Final Security Requirements Report

Mobile Platform	Hybrid Application
Application domain type	Entertainment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	MySQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Remote Storage (Cloud Database)
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	HTML5 + CSS + JavaScript
Input Forms	Yes
Upload Files	Yes
The system has logs	Yes
The system has regular updates	Yes
The system has third-party	Yes
System Cloud Environments	Hybrid Cloud
Hardware Specification	No
HW Authentication	Basic Authentication (user/pass)
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ;

Confidentiality

Confidentiality Requirement in Security Engineering

Confidentiality is one of the core principles of security engineering. This requirement focuses on ensuring that the viewed, modified or created data is accessible only to the applicable party and is not exposed to an unauthorized user. This guarantees the security and privacy of data.

Typically, confidentiality requirements are most applicable to the protection of financial information, medical records, military intelligence and other sensitive information. The following measures are typically employed to meet confidentiality requirements:

Encryption: Data should be encrypted using strong encryption algorithms to protect it from unauthorized access.

Access control: Access to data should be restricted to only authorized users.

Network security: Networks should be configured securely and monitored for any suspicious activities to protect data from external intrusions.

Application security: Applications should be tested for security weaknesses and patched to prevent any data breaches.

Physical security.

Warning:

If we fail to guarantee confidentiality requirements, then valuable data can be exposed and confidential information can be disclosed. This can cause serious harm to a system in multiple ways, such as:

1. Loss of trust from customers: customers may no longer trust the system because their confidential data was not adequately protected
2. Loss of revenue: customers may stop using the system, resulting in a loss of revenue for the business
3. Legal issues: failure to safeguard confidential data may result in legal action from affected customers, leading to fines and penalties
4. Damage to reputation: breach of trust can lead to negative publicity, which can damage the reputation of the system and the business as a whole.

Integrity

Integrity Requirements in Security Engineering

Integrity requirements are fundamental components of security engineering. Integrity involves establishing and maintaining data accuracy and consistency, preventing improper or unauthorized modifications, and ensuring that only authorized individuals can access or modify data. It is an important tool when it comes to protecting data, networks, and other systems from unauthorized interference or access.

- Ensure data accuracy and consistency
- Prevent improper or unauthorized modifications
- Only allow authorized individuals access or modify data
- Track and verify data modifications

- Use data encryption
- Utilize time stamps and secure log files
- Maintain an audit trail
- Implement secure access controls
- Implement strong authentication and authorization requirements
- Monitor networks and system events to detect abnormalities or potential threats

Warning:

When integrity requirements are not met, it can have disastrous consequences for the system. It can result in data corruption, which can lead to data loss and system crashes. Further, it can impair the fault-tolerance and reliability of the system, making it prone to attack. Additionally, it can lead to unauthorized access of confidential data, resulting in financial and reputational losses.

Availability

Availability Requirement in Security Engineering

Availability requirement in security engineering is the policy that ensures that authorized users are able to access the systems and data they are authorized for in a timely manner. This requirement is designed to guarantee that the organization's network, systems, services and data are available, reliable and secure.

The Availability requirement states that:

1. All authorized users must have access to the system or service they are authorized for, without restriction.
2. Access must be granted reliably and securely in a timely manner.
3. System performance must not be affected while granting access.
4. Information must be secured and confidential in accordance with the organization's security policies.
5. Data and information should be backed up regularly and stored securely.
6. Systems must be monitored and updated as needed to ensure availability and security.
7. Organizations must have plans and procedures in place to address any security threats or disruptions that may occur.

Warning:

If we fail to guarantee availability requirements, the system may go down or become unavailable, resulting in disruption to service. Additionally, user data may be corrupted, leading to data loss, and the system may become vulnerable to malicious attacks. This, in turn, can lead to adverse financial, legal, and reputational consequences.

Authentication

Authentication requirements are the criteria that must be fulfilled before a user can be granted access to a system or application. They are a critical part of security engineering. Authentication requirements typically ensure that the correct identity of a user is established and verified before granting them access to the system.

Authentication requirements are usually broken down into two categories:

Identification verification – this typically involves verifying a user’s identity through methods such as passwords, security questions, biometric data, or something similar.

Authorization verification – this typically involves verifying that a user has the appropriate permissions and rights to access a system or application. Authorization is usually based on pre-determined roles and privileges.

Warning:

If we fail to guarantee authentication requirements, the following may happen to the system:

Unauthorized access to confidential data: Without authentication, any malicious user can access confidential data and misuse it, leading to potential data leaks and financial losses.

System is prone to malicious attacks: Since no authentication is required, the system is vulnerable to malicious attacks like SQL injection and buffer overflows resulting in data and financial losses.

Ineffective application security: Without authentication and authorization, applications can be easily exploited by malicious users, leading to system instability and chances of data loss or corruption.

Increased exposure to data breach attacks: As authentication is not enforced, hackers can easily breach the system and get access to confidential data. This increases the risk of data loss and financial losses.

Authorization

Authorization Requirements in Security Engineering

Authorization requirements in security engineering refer to the processes and procedures by which access to systems, networks and resources is managed. Authorization requirements help to define who has access to what, when, and how.

These requirements specify the roles a user can occupy, the activities for which each user is authorized, and the security levels and controls associated with each user type or role. Authorization requirements should be well defined and enforced consistently across a system or network. It is important to ensure that only authorized personnel have the appropriate access to perform their respective tasks.

The following are the essential components of authorization requirements:

Identification and Authentication: A system must be able to verify the identity of users to ensure that only identified and authenticated individuals are granted access to a system or resource.

Role-based Access Control: Authorization requirements must specify or determine roles such as administrator, user or guest, etc. and set out associated access privileges.

Warning:

If authorization requirements are not guaranteed, the security of the system can be compromised. Access control policies may be weakened, which can allow malicious actors to gain access to restricted resources. This can result in data loss, data leakage, or system disruption. Furthermore, unauthorized users may be able to manipulate or delete important data. It is therefore essential to guarantee authorization requirements to ensure proper security and access control in a system.

Non-repudiation

Non-Repudiation Requirement In Security Engineering

Non-repudiation is an important security requirement in security engineering that is used to provide protection from repudiation attacks. It provides evidence that a user or party cannot deny the authenticity of a digital transaction or communication. Non-repudiation is a form of digital proof that helps to protect against malicious activities and other potential attacks. Non-repudiation can be implemented using digital signatures, cryptographic hashing and timestamps.

Warning:

If we fail to guarantee non-repudiation requirements, the system could be prone to various types of fraud and abuse. For example, malicious actors could falsely claim to have not participated in a transaction, or a sender could repudiate a transaction that was actually made. This could lead to significant financial losses for businesses and individuals and could also have serious legal ramifications.

Accountability

Accountability Requirements in Security Engineering

Security engineering includes a number of principles and practices that are aimed at reducing the risk of security breaches. One of the primary principles is accountability, which reflects the need for individuals and organizations to be held responsible for their actions. This is important for creating a secure environment and holding all parties accountable for their actions.

Accountability in security engineering requires that:

1. All security measures must be regularly monitored and evaluated to ensure that they are effective.
2. Any changes to the security system must be tracked and documented.
3. Any suspicious activity or security breaches must be immediately investigated and reported.
4. All users must be trained in the proper use of the system and security procedures.
5. All access to sensitive information must be logged and tracked.
6. Employees must be held responsible and accountable for their actions and any security violations.
7. All security policies and procedures must be regularly reviewed and updated as needed.

Warning:

Consequences of Failing to Guarantee Accountability Requirements

- Increased risk of fraud, waste, and abuse
- Loss of public confidence in the system
- Ineffective management of resources

- Lack of transparency and oversight
- Lack of visibility into system performance
- Unclear goals and objectives
- Lower efficiency and effectiveness of operations
- Increased costs for maintaining a system without accountability
- Inability to respond quickly to changing demands and situations
- Difficultly verifying data accuracy and integrity
- Potential loss of data or unauthorized access to system resources

Reliability

Reliability Requirements

Reliability requirements for security engineering involve ensuring that systems and services can be trusted to provide the necessary security functions consistently and flawlessly. This includes:

- Robustness: Systems must be designed to withstand high levels of stress and workload.
- Fault tolerance: Systems must be able to recover from, and/or mitigate, the effects of errors or malfunctions.
- Availability: Systems must be available and functioning properly at all times.
- Data integrity: Data must remain in its original form and not be excessively corrupted, altered, or unmasked.
- Auditing: System usage patterns and activities must be monitored, recorded, and audited periodically.
- Recovery: Systems must be designed to include fast, effective recovery protocols in the event of a security breach.

Warning:

Failing to guarantee reliability requirement can cause serious issues to a system, leading to an unreliable system.

- Unexpected malfunctioning of the system
- System outages
- Loss of data
- Poor performance of the system
- Loss of customer trust
- Damage to reputation of the business
- Financial losses

Data Freshness

Data Freshness Requirement in Security Engineering

Data freshness is essential in security engineering, as it ensures data remains valid and up-to-date. The need for data freshness goes beyond simple data integrity. By regularly refreshing data, security engineers are better able to detect changes in patterns of behavior and identify potential areas of vulnerability.

Data freshness requirements may vary depending on the security profile for a given system. Commonly, security engineers will expect data to be refreshed regularly, or in line with an established schedule. This could be daily, weekly, monthly or other user-defined interval.

In addition to a data freshness requirement, security engineers should also consider the following:

- Ensure data backups are up-to-date
- Automate any steps in the data refresh process
- Monitor for any data inconsistencies or conflicts
- Update any data protection policies and associated technology
- Perform usability testing on

Warning:

If we fail to guarantee data freshness requirements, it can have serious implications for the system. For example, it can lead to inaccurate or outdated information being stored and presented, or computational errors and incorrect analysis based on stale data. Additionally, it may introduce complications in terms of auditing and compliance requirements, with the system being unable to accurately reflect and report up-to-date information. The inability to ensure data freshness can also lead to decreased usability, with users not trusting or relying on the information the system is able to provide.

Confinement

Confinement Requirement in Security Engineering

The Confinement Requirement is a type of security engineering that ensures only authorized users have access to sensitive data. It is based on the principle of least privilege and state separation, which suggests that if a user has more privileges than they need, they may be able to access more information than they should. Confinement Requirements limit access to data, programs, and services to those users who are authorized to use them and segregate user accounts, reducing the opportunity for malicious activity. In addition, they also limit the ability of potential attackers to gain access to the system and its data.

Warning:

If We Fail to Guarantee Confinement Requirements

If we fail to guarantee confinement requirements, then the security of the system will be compromised. Without proper confinement, malicious actors can exploit existing vulnerabilities to gain unauthorized access to sensitive information or take control of the system. This can allow them to damage or steal data, disrupt operations, and even launch attacks against other systems. As a result, the system will be exposed to various security risks, including data breaches, malicious software, and denial of service attacks. Additionally, system stability and reliability will be compromised since users may be unable to use the system due to malicious activity.

Interoperability

Interoperability Requirements in Security Engineering

Interoperability requirements in security engineering are a set of policies and technical standards intended to ensure secure communication between different systems. They help ensure that different components of a system, such as software, hardware, and networking, are able to communicate and share information without compromising the security of the system.

The most common interoperability requirements in security engineering include: - Establishing a secure communication protocol between different components of the system - Establishing authentication and authorization controls to ensure only authorized users can access the system - Implementing data encryption and decryption protocols to protect the confidentiality of data in the system - Establishing secure file transfer protocols to provide secure data transmission - Implementing a secure log management system to keep track of system activities and detect suspicious activities

Warning:

If we failed to guarantee an interoperability requirement, there would be a variety of consequences to the system.

- The system could become unreliable, as components may not be able to communicate with each other effectively.
- Data integrity could be compromised, with different parts of the system not exchanging data correctly.
- System performance could suffer significantly, as communications between components could become sluggish or unresponsive.
- Security could be an issue, as the system may not be able to properly control access to data.
- Issues with scalability could arise, as the system may not have the capability to grow and adapt as needed.

In short, a system that does not meet the necessary interoperability requirements could suffer from a variety of issues that could severely hinder its ability to function properly.

Data Origin Authentication

Data Origin Authentication

Data origin authentication is an important security measure which focuses on ensuring that the data received is indeed from its purported source. The authentication process verifies the originator of the data, providing assurance that no data has been modified by third parties during transmission. Data origin authentication is most commonly used in network communications, but can also be applied to other systems to protect the integrity of data, including digital systems like electronic records.

Data origin authentication generally employs digital signatures, cryptographic hashes, or message authentication codes. Digital signatures help to verify the identity of the entity sending the communication, while cryptographic hashes provide integrity by verifying that the data being sent is the same as the data being received. Message authentication codes provide additional security by ensuring that the data has not been modified while in transit.

Data origin authentication adds an extra layer of security to data transmission, reducing the possibility that transmitted data has been tampered with or altered. This type of authentication is essential for secured networks and provides

Warning:

Consequences of Not Guaranteeing Data Origin Authentication Requirements

Without guaranteeing data origin authentication, the trustworthiness of data and applications on the system is put at risk. Here are the potential impacts that can affect the system:

Data integrity risks: If the data origin is not verified, it can be modified or manipulated before it reaches its destination. This can lead to security breaches, system downtime, and significant financial losses.

Increased chances of malicious attacks: Without validating the source of data, malicious actors can insert malicious content into system, which can lead to viruses, malware, and other malicious activities.

Loss of customer trust: When origin authentication is not ensured, customers and users become less trusting of the system and its ability to protect their data. This can lead to a decrease in customer loyalty and can ultimately lead to a decrease in revenue and profits.

Final Security Good Practices

Mobile Platform	Hybrid Application
Application domain type	Entertainment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	MySQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Remote Storage (Cloud Database)
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	HTML5 + CSS + JavaScript
Input Forms	Yes
Upload Files	Yes
The system has logs	Yes
The system has regular updates	Yes
The system has third-party	Yes
System Cloud Environments	Hybrid Cloud
Hardware Specification	No
HW Authentication	Basic Authentication (user/pass)
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ;

Security Best Practices Guidelines for Authentication

Authentication Best Practices

Security best practices for authentication should be followed to maintain the safety and integrity of user accounts and data. Below are some recommended steps for implementing secure authentication:

1. Require Multi-Factor Authentication

Where possible, implement multi-factor authentication (MFA) requiring users to provide two or more independent credentials. This ensures that even if one credential is compromised, the user account is still protected.

2. Use Secure Protocols

Wherever possible, use secure authentication protocols such as OAuth, OpenID Connect, and SAML. These protocols provide more secure ways of authenticating users compared to basic username and password combinations.

3. Strong Password Requirements

Enforce strong password requirements on user accounts including minimum length, complexity, and expiration.

4. Password Storage

Store passwords using secure hashing algorithms such as bcrypt. Do not store passwords in plaintext.

5. Monitor Failed Login Attempts

Monitor failed login attempts and lock out accounts after a certain number of failed attempts.

6. Utilize Two-Way Authentication

Where appropriate, use two-way authentication such as SMS or email-based codes or tokens. This ensures the user’s authenticity and provides an extra layer of security.

7. Utilize Token or Session Authentication

To avoid the need for reauthenticating users on every request, support token or session authentication. This allows the user to be authenticated once and all subsequent requests are authenticated using the token or session.

8. Limit Access

Implement access control measures ensuring that only authenticated users have access to sensitive data and systems.

9. Logging and Auditing

Enable logging and auditing to track authentication events and potential breaches.

10. Stay Up-To-Date

Constantly monitor the security of authentication protocols and algorithms and upgrade them as soon as newer, more secure versions become available.

By following these best practices, you can ensure that your authentication processes are secure and protect your data from malicious actors.

Security Best Practices Guidelines for Multifactor Authentication

Multifactor Authentication Best Practices

1. Use strong passwords with a combination of uppercase, lowercase, numeric, and special characters
2. Use different passwords for different accounts
3. Rotate passwords regularly
4. Avoid reusing passwords or writing down passwords
5. Place strong limits on failed login attempts
6. Enable two-factor authentication (2FA)
7. Implement identity verification processes
8. Use multi-factor authentication (MFA) whenever possible
9. Have policies in place to prohibit sharing passwords
10. Leverage other authentication methods such as biometrics (fingerprint, facial recognition)

Security Best Practices Guidelines for Authorization

Security Best Practice Guidelines for Authorization

Introduction

Authorization is a security principle associated with granting access to resources controlled by an organization or system. Authorization policies and procedures are essential for mitigating the risks associated with granting privileged users or applications access to sensitive systems and data.

This document outlines the security best practices that should be used to ensure proper authorization and secure access to organizational data and systems.

Best Practices

Single Sign-On (SSO): Establish a single sign-on solution to reduce the number of passwords required for access to multiple systems and applications.

Establish Access Grants: Develop authorization policies that clearly define who has the authority to access specific systems and data.

Role-Based Access Control (RBAC): Establish access control based on user roles and responsibilities.

User Access Logs: Keep detailed records of user access attempts to monitor for unauthorized authentication attempts.

User Authentication & Authorization: Authenticate and authorize user access to applications and systems on an individual basis.

Implement Least Privilege Principles: Restrict user access to only the privileges they need to perform their job function.

Separation of Duties & Privileges: Implement procedures to ensure that no single individual has complete control of system data or functions.

Encrypted Connections: Implement secure communications methods such as SSL when transmitting sensitive data.

Strong Password Guidelines: Ensure that strong passwords are used to protect access to systems and data.

Audit Access Logs: Conduct regular audits of user access logs to detect any unauthorized activity.

Conclusion

By following the security best practices outlined above, organizations can ensure that proper authorization procedures are in place to protect their data and systems from unauthorized access and data breaches.

Security Best Practices Guidelines for XSS

Security Best Practices for XSS

Enforce Input Validation – All input data received from users must be validated BEFORE processing and stored. No unvalidated user-provided data should ever be trusted.

Sanitize Input Data – Sanitize all input data by removing special characters and HTML tags that can be used to launch XSS attacks.

Escape Output Data – Make sure all output data is properly escaped. HTML entities should be used to escape data displayed on web pages.

Strict Content Security Policies – Implement a strict Content Security Policy (CSP) to ensure browsers only execute scripts and stylesheets that are explicitly allowed.

No Mixed Content – Avoid using both `http` and `https` resources in the same web page to prevent man-in-the-middle attacks.

Limit User Access – Provide users with only the necessary permissions to access the parts of your application that they need.

Regularly Monitor Log Files – Monitor log files for suspicious activity, such as suspicious and unexpected file uploads and downloads.

Regularly Perform Audits – Regularly check the code for any vulnerabilities that could result from XSS emails.

Disable MIME Type Sniffing – Make sure to disable MIME type sniffing in your web application to prevent attackers from uploading malicious files with unexpected MIME types.

Security Best Practices Guidelines for CSRF

CSRF Prevention Best Practices

Cross-site request forgery (CSRF) is a type of attack that allows an attacker to force an authorized user to initiate an action on a web application without their consent. These attacks are sneaky because they are hard to detect until it is too late. To prevent these attacks, it is important to implement proper CSRF protection best practices.

Generate Unique and Unpredictable Tokens

- Generate unique CSRF tokens when the user is authenticated. These tokens should be unpredictable generated and associated with the user session.

Validate Tokens on All Requests

- Make sure to check if the CSRF token is present in requests and validate it against the token stored in the session.

Use Same-Domain Cookies

- Make sure to keep the cookie domain and the website domain the same so that the cookies are not accessible cross-domain.

Use POST Requests Instead of GET Requests

- GET requests can be easily manipulated by attackers, so use POST requests instead. This way even if the request is compromised the data is not sent to the client.

Whitelist HTTP Referers

- Your website should only accept requests from whitelisted referrers. This way malicious requests can be easily identified.

Log Suspicious Activity

- Keep track of all suspicious requests sent to your website and log them. This can help you identify and investigate possible attacks.

Validate User Inputs

- Make sure to validate user inputs against known and acceptable values. This helps in filtering out malicious requests.

Implement Open Redirect Protection

- Open redirects can be used in CSRF attacks, so it is important to implement open redirect protection to avoid such attacks.

Following these CSRF prevention best practices can help keep your website and users safe from malicious attackers.

Security Best Practices Guidelines for Cryptographic Storage

Cryptographic Storage Security Best Practices

It is important to secure cryptographic storage in order to protect sensitive data from unauthorized access. The following best practices should be followed when protecting sensitive cryptographic storage:

1. Implement Access Control

Implement access control measures to limit access to cryptographic storage to only those who have legitimate need-to-know. Assign individual user rights to prevent unauthorized users from gaining access.

2. Enable Data Encryption

Enable data encryption to ensure that sensitive data is protected when stored or transmitted. Key-based encryption should be used to ensure the data is appropriately secured.

3. Implement File Protection Mechanisms

Implement file protection mechanisms to further ensure that sensitive data stored in the cryptographic storage system is secure. Utilize strict rules to protect sensitive data by setting different permissions for files so that unauthorized users cannot access the data.

4. Utilize a Firewall

A firewall should be implemented to protect the cryptographic storage system from malicious outsiders who are attempting to gain unauthorized access. Firewalls can also be used to help detect suspicious activity.

5. Monitor Network Traffic

Monitor the network traffic on the cryptographic storage system to identify any potential malicious activity. Monitor any changes to data on the system to detect any unauthorized access.

6. Backup and Archive Data

Regularly backup and archive the sensitive data stored in the cryptographic storage system to another secure location in order to protect against accidental or malicious data loss or modification.

7. Use Best Practices and Policies

Develop and implement best practices and policies such as strong passwords, multi-factor authentication, and physical security measures to ensure that the cryptographic storage system is protected and secure.

With these best practices in mind, organizations can protect their cryptographic storage systems from unauthorized access and data loss.

Security Best Practices Guidelines for Database Security

Database Security Best Practices

Introduction

Database security is essential for protecting sensitive data. The following best practices provide guidance into the implementation of secure database systems.

General Best Practices

1. Establish a secure environment by enforcing granular access control and limiting access to authorized users.
2. Grant only the minimum privilege levels required to perform specific tasks.
3. Implement strong password policies and regularly monitor user credentials.
4. Develop and implement processes to detect, respond, and report suspicious activity.
5. Analyze system logs and audit trails in order to detect and react appropriately to suspicious activity.
6. Encrypt sensitive data at all times, both at rest and in transit.
7. Protect the integrity of the data by implementing an access control system and regularly monitoring the actions of privileged users.
8. Implement data integrity controls to verify the accuracy and authenticity of data.

Physical Security

1. Secure the location and physical environment in which the database is stored.
2. Implement access control systems, such as biometrics.
3. Establish an incident response plan in the event of a breach.
4. Ensure that the hardware and data lines are securely and properly stored.

Network Security

1. Implement network firewalls in order to protect the network from malicious traffic.
2. Implement intrusion detection systems to monitor traffic and detect and respond to malicious activity.
3. Create and enforce secure configurations on the network in order to prevent unauthorized access and protect sensitive data.

Application Security

1. Develop and implement secure coding standards to ensure applications are built with security in mind.
2. Regularly audit applications and review code to ensure they are secure and up to date with the latest security patches.
3. Monitor application performance in order to detect and respond to suspicious activity.
4. Ensure all vulnerabilities are addressed in a timely manner.

Monitoring

1. Regularly monitor data access and operations in order to detect and react to suspicious activity.
2. Utilize automated tools for ongoing security assessments.
3. Establish alert mechanisms for potential threats or exploits.
4. Analyze system logs and audit trails for any suspicious activity.

Conclusion

In order to protect the security of databases, it is

Security Best Practices Guidelines for Denial of Service

Guidelines for Denial of Service Security Best Practices

Introduction

A Denial of Service (DoS) attack is an attack designed to make a computer or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. DoS attacks can be conducted in various ways, including flooding the network with traffic, crashing web servers, and hacking the application in order to gain control of the target device. By following these guidelines, organizations can better protect their systems and networks from DoS attacks.

Protocols

1. Install and configure firewalls to protect your perimeter.
2. Enable Network Intrusion Detection Systems to monitor your network activity and identify malicious activity.
3. Keep your operating systems and applications up to date with the latest security patches.
4. Monitor unusual network traffic and behavior.
5. Disable unused services and protocols such as Telnet, FTP, RSH, etc.

Network

1. Change default passwords for any device or application and use strong passwords.
2. Create separate networks for sensitive or mission-critical systems and segregate or isolate these from other networks.
3. Prepare for the worst, if you think you are a likely target for DoS attacks, then keep a “war chest” of resources ready in case one is needed.
4. Implement rate-limiting or resource quota controls on all services.

Communications

1. Encrypt communications, both internally and externally.
2. Use stronger authentication protocols such as TLS, IPSec, SSH, etc.
3. Implement traffic shaping to limit the amount of data per unit time that can enter or leave the network.

Resources

1. Maintain a backup of your data, so that in case of a successful DoS attack, you can restore your data.
2. Train your staff to be aware and knowledgeable of DoS attacks and the best practices that are in place to defend against them.

Hopefully, these security best practices will help organizations and individuals protect their systems against DoS attacks. However, as new techniques and technologies emerge, organizations should continue to monitor and update their security practices.

Security Best Practices Guidelines for File Upload

File Upload Security Best Practices

Creating a secure file upload system is important for preventing data breaches and protecting the privacy of your users. Here are some best practices to follow when setting up a file upload system.

Authentication

- Ensure that user accounts and data are securely authenticated with a strong password policy.
- Use secure methods such as two-factor authentication, biometrics, and other security controls.

Authorization

- Configure access control using the principle of least privilege.
- Log and monitor user activity levels and set up alerts for any suspicious activity.

File Size Limits

- Establish a file size limit that puts a reasonable cap on the amount of data that can be uploaded.
- Monitor file storage periodically to ensure users are adhering to the file size limit.

File Type Restrictions

- Restrict what file types can be uploaded, such as scripts, applications, and executables, which can be used for malicious purposes.
- When possible, set up a whitelist that only allows approved file types to be uploaded.

Validation

- Validate uploaded files for malicious content, such as viruses and malware.
- Validate uploaded files for expected content, such as file type, size, and other attributes.

Encryption

- Encrypt uploaded files at rest to prevent unauthorized access.
- Consider encrypting files in transit to prevent eavesdropping.

Logging & Monitoring

- Keep logs of all uploaded files and user activity.
- Monitor logs regularly to look for any suspicious activity or unusual activity.

Backups

- Implement a comprehensive backup strategy to ensure data can be restored quickly in the event of an emergency or data loss.

Update Security Measures

- Establish a process for regularly reviewing and updating security measures to ensure user data remains secure.
- Keep up to date with security best practices and consider engaging a third-party security expert periodically to audit the system.

Security Best Practices Guidelines for HTML5 Security

HTML5 Security Best Practices Guidelines

Minimizing Cross-Site Scripting Vulnerabilities

- Use HTML Escaping when rendering user-provided data in HTML by default
- When creating dynamic web pages, use Content Security Policy to restrict potentially dangerous operations like inline script and styles

Preventing Cross-Origin Resource Sharing (CORS) Attacks

- Limit who can access your resources with a whitelist policy. This should include all hosts who are legitimate users of your resource.
- Validate requests according to your CORS policy. This includes checking the origin and methods of each incoming request.

Securing Session Storage

-Use server-side sessions where possible to store user data and use HTTPOnly cookies for authentication tokens and session identifiers. - Never store sensitive user data in cookies.

Keeping Your HTML5 Applications Secure

- Validate user input on both the client and server-side.
- Use encryption on sensitive data and don't rely on client-side encryption.
- Limit who can exploit your application from the outside. Use authentication and authorization techniques to limit access to certain parts of the application to certain users.
- Regularly

Security Best Practices Guidelines for Securing CSS

Security Best Practices for Securing CSS

1. Keep strict access control for your stylesheets:
   - Ensure that any stylesheets used for confidential data are not publicly visible. Restrict access to those with a need to know.
   - Disable remote loading of stylesheets from other domains to protect from malicious code injections.
   - Be mindful of your folder structure to avoid any unintentional exposure of sensitive data.
2. Utilize effective data encryption:
   - Make sure to use secure HTTPS encryption when transferring sensitive data.
   - Use Secure Socket Layer (SSL) or Transport Layer Security (TLS) encryption when confidential data is exchanged between users and the web server.
3. Minimize vulnerabilities in the code:
   - Employ special tools and techniques to detect and eliminate CSS vulnerabilities in the code.
   - Ensure that the code is free of malicious and vulnerable content by regularly scanning the stylesheets.
4. Validate input sanitization:
   - Sanitize all user input before using it in the CSS file to prevent SQL injection and other malicious attacks.
   - Utilize input validation techniques (checking input against predefined rules) to verify the consistency of the data provided by the user.
5. Use secure content delivery networks (CDNs):
   - Employ a secure content delivery network (CDN) to secure the delivery of the stylesheets across the web.
   - CDNs can help protect against most common web application attacks, such as DDoS and Cross-Site Scripting (XSS).
6. Leverage web application firewalls (WAFs):
   - Install a web application firewall (WAF) to protect both the CSS files and other resources from malicious attacks.
   - WAFs can provide additional protection by blocking malicious requests and monitoring the traffic.
7. Use strong authentication and authorization mechanisms:
   - Make sure that all users have secure login credentials with strong passwords.
   - Implement multi-factor authentication, such as two-factor authentication, to enhance the security of the system.
8. Monitor the user activity:
   - Monitor user activity to detect any suspicious or malicious activities, such as unauthorized access to the system or changes to the stylesheets.
   - Utilize logging

Security Best Practices Guidelines for Injection Prevention

Injection Prevention Guidelines

What is Injection?

Injection is a type of attack where malicious code is inserted into an application by exploiting security vulnerabilities in the system. Injections can be used to gain access to or modify application data or to execute malicious code.

Security Considerations

The following security considerations should be taken when developing applications to protect against injection:

1. Validate user inputs: Application logic should always validate user input and enforce type, length, and format expectations.
2. Avoid dynamic SQL: Dynamic SQL can be exploited through user input and should be avoided where possible.
3. Use Object-Relational Mapping (ORM) frameworks: ORM frameworks can help minimize the risk of injection attacks.
4. Construct SQL queries with parameterized queries and stored procedures: Using parameterized queries and stored procedures can help shield the application from injection attacks.

What to Do If an Application is Vulnerable to Injection

If a vulnerability is detected within an application, the following steps should be taken to address the security risk from injection.

1. Identify the source of the vulnerability: Determine the source of the vulnerability and if it is exploitable.

2. Fix the security issue: Fix the underlying security issue by implementing the appropriate security defenses and validating user input.
3. Monitor the application: Implement logging and monitoring tools to continue to monitor and detect anomalies in user behaviors that could indicate potential vulnerabilities.
4. Test for vulnerabilities: Periodically test the application for potential vulnerabilities and security flaws.

