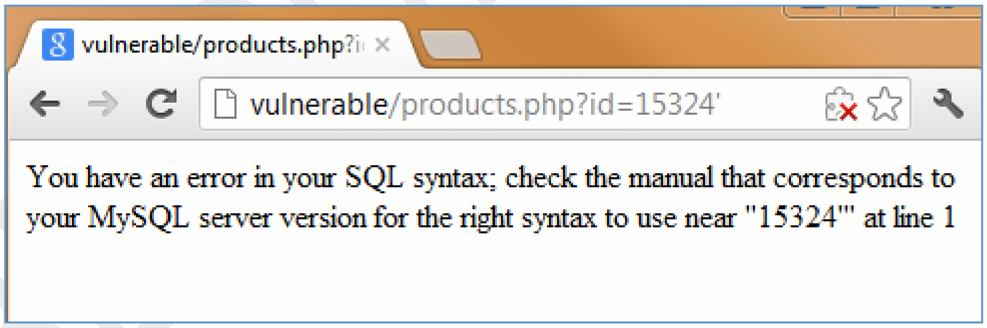
2. Therefore, if an attacker tampers the value the interpreter is going to parse whatever value is supplied by the attacker.

3. Consider the case where an attacker injects something malicious like the below:
   http://www.bestbank.com/users/AccountBalance.asp?uid=123; DROP TABLES--

4. The interpreter would execute the above command and drop the tables because it has no way to know that it is a malicious command. Thus, lack of separation of "data" from the "code" results in injection vulnerabilities.

How to identify?

The basic issue is to check whether all use of interpreters clearly separates untrusted data from the command or query. During pen testing this can be done by sending input to the application and seeing how it responds. Look for the areas where there is a scope to hit the database. For example to identify presence of SQL injection vulnerability, inserting a single quote might result in some error due to syntax issues as shown in the following screenshot:



You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''15324''' at line 1

However, this is not always true because there could be SQL injection even in cases where the application does not respond with an error message and instead shows some custom page. This case is called "Blind SQL Injection" which needs to be identified with true or false conditions.

True Condition: (returns normal page)

http://www.bestbank.com/users/AccountBalance.asp?uid=123 AND 1=1

False Condition: (returns some other page)

http://www.bestbank.com/users/AccountBalance.asp?uid=123 AND 1=2

Depending on the type of exploitation, the SQL injection flaws can also be categorized as Union based and error based.

What is the impact?

Injection vulnerabilities can have serious impact as they allow an attacker to:

- Extract all the details from the database
- Modify/Delete data present in the database
- Complete host take over (depending on the case)

With a simple well-crafted payload, an attacker can extract passwords and other sensitive data present in the database.

Remediation Suggestion:

SQL injection flaws can be prevented by separating untrusted data from commands.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface. Be careful with APIs, such as stored procedures that are parameterized, but can still introduce injection under the hood.
- If a parameterized API is not available, you should carefully escape special characters using the specific escape syntax for that interpreter. OWASP’s ESAPI provides many of these escaping routines.
- Positive or “white list” input validation is also recommended, but is not a complete defense, as many applications require special characters in their input. OWASP’s ESAPI has an extensible library of white list input validation routines.

Broken Authentication and Session Management

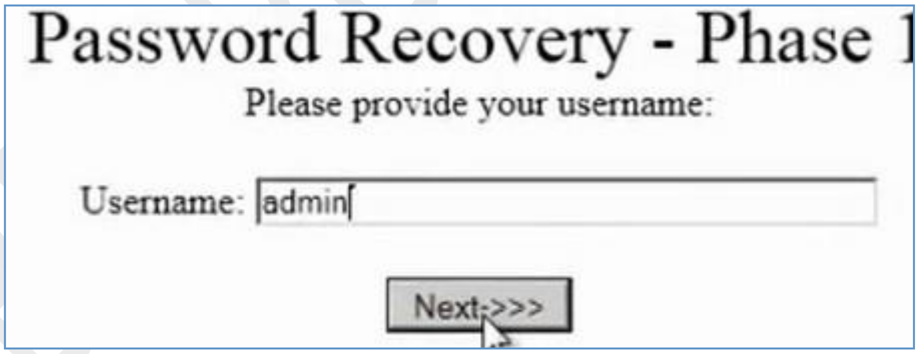
Session management is one of the crucial aspects of web application security. As HTTP is stateless (no inbuilt mechanism to handle sessions), developers have to handle this themselves using mechanisms such as cookies, session IDs etc. Due to bad coding practices, this custom code often has flaws in areas such as logout, password management, session timeout, remember me functionality, secret question during password reset, account update, etc. The flaws may include setting session cookie before authentication, not terminating session after logging out, using predictable tokens and so on.

Example:

As this vulnerability covers several areas, it may not be possible to provide a single example that can cover all the scenarios. Below is a simple example, which explains session management flaw in password reset process.

