Final Security Requirements Report

Mobile Platform	iOS App ; IoT System
Application domain type	m-Payment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Channel-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	PostgreSQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Both
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	C/C++/Objective-C
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	Public Cloud
HW Authentication	Symmetric Key
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ; NFC
Device or Data Center Physical Access	Yes

Confidentiality

Confidentiality Requirement in Security Engineering

Confidentiality is a critical security requirement in the engineering of any system. It ensures that only authorized users can access sensitive information, protecting the system from unwanted disclosure of information. Confidentiality can be achieved through the implementation of encryption, authentication, and access control mechanisms. Additionally, physical security measures such as restricted access to areas where confidential data is located and restricting the number of personnel authorized to access it can enhance the security of the system.

Confidentiality is also an important element of security engineering processes. Engineering teams must develop mechanisms that protect from unauthorized disclosure of information while still allowing authorized users to access it. Such mechanisms may involve the implementation of encryption, authentication, and access control mechanisms, as well as the use of secure coding practices to ensure that confidential information is not inadvertently disclosed through coding errors. Additionally, engineering teams must ensure that the system maintains an adequate level of confidentiality throughout its entire life cycle, as any

Warning:

If we fail to guarantee confidentiality requirements, the following could happen to the system:

A third-party could access sensitive data stored in the system, potentially leading to unauthorized disclosures, identity theft, financial losses, and legal issues.

Malicious actors may use the data to launch targeted attacks on the system, with the intent of disrupting business operations and gaining access to confidential information for their own gain.

The system may become vulnerable to exploits that can be used to gain access to private data, or to interfere with its operations.

Systems can become prone to denial of service attacks, where legitimate requests are blocked and malicious requests are allowed in order to slow down service or gain access to systems.

Sensitive information may be exposed to unauthorized personnel, leading to the potential loss of competitive advantage in the marketplace.

Integrity

Integrity Requirement in Security Engineering

Integrity is one of the key security requirements that must be addressed in security engineering. This requirement is used to ensure the accuracy and completeness of data and systems.

The integrity requirement helps to protect against malicious attacks, such as data tampering, data manipulation, or unauthorized access. It also helps to ensure that data is kept secure and stored in its original form.

Other aspects of integrity in security engineering include:

Data authentication: Data authentication is a process of verifying the accuracy of data by validating digital signatures, checksums, encryption, and other techniques.

Access control: Access control measures help to ensure that only authorized users have access to data and systems.

Backup and recovery: Backup and recovery processes help to maintain data integrity in case of system failure or malicious attacks.

Logging and auditing: Logging and auditing are processes to track user activity and ensure data integrity.

Warning:

If we fail to guarantee integrity requirements, the system may be compromised in a variety of ways. Some possible outcomes include:

- Data can be corrupted or modified without the knowledge of the user
- Hackers can break into the system and steal sensitive information
- Malicious software can be installed in the system
- Unauthorized users may be granted access to the system
- System performance can suffer due to malicious attacks or malicious code
- Security of the system can be compromised

Availability

Availability Requirement in Security Engineering

Availability requirement in security engineering refers to the need for secure systems to remain operational and available to users when required. Achieving availability without sacrificing security is often a challenge, as attackers may attempt to disrupt system availability in order to deny service or gain access to sensitive information.

Security engineering must therefore consider and account for the availability of system components, including network connections, storage systems, and web applications. Examples of availability requirements include:

- Ensuring that system services remain available despite distributed denial of service (DDoS) or other types of attacks.
- Preventing unauthorized users from accessing systems by restricting access privileges.
- Defending against malicious code, such as viruses, worms, and Trojans.
- Developing backup strategies and business continuity plans to ensure that systems maintain acceptable levels of service.
- Measuring service availability in order to identify areas of improvement.

Availability is a key concern in security engineering

Warning:

If availability requirements are not met, then the system may suffer from:

- Decreased performance or slow response times
- Outages or downtime
- Higher than expected resource usage
- Lowered security
- Loss of data
- Increased maintenance costs

Authentication

Authentication Requirement in Security Engineering

Authentication is a security measure which requires a user to prove their identity before accessing resources or taking certain actions on a system. Authentication requirements are essential for anything related to security engineering, as they are the basis for ensuring access to the system and resources is being done by authorized personnel. Authentication requirements can typically involve one or more of the following:

Username and Password: A unique username and password combination is often the most common way to authenticate an individual. Passwords should meet the security guidelines set by the organization and must be changed regularly.

Multi-Factor Authentication: This is an additional layer of security which requires users to provide two or more pieces of evidence to prove their identity. This could include personal information such as a security code, or additional authentication methods such as biometrics or a one-time token.

PIN/Password Combo: PIN numbers may also

Warning:

If we fail to guarantee authentication requirements, the system may become vulnerable to security threats. Malicious attackers may gain unauthorized access to the system and perform malicious activities such as data theft, manipulation of data, or denial of service. This could lead to significant financial losses, reputational damage, and legal ramifications.

Authorization

Authorization Requirements in Security Engineering

Authorization requirements are security measures that ensure only authorized personnel can access a system or database. These requirements are designed to protect systems and data from malicious activity or unauthorized access. Authorization requirements include authentication mechanisms, role-based access control, and audit logging.

Authentication mechanisms are designed to ensure that users or processes are who they say they are. Authentication can be done by combining something a user knows (e.g. a password) with something they have (e.g. a token) or something they are (e.g. a biometric fingerprint scan).

Role-based access control (RBAC) enables officials to assign user roles that limit access to certain functions and data. RBAC can be used to prevent access of sensitive information to prevent data leaks or damage to the system.

Audit logging is a process of tracking and recording changes in system activities and records. Auditing logs can be used for troubleshooting

Warning:

If we fail to guarantee authorization requirements, it can lead to a number of consequences for the system:

- Unauthorized users may have access to confidential information or make changes to the system without permission.
- Data stored in the system may be manipulated or corrupted by unauthorized users.
- System performance could be significantly impacted due to malicious activity.
- System security may be compromised, resulting in a breach of sensitive information.
- Legitimate users may be denied access to the system due to incorrect permissions.

Non-repudiation

Non-repudiation is a term used in information security that refers to a legal concept describing the assurance that someone cannot deny that they performed a certain action. It is a critical security requirement for many businesses, especially in the fields of finance and e-commerce.

In security engineering, non-repudiation refers to the technical capability of preventing a source from denying having performed an action, such as sending a message or making a payment. To achieve non-repudiation, various cryptographic techniques can be used, such as digital signatures and Secure Hash Algorithm (SHA).

Non-repudiation requirement in security engineering

Non-repudiation is a critical security requirement in many organizations, as it helps ensure that the source of a transaction or message cannot be denied at a later point in time. To guarantee non-repudiation, security engineering must employ various cryptographic techniques such as digital signatures, Secure Hash Algorithm (SHA), or other methods of

Warning:

Consequences of Failing to Guarantee Non-Repudiation Requirements

If a system fails to guarantee non-repudiation requirements, it can lead to a variety of serious consequences in both the short and long term. Some of these consequences include:

- Loss of customer confidence and potential decrease in revenue due to lack of trust
- Increased risk of fraudulent activities and unauthorized transactions
- Damage to brand reputation
- Legal issues and possible fines/penalties due to non-compliance with regulations
- Inability to prove ownership or responsibility for an action
- Difficulty in resolving disputes between parties

Accountability

Accountability Requirement in Security Engineering

Security engineering is the process of designing and building secure systems. A key feature of security engineering is the requirement for accountability. This means that when something goes wrong with a system, it must be possible to determine who was responsible for the incident and take appropriate action.