Conclusion

Injection attacks can cause serious damage to an application. To protect against injection attacks, application developers should validate user input, avoid dynamic SQL, use ORM frameworks and construct SQL queries with parameterized queries and stored procedures. Should a vulnerability be detected, a series of steps should be taken to fix the vulnerability, monitor the application and test for further potential vulnerabilities.

Logging

Introduction

This guidelines are focused on providing developers with concentrated guidance on building application logging mechanisms, especially related to security logging.

Many systems enable network device, operating system, web server, mail server and database server logging, but often custom application event logging is missing, disabled or poorly configured. It provides much greater insight than infrastructure logging alone. Web application (e.g. web site or web service) logging is much more than having web server logs enabled (e.g. using Extended Log File Format).

Application logging should be consistent within the application, consistent across an organization's application portfolio and use industry standards where relevant, so the logged event data can be consumed, correlated, analyzed and managed by a wide variety of systems.

Purpose

Application logging should be always be included for security events. Application logs are invaluable data for:

- Identifying security incidents
- Monitoring policy violations
- Establishing baselines
- Assisting non-repudiation controls (note that the trait non-repudiation is hard to achieve for logs because their trustworthiness is often just based on the logging party being audited properly while mechanisms like digital signatures are hard to utilize here)
- Providing information about problems and unusual conditions
- Contributing additional application-specific data for incident investigation which is lacking in other log sources
- Helping defend against vulnerability identification and exploitation through attack detection

Application logging might also be used to record other types of events too such as:

- Security events
- Business process monitoring e.g. sales process abandonment, transactions, connections
- Anti-automation monitoring
- Audit trails e.g. data addition, modification and deletion, data exports
- Performance monitoring e.g. data load time, page timeouts
- Compliance monitoring
- Data for subsequent requests for information e.g. data subject access, freedom of information, litigation, police and other regulatory investigations
- Legally sanctioned interception of data e.g application-layer wire-tapping
- Other business-specific requirements

Process monitoring, audit and transaction logs/trails etc are usually collected for different purposes than security event logging, and this often means they should be kept separate.

The types of events and details collected will tend to be different.

For example a PCIDSS audit log will contain a chronological record of activities to provide an independently verifiable trail that permits reconstruction, review and examination to determine the original sequence of attributable transactions. It is important not to log too much, or too little.

Use knowledge of the intended purposes to guide what, when and how much. The remainder of this cheat sheet primarily discusses security event logging.

Design, implementation and testing

Event data sources

The application itself has access to a wide range of information events that should be used to generate log entries. Thus, the primary event data source is the application code itself.

The application has the most information about the user (e.g. identity, roles, permissions) and the context of the event (target, action, outcomes), and often this data is not available to either infrastructure devices, or even closely-related applications.

Other sources of information about application usage that could also be considered are:

- Client software e.g. actions on desktop software and mobile devices in local logs or using messaging technologies, JavaScript exception handler via Ajax, web browser such as using Content Security Policy (CSP) reporting mechanism
- Embedded instrumentation code
- Network firewalls
- Network and host intrusion detection systems (NIDS and HIDS)
- Closely-related applications e.g. filters built into web server software, web server URL redirects/rewrites to scripted custom error pages and handlers
- Application firewalls e.g. filters, guards, XML gateways, database firewalls, web application firewalls (WAFs)
- Database applications e.g. automatic audit trails, trigger-based actions
- Reputation monitoring services e.g. uptime or malware monitoring
- Other applications e.g. fraud monitoring, CRM
- Operating system e.g. mobile platform

The degree of confidence in the event information has to be considered when including event data from systems in a different trust zone. Data may be missing, modified, forged, replayed and could be malicious – it must always be treated as untrusted data.

Consider how the source can be verified, and how integrity and non-repudiation can be enforced.

Where to record event data

Applications commonly write event log data to the file system or a database (SQL or NoSQL). Applications installed on desktops and on mobile devices may use local storage and local databases, as well as sending data to remote storage.

Your selected framework may limit the available choices. All types of applications may send event data to remote systems (instead of or as well as more local storage).

This could be a centralized log collection and management system (e.g. SIEM or SEM) or another application elsewhere. Consider whether the application can simply send its event stream, unbuffered, to stdout, for management by the execution environment.

- When using the file system, it is preferable to use a separate partition than those used by the operating system, other application files and user generated content
  - For file-based logs, apply strict permissions concerning which users can access the directories, and the permissions of files within the directories
  - In web applications, the logs should not be exposed in web-accessible locations, and if done so, should have restricted access and be configured with a plain text MIME type (not HTML)
- When using a database, it is preferable to utilize a separate database account that is only used for writing log data and which has very restrictive database , table, function and command permissions
- Use standard formats over secure protocols to record and send event data, or log files, to other systems e.g. Common Log File System (CLFS) or Common Event Format (CEF) over syslog; standard formats facilitate integration with centralised logging services

Consider separate files/tables for extended event information such as error stack traces or a record of HTTP request and response headers and bodies.

Which events to log

The level and content of security monitoring, alerting and reporting needs to be set during the requirements and design stage of projects, and should be proportionate to the information security risks. This can then be used to define what should be logged.

There is no one size fits all solution, and a blind checklist approach can lead to unnecessary "alarm fog" that means real problems go undetected.

Where possible, always log:

- Input validation failures e.g. protocol violations, unacceptable encodings, invalid parameter names and values
- Output validation failures e.g. database record set mismatch, invalid data encoding
- Authentication successes and failures
- Authorization (access control) failures
- Session management failures e.g. cookie session identification value modification
- Application errors and system events e.g. syntax and runtime errors, connectivity problems, performance issues, third party service error messages, file system errors, file upload virus detection, configuration changes
- Application and related systems start-ups and shut-downs, and logging initialization (starting, stopping or pausing)
- Use of higher-risk functionality e.g. network connections, addition or deletion of users, changes to privileges, assigning users to tokens, adding or deleting tokens, use of systems administrative privileges, access by application administrators,all actions by users with administrative privileges, access to payment cardholder data, use of data encrypting keys, key changes, creation and deletion of system-level objects, data import and export including screen-based reports, submission of user-generated content - especially file uploads

- Legal and other opt-ins e.g. permissions for mobile phone capabilities, terms of use, terms & conditions, personal data usage consent, permission to receive marketing communications

Optionally consider if the following events can be logged and whether it is desirable information:

- Sequencing failure
- Excessive use
- Data changes
- Fraud and other criminal activities
- Suspicious, unacceptable or unexpected behavior
- Modifications to configuration
- Application code file and/or memory changes

Event attributes

Each log entry needs to include sufficient information for the intended subsequent monitoring and analysis. It could be full content data, but is more likely to be an extract or just summary properties.

The application logs must record "when, where, who and what" for each event.

The properties for these will be different depending on the architecture, class of application and host system/device, but often include the following:

- When
  - Log date and time (international format)
  - Event date and time - the event timestamp may be different to the time of logging e.g. server logging where the client application is hosted on remote device that is only periodically or intermittently online
  - Interaction identifier `Note A`
- Where
  - Application identifier e.g. name and version
  - Application address e.g. cluster/hostname or server IPv4 or IPv6 address and port number, workstation identity, local device identifier
  - Service e.g. name and protocol
  - Geolocation
  - Window/form/page e.g. entry point URL and HTTP method for a web application, dialogue box name
  - Code location e.g. script name, module name
- Who (human or machine user)
  - Source address e.g. user's device/machine identifier, user's IP address, cell/RF tower ID, mobile telephone number
  - User identity (if authenticated or otherwise known) e.g. user database table primary key value, user name, license number
- What
  - Type of event `Note B`
  - Severity of event `Note B` e.g. `{0=emergency, 1=alert, ..., 7=debug}, {fatal, error, warning, info, debug, trace}`
  - Security relevant event flag (if the logs contain non-security event data too)
  - Description

Additionally consider recording:

- Secondary time source (e.g. GPS) event date and time
- Action - original intended purpose of the request e.g. Log in, Refresh session ID, Log out, Update profile
- Object e.g. the affected component or other object (user account, data resource, file) e.g. URL, Session ID, User account, File
- Result status - whether the ACTION aimed at the OBJECT was successful e.g. Success, Fail, Defer
- Reason - why the status above occurred e.g. User not authenticated in database check ..., Incorrect credentials
- HTTP Status Code (web applications only) - the status code returned to the user (often 200 or 301)
- Request HTTP headers or HTTP User Agent (web applications only)
- User type classification e.g. public, authenticated user, CMS user, search engine, authorized penetration tester, uptime monitor (see "Data to exclude" below)
- Analytical confidence in the event detection `Note B` e.g. low, medium, high or a numeric value
- Responses seen by the user and/or taken by the application e.g. status code, custom text messages, session termination, administrator alerts
- Extended details e.g. stack trace, system error messages, debug information, HTTP request body, HTTP response headers and body
- Internal classifications e.g. responsibility, compliance references
- External classifications e.g. NIST Security Content Automation Protocol (SCAP), Mitre Common Attack Pattern Enumeration and Classification (CAPEC)

For more information on these, see the "other" related articles listed at the end, especially the comprehensive article by Anton Chuvakin and Gunnar Peterson.

Note A: The "Interaction identifier" is a method of linking all (relevant) events for a single user interaction (e.g. desktop application form submission, web page request, mobile app button click, web service call). The application knows all these events relate to the same interaction, and this should be recorded instead of losing the information and forcing subsequent correlation techniques to re-construct the separate events. For example a single SOAP request may have multiple

input validation failures and they may span a small range of times. As another example, an output validation failure may occur much later than the input submission for a long-running "saga request" submitted by the application to a database server.

Note B: Each organisation should ensure it has a consistent, and documented, approach to classification of events (type, confidence, severity), the syntax of descriptions, and field lengths & data types including the format used for dates/times.

Data to exclude

Never log data unless it is legally sanctioned. For example intercepting some communications, monitoring employees, and collecting some data without consent may all be illegal.

Never exclude any events from "known" users such as other internal systems, "trusted" third parties, search engine robots, uptime/process and other remote monitoring systems, pen testers, auditors. However, you may want to include a classification flag for each of these in the recorded data.

The following should not usually be recorded directly in the logs, but instead should be removed, masked, sanitized, hashed or encrypted:

- Application source code
- Session identification values (consider replacing with a hashed value if needed to track session specific events)
- Access tokens
- Sensitive personal data and some forms of personally identifiable information (PII) e.g. health, government identifiers, vulnerable people
- Authentication passwords
- Database connection strings
- Encryption keys and other master secrets
- Bank account or payment card holder data
- Data of a higher security classification than the logging system is allowed to store
- Commercially-sensitive information
- Information it is illegal to collect in the relevant jurisdictions
- Information a user has opted out of collection, or not consented to e.g. use of do not track, or where consent to collect has expired

Sometimes the following data can also exist, and whilst useful for subsequent investigation, it may also need to be treated in some special manner before the event is recorded:

- File paths
- Database connection strings
- Internal network names and addresses
- Non sensitive personal data (e.g. personal names, telephone numbers, email addresses)

Consider using personal data de-identification techniques such as deletion, scrambling or pseudonymization of direct and indirect identifiers where the individual's identity is not required, or the risk is considered too great.

In some systems, sanitization can be undertaken post log collection, and prior to log display.

Customizable logging

It may be desirable to be able to alter the level of logging (type of events based on severity or threat level, amount of detail recorded). If this is implemented, ensure that:

- The default level must provide sufficient detail for business needs
- It should not be possible to completely deactivate application logging or logging of events that are necessary for compliance requirements
- Alterations to the level/extent of logging must be intrinsic to the application (e.g. undertaken automatically by the application based on an approved algorithm) or follow change management processes (e.g. changes to configuration data, modification of source code)
- The logging level must be verified periodically

Event collection

If your development framework supports suitable logging mechanisms use, or build upon that. Otherwise, implement an application-wide log handler which can be called from other modules/components.

Document the interface referencing the organisation-specific event classification and description syntax requirements.

If possible create this log handler as a standard module that can be thoroughly tested, deployed in multiple applications, and added to a list of approved & recommended modules.

- Perform input validation on event data from other trust zones to ensure it is in the correct format (and consider alerting and not logging if there is an input validation failure)
- Perform sanitization on all event data to prevent log injection attacks e.g. carriage return (CR), line feed (LF) and delimiter characters (and optionally to remove sensitive data)
- Encode data correctly for the output (logged) format

- If writing to databases, read, understand and apply the SQL injection cheat sheet
- Ensure failures in the logging processes/systems do not prevent the application from otherwise running or allow information leakage
- Synchronize time across all servers and devices `Note C`

Note C: This is not always possible where the application is running on a device under some other party's control (e.g. on an individual's mobile phone, on a remote customer's workstation which is on another corporate network). In these cases attempt to measure the time offset, or record a confidence level in the event timestamp.

Where possible record data in a standard format, or at least ensure it can be exported/broadcast using an industry-standard format.

In some cases, events may be relayed or collected together in intermediate points. In the latter some data may be aggregated or summarized before forwarding on to a central repository and analysis system.

Verification

Logging functionality and systems must be included in code review, application testing and security verification processes:

- Ensure the logging is working correctly and as specified
- Check events are being classified consistently and the field names, types and lengths are correctly defined to an agreed standard
- Ensure logging is implemented and enabled during application security, fuzz, penetration and performance testing
- Test the mechanisms are not susceptible to injection attacks
- Ensure there are no unwanted side-effects when logging occurs
- Check the effect on the logging mechanisms when external network connectivity is lost (if this is usually required)
- Ensure logging cannot be used to deplete system resources, for example by filling up disk space or exceeding database transaction log space, leading to denial of service
- Test the effect on the application of logging failures such as simulated database connectivity loss, lack of file system space, missing write permissions to the file system, and runtime errors in the logging module itself
- Verify access controls on the event log data
- If log data is utilized in any action against users (e.g. blocking access, account lock-out), ensure this cannot be used to cause denial of service (DoS) of other users

Deployment and operation

Release

- Provide security configuration information by adding details about the logging mechanisms to release documentation
- Brief the application/process owner about the application logging mechanisms
- Ensure the outputs of the monitoring (see below) are integrated with incident response processes

Operation

Enable processes to detect whether logging has stopped, and to identify tampering or unauthorized access and deletion (see protection below).

Protection

The logging mechanisms and collected event data must be protected from mis-use such as tampering in transit, and unauthorized access, modification and deletion once stored. Logs may contain personal and other sensitive information, or the data may contain information regarding the application's code and logic.

In addition, the collected information in the logs may itself have business value (to competitors, gossip-mongers, journalists and activists) such as allowing the estimate of revenues, or providing performance information about employees.

This data may be held on end devices, at intermediate points, in centralized repositories and in archives and backups.

Consider whether parts of the data may need to be excluded, masked, sanitized, hashed or encrypted during examination or extraction.

At rest:

- Build in tamper detection so you know if a record has been modified or deleted
- Store or copy log data to read-only media as soon as possible
- All access to the logs must be recorded and monitored (and may need prior approval)
- The privileges to read log data should be restricted and reviewed periodically

In transit:

- If log data is sent over untrusted networks (e.g. for collection, for dispatch elsewhere, for analysis, for reporting), use a secure transmission protocol
- Consider whether the origin of the event data needs to be verified
- Perform due diligence checks (regulatory and security) before sending event data to third parties

See `NIST SP 800-92` Guide to Computer Security Log Management for more guidance.

Monitoring of events

The logged event data needs to be available to review and there are processes in place for appropriate monitoring, alerting and reporting:

- Incorporate the application logging into any existing log management systems/infrastructure e.g. centralized logging and analysis systems
- Ensure event information is available to appropriate teams
- Enable alerting and signal the responsible teams about more serious events immediately
- Share relevant event information with other detection systems, to related organizations and centralized intelligence gathering/sharing systems

Disposal of logs

Log data, temporary debug logs, and backups/copies/extractions, must not be destroyed before the duration of the required data retention period, and must not be kept beyond this time.

Legal, regulatory and contractual obligations may impact on these periods.

Attacks on Logs

Because of their usefulness as a defense, logs may be a target of attacks. See also OWASP Log Injection and CWE-117.

Confidentiality

Who should be able to read what? A confidentiality attack enables an unauthorized party to access sensitive information stored in logs.

- Logs contain PII of users. Attackers gather PII, then either release it or use it as a stepping stone for further attacks on those users.
- Logs contain technical secrets such as passwords. Attackers use it as a stepping stone for deeper attacks.

Integrity

Which information should be modifiable by whom?

- An attacker with read access to a log uses it to exfiltrate secrets.
- An attack leverages logs to connect with exploitable facets of logging platforms, such as sending in a payload over syslog in order to cause an out-of-bounds write.

Availability

What downtime is acceptable?

- An attacker floods log files in order to exhaust disk space available for non-logging facets of system functioning. For example, the same disk used for log files might be used for SQL storage of application data.
- An attacker floods log files in order to exhaust disk space available for further logging.
- An attacker uses one log entry to destroy other log entries.
- An attacker leverages poor performance of logging code to reduce application performance

Accountability

Who is responsible for harm?

- An attacker prevent writes in order to cover their tracks.
- An attacker prevent damages the log in order to cover their tracks.
- An attacker causes the wrong identity to be logged in order to conceal the responsible party.

Related articles

- OWASP ESAPI Documentation.
- OWASP Logging Project.
- IETF syslog protocol.
- Mitre Common Event Expression (CEE) (as of 2014 no longer actively developed).
- NIST SP 800-92 Guide to Computer Security Log Management.
- PCISSC PCI DSS v2.0 Requirement 10 and PA-DSS v2.0 Requirement 4.
- W3C Extended Log File Format.
- Other Build Visibility In, Richard Bejtlich, TaoSecurity blog.
- Other Common Event Format (CEF), Arcsight.
- Other Log Event Extended Format (**LEEF**), IBM.

- Other Common Log File System (CLFS), Microsoft.
- Other Building Secure Applications: Consistent Logging, Rohit Sethi & Nish Bhalla, Symantec Connect.

Application Logging Vocabulary

This document proposes a standard vocabulary for logging security events. The intent is to simplify monitoring and alerting such that, assuming developers trap errors and log them using this vocabulary, monitoring and alerting would be improved by simply keying on these terms.

Overview

Each year IBM Security commissions the Ponemon Institute to survey companies around the world for information related to security breaches, mitigation, and the associated costs; the result is called the Cost of a Data Breach Report.

This logging standard would seek to define specific keywords which, when applied consistently across software, would allow groups to simply monitor for these events terms across all applications and respond quickly in the event of attack.

Assumptions

- Observability/SRE groups must support the use of this standard and encourage developers to use it
- Incident Response must either ingest this data OR provide a means by which other monitoring teams can send a notification of alert, preferably programmatically.
- Architects must support, adopt, and contribute to this standard
- Developers must embrace this standard and begin to implement (requires knowledge and intent to understand potential attacks and trap those errors in code).

Getting Started

As a reminder, the goal of logging is to be able to alert on specific security events. Of course, the first step to logging these events is good error handling, if you're not trapping the events, you don't have an event to log.

Identifying Events

In order to better understand security event logging a good high-level understanding of threat modeling would be helpful, even if it's a simple approach of:

1. **What could go wrong?**
- Orders: could someone order on behalf of another?
- Authentication: could I log in as someone else?
- Authorization: could I see someone else' account?
1. **What would happen if it did?**
- Orders: I've placed an order on behalf of another... to an abandoned warehouse in New Jersey. Oops.
- Then I bragged about it on 4Chan.
- Then I told the New York Times about it.
1. **Who might intend to do this?**
- Intentional attacks by hackers.
- An employee "testing" how things work.
- An API coded incorrectly doing things the author did not intend.

Format

*NOTE: All dates should be logged in ISO 8601 format **WITH** UTC offset to ensure maximum portability*

```
{ "datetime": "2021-01-01T01:01:01-0700", "appid": "foobar.netportal_auth", "event": "AUTHN_login_success:joebob1",
"level": "INFO", "description": "User joebob1 login successfully", "useragent": "Mozilla/5.0 (Macintosh; Intel Mac OS X
10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36", "source_ip": "165.225.50.94",
"host_ip": "10.12.7.9", "hostname": "portalauth.foobar.com", "protocol": "https", "port": "440", "request_uri":
"/api/v2/auth/", "request_method": "POST", "region": "AWS-US-WEST-2", "geo": "USA" }
```

The Vocabulary

What follows are the various event types that should be captured. For each event type there is a prefix like "authn" and additional data that should be included for that event.

Portions of the full logging format are included for example, but a complete event log should follow the format above.

Authentication [AUTHN]

authn_login_success[:userid]

Description All login events should be recorded including success.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authn_login_success:joebob1",
"level": "INFO", "description": "User joebob1 login successfully", ... }
```

---

authn_login_successafterfail[:userid,retries]

Description The user successfully logged in after previously failing.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"authn_login_successafterfail:joebob1,2", "level": "INFO", "description": "User joebob1 login successfully", ... }
```

---

authn_login_fail[:userid]

Description All login events should be recorded including failure.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authn_login_fail:joebob1",
"level": "WARN", "description": "User joebob1 login failed", ... }
```

---

authn_login_fail_max[:userid,maxlimit(int)]

Description All login events should be recorded including failure.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authn_login_fail_max:joebob1,3",
"level": "WARN", "description": "User joebob1 reached the login fail limit of 3", ... }
```

---

authn_login_lock[:userid,reason]

Description When the feature exists to lock an account after x retries or other condition, the lock should be logged with relevant data.

Level: WARN

Reasons:

- maxretries: The maximum number of retries was reached
- suspicious: Suspicious activity was observed on the account
- customer: The customer requested their account be locked
- other: Other

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"authn_login_lock:joebob1,maxretries", "level": "WARN", "description": "User joebob1 login locked because maxretries
exceeded", ... }
```

---

authn_token_delete[:appid]

Description When a token is deleted it should be recorded

Level:: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authn_token_delete:foobarapi",
"level": "WARN", "description": "The token for foobarapi has been deleted", ... }
```

---

authn_password_change[:userid]

Description Every password change should be logged, including the userid that it was for.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authn_password_change:joebob1",
"level": "INFO", "description": "User joebob1 has successfully changed their password", ... }
```

---

authn_password_change_fail[:userid]

Description An attempt to change a password that failed. May also trigger other events such as `authn_login_lock`.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authn_password_change:joebob1",
"level": "INFO", "description": "User joebob1 failed to changing their password", ... }
```

---

authn_impossible_travel[:userid,region1,region2]

Description When a user is logged in from one city and suddenly appears in another, too far away to have traveled in a reasonable timeframe, this often indicates a potential account takeover.

Level:: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"authn_impossible_travel:joebob1,US-OR,CN-SH", "level": "CRITICAL", "description": "User joebob1 has accessed the
application in two distant cities at the same time", ... }
```

---

authn_token_created[:userid, entitlement(s)]

Description When a token is created for service access it should be recorded

Level:: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "aws.foobar.com", "event":
"authn_token_created:app.foobarapi.prod,create,read,update", "level": "INFO", "description": "A token has been created
for app.foobarapi.prod with create,read,update", ... }
```

---

authn_token_revoked[:userid,tokenid]

Description A token has been revoked for the given account.

Level:: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "aws.foobar.com", "event":
"authn_token_revoked:app.foobarapi.prod,xyz-abc-123-gfk", "level": "INFO", "description": "Token ID: xyz-abc-123-gfk
was revoked for user app.foobarapi.prod", ... }
```

---

authn_token_reuse[:userid,tokenid]

Description A previously revoked token was attempted to be reused.

Level:: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "aws.foobar.com", "event":
"authn_token_reuse:app.foobarapi.prod,xyz-abc-123-gfk", "level": "CRITICAL", "description": "User app.foobarapi.prod
attempted to use token ID: xyz-abc-123-gfk which was previously revoked", ... }
```

Authorization [AUTHZ]

---

authz_fail[:userid,resource]

Description An attempt was made to access a resource which was unauthorized

Level:: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authz_fail:joebob1,resource",
"level": "CRITICAL", "description": "User joebob1 attempted to access a resource without entitlement", ... }
```

---

authz_change[:userid,from,to]

Description The user or entity entitlements was changed

Level:: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "authz_change:joebob1,user,admin",
"level": "WARN", "description": "User joebob1 access was changed from user to admin", ... }
```

---

authz_admin[:userid,event]

Description All activity by privileged users such as admin should be recorded.

Level:: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"authz_admin:joebob1,user_privilege_change", "level": "WARN", "description": "Administrtator joebob1 has updated
privileges of user foobarapi from user to admin", ... }
```

Excessive Use [EXCESS]

excess_rate_limit_exceeded[userid,max]

Description Expected service limit ceilings should be established and alerted when exceeded, even if simply for managing costs and scaling.

Level:: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"excess_rate_limit_exceeded:app.foobarapi.prod,100000", "level": "WARN", "description": "User app.foobarapi.prod has
exceeded max:100000 requests", ... }
```

File Upload [UPLOAD]

upload_complete[userid,filename,type]

Description On successful file upload the first step in the validation process is that the upload has completed.

Level:: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"upload_complete:joebob1,user_generated_content.png,PNG", "level": "INFO", "description": "User joebob1 has uploaded
user_generated_content.png", ... }
```

---

upload_stored[filename,from,to]

Description One step in good file upload validation is to move/rename the file and when providing the content back to end users, never reference the original filename in the download. This is true both when storing in a filesystem as well as in block storage.

Level:: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"upload_stored:user_generated_content.png,kjsdhkrjhwijhsiuhdf0000102020002", "level": "INFO", "description": "File
user_generated_content.png was stored in the database with key abcdefghijk101010101", ... }
```

---

upload_validation[filename,(virusscan|imagemagick|...):(FAILED|incomplete|passed)]

Description All file uploads should have some validation performed, both for correctness (is in fact of file type x), and for safety (does not contain a virus).

Level:: INFO|CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"upload_validation:filename,virusscan:FAILED", "level": "CRITICAL", "description": "File user_generated_content.png
FAILED virus scan and was purged", ... }
```

---

upload_delete[userid,fileid]

Description When a file is deleted for normal reasons it should be recorded.

Level:: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "upload_delete:joebob1,", "level":
"INFO", "description": "User joebob1 has marked file abcdefghijk101010101 for deletion.", ... }
```

Input Validation [INPUT]

input_validation_fail[:field,userid]

Description When input validation fails on the server-side it must either be because a) sufficient validation was not provided on the client, or b) client-side validation was bypassed. In either case it's an opportunity for attack and should be mitigated quickly.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"input_validation_fail:date_of_birth,joebob1", "level": "WARN", "description": "User joebob1 submitted data that failed
validation.", ... }
```

Malicious Behavior [MALICIOUS

malicious_excess_404:[userid|IP,useragent]

Description When a user makes numerous requests for files that don't exist it often is an indicator of attempts to "force-browse" for files that could exist and is often behavior indicating malicious intent.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"malicious_excess404:123.456.789.101,M@l1c10us-Hax0rB0t0-v1", "level": "WARN", "description": "A user at
```

```
123.456.789.101 has generated a large number of 404 requests.", ... }
```

---

malicious_extraneous:[userid|IP,inputname,useragent]

Description When a user submits data to a backend handler that was not expected it can indicate probing for input validation errors. If your backend service receives data it does not handle or have an input for this is an indication of likely malicious abuse.

Level: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"malicious_extraneous:dr@evil.com,creditcardnum,Mozilla/5.0 (X11; Linux x86_64; rv:10.0) Gecko/20100101 Firefox/10.0",
"level": "WARN", "description": "User dr@evil.com included field creditcardnum in the request which is not handled by
this service.", ... }
```

---

malicious_attack_tool:[userid|IP,toolname,useragent]

Description When obvious attack tools are identified either by signature or by user agent they should be logged.

TODO: A future version of this standard should link to known attack tools, signatures and user-agent strings. For instance, the tool "Nikto" leaves behind its user agent by default with a string like ***"Mozilla/5.00 (Nikto/2.1.6) (Evasions:None) (Test:Port Check)"***

Level: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"malicious_attack_tool:127.0.0.1,nikto,Mozilla/5.00 (Nikto/2.1.6) (Evasions:None) (Test:Port Check)", "level": "WARN",
"description": "Attack traffic indicating use of Nikto coming from 127.0.0.1", ... }
```

---

malicious_cors:[userid|IP,useragent,referer]

Description When attempts are made from unauthorized origins they should of course be blocked, but also logged whenever possible. Even if we block an illegal cross-origin request the fact that the request is being made could be an indication of attack.

NOTE: Did you know that the word "referer" is misspelled in the original HTTP specification? The correct spelling should be "referrer" but the original typo persists to this day and is used here intentionally.

Level: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"malicious_cors:127.0.0.1,Mozilla/5.0 (X11; Linux x86_64; rv:10.0) Gecko/20100101 Firefox/10.0,attack.evil.com",
"level": "WARN", "description": "An illegal cross-origin request from 127.0.0.1 was referred from attack.evil.com" ...
}
```

---

malicious_direct_reference:[userid|IP, useragent]

Description A common attack against authentication and authorization is to directly access an object without credentials or appropriate access authority. Failing to prevent this flaw used to be one of the OWASP Top Ten called **Insecure Direct Object Reference**. Assuming you've correctly prevented this attack, logging the attempt is valuable to identify malicious users.

Level: CRITICAL

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "malicious_direct:joebob1,
Mozilla/5.0 (X11; Linux x86_64; rv:10.0) Gecko/20100101 Firefox/10.0", "level": "WARN", "description": "User joebob1
attempted to access an object to which they are not authorized", ... }
```

Privilege Changes [PRIVILEGE]

This section focuses on object privilege changes such as read/write/execute permissions or objects in a database having authorization meta-information changed.

Changes to user/account are covered in the User Management section.