1. An attacker visits the BestBank application and clicks on the 'Forgot Password' link.

2. Now he enters some other user's ID (say admin) and clicks submit.



Password Recovery - Phase 1

Please provide your username:

Username: admin

3. After this, the attacker just requests some internal page such as viewprofile.jsp and he logs in as admin.

This works because the application wrongly sets the session attribute when forgot password process is initiated. The attacker takes advantage of this and exploits it by requesting it in a sequence. This is known as Session Puzzle attack.

How to identify?

Session management flaws are spread across the application. Below are a few cases that can help you in identifying some of the serious issues:

- Weak account management functions (e.g., account creation, change password, recover password, weak session IDs) which can result in account compromise.
- Session IDs are exposed in the URL
- Session fixation attacks: An application would be vulnerable to session fixation attacks if it uses same session cookies both before and after authentication. This would enable an attacker to fix the session in advance and then hijack it when the user logs in. Hence, check if the application resets or creates the session cookie values after authentication.
- Lack of session timeout
- Session not terminated on server after logout: When user clicks on log out, ideally the session needs to be terminated on the server. Nevertheless, some developers wrongly just delete the cookies on the client side leaving the session on the server still active. This can be identified by logging out of the application and re-sending an earlier captured request using a proxy tool such as Burp tool. If the application still shows a valid response even after logout, it confirms that the session is not terminated.

What is the impact?

Session management flaws can allow an attacker to compromise some or all accounts of the users. Once the account is compromised, an attacker can do anything that a normal user can do. The likelihood for the occurrence of these issues may vary greatly depending on the vulnerability in question. For instance a wrong implementation of forgot password functionality can be exploited by an attacker to totally compromise a victim’s account.

Remediation Suggestion:

Having proper authentication and session management controls is crucial to counter this vulnerability. Following are some of the important points that need to be considered by the developers to address this issue:

1. Setting a session: It is important to assess when the application sets a valid session and how it maintains it. Creating session tokens before authentication, creating session attributes at wrong locations can result in account compromise.

2. Generating session tokens: Since tokens act as a mechanism to identify a user, they should be random and unpredictable. Any guessable pattern within the tokens can be exploited by an attacker.

3. Handling session tokens: Session tokens are not be exposed in URL or displayed in the response. If they are sent in the response body, attacks can steal them by exploiting other attacks such as XSS.

4. Terminating the session: Most of the developers wrongly handle the session termination by simply clearing the cookies on the client side. This still leaves a chance for an attacker to connect to the server if in case he has the tokens. Therefore, it's important to terminate the session on the server properly.

5. Cookie Attributes: Cookie attributes such as 'HTTPOnly' and 'secure', add extra safety to the session tokens. For instance, HTTPOnly attribute ensures that a cookie cannot be accessed by a client side script used with XSS attacks. Similarly, secure attribute ensures that cookie can't be transmitted over unsecure channel.

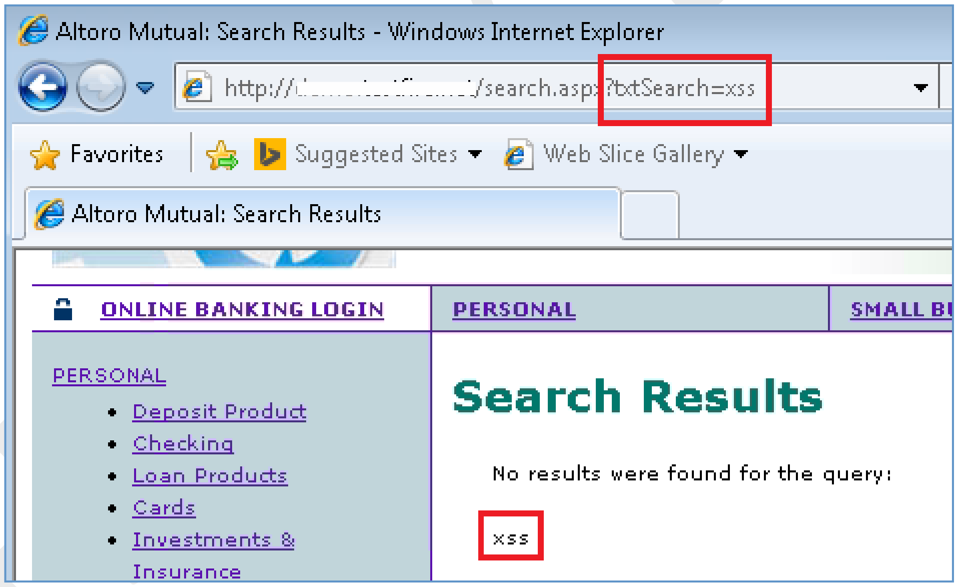
Cross site scripting

Cross-site scripting (XSS) is one of the most frequently encountered issues in web application security testing. XSS flaws occur when the application takes in user input and sends it back to the user without proper validation. In other words, an attacker will be able to execute scripts in the end user's browser by sending them as input to the application. The following example will help you to understand the issue better.