Accountability has several components including:

Auditable Events: Events in the system should be logged and tracked to allow for audit and investigation.

Identification: Access controls must be in place to identify and authenticate users who interact with the system.

Authorization: Users should only be given access to resources that they have been explicitly authorized to access.

Privileges and Access Control: Access to system components must be managed and restricted to only users who have the necessary privileges and clearance.

Data Protection: Sensitive data stored within the system must be protected from

Warning:

If we fail to guarantee accountability requirements, the system will become insecure and vulnerable. This could lead to data being exposed to unauthorized persons or malicious actors. It can also lead to data breaches, where confidential and sensitive information is leaked. This could result in financial or reputational damage to the organization. Furthermore, without accountability, it can be difficult to prove who is responsible for any wrongdoing or breaches of security.

Reliability

Reliability Requirement in Security Engineering

- The system must be able to detect and record any unauthorized access attempts.
- The system must provide an adequate level of fault tolerance.
- The system must be able to inform the users of any security breaches so action can be taken.
- The system must be able to withstand natural disasters or other forms of attack.
- The system must be able to protect the confidentiality, integrity, and availability of data.
- The system must be able to detect malicious code or errors that could cause potential data loss.
- The system must be able to restore any data that is lost or corrupted in the event of an attack.
- The system must be able to notify and inform appropriate personnel of any unauthorized access attempts and malicious activity.
- The system must be able to protect itself from malicious attack and be resilient to any changes in the environment.
- The system must be designed

Warning:

If reliability requirements are not met, the system may experience decreased performance, data loss, or downtime. This could result in a loss of user confidence in the system, decreased efficiency, and potentially loss of revenue. It could also result in customers going elsewhere for services and products, leading to a decline in profits and market share.

Physical Security

Physical Security

Physical Security is the protection of people, property, and information onsite. It involves protecting physical assets from potential risks such as fire, theft, vandalism, and natural disasters.

The following should be considered when designing physical security:

- Access Control: Controlling access to the facility, equipment, resources and data with authentication mechanisms such as lock and key, bio-metric, security guards, and CCTV surveillance.
- Environmental Management: Monitoring and controlling the environmental conditions within the facility, such as temperature, humidity, fire/smoke detection, seismic activity, and water leaks.
- Emergency Response: In the event of an emergency, it is important to have comprehensive procedures in place for responding quickly and effectively.
- Equipment Protection: Protecting all hardware and critical equipment with alarms/sensors and preventing tampering.
- Systems Security: Ensuring the integrity of the digital systems and networks within the facility by implementing security measures, such as

Warning:

Potential Consequences of Failing to Guarantee Physical Security Requirements

- Theft or destruction of hardware components and systems.

- Potential exposure of confidential data or information.
- Unauthorized access to sensitive systems, networks, or data.
- Increased risk of malicious attacks.
- Increased risk of denial-of-service attacks.
- Financial losses due to equipment damage or data theft.
- Loss of customer trust, resulting in decreased or lost business.
- Legal action due to data breaches.

Forgery Resistence

Forgery Resistance is an important requirement in security engineering that aims to protect data and systems from attempts to counterfeit, clone, counterfeit, or alter the identity of an entity. It can be achieved through various means, including:

Cryptography: Cryptography is the process of transforming data into a form that only the intended recipient can read. It can prevent forgery by making it impossible for anyone to create or alter data without knowing the recipientâ€™s authentication key.

Digital Signatures: A digital signature is a way of verifying the identity of a user or verifying the integrity of a message. It uses a private/public key system to ensure that the digital signature can only be created and verified by the proper party.

Tamper-proofing: Tamper-proofing techniques such as watermarking, sealing, and inlays help prevent data from being altered or forged without authorization.

Strong Authentication: Strong authentication methods like

Warning:

If we fail to guarantee the forgery resistance requirement, the system would be vulnerable to forgery or counterfeiting of documents, which could lead to potential fraud, illegitimate access to resources, data theft, and other malicious activities. This could have serious implications for the security and integrity of the system, as well as the data it contains. Furthermore, it could open up the system to legal and financial liabilities if it is determined that the failure to guarantee forgery resistance enabled a malicious attack.

Tamper Detection

Tamper Detection

Tamper detection is a requirement in security engineering that detects and alerts for any changes made to the system. This type of security helps to ensure that confidential information is safe and not accessible to unauthorized personnel. Tamper detection technology can detect any changes made to the system such as adding or removing files, changing configurations, and more. Additionally, tamper detection can trigger other protective measures such as locking down a system or triggering alerts when a malicious attack is detected.

Warning:

If we fail to guarantee tamper detection, the security of the system can be compromised. Attackers can try to break into the system, modify data, or even inject malicious code. This can cause a variety of problems such as system crashes, data corruption, and malicious activity. Without the assurance of tamper detection, the system may be vulnerable to malicious activity, and the risk of suffering from a security breach increases.

Data Freshness

Data Freshness is a requirement in security engineering which is concerned with ensuring that data requires updating periodically and is not outdated.

In order to ensure data freshness, organizations must have a defined and enforced policy regarding when and how often the data must be updated. Some organizations may require daily or even hourly updates, while others may adopt a more relaxed approach.

Good data freshness practices also require that data must not be allowed to become stale or out-of-date, and should be regularly monitored to ensure that the data is accurate and up-to-date.

Warning:

If the system fails to guarantee data freshness, there will be a number of consequences. These include:

Unreliable data and results: Data which is not up to date can lead to unreliable insights and inaccurate business decisions.

Missed opportunities and delayed decisions: Using stale data can lead to the loss of potential opportunities, as well as a delay in making decisions.

Lack of trust: By not maintaining fresh data, the system will lose credibility with its users and may be deemed untrustworthy.

Poor customer experience: Data that is not up to date can result in a poor customer experience, leading to dissatisfaction and a loss of customers.

Confinement

Confinement requirement in security engineering

Confinement requirements in security engineering are security requirements that ensure that privileged operations and activities (both internal and external) are constrained so that they cannot be abused or manipulated for malicious purposes. These requirements are generally implemented using a combination of hardware, software, processes, policies, and other safeguards. By confining privileged operations and activities within a secure boundary and ensuring that only authorized and authenticated parties can access these operations and activities, confidential information and systems remain safe and secure.

Warning:

When confinement requirements are not met, the system can be vulnerable to security vulnerabilities and breaches. Without proper boundaries, malicious actors can have unrestricted access to the system, allowing them to tamper with data, modify settings, or take complete control over the system. This could lead to malicious activities such as unauthorized data exfiltration, espionage, and sabotage. Furthermore, if the system is not properly secured, then attackers can use this access to launch Denial-of-Service (DoS) attacks, spread malware, or install malicious software.

Interoperability

Interoperability Requirement in Security Engineering

Interoperability is an important requirement in security engineering that requires different systems and components to be able to work together. It involves the ability to exchange data, process signals, interpret codes and store information, allowing systems to interact and coordinate their actions.

Interoperability allows for the secure transfer of data between different systems, ensuring the integrity and availability of the data being exchanged. It also helps to ensure that unauthorized persons are unable to access the data and that any changes to the data can be tracked and reversed.

Interoperability also helps to ensure that the security policies of different systems are respected and enforced and that any changes in security policies are properly implemented and updated. It also helps to reduce the risks associated with implementing incompatible systems and components.

By ensuring systems are interoperable, security risks can be minimized, data can be securely transferred between different entities, and businesses can better manage and control their networks and systems.