---

privilege_permissions_changed:[userid,file|object,fromlevel,tolevel]

Description Tracking changes to objects to which there are access control restrictions can uncover attempt to escalate privilege on those files by unauthorized users.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "malicious_direct:joebob1,
/users/admin/some/important/path,0511,0777", "level": "WARN", "description": "User joebob1 changed permissions on
/users/admin/some/important/path", ... }
```

Sensitive Data Changes [DATA]

It's not necessary to log or alert on changes to all files, but in the case of highly sensitive files or data it is important that we monitor and alert on changes.

---

sensitive_create:[userid,file|object]

Description When a new piece of data is created and marked as sensitive or placed into a directory/table/repository where sensitive data is stored, that creation should be logged and reviewed periodically.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sensitive_create:joebob1,
/users/admin/some/important/path", "level": "WARN", "description": "User joebob1 created a new file in
/users/admin/some/important/path", ... }
```

---

sensitive_read:[userid,file|object]

Description All data marked as sensitive or placed into a directory/table/repository where sensitive data is stored should be have access logged and reviewed periodically.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sensitive_read:joebob1,
/users/admin/some/important/path", "level": "WARN", "description": "User joebob1 read file
/users/admin/some/important/path", ... }
```

---

sensitive_update:[userid,file|object]

Description All data marked as sensitive or placed into a directory/table/repository where sensitive data is stored should be have updates to the data logged and reviewed periodically.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sensitive_update:joebob1,
/users/admin/some/important/path", "level": "WARN", "description": "User joebob1 modified file
/users/admin/some/important/path", ... }
```

---

sensitive_delete:[userid,file|object]

Description All data marked as sensitive or placed into a directory/table/repository where sensitive data is stored should have deletions of the data logged and reviewed periodically. The file should not be immediately deleted but marked for deletion and an archive of file should be maintained according to legal/privacy requirements.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sensitive_delete:joebob1,
/users/admin/some/important/path", "level": "WARN", "description": "User joebob1 marked file
```

```
/users/admin/some/important/path for deletion", ... }
```

Sequence Errors [SEQUENCE]

Also called a ***business logic attack***, if a specific path is expected through a system and an attempt is made to skip or change the order of that path it could indicate malicious intent.

---

sequence_fail:[userid]

Description When a user reaches a part of the application out of sequence it may indicate intentional abuse of the business logic and should be tracked.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sequence_fail:joebob1", "level":
"WARN", "description": "User joebob1 has reached a part of the application out of the normal application flow.", ... }
```

Session Management [SESSION]

session_created:[userid]

Description When a new authenticated session is created that session may be logged and activity monitored.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "session_created:joebob1", "level":
"INFO", "description": "User joebob1 has started a new session", ... }
```

---

session_renewed:[userid]

Description When a user is warned of session to be expired/revoked and chooses to extend their session that activity should be logged. Also, if the system in question contains highly confidential data then extending a session may require additional verification.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "session_renewed:joebob1", "level":
"WARN", "description": "User joebob1 was warned of expiring session and extended.", ... }
```

---

session_expired:[userid,reason]

Description When a session expires, especially in the case of an authenticated session or with sensitive data, then that session expiry may be logged and clarifying data included. The reason code may be any such as: logout, timeout, revoked, etc. Sessions should never be deleted but rather expired in the case of revocation requirement.

Level: INFO

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "session_expired:joebob1,revoked",
"level": "WARN", "description": "User joebob1 session expired due to administrator revocation.", ... }
```

---

session_use_after_expire:[userid]

Description In the case a user attempts to access systems with an expire session it may be helpful to log, especially if combined with subsequent login failure. This could identify a case where a malicious user is attempting a session hijack or directly accessing another persons machine/browser.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "session_use_after_expire:joebob1",
"level": "WARN", "description": "User joebob1 attempted access after session expired.", ... }
```

System Events [SYS]

sys_startup:[userid]

Description When a system is first started it can be valuable to log the startup, even if the system is serverless or a container, especially if possible to log the user that initiated the system.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sys_startup:joebob1", "level": "WARN", "description": "User joebob1 spawned a new instance", ... }
```

---

sys_shutdown:[userid]

Description When a system is shutdown it can be valuable to log the event, even if the system is serverless or a container, especially if possible to log the user that initiated the system.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sys_shutdown:joebob1", "level": "WARN", "description": "User joebob1 stopped this instance", ... }
```

---

sys_restart:[userid]

Description When a system is restarted it can be valuable to log the event, even if the system is serverless or a container, especially if possible to log the user that initiated the system.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sys_restart:joebob1", "level": "WARN", "description": "User joebob1 initiated a restart", ... }
```

---

sys_crash[:reason]

Description If possible to catch an unstable condition resulting in the crash of a system, logging that event could be helpful, especially if the event is triggered by an attack.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sys_crash:outofmemory, "level": "WARN", "description": "The system crashed due to Out of Memory error.", ... }
```

---

sys_monitor_disabled:[userid,monitor]

Description If your systems contain agents responsible for file integrity, resources, logging, virus, etc. it is especially valuable to know if they are halted and by whom.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "sys_monitor_disabled:joebob1,crowdstrike", "level": "WARN", "description": "User joebob1 has disable CrowdStrike", ... }
```

---

sys_monitor_enabled:[userid,monitor]

Description If your systems contain agents responsible for file integrity, resources, logging, virus, etc. it is especially valuable to know if they are started again after being stopped, and by whom.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"sys_monitor_enabled:joebob1,crowdstrike", "level": "WARN", "description": "User joebob1 has enabled CrowdStrike", ...
}
```

User Management [USER]

user_created:[userid,newuserid,attributes[one,two,three]]

Description When creating new users, logging the specifics of the user creation event is helpful, especially if new users can be created with administration privileges.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"user_created:joebob1,user1,admin:create,update,delete", "level": "WARN", "description": "User joebob1 created user1
with admin:create,update,delete privilege attributes", ... }
```

---

user_updated:[userid,onuserid,attributes[one,two,three]]

Description When updating users, logging the specifics of the user update event is helpful, especially if users can be updated with administration privileges.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event":
"user_updated:joebob1,user1,admin:create,update,delete", "level": "WARN", "description": "User joebob1 updated user1
with attributes admin:create,update,delete privilege attributes", ... }
```

---

user_archived:[userid,onuserid]

Description It is always best to archive users rather than deleting, except where required. When archiving users, logging the specifics of the user archive event is helpful. A malicious user could use this feature to deny service to legitimate users.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "user_archived:joebob1,user1",
"level": "WARN", "description": "User joebob1 archived user1", ... }
```

---

user_deleted:[userid,onuserid]

Description It is always best to archive users rather than deleting, except where required. When deleting users, logging the specifics of the user delete event is helpful. A malicious user could use this feature to deny service to legitimate users.

Level: WARN

Example:

```
{ "datetime": "2019-01-01 00:00:00,000", "appid": "foobar.netportal_auth", "event": "user_deleted:joebob1,user1",
"level": "WARN", "description": "User joebob1 has deleted user1", ... }
```

Exclusions

As important as what you DO log is what you DON'T log. Private or secret information, source code, keys, certs, etc. should never be logged.

For comprehensive overview of items that should be excluded from logging, please see the OWASP Logging Cheat Sheet.

Security Best Practices Guidelines for Password Storage

Security Best Practices Guidelines for Password Storage

1. Do not store passwords in plain text on any system or server.
2. Use strong, unique passwords for all accounts wherever possible, avoiding common words and phrases.
3. Encrypt passwords with a one-way hashing algorithm, such as SHA-2 or bcrypt, before storing them.
4. Salt passwords using a cryptographically secure pseudorandom function (CSPRNG) to generate a unique string for each password.
5. Never store passwords in browser cookies, local storage, or other unsecure locations.
6. Where possible, use multi-factor authentication (MFA) to help protect account access.
7. Audit systems regularly to check for suspicious behaviour related to password handling.
8. If a user's account is compromised, the password should be reset and the device should be running the latest security patches.
9. Do not share passwords with people outside the organisation, including family and friends.
10. Establish a password policy that emphasizes the importance of creating and maintaining strong, secure passwords.

Server-Side Request Forgery Prevention

Introduction

The objective of the cheat sheet is to provide advices regarding the protection against Server Side Request Forgery (SSRF) attack.

This cheat sheet will focus on the defensive point of view and will not explain how to perform this attack. This talk from the security researcher Orange Tsai as well as this document provide techniques on how to perform this kind of attack.

Context

SSRF is an attack vector that abuses an application to interact with the internal/external network or the machine itself. One of the enablers for this vector is the mishandling of URLs, as showcased in the following examples:

- Image on an external server (*e.g.* user enters image URL of their avatar for the application to download and use).
- Custom WebHook (users have to specify Webhook handlers or Callback URLs).
- Internal requests to interact with another service to serve a specific functionality. Most of the times, user data is sent along to be processed, and if poorly handled, can perform specific injection attacks.

Overview of a SSRF common flow

Notes:

- SSRF is not limited to the HTTP protocol. Generally, the first request is HTTP, but in cases where the application itself performs the second request, it could use different protocols (*e.g.* FTP, SMB, SMTP, etc.) and schemes (*e.g.* `file://`, `phar://`, `gopher://`, `data://`, `dict://`, etc.).
- If the application is vulnerable to XML eXternal Entity (XXE) injection then it can be exploited to perform a SSRF attack, take a look at the XXE cheat sheet to learn how to prevent the exposure to XXE.

Cases

Depending on the application's functionality and requirements, there are two basic cases in which SSRF can happen:

- Application can send request only to **identified and trusted applications**: Case when allow listing approach is available.
- Application can send requests to **ANY external IP address or domain name**: Case when allow listing approach is unavailable.

Because these two cases are very different, this cheat sheet will describe defences against them separately.

Case 1 - Application can send request only to identified and trusted applications

Sometimes, an application needs to perform a request to another application, often located on another network, to perform a specific task. Depending on the business case, user input is required for the functionality to work.

Example

> Take the example of a web application that receives and uses personal information from a user, such as their first name, last name, birth date etc. to create a profile in an internal HR system. By design, that web application will have to communicate using a protocol that the HR system understands to process that data. Basically, the user cannot reach the HR system directly, but, if the web application in charge of receiving user information is vulnerable to SSRF, the user can leverage it to access the HR system. The user leverages the web application as a proxy to the HR system.

The allow list approach is a viable option since the internal application called by the *VulnerableApplication* is clearly identified in the technical/business flow. It can be stated that the required calls will only be targeted between those identified and trusted applications.

Available protections

Several protective measures are possible at the **Application** and **Network** layers. To apply the **defense in depth** principle, both layers will be hardened against such attacks.

Application layer

The first level of protection that comes to mind is Input validation.

Based on that point, the following question comes to mind: *How to perform this input validation?*

As Orange Tsai shows in his talk, depending on the programming language used, parsers can be abused. One possible countermeasure is to apply the allow list approach when input validation is used because, most of the time, the format of the information expected from the user is globally known.

The request sent to the internal application will be based on the following information:

- String containing business data.
- IP address (V4 or V6).
- Domain name.
- URL.

Note: Disable the support for the following of the redirection in your web client in order to prevent the bypass of the input validation described in the section `Exploitation tricks > Bypassing restrictions > Input validation > Unsafe redirect` of this document.

String

In the context of SSRF, validations can be added to ensure that the input string respects the business/technical format expected.

A regex can be used to ensure that data received is valid from a security point of view if the input data have a simple format (*e.g.* token, zip code, etc.). Otherwise, validation should be conducted using the libraries available from the `string` object because regex for complex formats are difficult to maintain and are highly error-prone.

User input is assumed to be non-network related and consists of the user's personal information.

Example:

```
java //Regex validation for a data having a simple format if(Pattern.matches("[a-zA-Z0-9\\s\\-]{1,50}", userInput)){
//Continue the processing because the input data is valid }else{ //Stop the processing and reject the request }
```

IP address

In the context of SSRF, there are 2 possible validations to perform:

1. Ensure that the data provided is a valid IP V4 or V6 address.
2. Ensure that the IP address provided belongs to one of the IP addresses of the identified and trusted applications.

The first layer of validation can be applied using libraries that ensure the security of the IP address format, based on the technology used (library option is proposed here to delegate the managing of the IP address format and leverage battle-tested validation function):

> Verification of the proposed libraries has been performed regarding the exposure to bypasses (Hex, Octal, Dword, URL and Mixed encoding) described in this article.
>
> **JAVA:** Method InetAddressValidator.isValid from the Apache Commons Validator library.
> - **It is NOT exposed** to bypass using Hex, Octal, Dword, URL and Mixed encoding.
>
> **.NET**: Method IPAddress.TryParse from the SDK.
> - **It is exposed** to bypass using Hex, Octal, Dword and Mixed encoding but **NOT** the URL encoding.
> - As allow listing is used here, any bypass tentative will be blocked during the comparison against the allowed list of IP addresses.
>
> **JavaScript**: Library ip-address.
> - **It is NOT exposed** to bypass using Hex, Octal, Dword, URL and Mixed encoding.
>
> **Ruby**: Class IPAddr from the SDK.
> - **It is NOT exposed** to bypass using Hex, Octal, Dword, URL and Mixed encoding.
>
> **Use the output value of the method/library as the IP address to compare against the allow list.**

After ensuring the validity of the incoming IP address, the second layer of validation is applied. An allow list is created after determining all the IP addresses (v4 and v6 to avoid bypasses) of the identified and trusted applications. The valid IP is cross-checked with that list to ensure its communication with the internal application (string strict comparison with case sensitive).

Domain name

In the attempt of validate domain names, it is apparent to do a DNS resolution to verify the existence of the domain. In general, it is not a bad idea, yet it opens up the application to attacks depending on the configuration used regarding the DNS servers used for the domain name resolution:

- It can disclose information to external DNS resolvers.
- It can be used by an attacker to bind a legit domain name to an internal IP address. See the section `Exploitation tricks > Bypassing restrictions > Input validation > DNS pinning` of this document.
- An attacker can use it to deliver a malicious payload to the internal DNS resolvers and the API (SDK or third-party) used by the application to handle the DNS communication and then, potentially, trigger a vulnerability in one of these components.

In the context of SSRF, there are two validations to perform:

1. Ensure that the data provided is a valid domain name.
2. Ensure that the domain name provided belongs to one of the domain names of the identified and trusted applications (the allow listing comes to action here).

Similar to the IP address validation, the first layer of validation can be applied using libraries that ensure the security of the domain name format, based on the technology used (library option is proposed here in order to delegate the managing of the domain name format and leverage battle tested validation function):

> Verification of the proposed libraries has been performed to ensure that the proposed functions do not perform any DNS resolution query.

- **JAVA:** Method DomainValidator.isValid from the Apache Commons Validator library.
- **.NET**: Method Uri.CheckHostName from the SDK.
- **JavaScript**: Library is-valid-domain.
- **Python**: Module validators.domain.

  Ruby: No valid dedicated gem has been found.
  - domainator, public_suffix and addressable has been tested but unfortunately they all consider `<script>alert(1)</script>.owasp.org` as a valid domain name.
  - This regex, taken from here, can be used: `^(((?!-))(xn--|_{1,1})?[a-z0-9-]{0,61}[a-z0-9]{1,1}\.)*(xn--)?([a-z0-9][a-z0-9\-]{0,60}|[a-z0-9-]{1,30}\.[a-z]{2,})$`

Example of execution of the proposed regex for Ruby:

```
ruby domain_names = ["owasp.org","owasp-test.org","doc-test.owasp.org","doc.owasp.org",
"<script>alert(1)</script>","<script>alert(1)</script>.owasp.org"] domain_names.each { |domain_name| if ( domain_name
=~
/^(((?!-))(xn--|_{1,1})?[a-z0-9-]{0,61}[a-z0-9]{1,1}\.)*(xn--)?([a-z0-9][a-z0-9\-]{0,60}|[a-z0-9-]{1,30}\.[a-z]{2,})$/
) puts "[i] #{domain_name} is VALID" else puts "[!] #{domain_name} is INVALID" end }

bash $ ruby test.rb [i] owasp.org is VALID [i] owasp-test.org is VALID [i] doc-test.owasp.org is VALID [i]
doc.owasp.org is VALID [!] <script>alert(1)</script> is INVALID [!] <script>alert(1)</script>.owasp.org is INVALID
```

After ensuring the validity of the incoming domain name, the second layer of validation is applied:

1. Build an allow list with all the domain names of every identified and trusted applications.
2. Verify that the domain name received is part of this allow list (string strict comparison with case sensitive).

Unfortunately here, the application is still vulnerable to the `DNS pinning` bypass mentioned in this document. Indeed, a DNS resolution will be made when the business code will be executed. To address that issue, the following action must be taken in addition of the validation on the domain name:

1. Ensure that the domains that are part of your organization are resolved by your internal DNS server first in the chains of DNS resolvers.
2. Monitor the domains allow list in order to detect when any of them resolves to a/an: - Local IP address (V4 + V6). - Internal IP of your organization (expected to be in private IP ranges) for the domain that are not part of your organization.

The following Python3 script can be used, as a starting point, for the monitoring mentioned above:

```python

## Dependencies: pip install ipaddress dnspython

import ipaddress import dns.resolver

## Configure the allow list to check

DOMAINS_ALLOWLIST = ["owasp.org", "labslinux"]

## Configure the DNS resolver to use for all DNS queries

DNS_RESOLVER = dns.resolver.Resolver() DNS_RESOLVER.nameservers = ["1.1.1.1"]

def verify_dns_records(domain, records, type): """ Verify if one of the DNS records resolve to a non public IP address. Return a boolean indicating if any error has been detected. """ error_detected = False if records is not None: for record in records: value = record.to_text().strip() try: ip = ipaddress.ip_address(value) # See https://docs.python.org/3/library/ipaddress.html#ipaddress.IPv4Address.is_global if not ip.is_global: print("[!] DNS record type '%s' for domain name '%s' resolve
```

to a non public IP address '%s'!" % (type, domain, value)) error_detected = True except ValueError: error_detected = True print("[!] '%s' is not valid IP address!" % value) return error_detected

def check(): """ Perform the check of the allow list of domains. Return a boolean indicating if any error has been detected. """ error_detected = False for domain in DOMAINS_ALLOWLIST: # Get the IPs of the current domain # See https://en.wikipedia.org/wiki/List_of_DNS_record_types try: # A = IPv4 address record ip_v4_records = DNS_RESOLVER.query(domain, "A") except Exception as e: ip_v4_records = None print("[i] Cannot get A record for domain '%s': %s\n" % (domain,e)) try: # AAAA = IPv6 address record ip_v6_records = DNS_RESOLVER.query(domain, "AAAA") except Exception as e: ip_v6_records = None print("[i] Cannot get AAAA record for domain '%s': %s\n" % (domain,e)) # Verify the IPs obtained if verify_dns_records(domain, ip_v4_records, "A") or verify_dns_records(domain, ip_v6_records, "AAAA"): error_detected = True return error_detected

if **name**== "**main**": if check(): exit(1) else: exit(0) ```

URL

Do not accept complete URLs from the user because URL are difficult to validate and the parser can be abused depending on the technology used as showcased by the following talk of Orange Tsai.

If network related information is really needed then only accept a valid IP address or domain name.

Network layer

The objective of the Network layer security is to prevent the *VulnerableApplication* from performing calls to arbitrary applications. Only allowed *routes* will be available for this application in order to limit its network access to only those that it should communicate with.

The Firewall component, as a specific device or using the one provided within the operating system, will be used here to define the legitimate flows.

In the schema below, a Firewall component is leveraged to limit the application's access, and in turn, limit the impact of an application vulnerable to SSRF:

Network segregation (see this set of implementation advice can also be leveraged and **is highly recommended in order to block illegitimate calls directly at network level itself**.

Case 2 - Application can send requests to ANY external IP address or domain name

This case happens when a user can control a URL to an **External** resource and the application makes a request to this URL (e.g. in case of WebHooks). Allow lists cannot be used here because the list of IPs/domains is often unknown upfront and is dynamically changing.

In this scenario, *External* refers to any IP that doesn't belong to the internal network, and should be reached by going over the public internet.

Thus, the call from the *Vulnerable Application*:

- **Is NOT** targeting one of the IP/domain *located inside* the company's global network.
- Uses a convention defined between the *VulnerableApplication* and the expected IP/domain in order to *prove* that the call has been legitimately initiated.

Challenges in blocking URLs at application layer

Based on the business requirements of the above mentioned applications, the allow list approach is not a valid solution. Despite knowing that the block-list approach is not an impenetrable wall, it is the best solution in this scenario. It is informing the application what it should **not** do.

Here is why filtering URLs is hard at the Application layer:

- It implies that the application must be able to detect, at the code level, that the provided IP (V4 + V6) is not part of the official private networks ranges including also *localhost* and *IPv4/v6 Link-Local* addresses. Not every SDK provides a built-in feature for this kind of verification, and leaves the handling up to the developer to understand all of its pitfalls and possible values, which makes it a demanding task.
- Same remark for domain name: The company must maintain a list of all internal domain names and provide a centralized service to allow an application to verify if a provided domain name is an internal one. For this verification, an internal DNS resolver can be queried by the application but this internal DNS resolver must not resolve external domain names.

Available protections

Taking into consideration the same assumption in the following example for the following sections.

Application layer

Like for the case n°1, it is assumed that the `IP Address` or `domain name` is required to create the request that will be sent to the *TargetApplication*.

The first validation on the input data presented in the case n°1 on the 3 types of data will be the same for this case **BUT the second validation will differ**. Indeed, here we must use the block-list approach.

> **Regarding the proof of legitimacy of the request**: The *TargetedApplication* that will receive the request must generate a random token (ex: alphanumeric of 20 characters) that is expected to be passed by the caller (in body via a parameter for which the name is also defined by the application itself and only allow characters set `[a-z]{1,10}`) to perform a valid request. The receiving endpoint must only accept HTTP POST requests.

Validation flow (if one the validation steps fail then the request is rejected):

1. The application will receive the IP address or domain name of the *TargetedApplication* and it will apply the first validation on the input data using the libraries/regex mentioned in this section.

   The second validation will be applied against the IP address or domain name of the *TargetedApplication* using the following block-list approach: - For IP address:

   - The application will verify that it is a public one (see the hint provided in the next paragraph with the python code sample).
   - For domain name:
     1. The application will verify that it is a public one by trying to resolve the domain name against the DNS resolver that will only resolve internal domain name. Here, it must return a response indicating that it do not know the provided domain because the expected value received must be a public domain.
     2. To prevent the `DNS pinning` attack described in this document, the application will retrieve all the IP addresses behind the domain name provided (taking records *A* + *AAAA* for IPv4 + IPv6) and it will apply the same verification described in the previous point about IP addresses.
3. The application will receive the protocol to use for the request via a dedicated input parameter for which it will verify the value against an allowed list of protocols (`HTTP` or `HTTPS`).
4. The application will receive the parameter name for the token to pass to the *TargetedApplication* via a dedicated input parameter for which it will only allow the characters set `[a-z]{1,10}`.
5. The application will receive the token itself via a dedicated input parameter for which it will only allow the characters set `[a-zA-Z0-9]{20}`.
6. The application will receive and validate (from a security point of view) any business data needed to perform a valid call.
7. The application will build the HTTP POST request **using only validated information** and will send it (*don't forget to disable the support for redirection in the web client used*).

Network layer

Similar to the following section.

IMDSv2 in AWS

In cloud environments SSRF is often used to access and steal credentials and access tokens from metadata services (e.g. AWS Instance Metadata Service, Azure Instance Metadata Service, GCP metadata server).

IMDSv2 is an additional defence-in-depth mechanism for AWS that mitigates some of the instances of SSRF.

To leverage this protection migrate to IMDSv2 and disable old IMDSv1. Check out AWS documentation for more details.

Semgrep Rules

Semgrep is a command-line tool for offline static analysis. Use pre-built or custom rules to enforce code and security standards in your codebase. Checkout the Semgrep rule for SSRF to identify/investigate for SSRF vulnerabilities in Java https://semgrep.dev/salecharohit:owasp_java_ssrf

References

Online version of the SSRF bible (PDF version is used in this cheat sheet).

Article about Bypassing SSRF Protection.

Articles about SSRF attacks: Part 1, part 2 and part 3.

Article about IMDSv2

Tools and code used for schemas

- Mermaid Online Editor and Mermaid documentation.
- Draw.io Online Editor.

Mermaid code for SSRF common flow (printscreen are used to capture PNG image inserted into this cheat sheet):