Example:

Consider a search page on the BestBank site that takes in some user input and displays it back to the user.

1. For example enter the text "xss" and observe that the text reflects back in the same page as shown in the below screenshot.

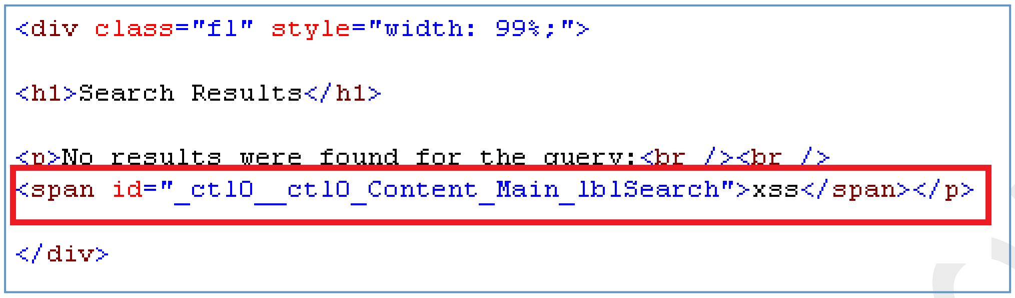


Search Results

No results were found for the query:

xss

2. Now this value sits in the response between some html tags as shown below:



```
<div class="fl" style="width: 99%;">

<h1>Search Results</h1>

<p>No results were found for the query:<br /><br />
<span id="_ctl0__ctl0_Content_Main_lblSearch">xss</span></p>

</div>
```

3. Thus inserting certain scripts will result in their execution as the browser parses the html normally unaware that it is sent by an attacker.

4. An attacker can create a malicious URL such as http://www.bestbank.com/search.asp?txtSearch=<script>window.location ="www.malicious.com"</script> and send it to the victim through mail or IM chat and redirect him to malicious or phishing sites. It executes because as seen in Step 2 the script tags sit in the response and will be parsed by the browser.

How to identify?

Before we talk about identifying XSS issues, it's important to note that there are three varieties of XSS vulnerabilities:

1. *Reflected:* The value inserted is reflected in the same page (Ex: Search page).

2. *Stored:* The value inserted is stored in the database and is reflected later on when the user visits the page. (Ex: comments posted by users in forums).

3. *DOM:* In this case, the value inserted does not hit the server i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser.

During pen testing look for the locations where user supplied values are reflected back in the response. This can be done by capturing the traffic using a proxy tool and observing the response to find the inserted value. The insertion points can be query parameters, POST parameters etc. while the reflection points can be anything in the response (including hidden fields). Once the reflection scenario is identified check if the application validates the input. To verify whether the application validates input, try sending special characters such as <, > etc. and see if the application responds in a different way. If not, based on the response location, frame a payload to test for XSS.

What is the impact?

XSS vulnerabilities exploit the trust a user has in a web site. The victim clicks on the link sent by the attacker because he trusts the domain. By exploiting XSS, an attacker can

- Steal session cookies
- Record key strokes
- Redirect users to malicious sites
- Deface web sites

Remediation Suggestion:

XSS can be remediated by following the below two options:

- Input Validation: Validate the input and allow only those characters that are applicable to the context. While validating do not go with black listing as hackers always find ways to bypass the filters. Instead, the white listing approach, which allows only limited known values, is recommended.

- Output Encoding: Output encoding the data will help browser understand what needs to be treated as html code and what needs to be displayed. Applications need to ensure that all variable output in a page are encoded before being returned to the end user. Encoding variable output substitutes HTML markup with alternate representations called entities. The browser displays the entities but does not run them. For example <script> gets converted to &lt;script&gt;.

Insecure direct object references

Applications often refer to their resources with actual names when generating the web pages. Developers need to include authorization checks in the application before granting access to any of the internal resources. If authorization checks are not implemented, an attacker can just tamper the parameter values to get access to unauthorized web pages that may contain sensitive data.

Example:

Consider a case where a user (say Tom) logs into his BestBank account and clicks on 'View Account Balance' link:

http://www.bestbank.com/ ViewBalance.jsp?user=tom

If the application had not implemented proper authorization checks, Tom can tamper the value of 'user' parameter and see the balance of some other user (say John) by requesting the below URL:

http://www.bestbank.com/ ViewBalance.jsp?user=John

How to identify?

Insecure direct object references can be identified by checking whether the application has proper authorization checks in place. As shown in the above example, try to tamper the values of parameters (GET/POST) and see how the application behaves. Even if the application implements an indirect reference, the mapping to direct reference needs to check whether the user is authorized to access that web page by requesting that web page directly when logged in as an unauthorized user.

What is the impact?

Impact of such vulnerabilities can be moderate to critical. It depends on what kind of data the object reference is pointing to. Exposure of any sensitive personal information leakage through these vulnerabilities can be severe.