Warning:

If we fail to guarantee interoperability requirements, the system could become unreliable. It would be difficult to connect with other systems or devices, and the system would be unable to exchange information with other systems and devices. This would lead to a lack of functionality as the system would not be able to communicate or interact with other systems and devices. Additionally, it could lead to security risks as the system would be more vulnerable to attacks and unauthorized access.

Data Origin Authentication

Data Origin Authentication

Data origin authentication is a security engineering requirement that aims to verify that data is sent securely and accurately, and that it is originating from an authenticated and trusted source. It aims to ensure that data sent from one location to another has not been modified in any way.

Data origin authentication typically involves techniques such as message authentication codes (MACs), digital signatures, and public-key infrastructure (PKI) protocols. It can also involve two-factor authentication and the use of cryptography. These techniques can be used to ensure that data is sent securely and with integrity, meaning that the data has not been tampered with or modified in transit.

Warning:

The consequences of failing to guarantee data origin authentication

- Untrusted data: Data integrity and authenticity could be compromised as untrusted sources may be allowed into the system, leading to data leakage, manipulation or other malicious activities.
- Reduced trust: Without authentication, it will be difficult to establish trust in any data or systems.
- Security breaches: It is much more likely that malicious actors could infiltrate the system and gain access to confidential information without authentication.
- Loss of data: Without authentication, there would be no way to confirm the accuracy or veracity of the data, leaving the system vulnerable to data loss.
- Increased risk: Without authentication, organizations may be more susceptible to cyber-attacks as malicious actors could easily access confidential data.

Final Security Good Practices

Mobile Platform	iOS App ; IoT System
Application domain type	m-Payment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Channel-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	PostgreSQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Both
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	C/C++/Objective-C
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	Public Cloud
HW Authentication	Symmetric Key
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ; NFC
Device or Data Center Physical Access	Yes

IoT Security Best Practices Guidelines

Internet of Things (IoT) devices fall into three main categories:

- Sensors, which gather data;
- Actuators, which effect actions;
- Gateways, which act as communication hubs and may also implement some automation logic.

All these device types may stand alone or be embedded in a larger product. They may also be complemented by a web application or mobile device app and cloud based service. IoT devices, services and software, and the communication channels that connect them, are at risk of attack by a variety of malicious parties.

Malicious intent commonly takes advantage of poor design, but even unintentional leakage of data due to ineffective security controls can also bring dire consequences to consumers and vendors. Thus it is vital that IoT devices and services have security designed in from the outset.

Classification of Data

- Define a data classification scheme and document it;
- Assess every item of data stored, processed, transmitted or received by a device and apply a data classification rating to it;
- Ensure the security design protects every data item and collections of items against unauthorised viewing, changing or deletion, to at least its classification rating or higher.

Physical Security

- Any interface used for administration or test purposes during development should be removed from a production device, disabled or made physically inaccessible;
- All test access points on production units must be disabled or locked, for example by blowing on-chip fuses to disable JTAG;
- If a production device must have an administration port, ensure it has effective access controls, e.g. strong credential management, restricted ports, secure protocols etc.;
- Make the device circuitry physically inaccessible to tampering, e.g. epoxy chips to circuit board, resin encapsulation, hiding data and address lines under these components etc;
- Provide secure protective casing and mounting options for deployment of devices in exposed locations;
- For high-security deployments, consider design measures such as active masking or shielding to protect against side-channel attacks.

Device Secure Boot

- Make sure the ROM-based secure boot function is always used. Use a multi-stage boot loader initiated by a minimal amount of read-only code;
- Use a hardware-based tamper-resistant capability (e.g. a microcontroller security subsystem, Secure Access Module (SAM) or Trusted Platform Module (TPM)) to store crucial data items and run the trusted authentication/cryptographic functions required for the boot process. Its limited secure storage capacity must hold the read-only first stage of the boot loader and all other data required to verify the authenticity of firmware;

- Check each stage of boot code is valid and trusted immediately before running that code. Validating code immediately before its use can reduce the risk of attacks;
- At each stage of the boot sequence, wherever possible, check that only the expected hardware is present and matches the stage's configuration parameters;
- Do not boot the next stage of device functionality until the previous stage has been successfully booted;
- Ensure failures at any stage of the boot sequence fail gracefully into a secure state, to ensure no unauthorised access is gained to underlying systems, code or data. Any code run must have been previously authenticated.

Secure Operating System

- Include in the operating system (OS) only those components (libraries, modules, packages etc.) that are required to support the functions of the device;
- Shipment should include the latest stable OS component versions available;
- Devices should be designed and shipped with the most secure configuration in place;
- Continue to update OS components to the latest stable versions throughout the lifetime of a deployed device;
- Disable all ports, protocols and services that are not used;
- Set permissions so users/applications cannot write to the root file system;
- If required, accounts for ordinary users/applications must have minimum access rights to perform the necessary functions. Separate administrator accounts (if required)will have greater rights of access. Do not run anything as root unless genuinely unavoidable;
- Ensure all files and directories are given the minimum access rights to perform the required functions;
- Consider implementing an encrypted file system.

Application Security

- Applications must be operated at the lowest privilege level possible, not as root. Applications must only have access to those resources they need;
- Applications should be isolated from each other. For example, use sandboxing techniques such as virtual machines, containerisation, Secure Computing Mode (seccomp), etc
- Ensure compliance with in-country data processing regulations;
- Ensure all errors are handled gracefully and any messages produced do not reveal any sensitive information;
- Never hard-code credentials into an application. Credentials must be stored separately in secure trusted storage and must be updateable in a way that ensures security is maintained;
- Remove all default user accounts and passwords;
- Use the most recent stable version of the operating system and libraries;
- Never deploy debug versions of code. The distribution should not include compilers, files containing developer comments, sample code, etc.;
- Consider the impact on the application/system if network connectivity is lost. Aim to maintain normal functionality and security wherever possible.

Credential Management

- A device should be uniquely identifiable by means of a factory-set tamper resistant hardware identifier if possible;
- Use good password management techniques, for example no blank or simple passwords allowed, never send passwords across a network (wired or wireless) in clear text, and employ a secure password reset process;
- Each password stored for authenticating credentials must use an industry standard hash function, along with a unique salt value that is not obvious (for example, not a username);
- Store credentials or encryption keys in a Secure Access Module (SAM), Trusted Platform Module (TPM), Hardware Security Module (HSM) or trusted key store if possible;
- Aim to use 2-factor authentication for accessing sensitive data if possible;
- Ensure a trusted & reliable time source is available where authentication methods require this, e.g. for digital certificates;
- A certificate used to identify a device must be unique and only used to identify that onedevice. Do not reuse the certificate across multiple devices;
- A "factory reset" function must fully remove all user data/credentials stored on a device.

Encryption

- When configuring a secure connection, if an encryption protocol offers a negotiable selection of algorithms, remove weaker options so they cannot be selected for use in a downgrade attack;
- Store encryption keys in a Secure Access Module (SAM), Trusted Platform Module (TPM), Hardware Security Module (HSM) or trusted key store if possible;
- Do not use insecure protocols, e.g. FTP, Telnet;
- It should be possible to securely replace encryption keys remotely;
- If implementing public/private key cryptography, use unique keys per device and avoid using global keys. A device's private key should be generated by that device or supplied by an associated secure credential solution, e.g. smart card. It should remain on that device and never be shared/visible to elsewhere.