```
text sequenceDiagram participant Attacker participant VulnerableApplication participant TargetedApplication
Attacker->>VulnerableApplication: Crafted HTTP request VulnerableApplication->>TargetedApplication: Request (HTTP,
FTP...) Note left of TargetedApplication: Use payload included<br>into the request to<br>VulnerableApplication
TargetedApplication->>VulnerableApplication: Response VulnerableApplication->>Attacker: Response Note left of
VulnerableApplication: Include response<br>from the<br>TargetedApplication
```

Draw.io schema XML code for the "case 1 for network layer protection about flows that we want to prevent" schema (printscreen are used to capture PNG image inserted into this cheat sheet).

Security Best Practices Guidelines for Session Management

Best Practices for Session Management

Ensure session data is encrypted: Encryption of session data should be used to ensure that session data is kept secure from interception or alterration.

Rotate session IDs: Accessing a site should create a new session ID after each login or a specified period of time, such as every 30 minutes.

Use cookies only for session IDs: Cookies should be used only for session ID and not for long term storage of preferences or other data such as login details.

Restrict access to session ID variables: Access to session ID variables should be restricted to only the necessary paths and functions of the website.

Limit session timeout period: Session timeouts should be set to a reasonable duration, such as 20 minutes.

Invalidate sessions on logout: All active sessions should be invalidated as soon as the user logs out.

Provide warning for session expiration: Web applications should give the user a warning about the impending session expiration before it takes place.

Avoid storing sensitive data in the session: Sensitive data such as user passwords or credit card numbers should never be stored in the session.

Restrict session access to HTTPS: All access to session variables should be restricted to HTTPS to ensure data is encrypted.

Use logoff buttons to clear sessions: Logoff buttons should be used to clear a user's active session.

Security Best Practices Guidelines for Transport Layer Protection

Transport Layer Protection: Security Best Practices

Transport Layer Protection (TLS) is a protocol that helps protect data that travels over a network by providing strong encryption. In order to safely use TLS, organizations must follow a set of security best practices.

Use TLS Versions that are Supported: Verify that the TLS version used is supported by the organization’s systems and software. Older versions of TLS are vulnerable to attack, so they must be updated to more secure versions.

Use Strong Cipher Suites: Cipher suites are sets of secure algorithms used to secure the connection between two devices. Organizations should use strong cipher suites that offer high levels of encryption, to keep data safe.

Use TLS as the Primary Encryption Protocol: TLS should be used as the primary encryption protocol, unless certain systems require other encryption methods. Stronger encryption protocols, such as SSL/TLS, should be used where applicable.

Require HTTPS Connections: Organizations should require HTTPS connections by default, rather than allowing an insecure HTTP connection. This ensures that all traffic is encrypted and secure.

Disable Weak Protocols: Weak protocols such as TLS 1.0 and SSL 3.0 must be disabled from the server, as they pose a risk to the organization’s security.

Test TLS: Organizations should regularly test their TLS implementations to ensure that it is properly configured and secure.

Encrypt all Sensitive Data: All sensitive data should be encrypted using strong encryption protocols, such as AES or RSA.

Keep TLS Up to Date: Organizations should use latest versions of TLS and regularly update their TLS implementation to the most secure version.

Log TLS Traffic: Organizations should log all TLS traffic, so that any suspicious activity can be identified and investigated.

Implement Access Control: Access control should be implemented to ensure that only authorized users can access the TLS-protected data.

Security Best Practices Guidelines for Input Validation

Security Best Practices for Input Validation

Introduction

Input Validation is the process of ensuring that user-supplied data conforms to a set of predetermined rules or specifications. It is an important security measure used to prevent malicious data from being submitted to an application. The purpose of input validation is to protect applications from malicious or malformed data which can potentially be used to exploit the application.

Guidelines

1. **Validate any user input** - All user input should be validated to ensure that it conforms to predetermined rules or specifications. This includes user-supplied data from web forms, query strings, cookies, and AJAX requests.
2. **Strictly enforce data type, range, length and format** - All user data should be checked for its data type (string, date, integer, number, etc.), range (minimum/maximum value), length (short/long strings) and format (e.g. date format, phone number format).

3. **Perform data sanitization** - Any user-supplied data should be properly sanitized before it is used. This includes removing any unexpected or malicious code from the data.
4. **Ensure input consistency** - All user inputs should be consistently enforced across the application. This includes validating the same data types, range and format in all parts of the application.
5. **Log and monitor user input** - All user input should be logged and monitored to detect potential malicious activities. If any suspicious activity is detected, the application should take appropriate security measures.
6. **Implement input validation routines server-side** - Any input validation routines should be implemented server-side to ensure client-side security is not compromised.
7. **Adopt a “whitelisting” approach** - A “whitelisting” approach should be adopted when validating user input. This means keeping a list of allowed values and only allowing user input if it is in that list. Any data that is not in the list should be considered invalid.
8. **Test input validation routines** - All input validation routines should be thoroughly tested to ensure that they are working properly. This includes testing all input types and ranges, and verifying that the input sanitization is working properly.

Security Best Practices Guidelines for User Privacy Protection

Security Best Practices for User Privacy Protection

Secure Your Networks - Implement firewalls and configure access restrictions for all users, including administrators, on all systems. - Regularly update your anti-virus and anti-spyware software. - Use encryption technologies such as HTTPS, SFTP, and IPSec. - Review logs for suspicious activity.

Train Your Staff - Ensure all personnel understand the importance of user privacy and security. - Educate your staff on how to identify and handle phishing emails and other potential security threats. - Train them on the proper way to dispose of sensitive data.

Enforce Strong Password Rules - Require that each user has a unique password. - Require longer passwords and complex characters. - Require periodic password changes.

Limit Third-Party Access - Screen and monitor vendors and service providers. - Establish data access policies that limit access to sensitive data to only those who need it.

Protect Data - Protect user data at all times. - Implement data retention policies to limit the amount of data stored. - Regularly review backups to ensure data is secure.

Follow Laws and Regulations - Understand and comply with any laws and regulations that affect the handling of personal user data. - Implement policies for handling user data in accordance with any applicable laws and regulations.

Review Processes and Policies - Perform routine audits of your processes, policies, and systems to ensure they are secure. - Review user access policies to ensure only authorized personnel have access to sensitive user data.

Security Best Practices Guidelines for Cryptography

Cryptography Security Best Practices

Cryptography is a powerful tool for keeping data secure, but it needs to be properly managed, configured and applied to ensure proper protection.

1. Keep Your Software and Protocols Up-To-Date

Regularly update the cryptography software, protocols and algorithms to take advantage of the latest security advancements.

2. Avoid Weak Algorithms

Be sure to use robust cryptography algorithms. Industry best practices generally lean towards using SHA-2 or AES-256 because of their security and efficiency.

3. Keep Keys and Passwords Secure

Be sure to store keys and passwords in a secure location, such as a password manager or hardware security module, and tightly manage access to them.

4. Ensure Proper Key Management

Keys are an essential part of cryptography and must be properly managed, rotated and securely stored.

5. Restrict Access to Keys and Encrypted Data

Don't give unnecessary access to keys and encrypted data. Ensure the data is accessible only to those who absolutely need it.

6. Use Strong Random Number Generation

Random number generation is essential to secure cryptographic processes, so use strong algorithms and techniques to generate them.

7. Run Security Tests

Perform various security tests on a regular basis to check for weaknesses and vulnerabilities. Understand the vulnerabilities within the system and take steps to address them.

8. Monitor Logs

System logs are an important part of any security system and should be monitored to detect any suspicious activity.

9. Implement a Solution for Revoking Compromised Keys

If a key is compromised, it's essential to be able to revoke it quickly and without disruption to the system. Implement a solution to enable secure and rapid revocation.

Security Best Practices Guidelines for Secure Application Update

Secure Update of Cloud-based Mobile Application

The following best practices should be followed to ensure a secure update of cloud-based mobile applications:

Infrastructure Security:

- Use of secure telecommunications to ensure secure transmission of mobile applications over the internet.
- Use encrypted protocols to ensure secure communication between the mobile application and the cloud.
- Ensure physical security for the cloud infrastructure, including personnel and physical access restrictions.
- Use of virtual firewalls and other security measures to protect the cloud infrastructure.
- Utilize cloud-based tools to monitor and detect suspicious network activity.

Application Security:

- Carefully review and approve all mobile applications before deployment.
- Develop strong authentication procedures to ensure that only authorized personnel can access the applications and perform updates.
- Utilize network segmentation and application firewalls to further protect applications from unauthorized access.
- Use secure coding practices and code reviews to ensure that applications are free from vulnerabilities.
- Utilize secure patch management processes to ensure timely response to application vulnerabilities.

User Authentication:

- Use strong authentication mechanisms including multi-factor authentication for user identity management.
- Ensure user access control lists are regularly updated to ensure user roles and permissions are correct.
- Utilize encryption protocols for user accounts and data transmission.
- Implement authentication time-outs and specify minimum password length to maintain security.

Data Security:

- Use encryption to protect sensitive data stored in the cloud.
- Implement access control measures to limit who can view certain sensitive data.
- Regularly audit user accounts, data access, and activities to maintain data security.
- Conduct regular backups to ensure data is securely stored and can be retrieved in the event of an incident.

Monitoring and Audit:

- Regularly monitor cloud-based applications and activities to detect any suspicious behavior.
- Implement a log management system to track user activities and application changes.
- Utilize cloud auditing software to record and review user activity, application changes, and other activities.

Security Best Practices Guidelines for Secure Third-party Application

Guidelines for Secure Third-party Cloud-based Mobile Application

Introduction

Mobile applications that leverage third-party cloud-based services are becoming more prevalent in our digital world today. Cloud services are used to store data and enable many of the features our customers expect. As such, adequate security measures must be taken to protect user data. This document outlines the best practices that should be followed when developing a secure third-party cloud-based mobile application.

Security Best Practices

Use robust authentication and authorization

- Utilize strong authentication mechanisms such as multi-factor authentication or adaptive authentication to ensure the identity of users.
- Utilize authorization mechanisms such as role-based access control or permission-based access control to ensure the appropriate access to user data.

Ensure data encryption

- Use industry-standard encryption algorithms and encryption keys to protect user data.
- Ensure that all data in transit between the mobile application and its cloud-based service is encrypted.

Enforce secure coding practices

- Follow secure coding best practices such as the Open Web Application Security Project (OWASP) standards.
- Utilize secure software development tools such as static and dynamic code analysis tools.

Monitor and log activity

- Monitor and log all activity related to the mobile application and its cloud-based service.
- Any unauthorized activity should be flagged and handled appropriately.

Perform regular security assessments

- Regularly assess the security posture of the mobile application and its cloud-based service to ensure that it is in compliance with security best practices.
- Perform threat modeling and vulnerability scanning of the application and its cloud-based service.

Utilize secure development process

- Utilize a secure software development life cycle to ensure that new features and updates are secure before release.
- Utilize automated testing tools and manual testing to validate the security of the application and its feature updates.

Educate and train staff

- Educate and train staff on secure coding best practices, secure coding development process, and general security best practices.

Conclusion

Following these security best practices will help ensure that user data is secure when using a third-party cloud-

Final Security Mechanisms Report

Mobile Platform	Hybrid Application
Application domain type	Entertainment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	MySQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Remote Storage (Cloud Database)
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	HTML5 + CSS + JavaScript
Input Forms	Yes
Upload Files	Yes
The system has logs	Yes
The system has regular updates	Yes
The system has third-party	Yes
System Cloud Environments	Hybrid Cloud
Hardware Specification	No
HW Authentication	Basic Authentication (user/pass)
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ;

Security Backup Mechanisms

Security backup mechanisms in cloud-based mobile apps refer to the measures taken to protect the data stored in the cloud and the mobile app itself. These measures can include having strong access control and authentication protocols, encrypted storage of data, secure data replication, server maintenance, regular software updates to protect against vulnerabilities, and the deployment of anti-malware and intrusion detection systems. Additionally, app developers should make use of mobile device management systems to prevent unauthorized access to the mobile app, and use secure protocols such as SSL/TLS to transmit data securely. It is also important to perform regular backups of the mobile app and its data, as well as to regularly inspect of the app's source code for any suspicious activities.

Backup Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Data Storage	iOS	iCloud	iCloud is Apple's cloud storage and backup system. It allows users to store music, photos, messages and more on the cloud, and access them from any device.	Application
Data Protection	Android	Google Drive	Google Drive is a cloud storage system provided by Google. It allows users to store documents and photos on their private Google Drive account, which can then be accessed from any device.	Application
Data Redundancy	Any	Version Control System	A version control system is a software system used to track and store files and any changes to them over time. This allows users to view previous versions of their code, which can help protect them in case of an accidental data loss.	Application

Security Audit Mechanisms

A security audit is a formal assessment of the security measures applied to a cloud-based mobile app, designed to identify any potential vulnerabilities or weaknesses. Security audits typically involve a review of the system architecture, including the application code, server configurations, coding practices and the underlying operating system. These audits recommend changes to improve the security posture and typically report on compliance with regulatory, industry or other security standards. Security audit mechanisms commonly include the following:

Source Code Review: A critical aspect of any security audit, a source code review will analyze the code for security vulnerabilities and make recommendations for improvements.

Penetration Testing: A common testing method for cloud mobile apps, penetration testing is designed to validate the security of the app by simulating a real-world attack.

System Log Reviews: A system log review allows security auditors to review the application's system logs for events such as suspicious logins, failed authentication attempts, and other suspicious activities.

Security Scans: A security scan is used to assess the external characteristics of the cloud-based mobile app, such as its ports, services, and file system.

Network Traffic Capturing: Auditors will analyze the communication between the app and its remote services, such as web servers, databases, and other external systems, as well as communication with the device itself.

User Behavioral Analysis: A review of user activity on the app will provide an additional layer of security and help ensure users are following best practices.

Mobile Device Management (MDM) Reviews: MDM reviews will assess configuration settings, such as device security and app usage policies,for potential threats or vulnerabilities.

Audit Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	iOS	Two-Factor-Auth.	Two-factor authentication provides an extra layer of security where the user has to provide two forms of identity verification to access the system. These forms are usually a password and an object, such as a token or OTP.	7
Access Control	Android	Fine-Grained Access	Fine-grained access control allows for advanced control of user access to certain resources, allowing for specific levels of permissions to be assigned to users.	6
Data Encryption	iOS	SSL/TLS Encryption	SSL/TLS encryption is a type of encryption that is used to secure data transmission over networks, ensuring that data is not readable by unauthorized individuals.	5
Authorization	Android	Access Control Lists	Access control lists are a type of authorization mechanism that assign privileges based on user roles and access levels. This allows for more granular control over access to certain resources within the system.	6
Intrusion Detection	iOS	Log Analysis	Log analysis is a method of intrusion detection that uses log data to detect suspicious activity on the system. Statistics and patterns are analyzed to detect anomalies that may signal a potential attack.	7
Vulnerability Scanning	Android	Static Analysis	Static analysis is a type of vulnerability scanning which involves looking at the code of an application and its respective libraries to identify potential vulnerabilities.	5

Cryptographic Algorithms Mechanisms

Cryptographic algorithms and mechanisms are used to secure cloud-based mobile applications to ensure data confidentiality and integrity, while maintaining user privacy. Cryptographic algorithms are mathematical functions used to encrypt and decrypt data, while mechanisms are protocols used to maintain secure communication. Common encryption algorithms used in cloud-based mobile apps include Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES), RSA, and Elliptic Curve Cryptography (ECC). Meanwhile, common cryptographic mechanisms used in cloud-based mobile apps include Transport Layer Security (TLS), IPsec, and SSH.

The use of cryptographic algorithms and mechanisms in cloud-based mobile apps helps to protect data from interception, modification, or deletion by malicious or unauthorized actors. This helps to keep user data and communications secure, while allowing users access across multiple devices and platforms with minimal effort.

Cryptographic Algorithms Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Confidentiality	Android	AES	Advanced Encryption Standard	Presentation
Integrity	iOS	SHA-256	Secure Hash Algorithm	Application
Authentication & Authorization	Android & iOS	RSA	Rivest, Shamir, Adleman	Network

Biometric authentication mechanisms are security measures that use biometric data to identify and authenticate a user's identity. Examples of biometric authentication mechanisms used in cloud-based mobile apps includes facial recognition, fingerprint scans, voice recognition and iris scans. These measures are used to verify users and are based on unique and identifiable physical attributes. They are increasingly being used in mobile apps due to the convenience they offer and the level of security they provide.

Biometric Authentication Mechanisms Examples:

Security Requirement	Mobile Plataform	Mechanism	Description	OSI Layer
Coding	Android	Fingerprint Recognition	Scans fingerprints	Application
		Vein Scanning	Scans veins in the hand	Presentation
Runtime	iOS	Voice Matching	Compares user's voice	Session
		Iris Scanning	Scans the eye's iris	Transport
		Facial Recognition	Verifies user's face	Network

Factors-based authentication mechanisms in cloud-based mobile apps are a security measure used to authenticate the identity of a user before granting access to confidential data and systems hosted in the cloud. They are built on the concept of using multiple layers of authentication to verify the user's identity and grant access. These factors can include something the user knows (such as a password), something the user has (such as a token or key fob), or something the user is (such as biometric data). This concept is especially important in today's mobile cloud environment as it provides an additional layer of security and helps to ensure that only those users with sufficient authentication can access confidential data and systems.

Factors-based Authentication Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer	Pre-coding/ Runtime
Authentication	Android	Secret Key	Hard to guess, stored securely	Application	Pre-coding
Authentication	iOS	Password/PIN	Use 4 digits to unlock the demo	Application	Pre-coding
Authorization	Android	Touch ID	Fingerprint based authentication	Application	Pre-coding/ Runtime
Authorization	iOS	Face ID	Facial recognition login system	Application	Pre-coding/Runtime#

ID-based Authentication Mechanisms

ID-based authentication mechanisms are technologies used in cloud-based mobile apps to verify the identity of a user by requiring the user to provide a unique ID, such as a username or email address, and a password. The system will then compare the provided information with a user record in a secure database to verify the user's identity. The verification process allows the system to identify users securely even when they are accessing the app remotely, helping to maintain secure access to the cloud-based mobile app.

ID-based Authentication Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	iOS	Passcode Unlock	User enters a predefined passcode	Application
		Biometrics	User authenticates via their own identity	Application
	Android	SMS-OTP	User receives an One-Time Password via SMS	Application
Authorization	iOS	Role-Based Access	A user's access is limited to predefined roles	Application
	Android	Policy-Based Access	User access is limited by network and security policies	Application
Coding	iOS	Data Encryption	Encryption of data stored in coding phase	Application
	Android	App Signing	Digital Signature of App in runtime	Network
Runtime	iOS	Token-based Access	User access is granted for a limited time via token	Application

	Android	Network Access Control	Network access is limited by predefined security policies	Network

Cryptographic Protocols Authentication Mechanisms

Cryptographic protocols and mechanisms used in cloud-based mobile apps are security protocols and algorithms employed to protect data stored in the cloud and to provide authentication of users and service providers. Common protocols and mechanisms include Transport Layer Security (TLS), public key infrastructure (PKI), encryption techniques such as Secure Socket Layer (SSL), Advanced Encryption Standard (AES), and Elliptic Curve Cryptography (ECC), along with the use of digital signatures and hashing techniques.

Cryptographic Protocols Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Data Confidentiality	iOS	TLS	Provides end-to-end encryption for data transmissions between mobile devices and servers	Layer 5
	Android	TLS	Provides end-to-end encryption for data transmissions between mobile devices and servers	Layer 5
Authentication & Authorization	iOS	OAuth 2.0	Enables authorization of mobile device requests based on user identity	Layer 5
	Android	OAuth 2.0	Enables authorization of mobile device requests based on user identity	Layer 5
Data Integrity	iOS	HMAC	Encrypts data using a shared secret for message authentication code generation	Layer 5
	Android	HMAC	Encrypts data using a shared secret for message authentication code generation	Layer 5

Access Control Mechanisms

Security access control mechanisms for cloud-based mobile apps refer to the policies and procedures that are used to protect and monitor the use of a mobile application. This includes the implementation of processes like user authentication, authorization, and encryption as well as monitoring user activities to prevent unauthorized access or changes to data or functions. Security access control also includes controlling physical access to the cloud servers and managing access to data or applications stored on the cloud.

Access Control Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	iOS	Trusted Network Connect (TNC)	TNC is a Network Access Control protocol that provides a secure authentication with mutual authentication of the user and the system.	Layer 2
Authorization	Android	Entitlement Management	Entitlement management uses digital identities, tokens and attributes to provide roles-based access control. It is a method of controlling user access to system resources by examining an identity to determine its access rights.	Layer 6
Encryption	iOS & Android	TLS/SSL	TLS/SSL is a standard security protocol used for establishing an encrypted communication channel between two machines. It is commonly used to provide secure communication over the internet.	Layers 4 & 7

Security Inspection Mechanisms

Security inspection mechanisms for cloud-based mobile apps involve analyzing and monitoring various aspects of the application to detect potential security vulnerabilities or threats. Here are some common security inspection mechanisms:

1. Vulnerability Scanning: Automated tools scan the application for known vulnerabilities in the code or configuration.
2. Penetration Testing: Ethical hackers attempt to exploit vulnerabilities in the application to identify weaknesses in the security.
3. Code Analysis: Analyzing the code for potential security flaws or vulnerabilities, such as buffer overflows or SQL injection.
4. Security Configuration Review: Evaluating the configuration of the application and associated infrastructure to ensure that security best practices are being followed.
5. Threat Modeling: Identifying potential threats and risks to the application and designing appropriate security controls to mitigate them.
6. Log Analysis: Analyzing application logs to detect suspicious behavior or potential security incidents.
7. Network Traffic Analysis: Analyzing network traffic to detect potential attacks or vulnerabilities in the network.

These mechanisms can be used to identify and remediate potential security vulnerabilities and help ensure that the application is secure. It's important to regularly perform these inspections to stay ahead of emerging threats and vulnerabilities. Additionally, it's important to involve security experts and conduct regular security audits to ensure that the application is meeting security standards and regulations.

Security Inspection Mechanisms Examples

Security Requirement	Mechanism	Description	OSI Model Layer
Access Control	Authentication Testing	Testing the effectiveness of authentication mechanisms in place to ensure access control.	Layer 7 (Application Layer)
	Authorization Testing	Testing the effectiveness of authorization mechanisms in place to ensure access control.	Layer 7 (Application Layer)
Data Protection	Data Encryption Testing	Testing the effectiveness of encryption mechanisms in place to protect sensitive data.	Layer 4-7 (Transport and Application Layers)
	Data Backup Testing	Testing the effectiveness of backup and recovery mechanisms in place to protect against data loss.	Layer 2-4 (Data Link, Network, and Transport Layers)
Network Security	Network Scanning	Scanning the network for vulnerabilities and potential security threats.	Layer 2-4 (Data Link, Network, and Transport Layers)
	Firewall Configuration Review	Reviewing the firewall configuration to ensure that it is properly configured to protect the network.	Layer 3-4 (Network Layer)
Security Monitoring	Log Analysis	Analyzing logs to detect potential security incidents or threats.	Layer 7 (Application Layer)
	Intrusion Detection and Prevention System (IDPS)	Monitoring network traffic to detect and prevent potential attacks.	Layer 2-4 (Data Link, Network, and Transport Layers)