Remediation Suggestion:

Insecure direct object references can be prevented by:

- Mapping the values: Instead of directly using the object references, the application can use identifiers (such as numbers from 1 to 6) per user which again map to actual database key on the server. In this case, attackers cannot directly target the unauthorized resources.

- Authorization checks: Ensure that the application has proper access control or authorization checks in place before grating access to the internal resources of the application.

Security Misconfiguration

This is related to misconfiguring settings that may belong to platform, web server, database, application server etc. This also includes missing security patches, use of default accounts and so on.

Example:

For example, consider a case where directory listing is not disabled on the BestBank's server. If an attacker discovers this vulnerability, he can simply list directories to find any file that is present on the BestBank Server. Attacker can then download sensitive documents or code file present on the server and then proceed to find a serious access control flaw in your application. Similarly, an application using Heart bleed vulnerable version of OpenSSL is an example for this kind of vulnerability.

How to identify?

Pen tester needs to analyze the security hardening posture of the application by asking below questions:

- Information Leakage: Check if the application reveals any information about the software used and its versions. For example, a Server response header might leak its version. Similarly, by exploiting SQL injection vulnerability we can know the current version of database in use.
- Use of unnecessary features: Unnecessary features (e.g., ports, services, pages, accounts, privileges) could be enabled. Check for these by running appropriate tools such as nmap etc.

- Default installations: Does the application allow use of default accounts and their passwords? For example if the application uses SAP, check for default usernames and passwords of SAP accounts.

- Information leakage through error messages: If the application is not properly configured, it could reveal information through error messages and stack traces. This can be dangerous as they might help an attacker in exploiting some other vulnerability.

- Security Settings: Check if the security settings in server, development frameworks and libraries not set to secure values. For example, some application servers come with sample pages that have reported vulnerabilities. Hence, check if the settings are updated to remove access to those pages.

The above is not a comprehensive list but a sample one to begin with. Depending on the underlying application and the technology, used pen tester needs to figure out other options, which can be explored to gather any information about misconfigurations.

What is the impact?

Impact can be critical as some of the vulnerabilities can even result in complete system compromise without the knowledge of the user. For example, heart bleed vulnerability can result in an attacker getting sensitive information present on the server's memory.

Remediation Suggestion:

Security hardening process should be continuous to update the systems with latest patches. Also while configuring, make sure that default accounts and passwords are changed. Running code scans and periodic audits can also help in identifying new threats that emerge on daily basis.

Sensitive data exposure

This is one of the most common vulnerabilities found in most of the websites. This vulnerability refers to the scenario where sensitive data is not encrypted and is exposed. Even when the data is encrypted, there could be a chance that the algorithm is broken or key generation and management is not proper.

Example:

Let us assume the BestBank application encrypts credit card numbers in a database using automatic database encryption. However, this means it also decrypts this data automatically when retrieved. If there were an SQL injection flaw in this application, it would allow an attacker to retrieve credit card numbers in clear text. Instead, the BestBank should have encrypted the credit card numbers using a public key, and only allowed back-end applications to decrypt them with the private key.

How to identify?

As a pen tester, it is important to identify which data is sensitive for a given application (Ex: passwords, credit card numbers, and salary details etc.). Next thing is to identify the level of protection that is required for each case. For example, passwords are to be hashed and stored in the database (not encrypted). Hence, after identifying the sensitive data try to ask these questions:

- Is sensitive data transmitted in clear text internally or externally? This would refer to lack of SSL mechanism for the application. In addition, the end-to-end transmission between various servers should also be secured. Of course the pen tester may not have the opportunity to verify all these cases but if there is a chance look for these areas too.

- Is the sensitive data stored on the server in clear text? This would allow database admins and attackers to grab the information. This may not be straightforward to identify unless you exploit some vulnerability such as SQL injection to get details from the database.

- Does the application log sensitive data anywhere?

- If the sensitive data is encrypted, what algorithms are used by the application? Are they secure or broken? Is the key size long enough?

- Does the application not use security directives or headers that address security issues at browser level? For example, lack of proper caching headers in the response results in the pages being cached in the browser. This can be verified by looking at the response header captured by the proxy tool. Any sensitive information present in these pages can later be accessed by an attacker.

What is the impact?

Since the issue is about sensitive data, the impact is always critical. Nevertheless, depending on the application and business type it may vary greatly.

Remediation Suggestion:

The following points can be helpful to remediate certain issues:

- Identify sensitive data & secure it: First step is to identify which areas deal with sensitive data. It could include SSN's, credit card numbers, and passwords and so on. Next thing is to ensure that this data is encrypted during transmission and when at rest.

- Eliminate unnecessary storage: Sensitive data should be used only when it is necessary. Do not store sensitive data unnecessarily. Discard it as soon as possible.