Network Connections

- Activate only those network interfaces that are required (wired, wireless - including Bluetooth etc.);
- Run only those services on the network that are required;

- Open up only those network ports that are required;
- Run a correctly configured software firewall on the device if possible;
- Always use secure protocols, e.g. HTTPS, SFTP;
- Never exchange credentials in clear text or over weak solutions such as HTTP Basic Authentication;
- Authenticate every incoming connection to ensure it comes from a legitimate source.
- Authenticate the destination before sending sensitive data.

Logging

- Ensure all logged data comply with prevailing data protection regulations;
- Run the logging function in its own operating system process, separate from other functions;
- Store log files in their own partition, separate from other system files.
- Set log file maximum size and rotate logs;
- Where logging capacity is limited, just log start-up and shutdown parameters, login/access attempts and anything unexpected;
- Restrict access rights to log files to the minimum required to function;
- If logging to a central repository, send log data over a secure channel if the logs carry sensitive data and/or protection against tampering of logs must be assured;
- Implement log "levels" so that lightweight logging can be the standard approach, but with the option to run more detailed logging when required;
- Monitor and analyse logs regularly to extract valuable information and insight;
- Passwords and other secret information should not ever be displayed in logs.

(https://www.iotsecurityfoundation.org/wp-content/uploads/2019/03/Best-Practice-Guides-Release-1.2.1.pdf)[https://www.iotsecurityfoundation.org/wp-content/uploads/2019

Security Best Practices Guidelines for Authentication

Security Best Practices for Authentication

Authentication is an important part of the security of any system. Here are best practices that should be followed to ensure a secure authentication process:

- Use strong passwords. Passwords should be at least 8 characters long and should include both upper and lowercase letters, numbers and special characters.
- Use multi-factor authentication whenever possible. This requires users to provide additional forms of authentication, such as a one-time code sent to a phone or email.
- Use a security question to protect accounts. This should be a question that is difficult for outsiders to answer but easy for the user to remember.
- Require users to change their password regularly. This helps reduce the risk of stolen credentials.
- Donâ€™t allow the same password to be used again after expiration or change.
- Limit log-in attempts. If too many invalid attempts are made, lock the account.
- Implement a lockout policy. After a certain number of failed attempts, lock the account and require the user to reset the password.
- Monitor user log-ins and suspicious activity.
- Donâ€™t store passwords in plain text. All passwords should be encrypted.
- Use security protocols such as TLS or SSL.
- Keep authentication systems up-to-date with the latest patches and security fixes.
- Ensure that all staff are properly trained on authentication best practices.

Security Best Practices Guidelines for Multifactor Authentication

Security Best Practices Guidelines for Multifactor Authentication

Implement Multi-Factor Authentication (MFA) where appropriate:

- Use MFA to protect critical systems, high-value assets, and sensitive data.
- Utilize a variety of authentication methods, such as biometrics, tokens, etc.

Use a password manager:

- Utilize strong, unique passwords for each of your accounts.
- Leverage an identity and access management system to securely store and manage user credentials.

Monitor user login attempts:

- Monitor user login attempts (e.g. IP addresses, time of day access, etc.).
- Set regular reviews and alerts to detect suspicious account activity.

Stay up-to-date on attack techniques:

- Utilize threat intelligence services to gain awareness about attack trends.
- Continuously monitor industry developments and stay aware of emerging threats.

Educate users:

- Regularly educate users on best practices and the importance of multi-factor authentication.
- Educate users on common attack techniques and how to recognize suspicious activity.

Establish a documented process for user onboarding and offboarding:

- Establish defined roles and detail user access requirements.
- Leverage automation and process documentation to ensure consistency in user provisioning.

Use strong credential standards:

- Use secure passphrases or passwords that are at least 12 characters.
- Utilize multi-factor authentication to reduce security risks associated with weak credentials and passwords.

Automate password rotation:

- Automate the password rotation process to ensure accounts remain secure.
- Require users to periodically update their passwords to detect suspicious activity.

Security Best Practices Guidelines for Authorization

Authorization: Security Best Practices Guidelines

Authorization refers to the process of determining what users or groups of users are able to access certain resources in a system. Ensuring the appropriate security of authorization processes is an important part of maintaining the privacy and security of systems and data.

The following are some best practices to help ensure the proper security of authorization processes:

- Implement multiple authentication factors to provide both authorization and identification.
- Regularly monitor and audit user access to data and systems and ensure that access is only granted to the necessary individuals.
- Follow the principle of least privilege when providing user access to systems and data - only provide users with the least level of access necessary to perform their tasks.
- Follow data segregation and separation of duties to reduce the potential risk of compromised authentication.
- Ensure authorization processes are enforced across all organizational devices and systems.
- Utilize an authorization system that allows for periodic audits and reviews, as well as the ability to track changes.
- Establish protocols and policies that clearly define grant and access management practices.
- Utilize a password management system in order to provide users with secure and easy access to authorization credentials.
- Ensure authorization processes are kept up-to-date with the latest security protocols.
- Monitor for unauthorized access attempts and investigate suspicious activities.
- Provide users and administrators with consistent and continuous authorization training.

Transaction Authorization

Purpose and audience

The Purpose of this cheat sheet is to provide guidelines on how to securely implement transaction authorization to protect it from being bypassed. These guidelines can be used by:

- **Banks** - to define functional and non-functional requirements for transaction authorization.
- **Developers** â€“ to design and implement transaction authorization without vulnerabilities.
- **Pentesters** â€“ to test for transaction authorization security.

Introduction

Some applications use a second factor to check whether an authorized user is performing sensitive operations. A common example is wire transfer authorization, typically used in online or mobile banking applications.

For the purpose of this document we will call that process: *transaction authorization*.

Usage scenarios are not only limited to financial systems. For example: an email with a secret code or a link with some kind of token to unlock a user account is also a special case of transaction authorization. A user authorizes the operation of account unlocking by using a second factor (a unique code sent to his email address). Transaction authorization can be implemented using various methods, e.g.:

- Cards with transaction authorization numbers (TAN),
- Time based OTP tokens, such as OATH TOTP (Time-based One-Time Password),
- OTP sent by SMS or provided by phone
- Digital signature using e.g. a smart card or a smart phone,
- Challenge-response tokens, including unconnected card readers or solutions which scan transaction data from the user's computer screen.

Some of these can be implemented on a physical device or in a mobile application.

Transaction authorization is implemented in order to protect for unauthorized wire transfers as a result of attacks using malware, phishing, password or session hijacking, CSRF, XSS, etc.. Unfortunately, as with any piece of code, this protection can be improperly implemented and as a result it might be possible to bypass this safeguard.

1. Functional Guidelines

1.1 Transaction authorization method has to allow a user to identify and acknowledge significant transaction data

User's computers cannot be trusted due to malware threats. Hence a method that prevents a user from identifying transaction on an external device cannot be considered as secure. Transaction data should be presented and acknowledged using an external authorization component.

Such transaction authorization components should be built using the *What You See Is What You Sign* principle. When a user authorizes a transaction they need to know what they are authorizing. Based on this principle, an authorization method must permit a user to identify and acknowledge the data that is significant to a given transaction. For example, in the case of a wire transfer: the target account and amount.

The decision about which transaction data can be considered as significant should be chosen based on:

- The real risk,
- The technical capabilities and constraints of the chosen authorization method,
- Positive user experience.

For example when an SMS message is used to send significant transaction data, it is possible to send the target account, amount and type of transfer. However, for an unconnected CAP reader it is perceived to be inconvenient for a user to enter these data. In such cases, entering only the most significant transaction data (e.g. partial target account number and amount) can be considered sufficient.