Security Configuration	Configuration Review	Reviewing the configuration of the application and associated infrastructure to ensure that security best practices are being followed.	Layer 3-4 (Network Layer)
Threat Management	Penetration Testing	Ethical hacking to identify potential security vulnerabilities and weaknesses in the application.	Layer 7 (Application Layer)
	Vulnerability Scanning	Scanning the application for known vulnerabilities in the code or configuration.	Layer 7 (Application Layer)
Code Security	Code Analysis	Analyzing the code for potential security flaws or vulnerabilities, such as buffer overflows or SQL injection.	Layer 7 (Application Layer)

Security Logging Mechanisms for Cloud-Based Mobile Apps

Security logging mechanisms for cloud-based mobile apps involve capturing and recording various events and activities within the application to help identify and investigate security-related incidents. Here are some common security logging mechanisms:

1. Login Logs: Records user login attempts, including the username, date, time, and source IP address.
2. Access Logs: Tracks which users are accessing which resources, including the date, time, and type of resource accessed.
3. Key Management Logs: Records the creation, modification, and destruction of encryption keys and certificates used to protect data.
4. Data Access Logs: Logs all access to sensitive data, including the user accessing the data, the date and time of access, and the action taken.
5. Network Logs: Captures network traffic, including the source and destination IP addresses, ports, and protocols, for analysis and troubleshooting.
6. Application Logs: Logs application-specific events and activities, including errors, warnings, and security-related events.
7. Security Incident Logs: Records security incidents, including the type of incident, date and time, and the steps taken to respond and remediate the incident.
8. Compliance Audit Logs: Captures all activity related to compliance audits, including the auditor, date and time, and results of the audit.

These logs can be used to identify and investigate security incidents, monitor compliance with regulations and policies, and provide evidence in the event of a breach or other security-related event. It's important to ensure that the logs are properly secured and encrypted to prevent unauthorized access or tampering.

Additionally, logs should be regularly reviewed and analyzed to detect any potential security issues.

Security Logging Mechanisms Examples

Security Requirement	Mechanism	Description	OSI Model Layer
Authentication	Login Logs	Records user login attempts, including the username, date, time, and source IP address.	Layer 7: Application Layer
Authorization	Access Logs	Tracks which users are accessing which resources, including the date, time, and type of resource accessed.	Layer 7: Application Layer
Data Encryption	Key Management Logs	Records the creation, modification, and destruction of encryption keys and certificates used to protect data.	Layer 7: Application Layer
Data Protection	Data Access Logs	Logs all access to sensitive data, including the user accessing the data, the date and time of access, and the action taken.	Layer 7: Application Layer
Network Security	Network Logs	Captures network traffic, including the source and destination IP addresses, ports, and protocols, for analysis and troubleshooting.	Layer 3: Network Layer
Application Security	Application Logs	Logs application-specific events and activities, including errors, warnings, and security-related events.	Layer 7: Application Layer
Incident Response	Security Incident Logs	Records security incidents, including the type of incident, date and time, and the steps taken to respond and remediate the incident.	Layer 7: Application Layer
Compliance	Compliance Audit Logs	Captures all activity related to compliance audits, including the auditor, date and time, and results of the audit.	Layer 7: Application Layer

Confinement Mechanisms

Security confinement mechanisms in cloud-based mobile apps are measures taken to protect cloud-based data and applications from unauthorized access and misuse. These mechanisms can include authentication and authorization protocols, encryption, data audit logging, as well as data masking, tokenization and sanitizing techniques. Additionally, security policies such as least privilege and least functionality can be implemented to mitigate risks associated with cloud-based solutions.

Confinement Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	Android	2-Factor Authentication	A combination of two different factors to prove the user's identity	Application
Access control	Android	Permission Model	Granular control over system resources based on user roles	Application
Data Confidentiality	iOS	Encription	A technique used for securing data by translating it into an unreadable format	Network
Data Integrity	iOS	Hashing	A cryptographic process of generating a unique value from input data to ensure integrity of the original data	Network

Filtering Mechanism Mechanisms

Security filtering mechanisms in cloud-based mobile apps are techniques used to ensure the privacy and security of user data, as well as to protect the app from malicious attacks. Such mechanisms can include authentication protocols, encryption techniques, data validation, virus and malware scanning, access control measures, and log analysis. These security filtering techniques are designed to protect user data, ensuring that only authorized users can access sensitive information. They can also help protect the app from possible attacks, such as distributed denial of service (DDoS) attacks, and limit unauthorized data access or information leakage.

Filtering Mechanism Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer

Authentication	Android	OAuth	A secure open protocol to allow user authentication on mobile apps	Application
Confidentiality	iOS	SSL/TLS	Uses cryptography to create a secure channel for data transmission and keeps data private	Transport
Integrity	Android	HMAC	A cryptographic protocol that uses keyed-hashing for message authentication, verifying the data integrity and authenticity of a message	Application
Availability	iOS	Database Replication	Keeping the databases synchronized by sharing information between the server and mobile apps, through the internet.	Data Link

Final Attack Models Report

Mobile Platform	Hybrid Application
Application domain type	Entertainment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	MySQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Remote Storage (Cloud Database)
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	HTML5 + CSS + JavaScript
Input Forms	Yes
Upload Files	Yes
The system has logs	Yes
The system has regular updates	Yes
The system has third-party	Yes
System Cloud Environments	Hybrid Cloud
Hardware Specification	No
HW Authentication	Basic Authentication (user/pass)
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ;

Man-in-th-Middle Attack

Man-in-the-Middle (MitM) attack is a type of cyber security attack where an attacker intercepts the communication between two parties in order to gain access to confidential information. It is also known as a bucket-brigade attack, monkey-in-the-middle attack, or ‘eavesdropping’ attack.

Man-in-the-Middle (MitM) attacks are typically used by attackers in order to gain access to usernames, passwords, and other sensitive information. The attacker can also modify or inject malicious code into the communication, allowing them to take control of a computer or system without the permission or knowledge of either party.

MitM attacks are usually difficult to detect as the attacker is usually positioned between two parties, making it difficult for either party to recognize the existence of an attack. Furthermore, MitM attacks can also be used to plant malware on systems, which can be used to gain access to even more sensitive information.

Man-in-th-Middle Architectural Risk Analysis:

Arquitectural Risk Analysis: Man-in-the-Middle Vulnerability (CVSS v3.1)

Base Score: 7.3

Attack Vector: Network (N)

Attack Complexity: Low (L)

Privileges Required: None (N)

User Interaction: None (N)

Scope: Changed (C)

Confidentiality Impact: High (H)

Integrity Impact: High (H)

Availability Impact: High (H)

Environmental Score: 8.4

Confidentiality Requirement: High (H)

Integrity Requirement: High (H)

Availability Requirement: High (H)

Overall Score: 8.4

Man-in-the-middle is an attack in which a malicious actor is able to intercept and manipulate network traffic between two legitimate users. While it does not technically require credentials, it does require carefully timing of the attack and the ability to perform an active man-in-the-middle attack on a secure connection. As such, it has a high potential to cause major impacts on confidentiality, integrity, and availability, giving it an Overall CVSS Score of 8.4.

Goal: To access on communication channel between two endpoints, and to manipulate their messages

OR
|- 1. Spoofing-based MITM Attack
| **OR**
| |- 1. ARP Spoofing-Based MITM Attack
| | **AND**
| | |- 1. Eve sends an ARP Reply message to Alice (which says that IP: 10.0.0.3 has MAC address: EE:EE:EE:EE:EE:EE. This message will update Alice's ARP table)
| | |- 2. Eve also sends an ARP Reply message to Bob, which says that IP: 10.0.0.2 has MAC address: EE:EE:EE:EE:EE:EE. This message will update Bob's ARP table
| | |- 3. When Alice wants to send a message to Bob, it will go to Eve's MAC address EE:EE:EE:EE:EE:EE, instead of Bob's BB:BB:BB:BB:BB:BB
| | |- 4. When Bob wants to sends a message to Alice, it will also go to Eve.
| |- 2. DNS Spoofing-Based MITM Attack
| | **OR**
| | |- 1. Hijacking or sniffing packets in the process of query response
| | |- 2. Cache poisoning through the birthday attack
| | |- 3. Hacking on authorised DNS
| |- 3. DHCP Spoofing-Based MITM Attack
| | **OR**
| | |- 1. Starvation Attack
| | |- 2. Rogue DHCP Server Attack
| |- 4. IP Spoofing-Based MITM Attack
| | **OR**
| | |- 1. Blind and Non-Blind spoofing
| | |- 2. ICMP spoofing
| | |- 3. TCP Sequence-Number prediction
|- 2. SSL/TLS MITM Attack
| **OR**
| |- 1. MITM+certificate
| |- 2. MITM+key
|- 3. Border Gateway Protocol MITM Attack
|- 4. FBS-BASED MITM Attack
| **OR**
| |- 1. MITM Attack on GSM
| | **OR**
| | |- 1. Attacker sets-upconnection between False BTS and Legitimate MS.
| | |- 2. False MSimpersonates the victim's MS to the real network by resending the identityinformation, which was received from the step 1
| | |- 3. Victim's MSsends its authentication information and cipher-suites to the False BTS
| | |- 4. Attackerforwards message from step 3 to the Legitimate BTS
| |- 2. MITM on Combined GSM/UTMS Network
| | **AND**
| | |- 1. Attacker launches 3G/4G/5G False BTS and catches IMSI of the victim's MS
| | |- 2. Attacker uses victim's IMSI to impersonate victim's MS to False MS
| | |- 3. Attacker starts 2G/4G/5G False BTS and forces victim to connect

Brute Force Attack

A brute force attack is a method used by attackers to gain access to user accounts or networks by systematically trying different combinations of usernames and passwords until the correct combination is discovered. This technique is used by attackers to gain access to various online accounts or networks without having to know the account password. Brute force attacks are conducted using automated scripts that try to guess the username and password until the correct combination is discovered. Attackers may also use tools such as dictionary files or specialized hardware to speed up the process of guessing passwords.

Brute Force Architectural Risk Analysis:

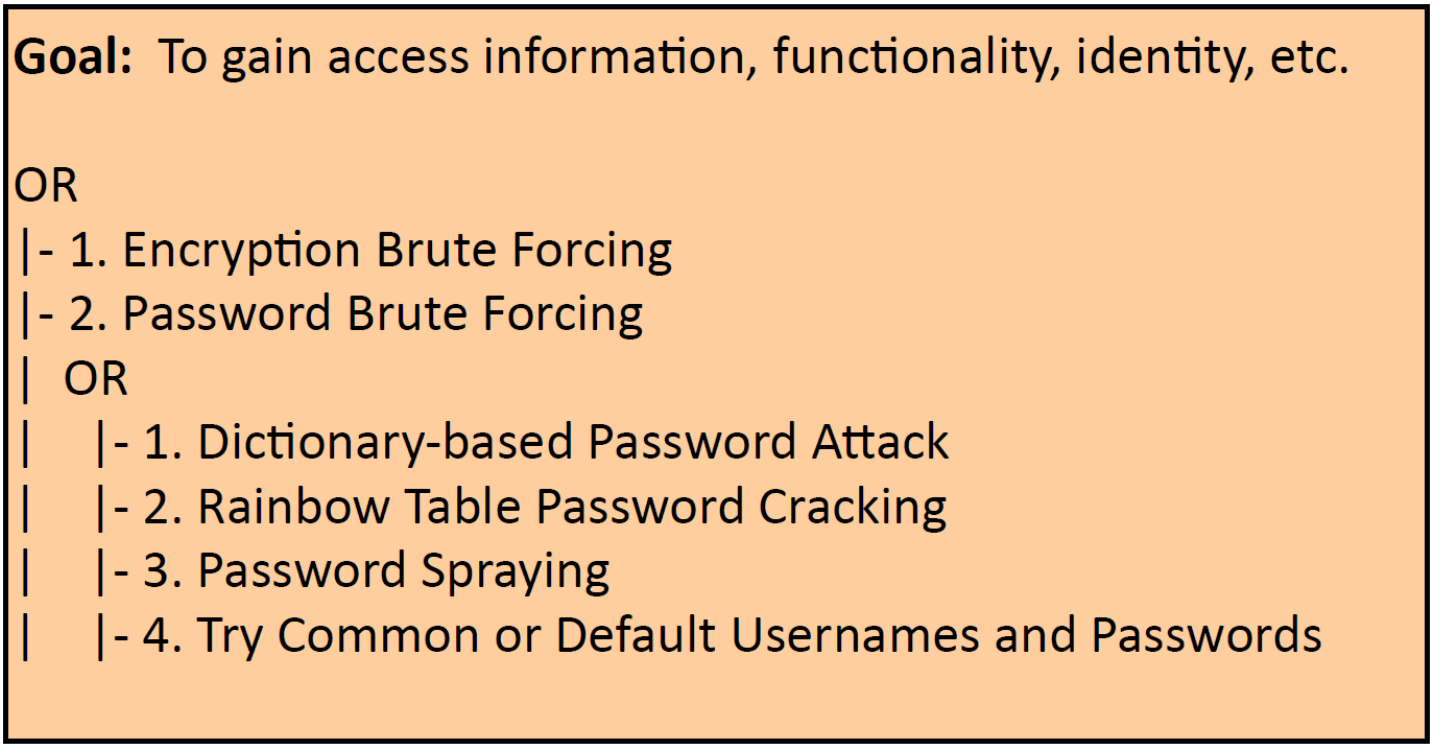
Arquitectural Risk Analysis of Brute Force Vulnerability

According to the *Common Vulnerability Scoring System (CVSS) v3.1*, the threat vectors for Brute Force Vulnerability are:

Attack Vector (AV): Network (N)
Attack Complexity (AC): Low (L)
Privileges Required (PR): None (N)
User Interaction (UI): None (N)
Scope (S): Changed (C)
Confidentiality (C): None (N)
Integrity (I): None (N)
Availability (A): None (N)

Given the components above, the resulting vulnerability score of Brute Force Vulnerability is 2.8 (Low).

The risk that this vulnerability entails is the possibility of an attacker to use brute-force techniques to guess user credentials, which can then be used to gain unauthorized access to a system. As such, mitigating this risk depends on the implementation of strong authentication methods to prevent brute-force attacks. Additionally, system administrators should regularly monitor for failed authentication attempts so that potential attacks can be identified and addressed immediately.



Eavesdropping Attack

Eavesdropping (also known as 'passive listening') is a type of network attack where an unauthorized individual uses various techniques to gain access to confidential information or data as it travels across a network.

Eavesdropping attacks usually occur over wireless networks and can be used to intercept emails, instant messages and other sensitive data, as well as access private networks and computers. To protect against such attacks, individuals and organizations should take steps to secure their networks with strong authentication, encryption, and other security measures.

Eavesdropping Architectural Risk Analysis:

Architectural Risk Analysis of Eavesdropping Vulnerability

Common Vulnerability Scoring System (CVSS) v3.1 Rating:

	CVSS:3.1
Base Score	6.1
Attack Vector	Network (N)
Attack Complexity	Low (L)
Privileges Required	None (N)
User Interaction	None (N)
Scope	Unchanged (U)
Confidentiality	High (H)
Integrity	None (N)
Availability	None (N)

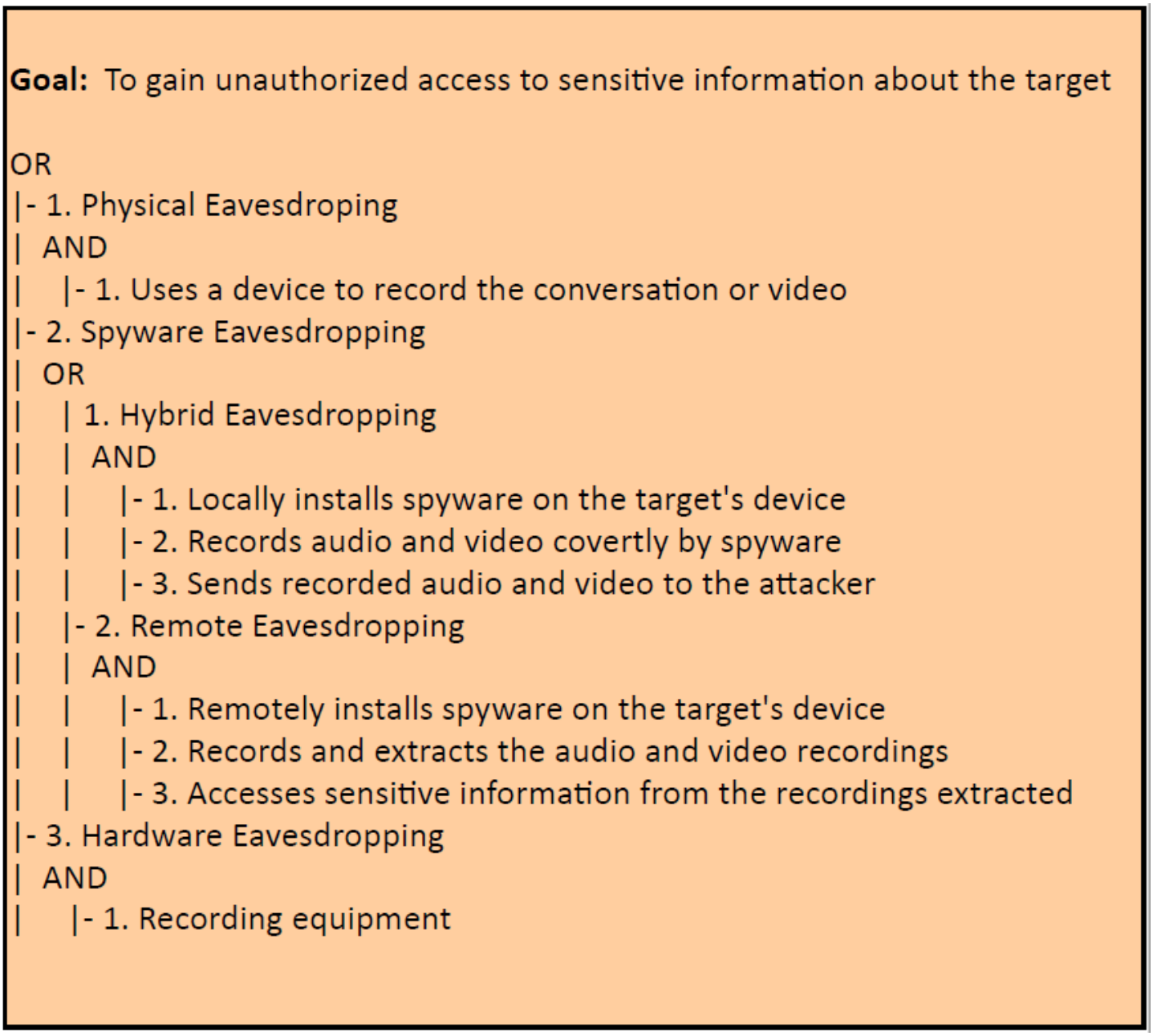
Eavesdropping Vulnerability Description:
Eavesdropping is a type of malicious attack which involves the interception of electronic communications by an unauthorized party. This type of attack is generally aimed at people or organizations, with the intent of gathering confidential information such as intellectual property, passwords, usernames, authentication tokens, and other sensitive data.

Impact of Vulnerability

Eavesdropping is a highly invasive attack that can have serious consequences, including the loss of confidential information. Depending on the nature of the data, an attacker can gain access to private or confidential information and use it to gain an advantage over the original sender. Eavesdropping attacks can also be used to disrupt the normal operations of an organization, or to launch other types of attacks. In addition, an attacker may be able to gain access to privileged user accounts or sensitive systems, leading to further consequences.

Recommendations

Organizations should implement measures to protect against eavesdropping attacks. Appropriate security protocols should be established and implemented to detect and stop unauthorized access to confidential information. Access control methods such as two-factor authentication (2FA) should be used to verify the identity of remote users. Encryption should also be used to secure data in transit. Furthermore, organizations should ensure that their employees are aware of the risks associated with eavesdropping and ensure that appropriate measures are taken to prevent any unauthorized access to confidential data.



XSS Attack

XSS, or Cross-Site Scripting, is a type of attack that exploits vulnerabilities in web application code to inject malicious code into web pages. XSS attacks use malicious JavaScript, VBScript, HTML, or other client-side code to steal user data and gain control over the user's web experience. XSS attacks can also be used to redirect users to malicious sites, hijack user sessions, and alter page content.

XSS Architectural Risk Analysis:

Arquitectural Risk Analysis - XSS Vulnerability

Rating based on Common Vulnerability Scoring System / Version 3.1 (CVSS:3.1)

- **Base Score**: 6.1
- **Impact Score**: 3.8
- **Exploitability Score**: 8.8

Vector String

`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:N`

Impact Subscore

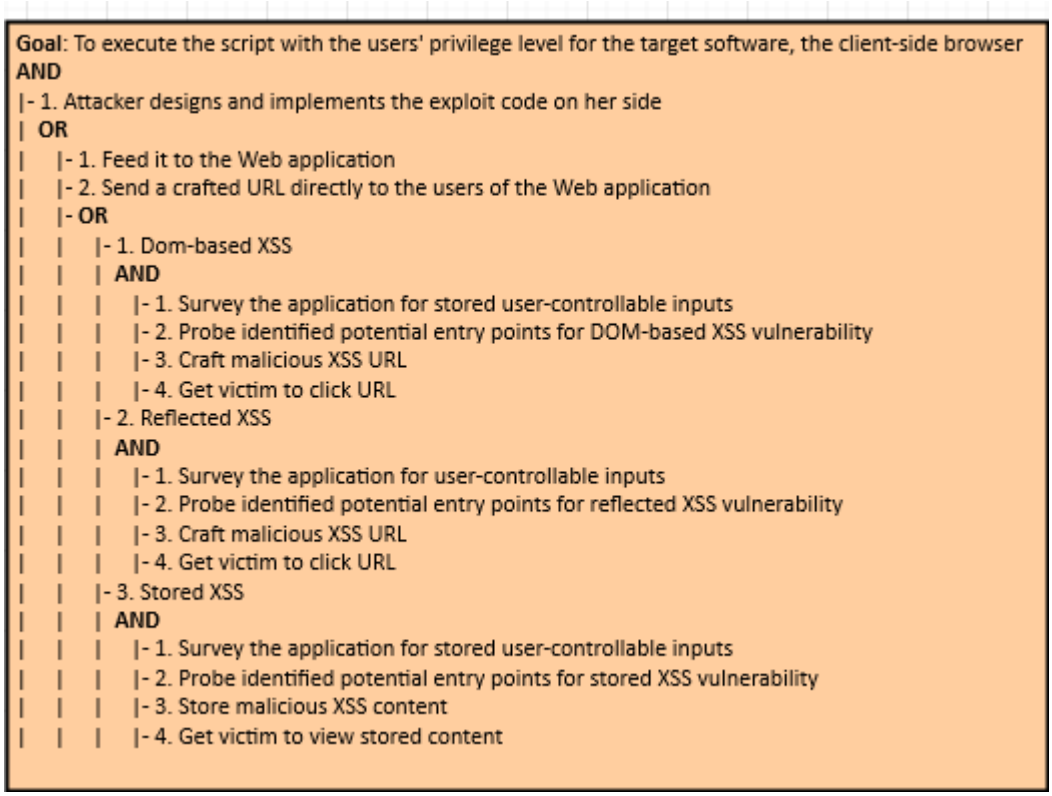
- **Availability Impact**: None (N)
- **Confidentiality Impact**: High (H)
- **Integrity Impact**: High (H)

Exploitability Subscore

- **Attack Vector**: Network (AV:N)
- **Attack Complexity**: Low (AC:L)
- **Privileges Required**: None (PR:N)
- **User Interaction**: Required (UI:R)
- **Scope**: Changed (S:C)

References:

- Common Vulnerability Scoring System / Version 3.1 (CVSS:3.1)



CSRF Arquitectural Risk Analysis:

Architectural Risk Analysis of CSRF Vulnerability

Description

Cross-Site Request Forgery (CSRF) is a type of attack which occurs when a malicious website, email, or message causes the victim's web browser to perform an unwanted action on a trusted site. It means that the malicious website can trick the user's browser into sending a malicious request to the vulnerable site without the user's knowledge.

Scoring

Category	Metric	Value
Exploitability	Attack Vector	Network
Exploitability	Attack Complexity	Low
Exploitability	Privileges Required	None
Exploitability	User Interaction	Required
Impact	Scope	Changed
Impact	Confidentiality	Low
Impact	Integrity	Low
Impact	Availability	None

Risk Score

A risk score of 2.6 (from 0.0 to 10.0) is calculated according to the Common Vulnerability Scoring System v3.1.

Mitigation Strategies

Strict measures should be taken to ensure that the web application is not vulnerable to the CSRF attack. The following approaches can be taken for protecting against CSRF attack:

1. Implement Cross-Site Request Forgery tokens.
2. Use Same-Site Cookies.
3. Use Captcha.
4. Set HTTP headers like X-Frame-Options, X-XSS-Protection and Content-Security-Policy.
5. Implement Post/Redirect/Get design to all requests.
6. Use one-time tokens while sending forms.
7. Use JavaScript token validation.
8. Validate each request on server side.

CSRF Arquitectural Risk Analysis:

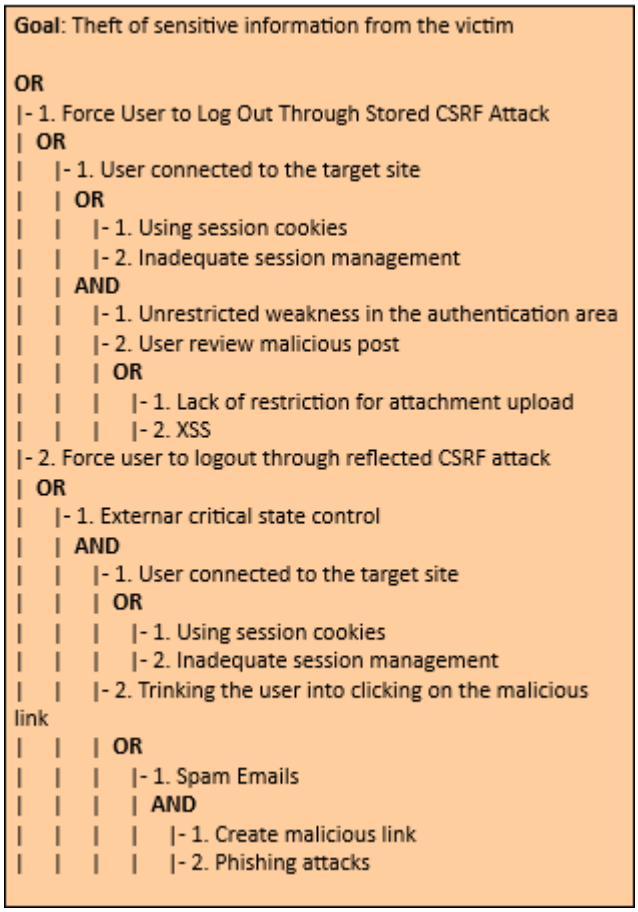
Arquitectural Risk Analysis for CSRF Vulnerability

In accordance with Common Vulnerability Scoring System (CVSS) v3.1, the arquitectural risk analysis of cross site request forgery (CSRF) vulnerability will focus on the following metrics:

Metric	Score
Attack Vector	Network (AV:N)
Attack Complexity	Low (AC:L)
Privileges Required	None (PR:N)
User Interaction	None (UI:N)
Scope	Changed (S:C)
Confidentiality	None (C:N)
Integrity	None (I:N)
Availability	None (A:N)

This risk is classified as a **CVSS v3.1 Base Score of 6.4**, with a **High Vector Severity (VSR) of 6.4**. The high vector severity indicates that the vulnerability has a high potential for exploitation and should be addressed with an urgency.

This risk should be remediated immediately, as exploiting the vulnerability may result in malicious code execution, data manipulation/corruption, and other undesired system behaviors. It is recommended to implement mitigating technologies, such as input and output validation, content security policies, and request tokens to reduce the risk associated with this vulnerability.



CSRF Attack

Cookie Poisoning is a type of attack that an attacker uses to modify a web browser's cookie data. It is used to gain unauthorized access to a user's account, steal their personal information, or inject malicious code into a website.

This type of attack usually involves the attacker sending out malicious scripts that modify a user's cookie data. The attacker can then use the cookies to gain access to the user's personal information or inject malicious code into a website.

Cookie poisoning attacks can also be used to disrupt a website's functionality and lead to denial of service attacks.

Fortunately, cookie poisoning attacks can be mitigated with proper security measures. For example, organizations can use encryption and other types of data protection techniques to make cookies more secure. They can also use firewalls and other security controls to protect their networks from attackers. Additionally, web developers can build safer websites by using anti-CSRF tokens and other security measures.

Cookie Poisoning Architectural Risk Analysis:

CVSS v3.1 Base Score: 7.3 (High)

Details

Vector: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

Calculating Impact Subscore Impact Subscore: 6.8 (High)

Calculating Exploitability Subscore Exploitability Subscore: 8.6 (High)

Summary

Cookie Poisoning Vulnerability is rated as High severity according to Common Vulnerability Scoring System v3.1. It has an impact subscore of 6.8 and exploitability subscore of 8.6 respectively.

Goal: Steal User Personal Information

AND
|- 1. Change the stored IP address to the attacker's fake site IP address
|- 2. The user queries the DNS Server
|- 3. The DNS responds to the customer query with the no real IP address
|- 4. The user reach the attackers fake site

Malicious QR Code Attack

Malicious QR Code attacks involve the use of QR codes (Quick Response Code) that, when scanned by unsuspecting users, can launch malicious applications or websites that contain malware. These QR Codes can be found in emails, websites, ads, or printed material, and can be used by hackers to launch malicious code on user devices.

In order to protect against malicious QR code attacks, users should be aware of what kind of content they are accessing through their devices and avoid clicking on any suspicious links. Furthermore, if a user is unsure of a QR code's origin, it is important to scan the code using specialized software (e.g. antivirus and anti-malware programs) to ensure it is safe before interacting with it.

Malicious QR Code Architectural Risk Analysis:

Arquitectural Risk Analysis of Malicious QR Code Vulnerability

CVSS v3.1 Base Score: 8.8 (High)

Vector: AV:L/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

Impact Subscore: 6.6

Malicious QR codes are physical objects which contain malicious code which may execute on devices when scanned by a camera. The risk posed by malicious QR codes is that they may allow malicious code to be executed on a device without the user's knowledge or consent. Such malicious code could be a payload that contains a malicious program or script, which may execute malicious code or exfiltrate data from the device.

The Arquitectural Risk Analysis of this vulnerability consists of the following:

Attack Vector (AV): Local The malicious code is embedded in the physical QR code which must be scanned in order to execute.

Attack Complexity (AC): High Scanning the code and executing the malicious code requires physical access to the QR codes as well as considerable technical expertise.

Privileges Required (PR): None No user privileges are necessary for the malicious code to execute.

User Interaction (UI): None No user interaction is necessary to exploit the vulnerability.

Scope (S): Unchanged The malicious code remains confined to the device in which it was executed.

Confidentiality (C): High Data on the device may be stolen or altered if the malicious code is successful.

Integrity (I): High The malicious code could corrupt data on the device, or inject malicious code that could remain hidden after the malicious code is executed.

Availability (A): High The malicious code could disrupt the operation of the device, or cause the device to become unresponsive.

Goal: To steal sensitive personal information
AND
|- 1. 1. Create a malicious QR Code or modify QR Code
| **OR**
| |- 1. Attacking automated processes
| | **OR**
| | |- 1. Command and Code Injections
| | |- 2. Malware
| | |- 3. Fraud
| |- 2. Attacking human interactions
| | **OR**
| | |- 1. Phishing
| | |- 2. Fraud
| | |- 3. Social Engineering Attack
| | |- 4. 4. Attacking Reader Software

SQLi Attack

Malicious QR Code attack is a form of cyber attack that takes advantage of Quick Response (QR) codes to perpetrate malicious activities. This type of attack works by embedding malicious code into QR codes that lead unsuspecting victims to malicious websites or applications when scanned. These malicious QR codes can be found in physical locations, such as on printed ads, posters, or packaging, or they can be sent electronically through email, text, or social media. If a victim unwittingly scans the malicious QR code, the malicious code embedded within it will be executed and could result in malicious activities such as data harvesting, installation of malware, or other malicious activities. Victims of malicious QR code attacks should always exercise caution when interacting with QR codes as they could lead to malicious websites and activities.

Malicious SQLi Architectural Risk Analysis:

Arquitectural Risk Analysis of SQLi Vulnerability

Common Vulnerability Scoring System v3.1:

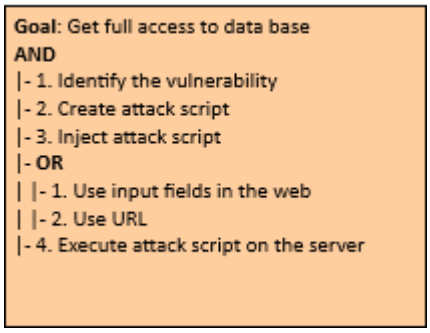
Metric	Score
Attack Vector	Network (AV:N)
Attack Complexity	Low (AC:L)
Privileges Required	None (PR:N)
User Interaction	None (UI:N)
Scope	Changed (S:C)
Confidentiality	High (C:H)
Integrity	High (I:H)
Availability	High (A:H)

Overall Score: 9.3 (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H)

SQLi Vulnerability presents a high risk arquitecture due to its potential negative consequences upoming from its exploitation, according to the Common Vulnerability Scoring System v3.1, the impact of this vulnerability is rated as a 9.3.

The Attack Vector score is Network (AV:N), meaning that the vulnerability exploit is deleivered through the network, such as Internet or an internal network, requiring no user interaction. The Attack Complexity score is Low (AC:L), implying that the vulnerability can be exlpoited with little effort and/or knowledge. The Privileges Required score is None (PR:N) so no privileged access is neccessary to exploit the vulnerability. The User Interaction score is None (UI:N), meaning that no user interaction is needed for the exploitation of this vulnerability. The Scope score is Changed (S:C), meaning that the risk affects other resources in the system due to the unauthorized access.

The Confidentiality score is High (C:H) because the attacker can find private information as a result of exploitation of SQLi Vulnerability, so providing access to confidential information could have a major impact. The Integrity score is High (I:H) because the attacker is able to modify or even delete private information which greatly impacts the integrity of the system. The Availability score is



Flooding Attack

What is Flooding attack?

A flooding attack is a type of denial of service (DoS) attack, in which the attacker seeks to overwhelm a targeted system or network through sending an excessive number of requests, either by using one machine or multiple computers. The intent is to render the target device or network incapable of responding to legitimate traffic, or to take up all of its available resources so that it can no longer provide normal service to its users. Flooding attacks can be achieved through a variety of methods, but they all revolve around sending large amounts of data to the target.

Flooding Architectural Risk Analysis:

Flooding Vulnerability

Category	Value	Vector String	Score
Attack Vector	Local	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	7.1
Attack Complexity	Low	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	0.7
Privileges Required	Low	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	0.7
User Interaction	None	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	0
Scope	Changed	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	1.7

Confidentiality Impact	High	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	5
Integrity Impact	High	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	5
Availability Impact	None	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	0
Total Score	**7.1**	AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N	**7.1**

The Flooding Vulnerability has been analyzed using the Common Vulnerability Scoring System v3.1, resulting in a total score of 7.1. This score indicates the level of risk associated with this vulnerability, as it is considered High. The breakdown of this score is as follows:

- Attack Vector: Local (7.1)
- Attack Complexity:

Goal: Stop the system or application availability
OR
|- 1. Launch Application Layer DoS/DDoS Attack
| **AND**
| |- 1. Compromise machine(s) to support in flooding via botnet
| |- 2. DOS/DDoS tool installation
| | **OR**
| | |- 1. Back
| | |- 2. RUDY
| | |- 3. SSL Flood
| | |- 4. HTTP Flood
| | |- 5. XML Flood
| | |- 6. DNS Amplification
| | |- 7. Voip Flooding
|- 2. Launch Protocol DDoS attack
| |- 1. Compromise machine(s) to support in flooding via botnet
| |- 2. DoS/DDoS tool installation
| | **OR**
| | |- 1. Land
| | |- 2. Neptune (TCP SYN flood)
| | |- 3. Ping of Death
| | |- 4. Teardrop
| | |- 5. TCP SYN-ACK flood
| | |- 6. ACK & PUSH ACK flood
| | |. 7. RST/FIN flood
| | |- 8. P2P
| | |- 9. BlueSmacking
|- 3. Launch Volume DDoS attack
| |- 1. Compromise machine(s) to support in flooding via botnet
| |- 2. DoS/DDoS tool installation
| | **OR**
| | |- 1. Smurf (ICMP Flood)
| | |- 2. Spoofed/non-spoofed UDP flood
| | |- 3. DNS flood
| | |- 4. VoIP Flood
| | |- 5. ICMP echo request
| | |- 6. Fraggle
| | |- 7. Amplification
| | |- 8. TCP Flood
| | |- 9. Mail Bomb
|- 4. Permanent DoS

Sniffing Attack

Sniffing attack is a type of cyber attack in which the attacker uses special software to intercept, monitor, and analyze data exchanged between computers on a network. A sniffer (also known as a network analyzer or protocol analyzer) can be used to collect, interpret, and save data exchanged over a network or a part of it for later analysis. Attacks of this type are especially dangerous since they can be used to intercept passwords, messages, or other sensitive data or to change, delete, or inject malicious data into a communication stream.

What Is a Sniffing Attack?

A sniffing attack is a type of cyber attack that uses special software to intercept, monitor, and analyze data exchanged between computers on a network. This type of attack is typically done with a tool known as a “sniffer” (or network analyzer, protocol analyzer). Sniffers can be used to collect, interpret, and save data that is being exchanged over a network (or a section of it) for later analysis.

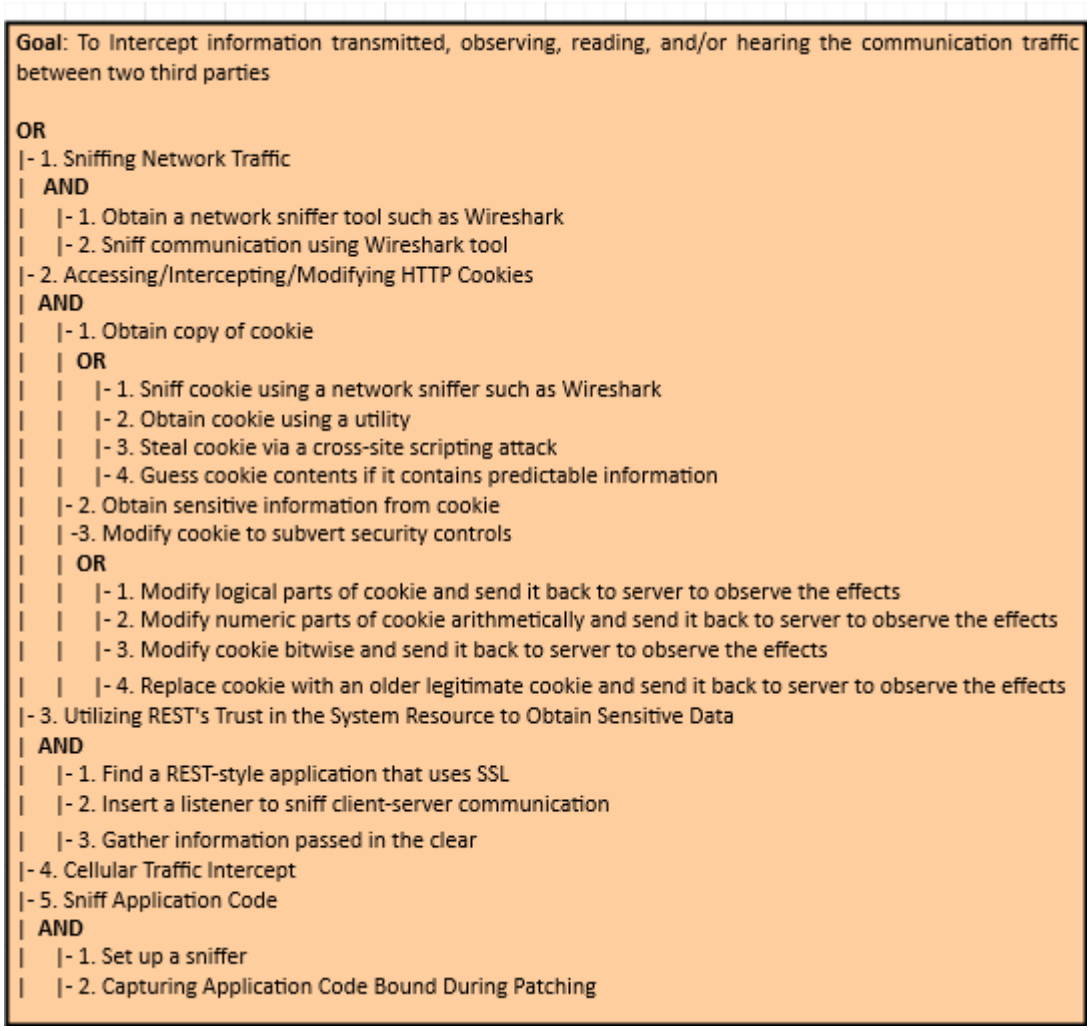
Sniffing attacks are especially dangerous because they can be used to intercept passwords, messages, or other sensitive data. Attackers may also use them to alter, delete, or inject malicious data into a communication stream.

Sniffing Architectural Risk Analysis:

Arquitectural Risk Analysis of Sniffing Vulnerability - Common Vulnerability Scoring System v3.1

Metric	Score
Attack Vector	Network (AV:N)
Attack Complexity	Low (AC:L)
Privileges Required	None (PR:N)
User Interaction	None (UI:N)
Scope	Unchanged (S:U)
Confidentiality	High (C:H)
Integrity	Low(I:L)
Availability	Low (A:L)
Score	5.4
Severity	Medium (CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:L)

Sniffing vulnerability has the potential to cause medium-level damage if exploited. Attackers can leverage this vulnerability to monitor communications between applications, find sensitive information like passwords, login credentials, social security numbers, etc. This can cause loss of confidentiality and integrity of data as well as disrupt availability of applications/services.



Phishing Attack

Phishing is a type of cyber attack in which attackers impersonate a legitimate entity or individual to gain access to sensitive personal or financial information of victims.

Phishing attacks may take the form of emails, texts, calls, or links to websites that prompt users to enter personal information, such as usernames, passwords, credit card numbers, etc. The attackers use this information for their own benefit to commit fraud or steal money from victims.

It is important to be aware of this threat and stay vigilant. If you suspect you may have been a victim of a phishing attack, it is important to take steps to protect yourself, such as changing your passwords, not clicking on suspicious links, and not providing sensitive information.

Phishing Architectural Risk Analysis:

Common Vulnerability Scoring System v3.1: Arquitectural Risk Analysis of Phishing Vulnerability

Attack Vector (AV): Network (N)

Attack Complexity (AC): Low (L)

Privileges Required (PR): None (N)

User Interaction (UI): Required (R)

Scope (S): Changed (C)

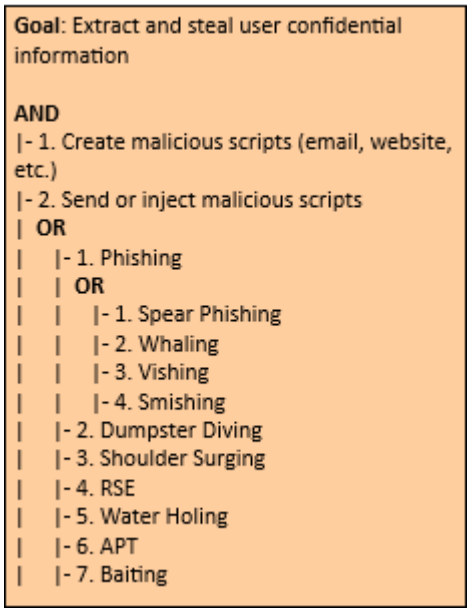
Confidentiality (C): High (H)

Integrity (I): High (H)

Availability (A): High (H)

Score (S): 7.8

Severity (Sev): High



Botnet Attack

Botnet attack is a type of cyberattack wherein a network of computer systems - collectively known as a botnet - are hacked and used to launch malicious activities like sending out spam, harvesting personal data, and launching distributed denial-of-service (DDoS) attacks. Botnet attacks are difficult to detect and counter due to the wide distribution of the malicious network.

What is a Botnet Attack?

A botnet attack is a type of cyberattack in which a network of compromised computers, known as bots or zombies, are used to launch malicious activities across the internet. This usually involves the distribution of spam, harvesting of personal information, and launching distributed denial-of-service attacks, whereby the target system is flooded with requests and rendered inaccessible. Botnets are typically organized by criminals who have taken control of several computers, using worms, viruses, or other malicious techniques.

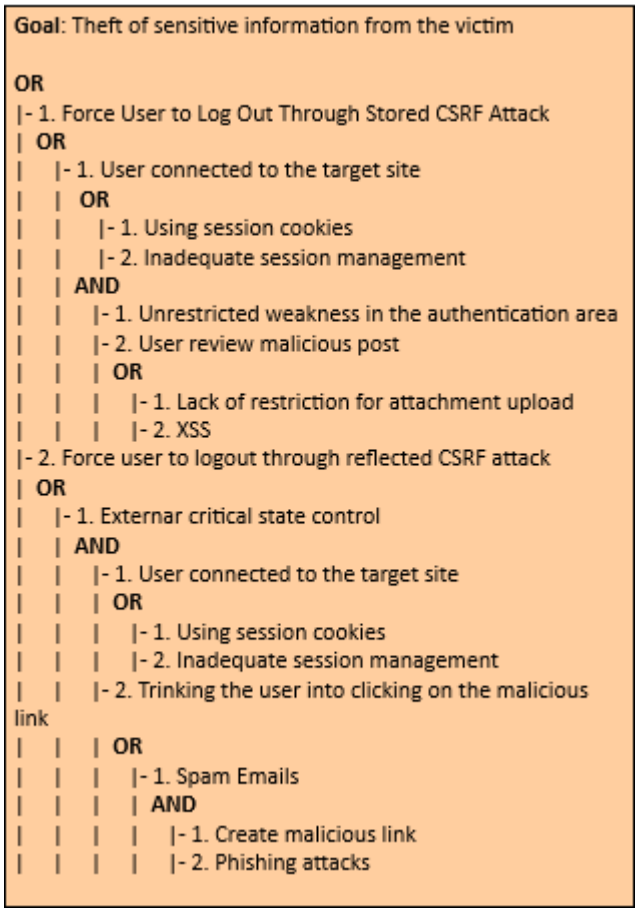
Botnet Architectural Risk Analysis:

Arquitectural Risk Analysis of Botnet Vulnerability as per CVSS v3.1

Attribute Group	Metrics	Value
Attack Vector	AV	Network
Attack Complexity	AC	Low
Privileges Required	PR	None
User Interaction	UI	None
Scope	S	Unchanged
Confidentiality	C	Low
Integrity	I	Low
Availability	A	Low

Overall CVSS Score: 5.4/10

Botnet Vulnerability as per Common Vulnerability Scoring System version 3.1 (CVSS v3.1) is an Arquitectural Risk with an overall score of 5.4/10. The attack vector is the network which requires low attack complexity, no privileges and user interaction. The scope is unchanged with respect to confidentiality, integrity and availability resulting low impact.



Session Hijacking Attack

Session Hijacking is a type of cyber attack in which an attacker takes control of a user's active session by taking control of the session ID. It is also known as session riding or SideJacking.

An attacker can hijack a session by stealing the session ID and using it to create a new session.

This attack can be used to access the user's account and data. It can be used to impersonate the user, perform malicious activities under their name, and gain access to other resources.

To protect against session hijacking, it is important to use strong authentication methods, establish secure network connections, and use session timeouts. Additionally, web applications can use encryption, secure cookies, and server-side session management.

Session Hijacking Architectural Risk Analysis:

Component	Attack Vector	Attack Complexity	Privileges Required	User Interaction	Scope	Confidentiality (C)	Integrity (I)	Availability (A)
Session Hijacking	Network (N)	Low (L)	None (N)	None (N)	Changed (C)	High (H)	Low (L)	Low (L)

CVSS Score: 8.2 (AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:L/A:L)

Impact: High

Goal: To gain unathorized access to the application

OR
|- 1. Reusing Session IDs (aka Session Replay)
| **AND**
| |- 1. The attacker interacts with the target host and finds authenticate users session IDs
| |- 2. The attacker steals a session ID from a valid user
| |- 3. The attacker tries to use the stolen session ID to gain access to the system
|- 2. Session Fixation
| **AND**
| |- 1. Setup the Attack (setup a session)
| | **OR**
| | |- 1. The attacker chooses a predefined identifier that they know
| | |- 2. The attacker creates a trap session for the victim
| |- 2. Attract a victim by fixate the session
| | **OR**
| | |- 1. Attackers can put links on web sites
| | |- 2. Attacker establish rogue proxy servers that give out the session ID and then redirect the connection to the legitimate service
| | |- 3. Attackers can email attack URLs to potential victims through spam and phishing techniques
| |- 3. Abuse the victim's Session via takeover the fixated session
| | **OR**
| | |- 1. The attacker loads the predefined session ID into their browser and browses to protected data or functionality.
| | |- 2. The attacker loads the predefined session ID into their software and utilizes functionality with the rights of the victim.
|- 3. Session Sidejacking
| **AND**
| |- 1. The attacker uses sniffing tools to capture a session token from traffic
| |- 2. The attacker attempts to insert a captured session token into communication with the targeted application to confirm viability for exploitation
| |- 3. The attacker leverages the captured session token to interact with the targeted application in a malicious fashion, impersonating the victim
|- 4. Cross Site Tracing
| **AND**
| |- 1. Determine if HTTP Trace is enabled at the web server with which the victim has an active session
| |- 2. Identify mechanism to launch HTTP Trace reques
| |- 3. Create a malicious script that pings the web server with HTTP TRACE request
| |- 4. Execute malicious HTTP Trace launching script
| |- 5. Intercept HTTP TRACE response

Spoofing Attack

Spoofing attacks are when an attacker masquerades as a legitimate user or other network device in order to gain access to a network, system, or application. It can be used to gain access to confidential data, gain control over a system, or launch other malicious activity. In most cases, spoofing attacks are difficult to detect, as the attacker goes to great lengths to make their identity difficult to verify. Depending on the type of spoofing attack, it could be a form of identity theft, masquerading, or IP spoofing.

Spoofing Architectural Risk Analysis:

Spoofing Vulnerability Arquitectural Risk Analysis

According to the Common Vulnerability Scoring System v3.1, the potential architectural risk posed by spoofing vulnerabilities can be summarized as follows:

- **Attack Vector (AV)**: Network (N) - Attack Vector complexity increases due to direct access from the network.
- **Attack Complexity (AC)**: Low (L) - Spoofing requires little or no user interaction.
- **Privileges Required (PR)**: None (N) - Attackers have no privileges required for the exploit.
- **User Interaction (UI)**: None (N) - Attackers don't require any user interaction to exploit the vulnerability.
- **Scope (S)**: Unchanged (U) - The impact of a successful attack is not limited to the vulnerable system.
- **Confidentiality Impact (C)**: High (H) - Information leakage and/or integrity violations can occur as a result of a successful attack.
- **Integrity Impact (I)**: High (H) - Information leakage and/or integrity violations can occur as a result of a successful attack.
- **Availability Impact (A)**: None (N) - Attacks have no impact on the availability of data or services.

Overall, the vulnerability has a score of 8.5, placing it in a medium risk category.

Goal: Accessing resources or obtain banking and other critical information

OR
|- 1. Content Spoofing
| **OR**
| |- 1. Checksum Spoofing
| |- 2. Spoofing ofUDDI/ebXML Messages
| |- 3. Counterfeit GPS Signals
|- 2. Action Spoofing
| **OR**
| |- 1. Clickjacking
| | **OR**
| | |- 1. Flash File Overlay
| | |- 2. iFrame Overlay
| |- 2. Android Activity Hijack
| |- 3. Credential Prompt Impersonation
| |- 4. Tapjacking
|- 3. Resource Location Spoofing
| **OR**
| |- 1. Redirect Access to Libraries
| | **OR**
| | |- 1. Symlink Attack
| | |- 2. Leveraging/Manipulating Configuration File Search Paths
| | |- 3. Search Order Hijacking
| | |- 4. DLL Side-Loading
| |- 2. Establish Rogue Location
| | **OR**
| | |- 1. Evil Twin Wi-Fi Attack
| | |- 2. Cellular Rogue Base Station
| | |- 3. Scheme Squatting
| | |- 4. BitSquatting
| | |- 5. TypoSquatting
| | |- 6. SoundSquatting
| | |- 7. Homograph Attackvia Homoglyphs
| | |- 8. Bluetooth Impersonation AttackS
|- 4. Identity Spoofing
| **OR**
| |- 1. Pharming
| |- 2. Phishing
| | **OR**
| | |- 1. Spear Phishing
| | |- 2. Mobile Phishing
| | |- 3. Voice Phishing
| |- 3. Fake the Source of Data
| |- 4. Principal Spoof
| |- 5. Signature Spoof

VM Migration Attack

VM Migration attack is a cyber attack that involves a hacker using malicious code to move a virtual machine (VM) from one host server to another. This attack typically targets cloud infrastructure and can be used to steal data or disrupt services. By moving a VM to a different location, the hacker can gain access to sensitive data or launch a denial-of-service attack. Additionally, VM Migration attacks can be used to alter the configuration of a server, such as changing the memory and/or CPU settings, which can lead to degraded performance or crashes.

In order to defend against VM Migration attacks, organizations should use Virtual Machine Intrusion Detection Systems (V-MIDS). These systems detect suspicious activities in virtual machine environments, alerting administrators to potential attacks. Additionally, organizations should closely monitor their cloud environment for unauthorized activities and keep hypervisors, virtual containers, and VMs up-to-date with the latest security patches.

VM Migration Architectural Risk Analysis:

Arquitectural Risk Analysis of VM Migration Vulnerability, according to Common Vulnerability Scoring System (CVSS) v3.1

The arquitectural risk analysis of VM migration vulnerability, according to Common Vulnerability Scoring System (CVSS) v3.1, is summarized in the following table:

Base Score	Vector String
5.9	AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N

This vulnerability is rated as having a **Medium** severity risk.

Attack Vector (AV): Network (N)
Attack Complexity (AC): High (H)
Privileges Required (PR): None (N)
User Interaction (UI): None (N)
Scope (S): Unchanged (U)
Confidentiality Impact (C): High (H)
Integrity Impact (I): None (N)
Availability Impact (A): None (N)

This vulnerability exists if a VM is allowed to migrate or “hop” from one host to another. it puts customers’ systems at risk of unauthorized access and data loss, as well as unplanned downtime and financial losses due to downtime.

Goal: Obtain user confidential data
AND
|- 1. Initiate migration of desired VM/data
|- 2. Obtain and process captured files
| **AND**
| |- 1. Remote captured files
| | **OR**
| | |- 1. Exfiltrate via network
| | |- 2. Exfiltrate via removable media
| | |- 3. Exfiltrate via capture device
| | |- 4. Exfiltrate via Wireless
| |- 2. Process captured files
| | **OR**
| | |- 1. Local
| | | **AND**
| | | |- 1. Obtain forensic apps
| | | | **OR**
| | | | |- 1. Downloads existing apps
| | | | |- 2. Write customs application
| | | |- 2. Introduce forensic apps into the system
| | | |- 3. Execute applications
| | | | **AND**
| | | | |- 1. Opportunity to run processor-intensive
| | | | |- 2. Write to run executable/binary Files
| | | | |- 3. Exclusive access to one or more PCs
| | | | |- 4. Suficiente storage
| | |- 2. Remote
| | | **AND**
| | | |- 1. Obtain forensic apps
| | | | **OR**
| | | | |- 1. Downloads existing apps
| | | | |- 2. Write custom apps
| | | |- 2. Execute applications
| | | | **OR**
| | | | |- 1. . File recovery
| | | | |- 2. Registry analysis
|- 3. Install network tap
| **AND**
| |- 1. Physical access to desired cable(s)
| | **OR**
| | |- 1. Trial and error
| | | **AND**
| | | |- 1. Access to significant proportion of cable infranstructure
| | | |- 2. Suficiente storage space
| | | |- 3. Time!
| | |- 2. Understand cable/network infrastructure
| |- 2. Tap and connect listening computer
| | **AND**
| | |- 1. Expose cable and connect tap device
| | |- 2. Install packet capture device
| | |- 3. Connect tap to capture device without dropping connect
| |- 3. Time!
| |- 4. Possession of dedicated hardware
| | **AND**
| | |- 1. Obtain
| | |- 2. Introduce into the organization covertly

VM Escape Attack

VM Escape Attack is an exploit in which an attacker exploits a vulnerability in the hypervisor to gain unauthorized access to the virtual machine. This attack allows the attacker to gain access to the underlying operating systems and software running on the host machine.

The attacker utilises this access to gain control of the host machine and access any data stored on the virtual machine. This attack can be performed in several ways, such as exploiting a vulnerability in the hypervisor or through malicious code.

In order to prevent this attack, it is important to keep the hypervisor updated with the latest security patches, ensure that the virtual machines are running the latest version of the operating system, and use strong access control measures to limit usage of the underlying resources on the host system.

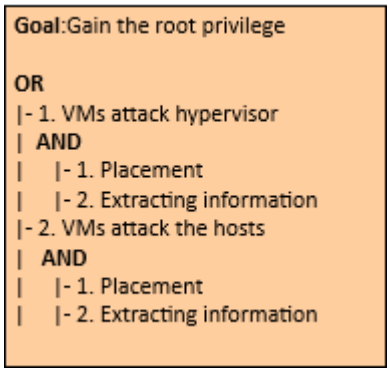
VM Escape Architectural Risk Analysis:

Architectural Risk Analysis of VM Escape Vulnerability

The Common Vulnerability Scoring System (CVSS) v3.1 rates the Architectural Risk of VM Escape Vulnerability as follows:

Metric	Calculated Value	Score
Attack Vector	Network	4.9
Attack Complexity	Low	1.9
Privileges Required	None	0.7
User Interaction	None	0.8
Scope	Unchanged	0.8
Confidentiality	High	3.2
Integrity	High	3.2
Availability	High	3.2
Overall Score	10.4	High Risk

VM escape vulnerability, which enables attackers to escape the virtual machine and gain access to the underlying host system, can have a severe impact on system availability, integrity and confidentiality, especially in cloud computing environment. Such attack can be launched with a low complexity and without user interaction, thus making it all the more dangerous. The network attack vector adds to the severity of the vulnerability, allowing attackers to launch attacks remotely. Therefore, the overall risk score generated by the CVSS v3.1 framework is 10.4, which indicates a high risk.



Side-Channel Attack

Side-channel attack is a type of security exploit which gains access to a system by taking advantage of physical characteristics of its implementation, such as power consumption, timing, sound, or electromagnetic radiation. It is a non-invasive technique which exploits the vulnerability caused by the physical characteristics of the system.

In a side-channel attack, the attacker relies on measurements from the physical system to measure data processing, timing and other information which can be used to determine system calculations, passwords, keys and other data. By analysing these characteristics, attackers are able to gain access to a system, or even to compromise its security.

Side-channel attacks are very difficult to detect and prevent due to the non-intrusive nature of the technique, and they can be very successful in compromising physical information. In order to protect against such attacks, it is important to ensure that the system's physical characteristics are kept secure and are monitored for any suspicious activity.

Side-Channel Architectural Risk Analysis:

Arquitectural Risk Analysis of Side-Channel Vulnerability

According to the **Common Vulnerability Scoring System (CVSS) v3.1**, an arquitectural risk analysis of Side-Channel Vulnerability yields the following scores:

Base Score	Vector String
8.2	AV:P/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

This vulnerability has been rated with a **base score of 8.2**, which indicates its severity. The vector string gives some additional insights:

Attack Vector (AV): Physical (P)

- Indicates that someone must be physically present in order to successfully exploit this vulnerability

Attack Complexity (AC): Low (L)

- Indicates that the attack can be performed with minimal complexity

Privileges Required (PR): None (N)

- Indicates that no elevated privileges are required to perform the attack

User Interaction (UI): None (N)

- Indicates that no user interaction is required for the attack to be successful

Scope (S): Unchanged (U)

- Indicates that the attack affects resources within the same scope

Confidentiality (C): High (H)

- Indicates the attack could result in high-impact data loss

Integrity (I): High (H)

- Indicates the attack could result in high-impact data corruption

Availability (A): High (H)

- Indicates the attack could result in high-impact system unavailability

This analysis reveals that Side-Channel Vulnerabilities have the potential to have a high-impact on the confidentiality, integrity and availability of resources. They can be exploited with minimal complexity, with no user interaction or elevated privileges, and they can affect resources within the same scope.

Goal: Stealing and accessing sensitive user or system information
OR
|- 1. Passive
| **OR**
| |- 1. Physical
| | **OR**
| | |- 1. Local (Chip, Device)
| | | **OR**
| | | |- 1. Power Analysis Attacks
| | | |- 2. Electromagnetic Analysis Attacks
| | | |- 3. Smudge Attacks
| | | |- 4. Shoulder Surfing and Reflections
| | | |- 5. Hand/Device Movements
| | |- 2. Vicinity (Wire/Communication)
| | | **OR**
| | | |- 1. Electromagnetic Analysis Attacks
| | | |- 3. Shoulder Surfing and Reflections
| | | |- 3. USB Power Analysis
| | | |- 4. Wi-Fi Signal Monitoring
| | |- 3. Remote (Sofware, Web)
| | | **OR**
| | | |- 1. Linux-inherited procfs Leaks
| | | |- 2. Data-Usage Statistics
| | | |- 4. Microarchitectural Attacks
| | | |- 4. Sensor-based Keyloggers
| | | |- 5. Fingerprinting Devices/Users
| | | |- 6. Location Inference
| | | |- 7. Speech Recognition
| | | |- 8. Soundcomber
| |- 2. Logical
| | |- 1. Local (Chip, Device)
| | | **OR**
| | | |- 1. Differential Computation Analysis
| | |- 2. Vicinity (Wire/Communication)
| | | **OR**
| | | |- 1. Network Traffic Analysis
| | |- 3. Remote (Sofware, Web)
| | | **OR**
| | | |- 1. Fingerprinting Devices/Users
| | | |- 2. Page Deduplication
| | | |- 3. Linux-inherited procfs Leaks
|- 2. Active
| **OR**
| |- 1. Physical
| | **OR**
| | |- 1. Local (Chip, Device)
| | | **OR**
| | | |- 1. Clock/Power Glitching
| | | |- 2. Electromagnetic Fault Injection
| | | |- 3. Laser/Optical Faults
| | | |- 4. Temperature Variation
| | | |- 5. NAND mirroring
| | |- 2. Vicinity (Wire/Communication)
| | |- 3. Remote (Sofware, Web)
| | | **OR**
| | | |- 1. Microarchitectural Attacks
| | | |- 2. Rowhammer
| |- 2. Logical
| | |- 1. Local (Chip, Device)
| | | **OR**
| | | |- 1. Differential Computation Analysis
| | |- 2. Vicinity (Wire/Communication)
| | | **OR**
| | | |- 1. Network Traffic Analysis
| | |- 3. Remote (Sofware, Web)

Bluejacking Attack

Bluejacking is a type of attack which uses Bluetooth technology to send anonymous messages to other devices. It is used for malicious purposes, like sending unsolicited messages to strangers or attempting to infect other devices with malware.

It is considered a form of “social engineering” attack, as it attempts to manipulate people into revealing information or allowing access to their devices. As with any other form of attack, it is important to keep Bluetooth turned off when not in use and pay attention to any suspicious messages.

Bluejacking Architectural Risk Analysis:

Arquitectural Risk	Score
Attack Vector	Base(4)
Attack Complexity	Low (3)
Privileges	None (0)
User Interaction	Required (3)
Scope	Changed (3)
Confidentiality	None (0)
Integrity	None (0)
Availability	None (0)

The Common Vulnerability Scoring System (CVSS) v3.1 assigns a score of **7.3** to the Bluejacking vulnerability. This score indicates that Bluejacking is a low-severity vulnerability that can be addressed through appropriate action.

Bluesnarfing Attack

Bluesnarfing is a type of attack used by hackers to gain unauthorized access to a Bluetooth enabled device. During a Bluesnarfing attack, a hacker uses specialized software to access confidential information from the device, such as email messages, contact lists, photos, text messages, calendar entries, and more. This type of attack does not require any type of authentication, allowing the hacker to gain access to the device without the victim’s knowledge and consent.

Bluesnarfing can also be used to make changes to the device, such as resetting the user's phone settings, dialing numbers, or sending text messages, which can be used for financial gain. As Bluetooth technology becomes more popular, the prevalence of Bluesnarfing attacks has also increased, making it important to take steps to protect yourself and your devices from this type of attack.

What You Can Do to Protect Yourself

1. Enable Bluetooth only when necessary and turn it off when not in use.
2. Adjust your device’s settings to ensure that other devices cannot detect it.
3. Make sure you have the latest security updates installed on your device.
4. Use a device-tracking application or add a passcode to help prevent unauthorised access to devices.
5. Use a Virtual Private Network (VPN) when accessing devices remotely.
6. Avoid storing personal and sensitive information on Bluetooth-enabled devices.
7. Be aware of the different types of bluetooth attacks and always remain vigilant.

Bluesnarfing Architectural Risk Analysis:

Arquitectural Risk Analysis of Bluesnarfing Vulnerability Using Common Vulnerability & Exploit Scoring System (CVSS v3.1)

CVSS v3.1 Base Score: 7.5/10

- **Attack Vector:** Network (AV:N)
- **Attack Complexity:** Low (AC:L)
- **Privileges Required:** None (PR:N)
- **User Interaction:** None (UI:N)
- **Scope:** Unchanged (S:U)
- **Confidentiality:** High (C:H)
- **Integrity:** High (I:H)
- **Availability:** High (A:H)

CVSS v3.1 Vector String: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

CVSS v3.1 Environmental Score: 7.0/10

- **Confidentiality Requirement:** High (CR:H)
- **Integrity Requirement:** High (IR:H)
- **Availability Requirement:** High (AR:H)

CVSS v3.1 Environmental Vector String: CR:H/IR:H/AR:H

Overall Risk Score: 7.5/10

Goal: To make services unavailable, to inject malware and to steal user/system sensitive information

AND
|- 1. Adquire and setup penetration testing tool
|- 2. Turn on Bluetooth on penetration testing tool, such as Kali Linux
|- 3. Installing “hcitool” and executing “hciconfig”
|- 4. From the terminal, scan the connected Bluetooth (hcitool scan) devices
|- 5. Make sure you can ping all device by using <l2ping 'MAC address'> command
| **OR**
| |- 1. Bluesmack/DoS Attack
| | **AND**
| | |- 1. Clone DOS attack script from GitHub or another repository
| | |- 2. Implement DoS attack using python3 Bluetooth-DOS-Attack.py from DoS attack script folder
| | |- 3. Specify the target address of the device using MAC address of them (Target addr > 'MAC address')
| | |- 4. Setup packages size and Treads counts (e.g., Packages size > 600; Treads counts > 512)
| |- 2. Bluejacking
| | **AND**
| | |- 1. Execute bluetooth-sendto --device='MAC Address' 'file.mp3' command from mobile device to target mobile device
| |- 3. Bluesnarfing
| | **AND**
| | |- 1. Execute bluesarfing attack using bluesnarfer –r 1-100 –b (Bluetooth device address) command from Kali Linux

GPS Jamming Attack

GPS Jamming attack is a type of attack where a signal jammer is used to disrupt the global positioning system (GPS) signal. This type of attack is used to deny or disrupt access to GPS services, either from ground receivers or satellites. GPS jamming is dangerous to navigation since it can cause a variety of navigation errors including decreased accuracy, navigation loop errors, and complete signal loss.

GPS Jamming Architectural Risk Analysis:

GPS Jamming Vulnerability

CVSS v3.1 Base Score: 8.2

Vector String: AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:H/A:H

Attack Vector (AV): N

GPS Jamming targets the physical infrastructure of a location and is thus not remotely accessible.

Attack Complexity (AC): L

GPS Jamming is relatively simple to achieve, as jammers are easily available online.

Privileges Required (PR): N

GPS Jamming does not require special privileges.

User Interaction (UI): N

GPS Jamming can be done without any user interaction.

Scope (S): U

GPS Jamming has the potential to impact the availability and integrity of other systems, due to its ability to disrupt navigation and location services.

Confidentiality (C): N

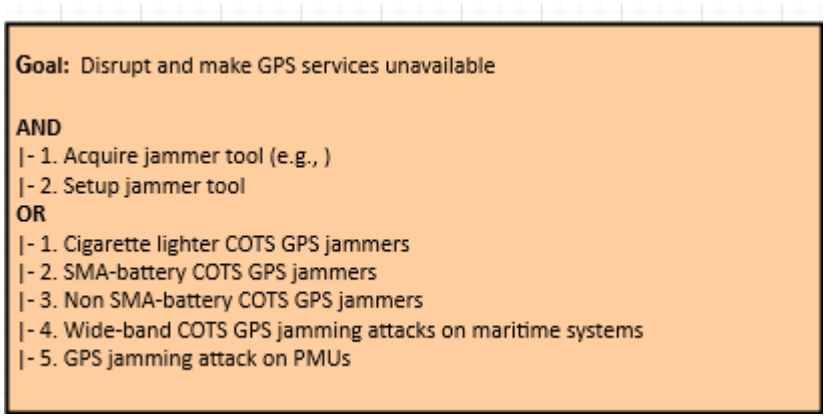
GPS Jamming does not directly affect the confidentiality of data.

Integrity (I): H

GPS Jamming can have a high impact on the integrity of other systems, due to its ability to disrupt navigation and location services.

Availability (A): H

GPS Jamming can have a high impact on the availability of other systems, due to its ability to disrupt navigation and location services.



Cellular Jamming Attack

Cellular Jamming is a form of attack in which an attacker uses a device that transmits signals on the same radio frequencies used by a mobile phone in order to interfere with its functioning. This attack makes it difficult or impossible for a mobile phone to connect to its cell tower, which means that any incoming and outgoing voice data, as well as other forms of data transmission, will be disrupted. This can limit the user's ability to communicate over the network effectively, or even prevent them from accessing the network altogether.

Cellular Jamming Architectural Risk Analysis:

Arquitectural Risk Analysis of Cellular Jamming Vulnerability scoring using Common Vulnerability Scoring System v3.1:

Attack Vector (AV): 4.9 (CWSS 2.0: Network)

Attack Complexity (AC): 2.2 (CWSS 2.8: Low)

Privileges Required (PR): None

User Interaction (UI): 0 (CWSS 2.4: None)

Scope (S): Changed (CWSS 2.6: Changed)

Confidentiality Impact (C): High (CWSS 3.2: High)

Integrity Impact (I): High (CWSS 3.2: High)

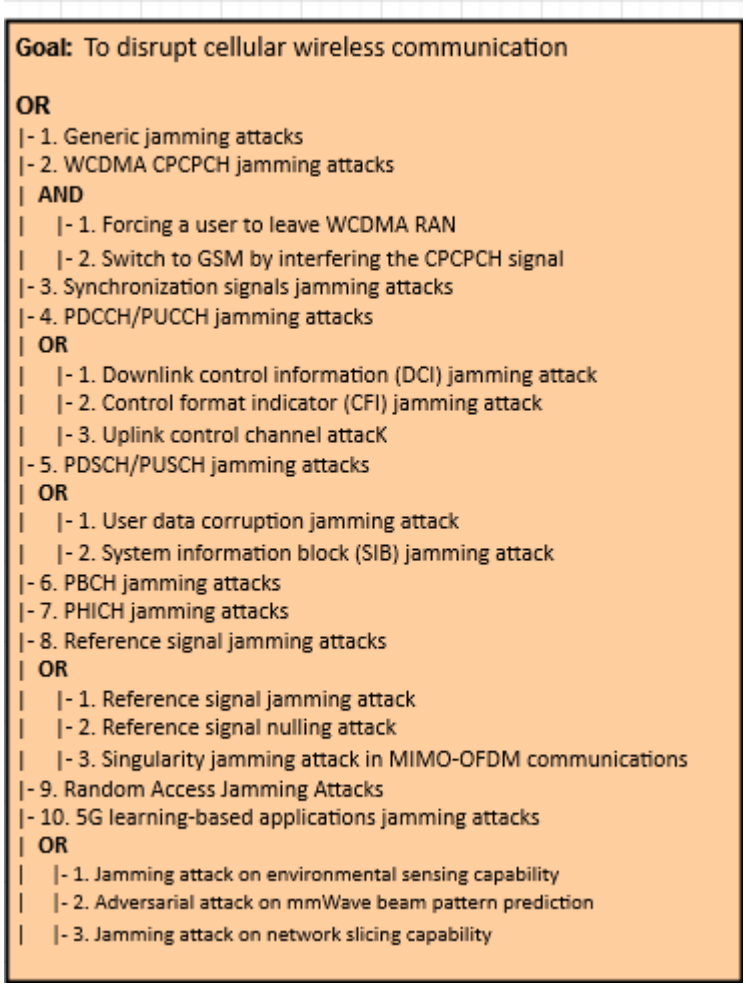
Availability Impact (A): High (CWSS 3.2: High)

Base Score: 9.9

Temporal Score: 8.5

Environmental Score: 9.9

Overall CVSSv3.1 score: 9.3 (High)



Cryptanalysis Attack

Cryptanalysis is an attack on cryptographic systems that involves analyzing and deciphering encrypted data. It involves techniques such as frequency analysis and brute-force methods to decipher data that has been encrypted by a cryptographic system. Cryptanalysis attacks can be used to gain access to confidential data or disrupt the secure channel between two parties.

Cryptanalysis Architectural Risk Analysis:

Cryptanalysis Vulnerability (CVSS v3.1)

- **Attack Vector (AV):** Network (N)
- **Attack Complexity (AC):** Low (L)
- **Privileges Required (PR):** None (N)
- **User Interaction (UI):** None (N)
- **Scope (S):** Unchanged (U)
- **Confidentiality (C):** High (H)
- **Integrity (I):** High (H)
- **Availability (A):** High (H)

Risk Score

The risk score is calculated by taking the base score and multiplying by different metric values for the environmental metrics.

Metric	Value
Base Score	8.8
Temporal	1
Environmental	1

Therefore, the risk score for the Cryptanalysis Vulnerability is **8.8**.

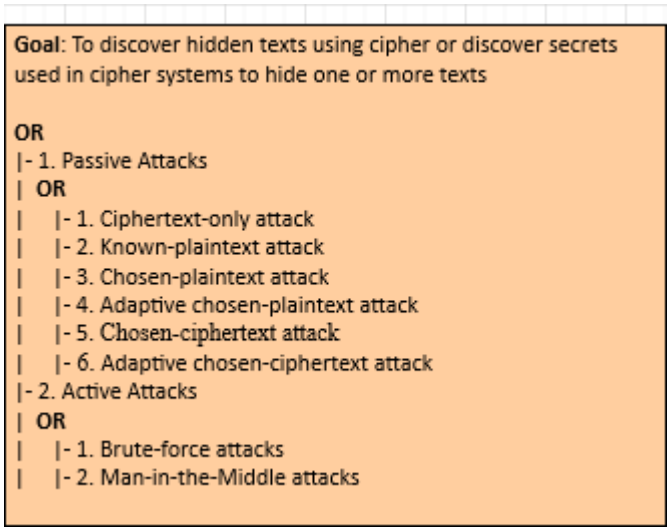
Risk Level

The risk level is based on the risk score and is determined by the following table:

Risk Score	Risk Level

8.0 – 10.0	Critical

Therefore, the risk level for the Cryptanalysis Vulnerability is **Critical**.



Reverse Engineering Attack

Reverse Engineering is a type of attack in which the attacker decompiles or disassembles the program code to gain access to the code or to learn how it works and potentially find hidden flaws or vulnerabilities. This type of attack is typically used by attackers to gain access to confidential or proprietary information. Reverse Engineering can also be used to bypass license protections by creating keys that allow software to run without authorization.

Protecting your software from reverse engineering is an important aspect of cybersecurity and often requires multiple layers of protection to prevent attackers from being able to gain access to confidential data and information. Proper encryption, access control, and enforcing reasonable usage limits are all important measures that should be taken to protect against reverse engineering attacks.

Reverse Engineering Architectural Risk Analysis:

Arquitectural Risk Analysis of Reverse Engineering Attack Vulnerability (according to CVSS v3.1)

Reverse engineering, in simplest terms, is the process of taking something apart to determine how it works. Attackers may reverse engineer an application in order to find flaws that they can exploit in order to gain access to sensitive data or otherwise disrupt the application's functionality.

CVSS v3.1 provides a set of metrics to measure the impact of reverse engineering attack vulnerability on an architectural level. The following table shows the severity score of this vulnerability according to each of the CVSS v3.1 metrics:

Metric	Value
Attack Vector (AV)	Network (N)
Attack Complexity (AC)	Low (L)
Privileges Required (PR)	High (H)
User Interaction (UI)	None (N)
Scope (S)	Unchanged (U)
Confidentiality Impact (C)	High (H)
Integrity Impact (I)	High (H)
Availability Impact (A)	High (H)