- Use of strong algorithms & keys: Broken algorithms such as MD5 are not supposed to be used with sensitive information. Only strong algorithms are to be used when encrypting values related to sensitive data. The key sizes should also be long enough to ensure that an attacker cannot perform brute force related attacks.

- Password Hashing: Passwords are to be stored in hashed format. This ensures that even if an attacker gets access to their values he cannot reverse them to get the original plain text values. Consider using 'salt' mechanism that adds extra layer of security for the password hashes.

Missing Function Level Access Control

This is a weakness in the application where access control checks are not properly implemented. Many a times, function level protection is managed via configuration, and the system may not be correctly configured. If the developers do not enforce proper code checks, it could result in missing function level access control flaw.

Example:

Let us assume the below page is supposed to be accessed only by a BestBank manager:

http://bestbank.com/app/employeesInfo.aspx

If the BestBank application does not properly implement authorization checks, an attacker simply browses to target URLs can view the data. Not only for unauthenticated users, but this also applies to all the users who are not of manager role. For instance if a normal employee logs in and then accesses this page successfully it a flaw.

How to identify?

The best way to find out if an application is vulnerable to function level access is to first try to login with a privileged account and identify sensitive functions, copy the URLs and then login with a less privileged account and request those URL's. If you can access those pages, it means that the access control checks are not properly implemented.

In short using a proxy, browse your application with a privileged role. Then try to access these pages using a less privileged role. If the server returns valid response, it means that the application doesn't have any access control mechanism.

What is the impact?

This flaw allows attackers to access unauthorized sensitive functionality such as admin functions etc. Impact is high as it allows a normal user to access high privileged functionality.

Remediation Suggestion:

Ensure that the application has authorization checks implemented at all levels before granting access to them. These authorization checks can be implemented as a separate module and invoked every time a request is handled. Please note that the problem is not solved by removing the link from UI. An attacker has several ways to know the internal URL's of privileged accounts.

Cross-Site Request Forgery (CSRF)

CSRF allows an attacker to perform unwanted actions (such as updating or deleting data) on a currently logged in user's account without his knowledge. CSRF takes advantage of the fact that most web apps allow attackers to predict all the details of a particular action. Browsers send cookies automatically for respective domains when a request is passed to them. Attackers take advantage of this and create malicious web pages that generate forged requests that are similar to the actual requests.

Example:

CSRF can be understood easily by considering the below scenario:

Consider a scenario where a BestBank user logs into his bank account to transfer money to his friend account.

1. As soon as the user (Tom) clicks on 'Send Money' button that allows him to send money (say 1000$) to his friend (Mark), the below request would be triggered:

   http://www.bestbank.com/SendMoney.aspx?From=Tom&To=Mark&Amt=1000

2. Now an attacker who has account in the BestBank studies this pattern and now crafts a request that would get triggered if Tom was to send money to attacker:

   http://www.bestbank.com/SendMoney.aspx?From=Tom&To=Attackr&Amt=20000000

3. The attacker now sends this link to Tom and lures him to click on it using social engineering techniques.

4. If Tom clicks on the above URL when he is signed in, the request would be processed because cookies automatically go along with the request. Hence, money would be transferred from Tom's account to attacker's account.

How to identify?

The easiest way to identify if an application is vulnerable to CSRF is to check if requests that deal with modification/deletion of data have unique tokens associated with them. If the forms lack these tokens, an attacker can craft malicious requests. Hence, for functions that involve updating or deleting, capture the traffic using a proxy tool and check if the POST request contains a unique token. In addition, when the tokens are present it is important to verify whether server is validating those tokens each time and if the tokens are random enough. This can be done by tampering the token values using tools such as Burp (Repeater tab). Focus on the links and forms that invoke state-changing functions, since those are the most important CSRF targets.

What is the impact?

Attackers can trick victims into performing any state changing operation the victim is authorized to perform, e.g., updating account details, making purchases and so on. Hence, depending on the underlying case the impact needs to be analyzed.

Remediation Suggestion:

The main reason why CSRF is possible is the fact that attacker is able to construct the request he knows its pattern and the values. However, if there is some unique token whose value is not guessable and which is validated on the server for every request, then that solves the problem. Therefore, the best solution is to include the unique token in a hidden field. This causes the value to be sent in the body of the HTTP request, avoiding its inclusion in the URL, which is more prone to exposure. Having a CAPTCHA also solves the issue but it might not be practical to have CAPTCHA in every page of the application.

Using Components with Known Vulnerabilities

Web applications use different components such as libraries, frameworks, and other software modules. Lot of vulnerabilities are reported on these components on a daily basis and if a vulnerable component is exploited, such an attack could result in serious data loss or even server takeover in some cases. Applications using components with known vulnerabilities may be thus vulnerable in spite of having other defense mechanisms.