In general, significant transaction data should always be presented as an inherent part of the transaction authorization process. Whereas the user experience should be designed to encourage users to verify the transaction data.

If a transaction process requires a user to enter transaction data into an external device, the user should be prompted for providing specific value (e.g. a target account number). Entering a value without meaningful prompt could be easily abused by malware using social engineering techniques as described in the example in paragraph 1.4. Also, for more detailed discussion of input overloading problems, see here.

1.2 Change of authorization token should be authorized using the current authorization token

When a user is allowed to change authorization token by using the application interface, the operation should be authorized by using his current authorization credentials (as is the case with password change procedure). For example: when a user changes a phone number for SMS codes an authorization SMS code should be sent to the current phone number.

1.3 Change of authorization method should be authorized using the current authorization method

Some applications allow a user to chose between multiple methods of transaction authorization. In such cases, the user should authorize the change in authorization method using his current authorization method. Otherwise, malware may change the authorization method to the most vulnerable method.

Additionally, the application should inform the user about the potential dangers associated to the selected authorization method.

1.4 Users should be able to easily distinguish the authentication process from the transaction authorization process

Malware can trick users in authorizing fraudulent operations, when an application requires a user to perform the same actions for authentication as for transaction authorization. Consider the following example:

- An application is using the same method for user authentication (usually as a second factor to traditional login/password) and for transaction authorization. E.g. by using a OTP token, Challenge-response codes, operation signing using external smartcard, ...
- A malware may present the user a false error message after the first step (authentication to the application) and trick the user into repeating the authentication procedure. The first authentication code will be used by the malware for authentication, whereas the second code would be used to authorize a fraudulent transaction. Even challenge-response schemes could be abused using this scenario as malware can present a challenge taken from a fraudulent transaction and trick the user to provide response. Such an attack scenario is used widely in malware attacks against electronic banking.

In the abovementioned scenario, the same method was used to authenticate the user and to authorize the transaction. Malware can abuse this behavior to extract transaction authorization credentials without the user's knowledge. Social engineering methods can be used despite utilized authentication and operation authorization methods but the application shouldn't simplify such attack scenarios.

Safeguards should allow the user to easily distinguish authentication from transaction authorization. This could be achieved by:

- Using different methods to authenticate and to authorize,
- Or using different actions in an external security component (e.g. different mode of operation in CAP reader),
- Or presenting the user a clear message about what they are "signing" (What You See Is What You Sign Principle).

1.5 Each transaction should be authorized using unique authorization credentials

Some applications are asking for transaction authorization credentials only once, e.g. static password, code sent through SMS, token response. Afterwards a user is able to authorize any transaction during the whole user's session or at least they have to reuse the same credentials each time they need to authorize a transaction. Such behavior is not sufficient to prevent malware attacks because malware will sniff such credentials and use them to authorize any transaction without the user's knowledge.

2. Non-functional guidelines

2.1 Authorization should be performed and enforced server-side

As for all other security controls transaction authorization should be enforced server-side. By no means it should be possible to influence the authorization result by altering data which flows from a client to a server, e.g. by:

- Tampering with parameters that contain transaction data,
- Adding/removing parameters which will disable authorization check,
- Causing an error.

To achieve this, security programming best practices should be applied, such as:

- Default deny.
- Avoiding debugging functionality in production code.

To avoid tampering, additional safeguards should be considered. For example by cryptographically protecting the data for confidentiality and integrity and while decrypting and verifying the data server side.

2.2 Authorization method should be enforced server side

When multiple transaction authorization methods are available to the user. The server should enforce the use of the current authorization method chosen by the user in the application settings or enforced by application policies. It should be impossible to change an authorization method by manipulating the parameters provided from the client. Otherwise, malware can downgrade an authorization method to a less or even the least secure authorization method.

This is especially important when an application is developed to add a new, more secure authorization method. It is not very rare,that a new authorization method is built on top of an old codebase. As a result, when a client is sending parameters using the old method, the transaction may be authorized, despite the fact that the user has already switched to a new method.

2.3 Transaction verification data should be generated server-side

When significant transaction data are transmitted programmatically to an authorization component, extra care should be put into denying client modifications on the transaction data at authorization. Significant transaction data that has to be verified by the user, should be generated and stored on a server, then passed to an authorization component without any possibility of tampering by the client.

A common anti pattern is to collect significant transaction data client-side and pass it to the server. In such cases, malware can manipulate these data and as a result, show faked transaction data in an authorization component.

2.4 Application should prevent authorization credentials brute-forcing

When transaction authorization credentials are sent to the server for verification, an application has to prevent brute-forcing. The transaction authorization process must be restarted after number of failed authorization attempts. In addition other anti brute-forcing and anti-automation techniques should be considered to prevent an attacker from automating his attacks,see OWASP Authentication Cheat Sheet.

2.5 Application should control which transaction state transitions are allowed

Transaction authorization is usually performed in multiple steps, e.g.:

1. The user enters the transaction data.
2. The user requests authorization.
3. The application initializes an authorization mechanism.
4. The user verifies/confirms the transaction data.
5. The user responds with the authorization credentials.
6. The application validates authorization and executes a transaction.

An application should process such business logic flow in sequential step order and preventing a user from performing these steps out of order or in even skipping any of these steps (see OWASP ASVS requirement **15.1**).

This should protect against attack techniques such as:

- Overwriting transaction data before user will enter the authorization credentials,
- Skipping transaction authorization.

2.6 Transaction data should be protected against modification

The transaction authorization process should protect against attack scenarios that modify transaction data after the initial entry by the user. For example, a bad implementation of a transaction authorization process may allow the following attacks (for reference, see steps of transaction authorization described in paragraph 2.5):

- Replaying step 1 (sending transaction data) in the background and overwriting transaction details with fraudulent transaction, before the user enters authorization credentials.
- Adding parameters with transaction data to a HTTP request which authorizes the transaction. In such a case, poor implementation will authorize the initial transaction and then execute a fraudulent transaction (specific example of Time of Check to Time of Use vulnerability).

The protection against modification could be implemented using various techniques depending on the framework used, but one or more of the following should be present:

- Any modification of transaction data should trigger invalidation of any previously entered authorization data. E.g. Generated OTP or challenge is invalidated.
- Any modification of transaction data should trigger reset of the authorization process.
- Any attempts to modify transaction data after the initial entry by the user is a symptom of tinkering with an application and should be logged, monitored and carefully investigated.

2.7 Confidentiality of the transaction data should be protected during any client / server communications

The transaction authorization process should protect the privacy of transaction data being presented to the user to authorize i.e. at section 2.5, steps 2 and 4.

2.8 When a transaction is executed, the system should check whether it was authorized

The result of the transaction entry and the authorization process described in paragraph 2.5 is the transaction execution. Just before the transaction is executed there should be a final control gate which verifies whether the transaction was properly authorized by the user. Such control, tied to execution, should prevent attacks such as:

- Time of Check to Time of Use (TOCTOU) â€“ example in paragraph 2.6
- Skipping authorization check in the transaction entry process (see. paragraph 2.5)

2.9 Authorization credentials should be valid only by limited period of time

In some malware attacks scenarios, authorization credentials entered by the user is passed to malware command and control server (C&C) and then used from an attacker-controlled machine. Such a process is often performed manually by an attacker. To make such attacks difficult, the server should allow authorizing the transaction only in a limited time window between generating of challenge or OTP and the transaction authorization. Additionally, such safeguard will also aid in preventing resource exhaustion attacks. The time window should be carefully selected to not disrupt normal users' behavior.