The base score according to CVSS v3.1 is 9.7 (Emergency) which poses a high risk to systems, as the attacker may be able to take apart the application and find flaws that can be exploited.

An architectural approach to mitigate this risk is to separate the codebase into multiple layers and to limit access to critical components of the application to authorized personnel only. Additionally, an access control mechanism should be put in place to track and log access attempts to these components. Finally, regular code review and security review should be performed to ensure the code is free of flaws.

Goal: To gain access to sensitive information from systems and users
OR
|- 1. Black Box Reverse Engineering Attacks
| **OR**
| |- 1. Analysis of Packet Timing and Sizes
| |- 2. Electromagnetic Side-Channel Attack
| |- 3. Compromising Emanations Attack
|- 2. White Box Reverse Engineering Attacks
| **OR**
| |- 1. RetrieveEmbedded Sensitive Data
| |- 2. ReverseEngineer an Executable to Expose Assumed Hidden Functionality
| |- 3. ReadSensitive Constants Within an Executable
| |- 4. LiftingSensitive Data Embedded in Cache

Audit Log Manipulation Attack

Audit Log Manipulation attack is a form of attack whereby malicious actors attempt to modify or delete audit logs in order to cover their tracks, hinder incident response and investigation, or even compromise the integrity of systems and data. By manipulating or deleting audit logs, attackers can hide their malicious activities, including malicious file or system modifications and unauthorized access attempts.

Audit log manipulation poses a serious security risk, as it enables attackers to cover their malicious activities, thereby making detected attacks more difficult to investigate and increasing the damage caused by them. Additionally, without the necessary audit logs, organizations cannot comply with cybersecurity regulations and may face significant legal repercussions.

Therefore, it is important for organizations to protect their audit logs and ensure they are unable to be manipulated by malicious actors. Organizations should implement appropriate security controls such as logging policies, logging server security, and log integrity verification to protect their audit logs, and an appropriate audit log review process to detect malicious activities. Additionally, organizations should monitor their system and network logs for any unauthorized access and suspicious anomalies.

Audit Log Manipulation Architectural Risk Analysis:

Audit Log Manipulation Attack (CVSS v3.1)

CVSS v3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:H

Vector String: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:H

Attack Vector (AV): Network (N).

Attack Complexity (AC): Low (L).

Privileges Required (PR): None (N).

User Interaction (UI): None (N).

Scope (S): Unchanged (U).

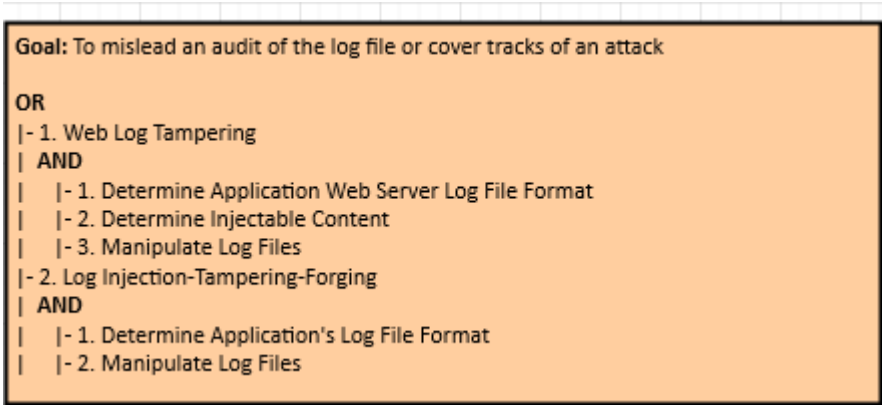
Confidentiality (C): Low (L).

Integrity (I): None (N).

Availability (A): High (H).

Impact: An attacker with access to the audit log system can manipulate it by adding, deleting, and/or modifying audit log entries, in order to cover up malicious activities and/or create false audit log entries.

Recommendations: Access to the audit log system should be restricted to privileged users only, and appropriate monitoring should be implemented to detect any manipulation of the audit log.



Wi-Fi Jamming Attack

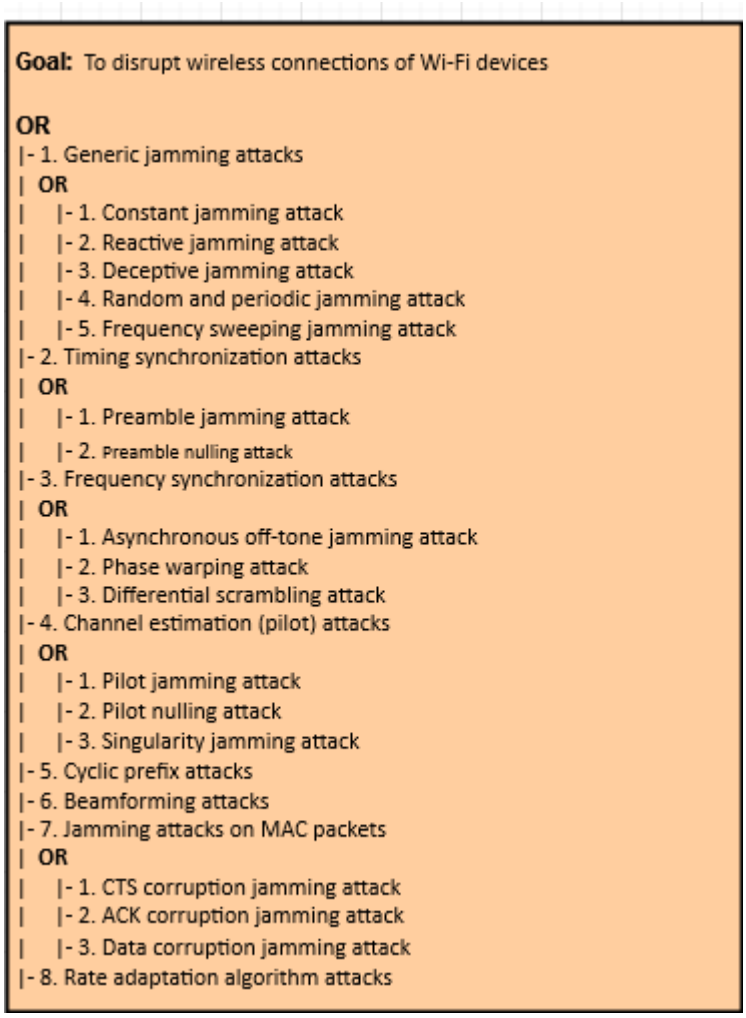
Wi-Fi Jamming attack is a type of attack that uses radio frequency signals to interfere with and/or disrupt legitimate Wi-Fi traffic between two or more connected devices. By sending out high-powered radio signals, the attacker is able to cause a denial of service in the targeted Wi-Fi network, preventing users from accessing the Internet or obstructing any other communication activities.

Wi-Fi Jamming Architectural Risk Analysis:

Wi-Fi Jamming Attack Vulnerability - CVS v3.1 Arquitectural Risk Analysis:

- **Attack Vector (AV):** Network (N)
- **Attack Complexity (AC):** Low (L)
- **Privileges Required (PR):** None (N)
- **User Interaction (UI):** None (N)
- **Scope (S):** Unchanged (U)
- **Confidentiality (C):** High (H)
- **Integrity (I):** Low (L)
- **Availability (A):** High (H)
- **Base Score (BS):** 6.8
- **Temporal Score (TS):** 6.8
- **Environmental Score (ES):** 6.8

Risk Rating: High (CVSS: 6.8/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:H).



Wi-Fi SSID Tracking Attack

Wi-Fi SSID Tracking Attack is a type of attack that allows a malicious actor to leverage the information of an access point, such as its Service Set Identifier (SSID), and use it to track their targets. This attack is possible due to the fact that the SSID is broadcasted by the access point and can be used to track users within a network. It is possible for an attacker to use this information to determine who is using the device, when the device is being used, and even the physical location of the user.

In order to protect against this type of attack, users should ensure that they are not broadcasting their SSID, as this creates an easy target for those looking to track them. Additionally, users can enable encryption to further protect their identities, as the SSID alone cannot be used to determine the user's identity. By following these best practices, users can ensure that their privacy and data is kept secure.

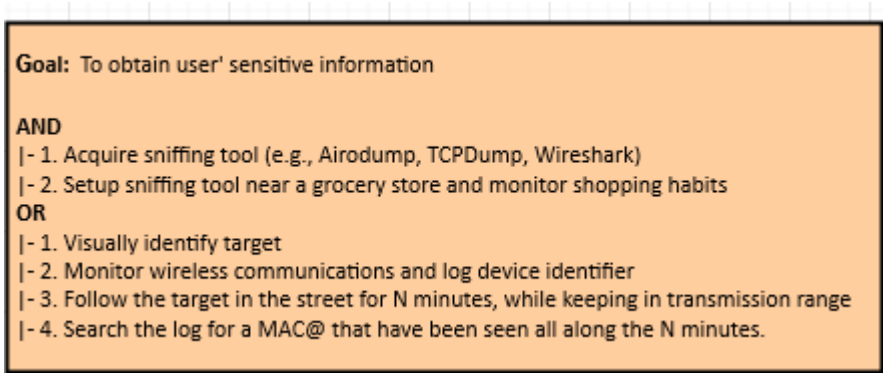
Wi-Fi SSID Tracking Architectural Risk Analysis:

Wi-Fi SSID Tracking Attack Vulnerability - Arquitectural Risk Analysis - CVS3.1

| Attack Vector (AV) | Attack Complexity (AC) | Privileges Required (PR) | User Interaction (UI) | Scope (S) | Confidentiality (C) | Integrity (I) | Availability (A) | Base Score | Exploit Code Maturity (E) | Remediation Level (RL) | Report Confidence (RC) |

| Network (N) | Low (L) | None (N) | None (N) | Changed (C) | High (H) | None (N) | Low (L) | 7.8 | Not Defined (X) | Not Defined (X) | Confirmed (C) |

This Wi-Fi SSID Tracking Attack Vulnerability has an Attack Vector (AV) of Network (N), Attack Complexity (AC) of Low (L), Privileges Required (PR) of None (N), User Interaction (UI) of None (N), Scope (S) of Changed (C), Confidentiality (C) of High (H), Integrity (I) of None (N), Availability (A) of Low (L). The Base Score with these inputs is 7.8, Exploit Code Maturity (E) is Not Defined (X), Remediation Level (RL) is Not Defined (X) and Report Confidence (RC) is Confirmed (C).



Byzantine Attack

Byzantine Attack

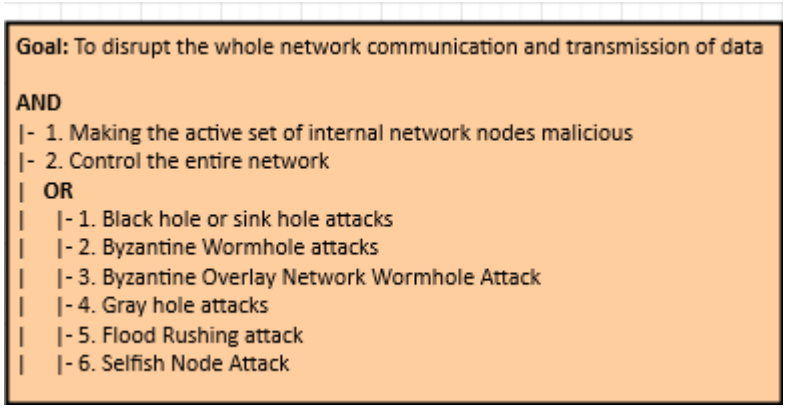
A Byzantine Attack is a type of attack that exploits flaws in a system's communication protocol in order to cause the system to fail in one or more of its operations. It is named after the Byzantine Generals' Problem, a classic problem used in computer science to describe the difficulty of achieving consensus within a distributed system. In a Byzantine Attack, malicious actors send malicious or incorrect messages to the system, causing it to malfunction or reach an unintended result. The system can become vulnerable to this type of attack if its communication protocols are poorly designed and do not adequately protect against malicious or incorrect messages.

Byzantine Architectural Risk Analysis:

Byzantine Attack Vulnerability

- **Attack Vector (AV)**: Network (N)
- **Attack Complexity (AC)**: Low (L)
- **Privileges Required (PR)**: None (N)
- **User Interaction (UI)**: None (N)
- **Scope (S)**: Unchanged (U)
- **Confidentiality Impact (C)**: Medium (M)
- **Integrity Impact (I)**: Low (L)
- **Availability Impact (A)**: Medium (M)
- **CWE-ID**: CWE-400
- **CVSS Base Score**: 4.9

Byzantine attack vulnerability refers to a kind of attack in which the attacking node pretends to be multiple nodes to cause disruption in the network. It is a type of attack that can be successful even with low complexity requirements, and does not require privileged access or user interaction. The scope of the attack does not change and the impacts of this type of attack can result in a moderate impact on the confidentiality, integrity and availability of the system. The CWE-ID associated with this vulnerability is CWE-400 and the CVSS Base Score for this vulnerability is 4.9.



Rowhammer Attack

Rowhammer is an exploit of a hardware security vulnerability that occurs on dynamic random-access memory (DRAM) chips. It exploits a phenomenon whereby repeatedly accessing a row of memory can cause bit flips in adjacent rows, resulting in corruption of data in DRAM. As DRAM density increases, it can become increasingly difficult for memory controllers to keep track of which rows are being accessed, resulting in more frequent bit flips from Rowhammer attacks.

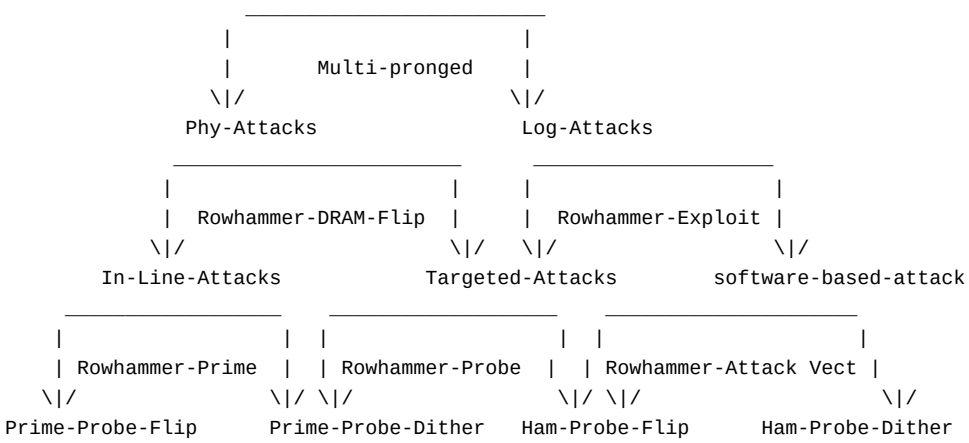
Rowhammer Arquitectural Risk Analysis

Arquitectural Risk Analysis of Rowhammer Attack Vulnerability (CVSS v3.1)

Score	Metric	Value
Base Score		5.9
Vector String		AV:L/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
Attack Vector (AV)		Local (L)
Attack Complexity (AC)		Low (L)
Privileges Required (PR)		None (N)
User Interaction (UI)		None (N)
Scope (S)		Unchanged (U)
Confidentiality (C)		None (N)
Integrity (I)		None (N)
Availability (A)		High (H)

Rowhammer Attack Tree

Rowhammer Attack



Rowhammer Attack

Phy-Attacks

- **Rowhammer-DRAM-Flip**
- In-Line-Attacks
  - Rowhammer-Prime
  - Rowhammer-Probe
- Targeted-Attacks
  - Prime-Probe-Flip
  - Prime-Probe-Dither

Log-Attacks

- **Rowhammer-Exploit**
- Software-Based Attack
  - Ham-Probe-Flip
  - Ham-Probe-Dither

Final Security Test Specification and Tools Report

Mobile Platform	Hybrid Application
Application domain type	Entertainment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	MySQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Remote Storage (Cloud Database)
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	HTML5 + CSS + JavaScript
Input Forms	Yes
Upload Files	Yes
The system has logs	Yes
The system has regular updates	Yes
The system has third-party	Yes
System Cloud Environments	Hybrid Cloud
Hardware Specification	No
HW Authentication	Basic Authentication (user/pass)
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ;

Testing the DoS or Cellular Jamming Attack

1. **Assessment:** Set up a denial-of-service (DoS) or cellular jamming attack testing to check for potential vulnerabilities/risk.
2. **Preparation:** Before testing, make sure to have the necessary tools and materials, including a network-testing tool that can be used to simulate DoS/jamming attacks, a test environment for the attack, and a virtual machine environment.
3. **Testing Procedure:** * Step 1: Configure the test environment with the attack type chosen. * Step 2: Execute the attack on the test environment. * Step 3: Monitor for any signs of the attack's success such as decreased network performance. * Step 4: If the attack has been successful, record the data or screenshot the results. * Step 5: Repeat steps 1-4 as needed with different attack types and configurations.
4. **Analysis:** Analyze the results of the tests to determine any potential DoS/jamming vulnerabilities or risk in the system.
5. **Conclusion:** Analyze the results and make conclusions on the potential impacts of DoS/jamming attacks on the system.

Testing Tools:

Target Testing | Testing Technique | Test Analysis | Test Method | Test Tool | Mobile Platform

DoS Attack | White-box | Dynamic | Penetration Testing | Nmap | iOS/Android Cellular Jamming | Gray-box | Static | Vulnerability Scan | Nessus | iOS/Android | Black-box | Hybrid | Activity Monitoring | Wireshark | iOS/Android

Testing the Wi-Fi Jamming Attack

1. Test equipment

In order to test a Wi-Fi Jamming attack, the following test equipment is typically needed: - Wi-Fi Jamming device(s) - A laptop or desktop computer with a wireless adapter - External antennas

1. Set up test environment

The best way to test a Wi-Fi Jamming attack is to create a simulated Wi-Fi environment: - Create a dedicated wireless network or VLAN that has no other Wi-Fi traffic on it. - Connect the test laptop or desktop to the dedicated Wi-Fi network. - Connect the jamming device(s) to a power source.

1. Test the jamming attack

Once the test environment is set up and all the equipment is connected, the jamming attack can be tested:

- Verify that the jamming device(s) are broadcasting the jamming signal and that the test laptop or desktop can detect it.
- Try to make a connection on the test laptop or desktop.
- Verify that the jamming device(s) is blocking or degrading any attempts to connect to the Wi-Fi network.
- Try to connect to other Wi-Fi networks in range to verify that the jamming attack is blocking or degrading any attempts to connect to other Wi-Fi networks outside of the test environment.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Wi-fi Jamming Attack	White-box	Dynamic	Parameter Testing	InSSIDer	iOS, Android
Wi-fi Jamming Attack	Grey-box	Static	Code Review	Nmap	iOS, Android
Wi-fi Jamming Attack	Black-box	Hybrid	Network Sniffing	Wireshark	iOS, Android

Testing the Orbital Jamming Attack

Testing of Orbital Jamming Attack

Orbital jamming attacks aim to hamper satellite communications and can be difficult to detect. Here are some test procedures that can be implemented to detect such attacks:

Spectral Analysis using Software Defined Radios (SDRs): SDRs can be used to analyze the frequency spectrum and detect any deviations in the spectrum that may indicate the presence of the jamming signals.

Direct Signal Detection: If a direct signal is detected, then it can be used to detect the presence of a jamming attack.

Power Analysis: Power analysis can be used to identify any abnormalities in the received signal strength. A rapidly fluctuating signal could be an indication of an Orbital Jamming attack.

Timing Analysis: Timing analysis can be used to determine the timing of the jamming signals. If the timing is irregular or inconsistent, then it could be an indication of an attack.

Signal-to-noise Ratio Analysis: Signal-to-noise ratio analysis can be used to determine the accuracy of the received signal. If the signal-to-noise ratio is low or fluctuating, then it could indicate an orbital jamming attack.

These tests can help detect any presence of a jamming attack, and ultimately lead to further investigation.

Testing Tools:

Target	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Orbital Jamming Attack	White-box	Dynamic	Penetration Testing	Burp Suite	iOS
Orbital Jamming Attack	Grey-box	Static	Fuzz Testing	OWASP ZAP	Android
Orbital Jamming Attack	Black-box	Hybrid	Automated Security Testing	AppScan	iOS, Android

Testing the GPS Jamming Attack

Testing GPS Jamming Attack

Check if possible to jam satellite signals. This can be done by examining the radio frequencies around the target area.

Once a jamming source is spotted, use a GPS receiver to measure the impact of the interference.

Use a spectrum analyzer to further analyze the spectrum of the jammer. This will allow one to see any changes in the jammer power.

Test the GPS receiver to ensure that it is functioning properly, and that its connections are not being disrupted by the jammer.

Install different types of antennae to measure the range of the jamming attack and to determine how effective it is.

Take readings at different distances to measure the impact of the jammer. This will help determine the strength of the jammer and its likely range.

Test the GPS in different environments such as buildings, forests, and open areas to get an idea of its capabilities.

Finally, use a field-testing kit to measure the impact of the jammer in a more realistic environment. This will provide an accurate measure of the effectiveness of the jamming attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
GPS Functionality	White-box	Dynamic	Regression testing	Google Test	Android
GPS Functionality	Grey-box	Static	Security Testing	Nessus	iOS
GPS Jamming	Black-box	Hybrid	Penetration Testing	Metasploit	Cross-platform
GPS Jamming	Black-box	Hybrid	Exploratory Testing	Burp Suite	Cross-platform

Testing the Bluesnarfing Attack

1. First, set up a Bluetooth enabled device with a vulnerable version of Bluetooth and make sure the device is in Discoverable mode.

2. Scan the device with a tool such as BlueSnarf, a suite of Bluetooth security tools, to determine the services offered by the Bluetooth enabled device.
3. If the services offered contain authentication and/or authorization, identify and exploit vulnerabilities to complete a successful Bluesnarfing attack.
4. Once the attack is complete, a log of information should be generated and the activity should be monitored to verify the success of the attack.
5. Further testing of the security measures in place should be performed to reduce the likelihood of a successful Bluesnarfing attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
VulnerableBT	White-Box	Dynamic	Manual	No	Android, iOS
Vulnerable BT	White-Box	Static	Manual	No	Android, iOS
Vulnerable BT	Grey-Box	Dynamic	Manual	No	Android, iOS
Vulnerable BT	Grey-Box	Static	Manual	No	Android, iOS
Vulnerable BT	Black-Box	Dynamic	Manual	No	Android, iOS
Vulnerable BT	Black-Box	Static	Manual	No	Android, iOS
Vulnerable BT	White-Box	Hybrid	Manual	No	Android, iOS
Vulnerable BT	Grey-Box	Hybrid	Manual	No	Android, iOS
Vulnerable BT	Black-Box	Hybrid	Manual	No	Android, iOS

Testing the Bluejacking Attack

1. Set up the environment:
    - Install a Bluetooth-capable device, such as a laptop or a smartphone.
    - Install a Bluetooth scanner or proximity detection software such as Bluediving.
2. Perform the attack:
    - Enable Bluetooth on both the target device and the attacker's device.
    - On the attacker's device, scan for Bluetooth devices in the area.
    - Once the target device is detected, the attacker can send a message or picture to the target device.
3. Test the results:
    - Observe whether the message or picture was successfully sent.
    - Observe whether the target device has responded to the message or picture.
    - Verify that the data was successfully transferred between devices.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Bluejacking Attack	White-box	Dynamic	Fuzz Testing	Metasploit Framework	Android
Bluejacking Attack	Grey-box	Static	Security Code Review	Code Radar	iOS
Bluejacking Attack	Black-box	Hybrid	Exploratory Testing	Burp Suite	Windows Phone

Testing the Wi-Fi Jamming Attack

Testing a Wi-Fi SSID Tracking attack:

1. Scan the local wireless network to record all available SSID (Service Set Identifier) information, including the SSID name, MAC address, and signal strength.
2. Monitor the same wireless network to track changes in SSID information over time.
3. Compare the current and previous records of SSID information to identify any changes.
4. Collect data points using tracking software that can map a device’s route through an area based on the changes in SSID information.
5. Visualize the data points on a map and assess the accuracy of the location data.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Wi-Fi SSID Tracking	White Box	Dynamic	Penetration Testing	Nmap	Android/iOS
Wi-Fi SSID Tracking	Grey Box	Dynamic	Exploratory Testing	Selenium	Android/iOS
Wi-Fi SSID Tracking	Black Box	Static	Code Review	SonarQube	Android/iOS
Wi-Fi SSID Tracking	White Box	Hybrid	Security Testing	OWASP Zed Attack Proxy	Android/iOS

Testing the Byzantin Attack

One way to test for a Byzantine attack is to use the "Byzantine Fault Tolerance Test" (BFTT). This is a test designed to detect any malicious behavior in a distributed system. The test can be broken up into the following steps:

Setup the test environment: - Choose a testbed and deploy the system components according to the required configuration. - Install monitoring tools to measure the performance of the system.

Perform the test: - Launch the Byzantine Fault Tolerance Test to detect any instances of malicious behavior on the distributed system. - Generate test transactions that could be potentially malicious. - Measure the system's response to the malicious transactions.

Analyze the results: - Analyze the system's behavior in terms of speed, accuracy, and time to detect malicious transactions. - Compare the results of the test with the theoretical baseline or established standards.

Make changes to secure the system: - Make changes to the system configuration or software to ensure the system is secure. - Monitor the system's performance after the changes are implemented to ensure the system is secure.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tools	Mobile Platform
Byzantine Attack	White-box	Dynamic	Penetration Testing	Nmap, Wireshark	Android, iOS
Byzantine Attack	Grey-box	Static	Code Review	SonarQube	Android, iOS
Byzantine Attack	Black-box	Hybrid	Stress Testing	LoadRunner	Android, iOS

Testing the On-Off Attack

Testing an On-Off attack is done by conducting a series of tests against a target system to attempt to determine its susceptibility to the attack.

The testing process consists of the following steps:

1. Identify the target system and configure it for the attack.
2. Conduct active reconnaissance to discover the presence and types of services running on the target system.
3. Launch a series of On-Off attacks against the identified services.
4. Analyze the response from the target system to determine if the attack was successful.
5. If successful, note the attack vector(s) used and document the vulnerability that was exploited.
6. If unsuccessful, repeat the process until the target system has been fully tested.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Plataform
Endpoints	White-box	Dynamic	Exploitation	Burp	Android & iOS
Services	Grey-box	Static	Pen. Test	Zed Attack Proxy	Android & iOS
Network	Black-box	Hybrid	Scan	Nmap	Android & iOS

Testing the Malicious Insider Attack

Educate Employees: Provide regular training to employees on data protection, cybersecurity, and mitigation of malicious insider activity.

Monitor Network Activity : Monitor and audit employees' access to data and system. Look for suspicious activity such as unusually large data transfers or out-of-the-ordinary access attempts.

Separate Duties : Thoroughly separate duties between different personnel to prevent one person from having too much access or control.

Implement Least Privilege Measures: Limit user access by granting only the privileges that are needed, rather than granting full access to all users.

Establish Access Protocols: Establish access protocols and rules regarding data access, such as when, what, and how users are allowed to access certain data.

Conduct Security Audits: Regularly perform security audits on employee accounts to detect malicious activity.

Utilize Network Logging : Utilize network logging to monitor user activity, such as web browsing and application usage.

Install SSL Monitoring: Install an SSL monitoring system to detect any suspicious activity, such as unauthorized access to sensitive data.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Detection	White-box	Dynamic	Functional Testing	OWASP ZAP Proxy	iOS
Prevention	Grey-box	Static	Fault Tree Analysis	HP Fortify	Android
Response	Black-box	Hybrid	Penetration Testing	Burp Suite	All

Testing the Sniffing Attack

Sniffing attacks can be tested using the tools included in most network security programs.

The following steps should be taken to test Sniffing Attacks:

Set up a network and create a designated test environment.

Monitor all of the traffic on the network and detect any signs of suspicious activity. This would include anything out of the ordinary, such as changes in IP addresses, traffic amounts, or unexpected errors.

Use the tools included in the security program to detect the sources of the suspicious traffic.

Actively scan the network in order to detect any malicious programs that might be present.

Use a packet sniffer to log all packets and analyze any anomalous behavior.

Detect and block any attempts to hijack communications or take control of any system on the network.

Perform a vulnerability assessment of the network to ensure that no weak spots are present that could be taken advantage of by a sniffing attack.

Test to see if your organization's security measures are able to detect and block against any sniffing activity.

Once the tests have been completed, perform a validation process to ensure that all of the results are accurate and the attack can be blocked and remediated properly.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Sniffing Attacks	White-box	Static	Code Review	Wireshark	No
Sniffing Attacks	Grey-box	Hybrid	Fuzz Testing	Tcpdump	No
Sniffing Attacks	Black-box	Dynamic	Protocol Analysis	Scapy	No

Testing the Man-in-the-Middle Attack

Testing Man-in-the-Middle Attack

Man-in-the-middle (MitM) attacks are a type of cyber attack where an attacker intercepts the communication between two parties, allowing them to listen in on, modify, or block the communication. In order to test for MitM attacks, you must be able to detect if a third-party is attempting to intercept or modify traffic between two parties.

1. Disable your firewall temporarily and ensure that any antivirus software is turned off.
2. Use command line tools such as **netstat** to monitor incoming and outgoing connections to your system.
3. Monitor inbound and outbound traffic for suspicious IP addresses or packets.
4. Use **traceroute** and **ping** tests to identify malicious intermediaries on the network.
5. Use **arp spoofing** to detect and stop man-in-the-middle attacks.
6. Monitor log files for suspicious or irregular behavior.
7. Implement solutions such as SSL/TLS or IPSec encryption to ensure the authenticity of data transmitted over the network.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Man-in-the-Middle	White-box	Dynamic	Penetration	Nmap	iOS
Man-in-the-Middle	Grey-box	Static	Source Code	Infer	Android
Man-in-the-Middle	Black-box	Hybrid	Vulnerability	Burp Suite	Windows
Man-in-the-Middle	Grey-box	Dynamic	Performance	WebLink	iOS
Man-in-the-Middle	White-box	Static	Network	Wireshark	Android
Man-in-the-Middle	Black-box	Hybrid	Fuzzing	Afl	Windows

Testing the Eavesdropping Attack

Eavesdropping attacks are one of the most common and widespread cyberattacks, in which an attacker attempts to gain access and collect sensitive or confidential information sent or stored by individuals or businesses.

Testing for an eavesdropping attack can be accomplished in several ways. Some of the methods that can be used are outlined below:

Network Sniffing: This involves monitoring network traffic for any suspicious activity. Network packet sniffing tools and protocols such as TCPdump and Wireshark can be used to help detect suspicious activity and potential eavesdropping attempts.

System Logs: Reviewing system logs can help to identify any suspicious activity that may be related to an eavesdropping attack. Administrators can also monitor application and network logs for any unauthorized access or changes that may have been made to the system.

Password Testing: Utilizing password-cracking tools such as John the Ripper, administrators can test the strength of passwords that are in use against a list of thousands of common passwords. This offers the best idea of how secure a network is against eavesdropping attacks.

Penetration Testing: This is a more advanced method of testing for eavesdropping attacks and involves launching sophisticated simulated attacks on an organization's system. Penetration testers use tools such as Metasploit and Nmap to assess the effectiveness of an organization's security measures against different types of attack vectors, including eavesdropping attacks.

Antivirus Scanning: This is an important step that should be undertaken regularly. Regularly scanning the system with a reliable antivirus will help to identify any malicious software that may have been installed as part of an eavesdropping attack.

Testing Tools:

Target Testing	Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Eavesdropping Attack	White-box	Dynamic	Penetration Testing	Nexpose, Nmap, Metasploit	Android, iOS

Testing the CSRF Attack

Below are the steps to test for a CSRF attack:

1. Ensure that the application is properly allowing and validating CSRF tokens.
2. Identify all form entry points in the application.
3. Use a proxy tool such as Burp Suite to intercept the request.
4. Modify the request on the proxy tool with an incorrect CSRF token.
5. Submit the request and observe the response.
6. If an error is returned, the application is properly validating the token and the attack is not successful.
7. If the request is accepted, the application is not properly validating the CSRF token and the attack is successful.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
CSRF Attack	White-Box	Dynamic	Manual	Burp Suite	iOS & Android
	Grey-Box	Static	Automated	Nikto	IOS & Android
	Black-Box	Hybrid	Automated	Owasp Zap	iOS & Android

Testing the SQLi Attack

Testing SQLi Attack

1. Cross-Site Scripting (XSS)

Cross-site scripting (XSS) is a type of attack that involves a malicious script being injected into a website. When a user visits the website, the script is executed in their browser. To test for XSS, we can insert a malicious script into the website and see if the script is executed.

2. SQL Injection

SQL injection is a type of attack where malicious code is inserted into the SQL query in order to gain access to data stored in a database. To test for SQLi attacks, we can insert a malicious query into the website and check if it is executed on the database.

3. Parameter Tampering

Parameter tampering is a type of attack where parameters of a URL are modified to gain access to restricted information or data. To test for parameter tampering, we can change the values of URL parameters and see if the website is responding with different outputs.

4. Brute-Force Attack

Brute-force attacks are attempts to guess the username and password of a system using automated tools. To test for brute-force attacks, we can use tools that try different combinations of usernames and passwords and check if the system is able to detect the attempts.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
SQLi Attack	White-box	Dynamic	Manual	SQLMap	iOS / Android
SQLi Attack	Grey-box	Static	Automated	OWASP ZAP	iOS / Android
SQLi Attack	Black-box	Hybrid	Exploration	Burp Suite	iOS / Android

Testing the XSS Attack

Testing XSS Attack

1. Identify the potential injection points: - Take a look at webpages for available input fields, parameters in a URL, cookies, and source code.