Example:

There are several examples that fall under this category. For instance, check if BestBank is vulnerable to Heart bleed attack on OpenSSL cryptography library, a widely used component in the implementation of the Transport Layer Security (TLS) protocol. This flaw allows a remote attacker to expose sensitive data, possibly including user authentication credentials and secret keys, through incorrect memory handling in the TLS heartbeat extension. This can be checked using Heartbleed scripts in nmap tool. Alternately there are several online sites available (such as https://filippo.io/Heartbleed/) which give the results instantly. Similarly, Apache Struts remote code execution, Apache Cookie disclosure vulnerability etc. are other examples.

How to identify?

Although it may sound easy to figure out if you are using some vulnerable component, it is not so in reality. This is because there are no such tools that point out all the vulnerable libraries or components used in a standard and searchable way. This is because not all libraries use a proper versioning system. In addition, there is no central database where we can search for vulnerable versions of all the components (CVE & NVD are now slowly filling up this gap). Apart from this, vulnerabilities are reported everyday on several components. So a pen tester has to keep in touch with these updates and look for them while pen testing the application.

What is the impact?

The impact could vary from low to critical and usually depends on what the vulnerability is about and how it can be exploited.

Remediation Suggestion:

The best solution is to have a process in place to address the issue of updating to newer versions. Many a time's development teams have no idea about using old and obsolete software and its danger effects. Below are other things that need to be considered:

- Identify and monitor components: Identify all components and their versions used by the application. Track and follow the security of these components in public forums and security mailing lists, and keep them up to date.

- Establish security policies & standards: Have security standards for the component use, such as requiring clearances from an authorized team for third party component usage.

Unvalidated Redirects and Forwards

Applications frequently have the necessity to redirect the users to different sites. For this, a target page is specified in a parameter that is generally appended to the URL. The unvalidated redirect issue arises if this parameter is not validated allowing the attackers to define the destination site.

Example:

Consider BestBank link that has a page called 'redirect.jsp' which takes a single parameter named 'url'. As soon as the user clicks on a link the below link would be triggered:

http://www.bestbank.com/redirect.jsp?url=desination.com

If the application does not validate the value coming from the 'url' parameter, an attacker can craft a malicious URL that redirects users to a malicious site that performs phishing and installs malware.

http://www.bestbank.com/redirect.jsp?url=evil.com

How to identify?

Try to identify on what basis the application is redirecting the user to another page. If the destination URL value is sent from the client end, try to tamper the value and see how the application responds.

What is the impact?

Impact is high because successful exploitation of this might result in stealing credentials of a valid user by redirecting him to a phishing site. The user can also be redirected to malicious sites that can install malware.

Remediation Suggestion:

The issue can be remediated using the below solutions:

- Do not redirect the user based on user-supplied value. Make sure to validate the same on the server.

  The destination parameters can be a mapping value, rather than the actual URL or portion of the URL, and that server side code translate this mapping to the target URL.

-

Automated Scanning

Automated scanning refers to the process of using vulnerability scanners to identify security issues within the web applications. Quite often, several security teams use both manual and automated scanning in their security approach.

Automated scanning cannot replace the manual process described above but it can certainly aid penetration tester to some extent. Below are some of the advantages and disadvantages of automated scanning process:

Advantages:

- Tools may not identify all the vulnerabilities but they can help you with the low hanging fruits or the obvious ones. This could in turn help the pen tester to focus on critical ones while the tool looks for easy and straightforward issues.

- Tools run fast. In other words, they cover lot of ground in a fraction of time. While a manual penetration testing might take somewhere between 10-15 days depending on the size of the application, a scan for the same application takes only a few hours to complete. Hence, if you want to test lot of application in a limited period of time, automated scanning can help you.

- They help you maintain advanced, centralized, enterprise wide global security programs in a distributed manner.

Disadvantages:

- A tool cannot replace human mind. A tool will not identify logical flaws present in the application, as it cannot understand the functionality.

- There will be several false positives and hence it is necessary to confirm manually all the vulnerabilities before they are raised.

- You cannot be sure that the tool has scanned all the pages and for all the vulnerabilities.

- Tools may not work with new technologies and may not identify issues such as information leakage etc.

With these points in mind, one needs to have mix of both manual and automated scanning during the security assessments. There are several open source tools as well as commercial tools available in the market as listed below.