2.10 Authorization credentials should be unique for every operation

To prevent all sorts of replay attacks, authorization credentials should be unique for every operation. It could be achieved using different methods depending on the applied transaction authorization mechanism. For example: using a timestamp, a sequence number or a random value in signed transaction data or as a part of a challenge.

Remarks

We identify other issues that should be taken into consideration while implementing transaction authorization. However we deem to be beyond the scope of this cheat sheet:

- Which transactions should be authorized? All transactions or only some of them. Each application is different and an application owner should decide if all transactions should be authorized or only some of them, considering risk analysis, risk exposition of given application, and other safeguards implemented in an application.
- We recommend the use of cryptographic operations to protect transactions and to ensure integrity, confidentiality and non-repudiation.
- Device enrolment or "pairing" of an external authorization device (or a mobile application) with the user account.
- Provisioning & protection of the device signing keys, during device "pairing" is as critical as the signing protocol itself. Malware may attempt to inject/replace or steal the signing keys.
- User awareness. E.g.: For transaction authorization methods, when a user types-in significant transaction data to an authorization component (e.g. an external dedicated device or a mobile application), users should be trained to rewrite transaction data from trusted source and not from a computer screen.
- There are some anti-malware solutions that protect against malware threats but such solutions do not guarantee 100% effectiveness and should be used only as an additional layer of protection.
- Protection of the signing keys using a second factor either be password, biometric, etc..
- Protection of the signing keys leveraging secure elements (TEE, TPM, Smart card..)

References and future reading

References and future reading:

- Wojciech Dworakowski: E-banking transaction authorization - possible vulnerabilities, security verification and best practices for implementation. Presentation from AppSec EU 2015.
- Saar Drimer, Steven J. Murdoch, and Ross Anderson: Optimised to Fail - Card Readers for Online Banking.
- Jakub KaÅ‚uÅ¼ny, Mateusz Olejarka: Script-based Malware Detection in Online Banking Security Overview.
- List of websites and whether or not they support 2FA.
- Laerte Peotta, Marcelo D. Holtz, Bernardo M. David, Flavio G. Deus, Rafael TimÃ³teo de Sousa Jr: A Formal Classification Of Internet Banking Attacks and Vulnerabilities.
- Marco Morana, Tony Ucedavelez: Threat Modeling of Banking Malware-Based Attacks.
- OWASP Anti-Malware - Knowledge Base.
- OWASP Anti-Malware Project - Awareness Program.
- Arjan Blom , Gerhard de Koning Gans , Erik Poll , Joeri de Ruiter , and Roel Verdult: Designed to Fail - A USB-Connected Reader for Online Banking.

Security Best Practices Guidelines for Cryptographic Storage

Security Best Practices for Cryptographic Storage

Overview

Cryptographic Storage is the practice of maintaining sensitive data in an encrypted form. It helps to protect the confidentiality of your data even if it is stolen.

Security Practices

Identify confidential data to be protected: Identify the data that needs to be stored in encrypted form. This includes data such as user credentials, Personally Identifiable Information (PII), and proprietary information.

Implement strong cryptographic protocols: Use strong cryptographic protocols to encrypt the data. The cryptographic keys should never be shared or stored in plaintext.

Store the cryptographic keys securely: Use a secure mechanism such as hardware security modules (HSMs) to store the cryptographic keys.

Protect the cryptographic keys: Use access controls, such as authentication tokens, to protect the cryptographic keys. Do not allow unauthorized access to the keys.

Review security regularly: Perform periodic audits to check for any unauthorized access to the cryptographic keys.

Train staff on cryptographic storage: Ensure that your staff is trained on secure cryptographic storage practices.

Conclusion

By following the security best practices outlined above, you can ensure the safety of your data and your organizationâ€™s security.

Security Best Practices Guidelines for Database Security

Database Security Best Practices

1. Establish Separation of Duties

To help reduce the potential for fraud or unauthorized access, establish a separation of duties between those responsible for administering the database, those responsible for defining security policies, and those able to access the data.

2. Encrypt Data in Transit and at Rest

Where possible, use encryption techniques for data stored in the database and for data while it is in transit. This helps protect the data from malicious activity.

3. Restrict Database Access

Ensure that only authorized personnel have access to the database. Implement security measures such as user authentication, user profiles, role-based access control, two-factor authentication, etc.

4. Regularly Monitor Database Activity

Regularly monitor database activity and user access. Monitor authentication activities, login attempts, data modification requests, etc. Review logs regularly and ensure that access requests are authorized.

5. Update Databases Regularly

Databases can quickly become outdated and insecure. Make sure to regularly patch, update, and upgrade the database and applications running on it.

6. Regularly Test Database Security

Regularly test the security of the database to identify potential vulnerabilities. Also, test the strength of passwords and other security controls.

7. Implement An Active Database Backup Strategy

To minimize disruption in the event of a data breach or other security incident, maintain an active and testable database backup strategy.

8. Use Intrusion-Prevention Systems

Implement intrusion-prevention systems to monitor and protect the database from malicious activity.

Security Best Practices Guidelines for Denial of Service

Denial of Service (DoS)

A Denial-of-Service (DoS) attack is a malicious attempt to make a system unavailable, by consuming all of its resources so that legitimate requests canâ€™t be served. The main goals of DoS attacks are to render systems unusable or significantly slow them down.

Best Practices

- **Secure Your Firewall and Perimeter Devices:** Ensure that your firewall rules and configurations are updated and actively managed. Monitor and audit these components regularly for any changes or weaknesses that could be exploited.
- **Implement an Intrusion Detection and/or Prevention System (IDS/IPS):** Detect and respond to malicious traffic, as early as possible. This can be done with an Intrusion Detection System (IDS) and/or an Intrusion Prevention System (IPS).
- **Monitor Network Activity and Logs:** Track the source and duration of all incoming and outgoing traffic. Create rules that will alert you immediately when you detect suspicious activity. This will allow you to take action quickly and prevent the attack from escalating.
- **Establish Network Behavior Baselines:** Establish a baseline for normal network traffic patterns and be prepared to identify any sudden spikes or abnormal activity.
- **Reduce Network Flows and Data:** Take steps to reduce the amount of data flowing across your network. This can be done by limiting what services are accessible, or by setting up traffic filtering and prioritization rules.
- **Deploy Resources Appropriately:** Make sure that load balancers, firewalls, and other networking devices are deployed in such a way that is capable of handling large amounts of traffic.
- **Periodically Sandbox:** Periodically subject parts of the network to simulated DoS attacks. Use the results to identify weak spots and areas of improvement. This can be done using packet analyzers or DoS vulnerability assessment tools.
- **Be Prepared to Respond:** Create a plan for responding to a DoS attack, anticipate different attack scenarios, and know how to identify any potential indicators of an attack in progress.
- **Educate Your Staff:** Make sure that your staff is aware of the risks associated with DoS attacks and how to recognize suspicious activity. Train them periodically to ensure they are up-to-date on the

Security Best Practices Guidelines for File Upload

File Upload Security Best Practices

It is essential to establish effective security guidelines for the file upload process. The following are best practices that can help ensure privacy and security of systems and data:

- **Limiting Accessibility:** Access to file uploads should be restricted, and user credentials should be verified each time an upload is attempted.
- **Requiring Authentication:** Establish strong authentication methods such as two-factor authentication for any user attempting to upload files.
- **Monitoring Uploads:** Monitor the uploads occurring on an ongoing basis for any malicious activity.
- **Backups:** Establish a backup policy that addresses the time frequency of file uploads and the recovery process in the event of a security incident or disaster.
- **File Size Limitations:** Place limits on the size of files that can be uploaded to help prevent malicious file uploads.
- **Data Validation:** Establish data validation processes such as virus scanning to help ensure that only clean files are being uploaded into the system.