2. Input test payloads: - One of the most common XSS payloads is the `<script>alert(1)</script>` tag.
3. Monitor the response: - If your payload is successfully executed, you'll likely get an alert box or some other type of notification.
4. Assess for vulnerabilities: - Once you've confirmed where the vulnerabilities are, you can start to control and mitigate the attack.

Prevention of XSS Attack

1. Employees: - Train employees to be aware of XSS attacks so they can spot and report suspicious activities.
2. Validate input: - Validate and filter any user input to ensure that only properly formatted input is accepted.
3. Sanitize input: - Sanitizing data is the process of removing Javascript, HTML, and other malicious code from user input.
4. Use Trusted data: - Be sure to use trusted input whenever possible to minimize the chances of XSS attacks.
5. Use Security Headers: - Use security headers, such as HTTP Strict Transport Security (HSTS), to help prevent XSS attacks.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
XSS Attack	White-box	Dynamic	Manual	Burp Suite	iOS, Android
	Grey-box	Static	Automated	HP WebInspect	
	Black-box	Hybrid		Acunetix	

Testing the SSRF Attack

Testing SSRF Attack

Set up an attacker controlled server: This server needs to be able to accept connections from the target server and should log the requests from the target server like the source IP address, used port, timestamp, and any data sent over with the request.

Prepare a crafted URL: This URL should lead to the attacker controlled server and is used for the SSRF attack.

Send the crafted URL to the target server: This can be done manually by providing the URL as an input for a user or programmatically by sending the URL with a POST request to the target server.

Check the logs of the attacker controlled server: After the URL is sent to the target server, the attacker controlled server should log the requests sent by the target server. This is used to analyze the type of requests being sent, the data being sent, and the source IP address.

Monitor the environment of the target server: When the target server makes requests to the attacker controlled server, the attacker can set up environment variables, files, resources, etc. in order to exploit the target server.

Try to gain access to the target server: If the SSRF attack was successful, the attacker can try to access the target server from the attacker controlled server by using the access gained from the exploited environment of the target server.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Server	White-Box	Dynamic	Input Validation	DSSS - Detection of SSRF	No
Server	Gray-Box	Dynamic	Fuzzing	Burpsuite	No
Server	Black-Box	Dynamic	Vulnerability Scanner	NTOSpider	No
Server	White-Box	Static	Source Code Analysis	Code Editors	No
Server	Gray-Box	Static	Source Code Auditing	Vega	No
Server	Black-Box	Static	Manual Review	Manual Review	No
Server	White-Box	Hybrid	Interactive Testing	Manual Review	No
Server	Gray-Box	Hybrid	Penetration Testing	Metasploit	No
Server	Black-Box	Hybrid	Penetration Testing	Zed Attack Proxy	No
Mobile	White-Box	Dynamic	Input Validation	DSSS - Detection of SSRF	Yes
Mobile	Gray-Box	Dynamic	Fuzzing	Burpsuite	Yes
Mobile	Black-Box	Dynamic	Vulnerability Scanner	NTOSpider	Yes
Mobile	White-Box	Static	Source Code Analysis	Code Editors	Yes
Mobile	Gray-Box	Static	Source Code Auditing	Vega	Yes
Mobile	Black-Box	Static	Manual Review	Manual Review	Yes
Mobile	White-Box	Hybrid	Interactive Testing	Manual Review	Yes
Mobile	Gray-Box	Hybrid	Penetration Testing	Metasploit	Yes
Mobile	Black-Box	Hybrid	Penetration Testing	Burp Suite	Yes

Testing the Command Injection Attack

1. **Testing for Command Injection**

Command Injection is an attack in which malicious commands are injected into a vulnerable application in order to gain access or obtain privileged information. This type of attack involves taking user input and executing it as code on the system, making it highly dangerous and difficult to detect.

In order to test for command injection, it is important to first identify any areas of the application that may be vulnerable to attack. Applications which take user input and execute it as code are particularly at risk.

Once potential vulnerable areas have been identified, test cases can be created to simulate attempts at injecting malicious code. This can be done by attempting to insert special characters or meta-characters into input fields, such as semi-colons, backticks, or quotation marks. If the application responds in an unexpected manner or creates an externally visible result, such as creating an extra file in the directory, it is likely that command injection is possible.

It is important to note that command injection may not always be easy to spot, as the application may attempt to sanitize user input or where malicious code can be 'hidden' within benign input. As such, testing using all potential input combinations is recommended in order to ensure that vulnerabilities have been adequately identified and addressed.

Once all potential vulnerabilities have been assessed and remediated, it is recommended to repeat the testing process in order to validate the changes.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Command Injection Attack	White-box	Dynamic	Input Fuzzing	Metasploit	iOS/Android
Command Injection Attack	Grey-box	Static	Variant Analysis	CodeSonar	iOS/Android
Command Injection Attack	Black-box	Hybrid	Interactive Fuzzing	Burp	iOS/Android

Testing the Code Injection Attack

Testing code injection attacks involve attempting to inject malicious code into a web application. Testing code injection is typically done using a few different methods.

Manual Tests: Manual tests involve manual input of potentially malicious code into user inputs and forms within the application. If the application does not appropriately filter or reject suspicious input, this can lead to the vulnerability.

Automation Tools: Automation tools, such as Security Scanners, can be used to test for code injection attacks. Tools such as these can detect code injection vulnerabilities with very high accuracy, allowing security teams to quickly identify and mitigate such risks.

Penetration Testing: Penetration testing is a manual assessment of security. During this process testers attempt to gain access to restricted areas within an application, by entering or manipulating code in user input fields. This will allow testers to identify potential security weaknesses in an application, and recommend remediation strategies.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Code Injection	White-box	Dynamic	Unit Testing	HP QTP	Android
Code Injection	Grey-box	Dynamic	Penetration Testing	HP Fortify	Android
Code Injection	Black-box	Static	Security Scanning	Burp Suite	iOS
Code Injection	White-box	Hybrid	Code Review	Splint	iOS

Testing the Phishing Attack

1. Create a mock Phishing website: Create a fake website that looks and behaves like the real one in order to fool users into providing their login credentials or other sensitive information.
2. Set up traffic monitoring: Monitor incoming traffic to your website to ensure that the fake site isn’t receiving any legitimate requests.
3. Analyze the behavior of malicious users: Monitor user behavior on your site and look for any suspicious activity that may indicate a potential attack.
4. Send out test emails: Send out phishing emails to members of your staff and monitor their responses. If the email is opened and a link is clicked, you can be sure that your staff needs more education on Phishing attacks.
5. Analyze the results: Once all of the test emails have been sent out and the responses analyzed, prepare a report outlining what worked and what didn’t.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Phishing Attack	White-box	Dynamic	Penetration	Metasploit	iOS, Android
Phishing Attack	Grey-box	Static	Source Code	OWASP ZAP	iOS, Android
Phishing Attack	Black-box	Hybrid	Fuzzing	Wfuzz	iOS, Android

Testing the Pharming Attack

Testing a pharming attack requires several steps:

1. Use a vulnerability scanner to identify and assess the security of the computer systems, networks, and applications.
2. Access the system and perform a detailed analysis of the system logs and audit trails.
3. Perform a packet sniffing test to detect any malicious activity.
4. Check for any Domain Name System (DNS) anomalies and any suspicious domain redirection.
5. Check if the DNS cache poisoning technique has been employed.
6. Use a URL analyzer tool to detect any malicious redirects or domain name changes.
7. Run a malware scan on the system to detect any malicious code or program.
8. Use a honeypot system to detect any suspicious activity on the network.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Network	White-box	Dynamic	Automation	Nmap	N/A
Network	White-box	Static	Manual	Wireshark	N/A
System	Grey-box	Dynamic	Automation	Nessus	N/A
System	Grey-box	Static	Manual	Regshot	N/A
Host	Black-box	Dynamic	Automation	Core Impact	Android, iOS

Testing the Spoofing Attack

Testing for Spoofing Attacks

Scanning For Open Ports: By using network scanning tools, such as NMAP, one can identify open ports that may be vulnerable to spoofing attacks.

Creating Honeypots: Honeypots can be created and deployed on the network to detect and identify malicious behavior.

Disabling Unused Services: Unused services can be disabled on the system to reduce the attack surface available to an attacker.

Restrict Network Traffic: Network traffic can be restricted on the system to avoid spoofing attacks.

Whitelisting: Only applications, IP addresses and services that are known and trusted should be allowed to access the system. All other traffic should be blocked.

Implement Firewalls and Intrusion Detection Systems: Firewalls and Intrusion Detection Systems (IDS) should be implemented to detect and prevent unauthorized access and activity.

Monitor Network Activity: The network activity should be monitored on a regular basis to identify any suspicious activity that may indicate a spoofing attack.

Regularly Update System Software: Systems should be updated regularly with the latest security patches to address known vulnerabilities that can be exploited by attackers.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Spoofing Attack	White-box	Dynamic	Exploratory Testing	SoapUI	iOS
Spoofing Attack	Grey-box	Static	Source Code Analysis	Veracode	Android
Spoofing Attack	Black-box	Hybrid	Penetration Testing	Nessus	iOS & Android

Testing the Session Fixation Attack

Testing for Session Fixation Attack

Start by testing for session fixation with known tokens. Generate a known token, log into the application and submit the known token in the login form. If the application successfully logs into the system, the application may be vulnerable to session fixation.

If the form does not accept known tokens and does not reissue the token, the application may be vulnerable to session fixation.

Insert a tracking or logging code into the application and try to use different known tokens and see if they are accepted by the application.

After logged in, renew the session and check if the session ID is changed. If it is same as before, it might be vulnerable to session fixation.

Log out of the application and try to use the session ID that was captured during login. If the application continues to use the same session ID, it might be vulnerable to session fixation.

Create a honeypot page that only registered users can access. Log in with a known token and try to access the page. If the application allows access to the page using the known token, the application might be vulnerable to session fixation.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Session Fixation Attack	White-box	Dynamic	Automated	Web security scanner (e.g. Acunetix, Netsparker, Burp Suite)	N/A
Session Fixation Attack	Grey-box	Dynamic	Automated	Web security scanner (e.g. Acunetix, Netsparker, Burp Suite)	N/A
Session Fixation Attack	Black-box	Dynamic	Automated	Web security scanner (e.g. Acunetix, Netsparker, Burp Suite)	N/A
Session Fixation Attack	White-box	Static	Automated	Code security scanner (e.g. Checkmarx, Veracode, SonarQube)	N/A
Session Fixation Attack	Grey-box	Static	Automated	Code security scanner (e.g. Checkmarx, Veracode, SonarQube)	N/A
Session Fixation Attack	Black-box	Static	Automated	Code security scanner (e.g. Checkmarx, Veracode, SonarQube)	N/A
Session Fixation Attack	White-box	Hybrid	Automated	Code and web vulnerability scanner (e.g. Appscan, Appspider, Arachni)	N/A
Session Fixation Attack	Grey-box	Hybrid	Automated	Code and web vulnerability scanner (e.g. Appscan, Appspider, Arachni)	N/A
Session Fixation Attack	Black-box	Hybrid	Automated	Code and web vulnerability scanner (e.g. Appscan, Appspider, Arachni)	N/A
Session Fixation Attack	White-box	Dynamic	Manual	N/A	N/A
Session Fixation Attack	Grey-box	Dynamic	Manual	N/A	N/A
Session Fixation Attack	Black-box	Dynamic	Manual	N/A	N/A
Session Fixation Attack	White-box	Static	Manual	N/A	N/A
Session Fixation Attack	Grey-box	Static			

Testing the Session Hijacking Attack

Testing Session Hijacking attack requires two machines connected to the same network. The attacker would need to intercept the network traffic from the victim's machine where the session data is being exchanged. It is also important to note that it is possible to hijack cookies from other machines on the same network.

Here are the steps to test a Session Hijacking attack:

Install a network sniffing tool (such as Wireshark) on both machines to monitor network traffic.

Use the sniffing tool to identify any cookies used in the session interaction between the two machines. Note that certain browsers may encrypt the cookie data so the cookie might appear to be encrypted.

Intercept the cookie data on the attacker’s machine - the attacker should now have temporary access to the session data.

Use the sniffing tool to analyze the data transferred between the two machines and confirm if the session data has been compromised.

Try to login to the victim’s session with the stolen credentials or cookies.

If successful, the attacker has hijacked the session and should be able to access the victim’s session data.

Log out of the session on the attacker's machine and remove all evidence of the session hijacking attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Session Hijacking Attack	White-box	Dynamic	Code review	Burp Suite	N/A

Session Hijacking Attack	Grey-box	Static	Vulnerability assessment	Nessus	N/A
Session Hijacking Attack	Black-box	Hybrid	Penetration testing	Nmap	N/A

Testing the Access Point Hijacking Attack

Testing Access Point Hijacking Attack

To test an Access Point (AP) Hijacking attack, follow these steps:

1. Set up a rogue AP that mimics the legitimate AP. - Ensure that both SSIDs (Service Set Identifiers) and MAC addresses are identical.
2. Create a beacon frame that advertises the rogue AP.
3. Capture the beacon frame using a network monitoring tool. - This will give you an idea of when the legitimate AP is being impersonated.
4. Monitor for clients that attempt to authenticate with the rogue AP. - This will provide further evidence of the attack.
5. Confirm that the rogue AP is functioning by trying to access it using a client device.
6. Review logs for a record of the attack.
7. Attempt to decrypt any traffic captured from the attack. - This will give you an understanding of the data that was stolen.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Access Point Hijacking Attack	White-box	Dynamic	Manual	N/A	N/A
Access Point Hijacking Attack	Grey-box	Dynamic	Automation	Nessus	N/A
Access Point Hijacking Attack	Black-box	Static	Manual	Nmap	IOS/Android

Testing the Cellular Rogue Base Station Attack

Prerequisite:

- Wireless Access Point
- Wireless testing tool/software

Setup:

- Place the wireless access point with in the testing environment.

Testing Procedure:

- Using wireless testing tool/software, create a wireless environment and record the signal strength and wireless connection status of the legitimate wireless base station.
- Place the rogue base station in the same environment and power it up.
- Again measure the signal strength and wireless connection status of the rogue base station and compare it with the legitimate base station.
- If the rogue base station is showing higher signal strength than the legitimate one, then it indicates that the rogue base station is stronger than the legitimate one.

Analysis:

- Analyze the result of the comparison and determine whether a cellular rogue base station attack is taking place in the environment or not.
- Use the information gathered from the testing to focus the countermeasures for mitigating the attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Cellular Rogue Base Station Attack	White-box	Dynamic	Manual testing	N/A	Android/iOS
Cellular Rogue Base Station Attack	Grey-box	Static	Automated testing	Veracode	Android/iOS
Cellular Rogue Base Station Attack	Black-box	Hybrid	Penetration testing	Metasploit	Android/iOS

Testing the GPS Spoofing Attack

Testing GPS spoofing can be done in several ways:

Open Path Verification: This involves verifying the reported path by comparing it to the landscape or terrain around it. This method can be used to identify any inconsistencies or anomalies in the path reported by the GPS device.

Dual Path Verification: This involves using two different methods to verify the accuracy of the path reported by the GPS device. The two methods are usually triangulation and celestial navigation.

Data Acquisition and Evaluation: This method involves collecting and analyzing data from the GPS device in order to identify any discrepancies or errors in its reported path. This is particularly useful when multiple GPS devices are being used in order to verify their reported paths.

Monitoring: This technique involves continuously monitoring the reported path of the GPS device in order to detect any suspicious behavior or inconsistencies. This method is often used when attempting to detect a spoofed signal.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
GPS Spoofing Attack	White-box	Dynamic	Automation	Appium	Android & iOS
GPS Spoofing Attack	Grey-box	Static	Manual	GPSLogger	Android & iOS
GPS Spoofing Attack	Black-box	Hybrid	Automation	MonkeyTalk	Android & iOS

Testing the Botnet Attack

Testing a Botnet Attack

Botnets are malicious networks of computers infected with malware. They are typically used to generate unwanted traffic on websites, servers, networks, and services. In this article, we will discuss how to test a botnet attack.

Prerequisites

- Understanding of Botnet attacks
- Knowledge of common botnet testing tools

Preparation

1. Install and configure any necessary software, such as emulators and virtual machines.
2. Prepare a secure environment for testing the botnet attack. This includes establishing security measures for the testing environment and isolating it from the production environment.
3. Configure the necessary hardware and software.

Running the Test

1. Establish a connection between the test environment and an external source system.
2. Introduce the botnet code into the test environment.
3. Activate and monitor the botnet code, tracking how it propagates and gathers information.
4. Test the botnet's resilience against security measures.
5. Collect data on botnet activity and performance and analyze it.

Conclusion

Testing a botnet attack is essential in order to assess the effectiveness of security measures and ensure that all systems are adequately protected. With the right preparation and the right tools, it is possible to conduct effective tests and prevent botnet attacks.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Botnet Attack	White-box	Dynamic	Protocol fuzzing	nmap	Android
Botnet Attack	Black-box	Dynamic	Activity monitoring	Wireshark	iOS
Botnet Attack	Grey-box	Static	Source code review	CheckMarx	Android/iOS

Testing the Malware as a Service Attack

Testing Malware as a Service (MaaS) involves assessing the efficacy of a MaaS solution in detecting, preventing, and responding to potential malicious actors. It is essential to ensure that the MaaS provider has the resources and capabilities to provide effective security and protection while adhering to industry best practices.

The following steps should be taken in order to test the efficacy of a MaaS solution:

Define test objectives: The first step before testing is to clearly define what needs to be tested and what is to be achieved by the end of the testing process. For example, the objective could be to test and verify the MaaS solution's ability to detect, prevent, and respond to malicious activity.

Select test environment: Selecting the appropriate environment for testing is essential to ensure a successful test. The environment should be as similar to the actual production environment as possible.

Execute test scenarios: Once the environment is set up, the next step is to execute the various test scenarios which are designed to evaluate the effectiveness of the MaaS solution.

Record observations and results: During the testing process, the tester should make a record of the observations and results from the various test scenarios. This data can then be used to analyze the efficacy of the MaaS solution.

Analyze test results and draw conclusions: After the testing is complete, the testers should analyze the results in order to determine the efficacy of the MaaS solution. They should also draw conclusions on the ability of the solution to detect, prevent, and respond to various types of malicious activity.

Testing the efficacy of a MaaS solution is essential to ensure that the solution can provide the appropriate level of security and protection. Following the steps outlined above will help ensure that tests are successful and meaningful.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
API	White-Box	Dynamic	Automated	Webproxy	Android & iOS
Network Environment	White-Box	Static	Manual	Nessus Security Center	Android & iOS
Application Environment	Grey-Box	Hybrid	Automated	Burp Suite	Android & iOS
End-User Environment	Black-Box	Dynamic	Automated	Nmap	Android & iOS

Testing the Bypassing Physical Security Attack

Testing Bypassing Physical Security

Physical security is an important part of any security system and must be tested regularly to ensure it is effective. Here are some tips for testing the physical security of your system:

Perform regular patrols around your premises to assess the effectiveness of your physical security measures.

Monitor your security cameras to ensure they are in proper working order and can adequately capture any suspicious activity.

Use physical penetration tests to evaluate the security of doors, windows, and other access points.

Regularly inspect locks, alarms and other security equipments.

Test access control systems to verify that identities can be authenticated and access is denied properly.

Cart tests can also be used to determine if it is possible to bypass interaction with security personnel.

Assess your system for weaknesses, such as unlocked doors, weak locks, unmonitored cameras and access points not adequately secured.

Regularly review your system’s response to alarm systems and other security breaches.

Use physical security audit to track changes in physical security measures and document any breaches.

By following these tips and regularly testing your physical security systems, you can ensure the safety of your premises and the data contained within.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Physical Security	White-box	Dynamic	Application Penetration Test	Metasploit, Core Impact	Android, iOS
Physical Security	Black-box	Static	Code Review	Retire.js, Checkmarx	Android, iOS
Physical Security	Grey-box	Hybrid	Hybrid Analysis	Burp Suite Proxy, Acunetix	Android, iOS

Testing the Physical Theft Attack

Testing physical theft is not something that can be easily tested, as it requires physical guards and surveillance of the premises. However, here are some steps that can be taken to reduce the likelihood of theft:

1. Install motion detectors at all entry and exit points of the building.
2. Ensure adequate lighting in and around the premises.

3. Implement security cameras both inside and outside the building.
4. Install tamper-proof locks on all entrance and exit points.
5. Issue swipe cards or security badges to all personnel.
6. Vet all personnel before hiring them.
7. Ensure that staff do not leave valuables unattended.
8. Carry out regular security audits.
9. Make sure that all personnel report any suspicious activities.
10. Ensure that valuables are not stored in unsecured areas.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Hardware Security	White-box	Dynamic	fuzz testing	[Oss Fuzz] (https://oss-fuzz.com/)	Android, iOS
Hardware Security	White-box	Static	code reviews	[Black Duck] (https://www.blackducksoftware.com/)	Android, iOS
Hardware Security	White-box	Hybrid	vulnerability assessment	[Acunetix] (https://www.acunetix.com/)	Android, iOS
Hardware Security	Grey-box	Dynamic	payload-based attack	[Burp Suite Pro] (https://portswigger.net/burp/pro)	Android, iOS
Hardware Security	Grey-box	Static	architecture review	[Microsoft Security Risk Detection] (https://www.microsoft.com/en-us/security-risk-detection/)	Android, iOS
Hardware Security	Grey-box	Hybrid	penetration testing	[OWASP Zed Attack Proxy] (https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project)	Android, iOS
Hardware Security	Black-box	Dynamic	fuzz testing	[Sulley Fuzzer] (https://github.com/OpenRCE/sulley)	Android, iOS
Hardware Security	Black-box	Static	code reviews	[CheckMarx] (https://www.checkmarx.com/)	Android, iOS
Hardware Security	Black-box	Hybrid	vulnerability assessment	[Netsparker] (https://www.netsparker.com/)	Android, iOS

Testing the VM Migration Attack

Testing VM Migration

To test Virtual Machine (VM) Migration, the following steps should be taken:

Prepare the source and target environment: Set up the source/target environments and make sure all necessary applications and services that the VM needs is installed and running properly.

Configure Migration: Configure the VM's settings such as network, CPU, memory, storage, etc. to be compatible with the source/target environment and verify they are the same between both environments.

Test Network Connectivity: Make sure that the source and target environments are properly connected over the network and that the necessary ports are open and accessible.

Test Migration Tool: Test the migration tool being used, if any, to make sure it is working properly.

Perform Initial Backup: Take an initial backup of the source environment to ensure the VM is properly backed up and can be restored if the migration fails.

Begin Migration: Begin migrating the VM from the source to the target environment.

Monitor Progress: Monitor the migration process and make sure no errors or warnings occur during the migration.

Validate Migration: Once the migration is completed, validate that all services, applications, and settings are intact and working properly.

Restore Backup: Restore the initial backup as a contingency plan against any unforeseen errors or failures.

Clean Up Source: Clean up the source environment, including removing any unnecessary files, services, or applications that are no longer needed or can be moved over to the target environment.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
VM Migration	White-box	Dynamic	Manual Testing	N/A	N/A
VM Migration	Grey-box	Dynamic	Exploratory Testing	N/A	N/A

VM Migration	Black-box	Dynamic	Automated Testing	Burp Suite	N/A
VM Migration	White-box	Static	Code Review	Veracode	N/A
VM Migration	Grey-box	Static	Risk Assessment	N/A	N/A
VM Migration	Black-box	Static	Penetration Testing	Metasploit	N/A
VM Migration	White-box	Hybrid	Traceability Testing	N/A	N/A
VM Migration	Grey-box	Hybrid	Security Testing	HP WebInspect	N/A
VM Migration	Black-box	Hybrid	Application Security Testing	Acunetix	iOS/Android

Testing Rowhammer Attack

Rowhammer is a security vulnerability due to which a malicious user can gain access to memory by repeatedly accessing certain memory locations (rows of memory cells).

The best way to test for Rowhammer is to use a specialized tool known as RAMBleed. RAMBleed is an open source tool developed by Google researchers which can detect Rowhammer attack traces in the RAM. The following steps can be followed to use RAMBleed to test for Rowhammer:

1. Install RAMBleed on the system.
2. Start the RAMBleed server.
3. Configure the RAMBleed client on the system to be tested.
4. Execute the RAMBleed client and wait to see if any attack traces are detected.
5. Follow the instructions to interpret the output of RAMBleed.
6. If any attack traces were detected, take the necessary steps to fix the vulnerability.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Memory	White box	Dynamic	Assertion	DRAMstress	Not applicable
Memory	Grey box	Static	Verification	Memsmash	Not applicable
Memory	Black box	Hybrid	Execution	Hammertime	Not applicable

Testing the VM Escape Attack

There are a few approaches to testing for VM Escape (also known as Virtual Machine Escape).

Code Review: A comprehensive code review can help identify potential vulnerabilities present in the code which, if exploited, could lead to a VM Escape. This involves a thorough, line-by-line examination of the source code, using techniques such as manual inspection, automated static code analysis and fuzzing.

Exploit Testing: A series of exploitation techniques can be used to try to break out of the virtualized environment. These could include things such as exploiting buffer and account overflow vulnerabilities, command injection and malicious software attempts.

Penetration Testing: Penetration testing involves the use of specialized tools and techniques to break into the virtual environment and gain access to critical resources. This could involve standard methods such as brute force attacks, port scanning, and social engineering.

External Auditing: External auditing involves examining the operating environment from the outside, examining access controls, security protocols and other measures. This can identify any weak points that would allow for a successful VM Escape.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
VM Escape Attack	White-box	Dynamic	Fuzzing	Peach Fuzzer	N/A
VM Escape Attack	Grey-box	Static	Signature Detection	Codenomicon Defensics	N/A
VM Escape Attack	Black-box	Hybrid	Exploitation	Metasploit	N/A