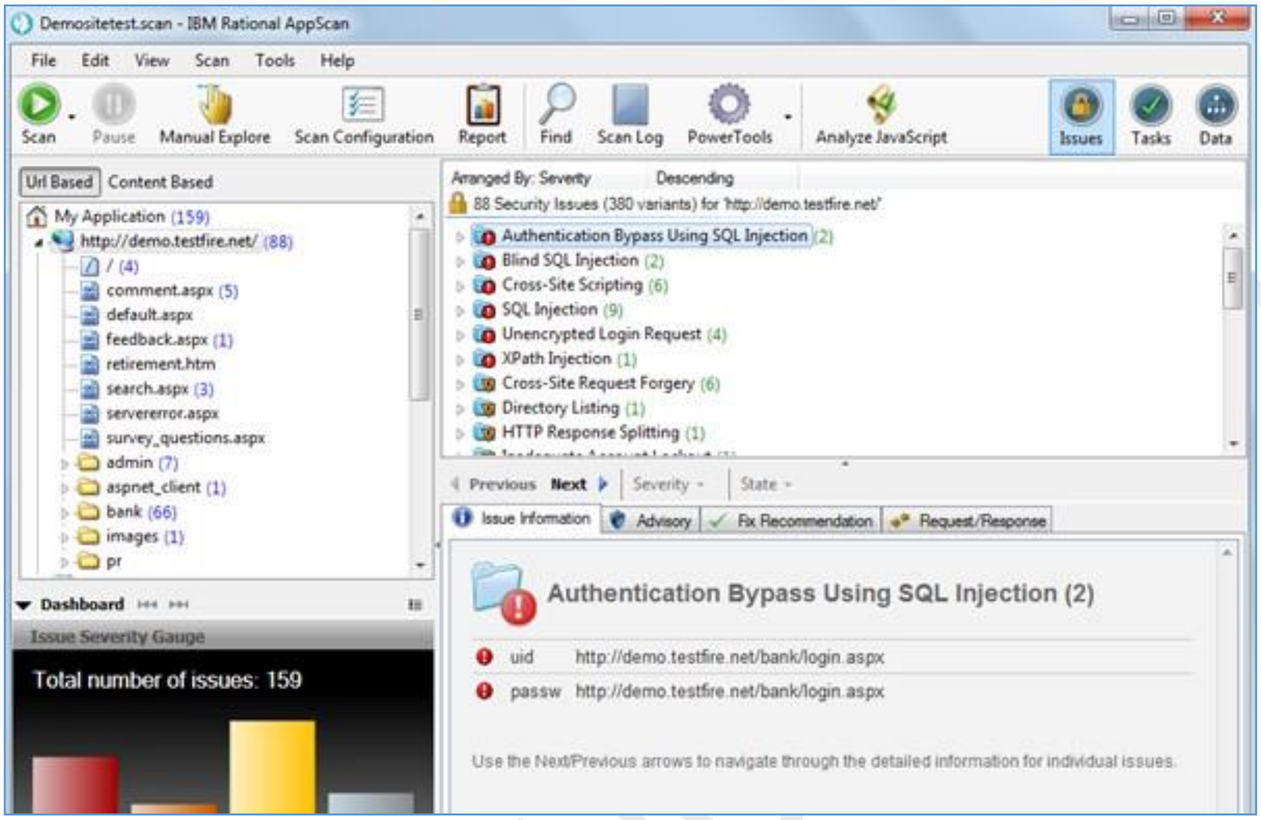
Commercial Tools:

Acunetix, AppScan Standard Edition, Burp Suite Professional, Hailstorm, N-Stalker, Nessus, NetSparker, NTOSpider, ParosPro, Retina Web Security Scanner, WebInspect, Websecurify

Free / Open Source Tools:

Arachni, Grabber, Zed Attack Proxy, Powerfuzzer, W3AF, Wapiti, Watcher, Zero Day Scan.

All of these scanners require you to login to the web application by configuring the tool. The tool then parses through all the pages and constructs a hierarchical structure. After this the tool sends payloads to test for various vulnerabilities. Following is a screenshot of IBM Appscan tool reporting various vulnerabilities:



My Application (159)
http://demo.testfire.net/ (88)
/ (4)
comment.aspx (5)
default.aspx
feedback.aspx (1)
retirement.htm
search.aspx (3)
servererror.aspx
survey_questions.aspx
admin (7)
aspnet_client (1)
bank (66)
images (1)
pr

Total number of issues: 159

Arranged By: Severity Descending
88 Security Issues (380 variants) for 'http://demo.testfire.net/'

Authentication Bypass Using SQL Injection (2)
Blind SQL Injection (2)
Cross-Site Scripting (6)
SQL Injection (9)
Unencrypted Login Request (4)
XPath Injection (1)
Cross-Site Request Forgery (6)
Directory Listing (1)
HTTP Response Splitting (1)

Authentication Bypass Using SQL Injection (2)

uid http://demo.testfire.net/bank/login.aspx
passw http://demo.testfire.net/bank/login.aspx

Use the Next/Previous arrows to navigate through the detailed information for individual issues.

Pen tester has to evaluate these by manually verifying them so as to remove the false positives. Once the false positives are removed, report about remaining findings can be generated using these tools which we will see under reporting section.

Risk Ratings & Assessment

Risk assessment with respect to web application is the process of identifying actual impact caused to the organization in the event of successful exploitation of the underlying vulnerability. Although identifying vulnerabilities is important it is equally important to assess the actual risk involved to the business. For instance the risk involved in having SQL injection vulnerability in the BestBank's login page is more than the risk involved in having an information leakage issue. Hence it is important to have an approach to calculate the severity of these vulnerabilities. Below are the steps involved in this direction:

- Identify vulnerability: The first test is to identify vulnerabilities in the application through security assessments and penetration testing as described in the earlier sections. Risk is considered to be product of likelihood and impact. Hence the next step is to identify both likelihood and impact of the issue.