Encryption: Encrypt the data being uploaded to help ensure its privacy and security.

Auditing: Establish an audit policy to review the upload process and ensure that it is operating securely.

Logging: Log any file uploads to track and monitor activity.

Patching: Update the system with the latest security patches to ensure the latest protections are in place.

Security Best Practices Guidelines for Injection Prevention

Injection Prevention Best Practices

Injection vulnerabilities occur when user input is unexpectedly executed as code. Injection attacks can come in many forms, including SQLi, OS, and LDAP injections, and can cause substantial data loss and server damage. It is important to take precautions to prevent injection attacks from occurring.

General Best Practices

- Validate user input using whitelisting, type conversion, or other techniques
- Enforce input length and format constraints
- Implement output encoding for dynamic data
- Reduce attack surface area, minimizing the amount of code accessible to malicious users
- Sanitize and filter user input
- Check input strings for any malicious code
- Escaping special characters
- Use prepared statements, parameterized queries, and stored procedures for database interaction
- Audit and log all input and output operations
- Use API Gateway to control access to APIs

Web Security Best Practices

- Disable the use of backslash and commas in web applications
- Filter out SQL injection attempts from user input
- Filter out "naughty strings" (e.g. "DROP TABLE")
- Limit the number of characters in forms
- Sanitize regular expression data
- Use HTTPs for all network traffic
- Permit the use of only known file types
- Disallow the execution of arbitrary command line parameters
- Validate URL requests
- Use CAPTCHA for authentication

Following these best practices can help prevent injection attacks and ensure the safety of your system.

References: - https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet - https://www.veracode.com/security/injection-prevention - https://www.netsparker.com/blog/web-security/prevent-sql-injection-attacks/ - https://www.zeropointsecurity.com/injection-attacks

Security Best Practices Guidelines for Logging

Security Best Practices for Logging

Introduction

Logging is a critical component of operational security, which can be used to detect potential security incidents, verify compliance with internal and external regulatory requirements, and provide an audit trail for later forensic activities. Proper logging configurations and practice can help you protect the confidentiality, integrity, and availability of your system, as well as reduce the chances of data privacy breaches.

This guide provides an overview of some of the best practices for configuring, maintaining, and viewing logs.

Logging Practices

Establish a logging policy: Decide which logs need to be saved and how long the logs need to be retained, and share the policy with stakeholders.

Set up log aggregation and storage: Ensure that access and storage controls are configured on log files to protect the log data from manipulation or unauthorized access.

Choose appropriate log retention periods and awareness programs: Decide how long the logs should be retained before disposal.

Utilize log monitoring and analysis tool: Use a tool to automate the collection, analysis, and alerting on log data.

Configure the system to generate detailed logs: Detail the events required to capture in the logs including, but not limited to, user authentication, attempted logins, system startup/shutdown, system modifications, etc.

Encrypt data in transit and at rest: Ensure data is encrypted both in transit and at rest to protect sensitive data from unauthorized access.

Test logging regularly: Test the logging system regularly to ensure that all relevant data is being logged as desired.

Ensure only authorized users can view the logs: Apply role-based access control and passwords to prevent unauthorized access to log data.

Educate users on logging: Inform users about logging best practices and policies to avoid inadvertent violations.

Security Best Practices Guidelines for Logging Vocabulary

Logging Vocabulary Security Best Practices

Be Aware of Log-Levels: Understand the context of the information your application collects and what purpose it serves. For example, too much logging could impact performance and increase storage and processing overhead.

Limit Access to Logs: Make sure to limit access to log information to individuals and groups that really need it. Logging should never be exposed to the public.

Create Secure Log Storage: Choose a secure storage system for logs to minimize chances of tampering or unauthoirzed access.

Keep Track of Log "Events": Document and maintain a record of changes and additions to the log information.

Securely Delete Log Information: Log information should be securely deleted once it has served its purpose.

Configure and Enable Security Logging: Set up and enable logging for any security events, like failed authentication attempts, etc.

Audit Logging Systems: Periodically audit log systems to ensure that they are properly configured and functioning correctly.

Log File Integrity Monitoring: Monitor log files for integrity and ensure that they are not modified, overwritten, or deleted.

Follow Directive Rules: If your organization uses directives such as the European Union's GDPR, HIPAA, and Sarbanes-Oxley Act, make sure your logging practices comply with these regulations.

Security Best Practices Guidelines for Password Storage

Password Storage Best Practices Guidelines

Make your passwords long: Use a minimum of 8-10 characters; longer passwords are more secure.

Make your passwords complex: Include a mix of uppercase and lowercase letters, numbers, and special characters.

Avoid using personal and easily guessed details: Do not use your name, birthdate, address, or any other personally identifiable information in your password.

Do not use the same password for multiple accounts: It is more secure to use unique passwords for each account.

Keep your passwords safe: Store them in a secure password manager or use two-factor authentication when available. If you need to write down your passwords, keep them in a secure, locked place.

Change passwords regularly: Change your passwords at least every 3 months.

Be careful of suspicious links or email attachments: Never click on links or open attachments in emails from unknown or untrusted sources.

Be alert when logging in: Always check to make sure you are on a secure, legitimate website.

Security Best Practices Guidelines for SSRF Prevention

Security Best Practices Guidelines for SSRF Prevention

Server-Side Request Forgery (SSRF) is an attack that forces a server to perform requests on behalf of an attacker. It can be used to compromise data, bypass authentication, and gain access to internal systems.

The following security best practices can help prevent SSRF and protect against related attacks:

- Develop and deploy applications securely and ensure that any input from an untrusted source is sanitized and validated.
- Block access to all unnecessary services, especially those that can be used to send requests to other systems. This includes the likes of external APIs, databases, and filesystems.
- Set up an internal firewall to prevent external requests from entering the network.
- Implement strong authentication and access control restrictions to verify that only authorized users can access critical resources.
- Monitor and log all requests to internal and/or external services.
- Patch and maintain all servers, web applications, and operating systems regularly to keep them up to date.
- Educate and train all staff to be aware of the risks associated with SSRF attacks.
- Make sure third-party APIs and services are configured securely and have adequate security measures in place.

Security Best Practices Guidelines for Session Management

Session Management Best Practices

Session Management is an important part of securing web applications. Implementing good session management practices helps protect user data, prevent unauthorized access, and offers a more secure user experience.

Below are some best practices to help to ensure that you are properly managing user sessions and protecting user data:

Use Secure Cookie Policies when Storing Session Data

- Use secure HTTPS protocol when sending session data.
- Specify short expiration times on session cookies.
- Use "secure" and "httponly" attributes to further enhance cookie protection and disable cookie access from JavaScript code.
- Renew the session ID when sensitive data is updated.

Set Appropriate Access Controls

- Restrict access to authenticated users only.
- Enforce strong passwords and multi-factor authentication when possible.
- Limit access to resources to a specific IP address or range of addresses.

Monitor User Activity

- Monitor session data for signs of suspicious activities.
- Log failed login attempts.
- Implement an audit logging system to track user activities over time.

Implement Timeouts

- Use server side session timeouts to ensure that a user session is terminated when a set period of time has expired.
- Implement shorter timeouts for important transactions like online banking transactions.