- Estimate Likelihood: Likelihood factor refers to how likely the vulnerability is to be uncovered and exploited by an attacker. This can be done by identifying threat agents and then asking questions such as below:

  I. How easy is it for the threat agents to discover the issue?
  II. How easy is it for the threat agents to exploit the issue?
  III. What should be the technical level of threat agent to exploit the issue?
  IV. How large is the size of threat agents?

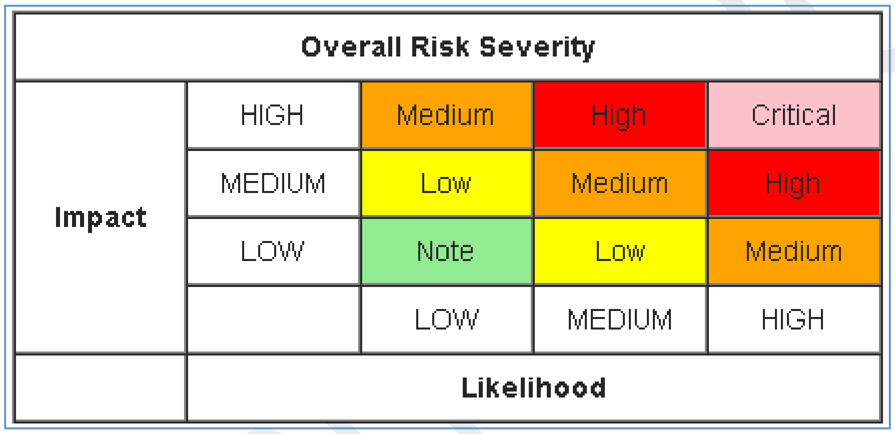
- Estimate impact: Impact is estimated by asking what damage could be caused by exploiting the vulnerability. Here it's important to differentiate between 'technical impact' and 'business impact'. Consider the case of SQL injection in BestBank application. In this scenario, technical impact would be that an attacker can access all details in the database including the SSN's and credit card numbers of 80 million users of BestBank whereas business impact would deal with how much would this actually cost to the company. For example business impact would include cost of issuing new credit cards for all those 80 million users, costs of providing credit monitoring services for the affected users, legal fees in case of litigations and so on. So business impact gives us the actual impact caused to the organization and hence is more important than technical impact. Impact can be estimated by asking questions such as below:

  I. What is the financial damage caused to the organization?
  II. Would this result in reputation damage that in turn can affect the business?

III. Would this result in non compliance to any state laws?

IV. Would this affect CIA (Confidentiality, integrity, authenticity)?

- Determine severity of the risk: In this step the likelihood estimate and the impact estimate are put together to calculate an overall severity for this risk. This can be done by figuring out whether the likelihood is low, medium, or high and then do the same for impact. OWASP proposes a 0-9 scale rating and overall risk rating based on the below table.



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

It may not be possible for an external pen tester to adequately assess the actual business impact caused by successful exploitation of a vulnerability.

Reporting

Penetration testing includes not only technical assessment but also proper and detailed documentation of the identified findings. Remember that the client may not be aware of any of the terminologies such as cross site scripting or cross site request forgery. Hence a detailed report highlighting important issues needs to be presented at the end of the test. A typical report contains the below sections:

Summary:

This gives an overview of the security posture of the application and sums up the overall findings of the assessment and provides a high level view of the issues present in the application. This is intended for people at executive level who may not be very much aware of the technical issues. Hence the summary needs to be in simple words explaining how the issues present in the application can impact it and what needs to be in place to improve the overall security.

Test Details:

This section provides information on the test details as explained below:

- Objective: This section outlines the project objectives and the expected outcome of the assessment.
- Scope: This section outlines the scope of the test.
- Schedule: This section outlines start and end dates of the test.
- Targets: This section lists the number of applications or targeted systems (BestBank's application).
- Limitations: This section outlines every limitation which was faced throughout the assessment. For example, limitations of project-focused tests, limitation in the security testing methods, performance or technical issues that the tester come across during the course of assessment, etc.
- Findings Summary: This section outlines the vulnerabilities that were discovered during testing.
- Remediation Summary: This section outlines the action plan for fixing the vulnerabilities that were discovered during testing.

Vulnerability details:

This section includes detailed technical information about the vulnerabilities identified and the remediation steps needed to resolve them. This section is aimed at a technical level and should include all the necessary information for the development teams to understand the issue and resolve it. Each finding should be clear and concise and give the reader of the report a full understanding of the issue at hand. This section generally includes:

- A technical description of the issue and the affected function or object
- Screenshots taken during the execution of a test case
- The affected item
- Remediation steps