Take Advantage of Automated Tools

- Use automated tools to help identify and track session data.
- Use automated tools to update application code and ensure that security issues are proactively addressed.

Following these best practices will help ensure that user data is secure and protected, and that web applications are operating in a safe and secure manner.

Security Best Practices Guidelines for Transport Layer Protection

Transport Layer Protection: Security Best Practices

It is important to ensure that your transport layer is secure to protect the confidentiality, integrity, and availability of your data. The following best practices should be followed when using transport layer protection:

Encryption

1. Use TLS/SSL whenever possible for secure transit of data between clients and servers.
2. Use strong encryption algorithms such as AES-256 and RSA-2048 to protect data.

3. Use Elliptic-curve Cryptography (ECC) for its smaller key size and higher encryption strength.

Certificate Management

1. Use only valid and trusted SSL certificates.
2. Regularly check for revoked and expired certificates and take necessary steps to update them.
3. Make sure all certificates used by the organization are up to date and properly configured.

Firewall & Network Security

1. Make sure to enable firewall rules to allow only secure protocols like HTTPS/TLS.
2. Use Intrusion Detection and Prevention Systems to prevent malicious packets from entering the network.
3. Utilize monitoring and logging tools to detect and respond to suspicious or malicious activity on the network.

Authentication & Authorization

1. Enable two-factor authentication when available, and use a secure password policy.
2. Implement Role-Based Access Control (RBAC) to separate users and enforce access control.
3. Use strong authentication methods such as digital certificates or biometrics.

Physical Security

1. Implement appropriate physical security measures such as access control and CCTV surveillance.
2. Monitor all external device connections such as USB drives.
3. Ensure the secure storage of data center devices.

Security Best Practices Guidelines for Input Validation

Input Validation Security Best Practices

1. **Whitelisting:** Use whitelisting to ensure only known reliable data enters the system.
2. **Data Minimization:** When possible, minimize the amount of user supplied input data.
3. **Data Size Limitation:** Restrict input data to a reasonable length.
4. **Data Type Limitation:** Restrict input data to expected types and formats.
5. **Input Data Sanitization:**Sanitize input data to strip out malicious content (e.g. tags, scripts).
6. **Input Data Encoding:**Encode input data (e.g. HTML encoding) to prevent attackers from exploiting a known vulnerability.
7. **Verify Server Side:** Perform checks and validation on the server side for all user supplied data.
8. **Data Format Validation:** Validate any input data is in the required format.
9. **Reduce False Positives:** Try to reduce any false positives that impede users from submitting their input data (e.g. CAPTCHAs).
10. **Logging and Monitoring:** Monitor suspicious or malicious activity (e.g. failed logins attempts) around user input.

Security Best Practices Guidelines for User Privacy Protection

User Privacy Protection Best Practices

1. Ensure explicit user consent for the collection and use of personal data.
2. Collect and process only the necessary personal data to fulfil your organizations purpose.
3. Securely store all collected personal data.
4. Implement data access controls so that only those that need it have access to personal data.
5. Ensure your data processing activities are documented.
6. Only share personal data with third parties if necessary and if the third party has the right procedures and controls in place to protect the data.
7. Give users the right to access, update, and delete their personal data.
8. Notify users of any data breaches promptly and as required by law.
9. Regularly reassess and revise your user privacy protection standards.
10. Educate all personnel who have access to personal data on user privacy protection best practices.

Security Best Practices Guidelines for Cryptography

Security Best Practices for Cryptography

Cryptography is one of the most important tools when it comes to securing sensitive information. The following best practices should be implemented when using cryptography:

Key Management

1. Generate strong cryptographic keys and store them securely.
2. Back up cryptographic keys regularly in multiple secure locations.
3. Properly revoke cryptographic keys that will no longer be used.
4. Implement access control measures for cryptographic keys to prevent unauthorized access.
5. Limit the number of administrators that have access to cryptographic keys.

Use of Cryptographic Algorithms

1. Use only well-tested cryptographic algorithms and implementations.
2. Regularly assess and update cryptographic algorithms if they become outdated or vulnerable.
3. Use strong cryptographic algorithms such as AES and RSA.
4. Utilize separate cryptographic implementations for different systems for better security.

Encryption

1. Encrypt data at rest, in transit, and in memory.
2. Never store unencrypted data or passwords.
3. Ensure secure transmission of data over the network and across systems.
4. Use separate encryption keys for different systems for better security.

Security Monitoring

1. Implement proper security monitoring of cryptographic systems.
2. Regularly audit cryptographic systems to ensure that they are secure and compliant.
3. Monitor for unauthorized access to cryptographic keys and systems.
4. Implement proper incident response measures for security breaches.

Security Best Practices Guidelines for Secure Application Update

Secure Update of Cloud-based Mobile Application

Best Practices Guidelines

This document details the best security practices for performing a secure update of a cloud-based mobile application.

1. Prepare a Secure Infrastructure

- Leverage a secure cloud infrastructure designed to ensure the security of the mobile application.
- Use a secure cloud environment such as a virtual private cloud (VPC) with dedicated firewalls and access control mechanisms.
- Ensure that the VPC is fully isolated from any other public services to minimize the risk of unauthorized access.
- Ensure that all security settings related to the VPC, such as ports, protocols, and authentication mechanisms, are properly configured to prevent potential threats and attacks.

2. Encrypt Sensitive User Data

- Ensure that sensitive user data is encrypted both at rest and in transit, using end-to-end encryption to protect against data leakage and malicious actors.
- Use strong cryptographic algorithms and regularly update them in order to remain up-to-date with the latest industry standards.

3. Use Multi-Factor Authentication

- Make sure that multi-factor authentication (MFA) is implemented for all users to provide an extra layer of security.
- Utilize different means for authentication, such as physical tokens, biometrics, one-time passwords, or mobile applications.

4. Implement Proper Access Controls

- Ensure that users and administrators are granted access to only those resources that are absolutely necessary.
- Implement least privilege principles to reduce the risk of unauthorized access of sensitive user data.
- Ensure that sensitive information is stored on secure servers with up-to-date access controls.

5. Ensure Regular Vulnerability Scanning

- Perform regular security scans in order to identify potential vulnerabilities before they can be exploited.
- Utilize web application scanning tools to identify and address any security issues in the code.
- Make sure that all servers are regularly updated with the latest security patches and fixes.

6. Monitor Logs and Monitor Network Activity

- Monitor all system logs and network activities in order to detect any suspicious or malicious activities.
- Utilize automated intrusion detection systems to detect any malicious attempts to break into the system.

7. Develop Secure Application Code

*

Security Best Practices Guidelines for Secure Third-party Application

Security Best Practices Guidelines for Secure Third-party Cloud-Based Mobile Applications

The following best practices are designed to ensure secure use of Cloud-Based Mobile Applications.

Proper User Authentication

Authentication should be based on strong credentials such as two-factor authentication whenever possible.

Application passwords should be strong and updated regularly. Make sure to store them securely.

User accounts should be locked out after multiple failed attempts to discourage brute force attacks.

Secure Communications

All communications should be encrypted and authenticated using industry-standard encryption protocols such as HTTPS and SSL/TLS.

Mobile Applications should only communicate with backend services over a secure data channel or VPN

Secure Data Storage

All sensitive data should be stored in an encrypted format.

Data should be stored on secure servers that are regularly patched with the latest security updates.

Access to sensitive data should be limited only to authenticated users.

Secure Data Transmission

All data transmitted between mobile devices and backend services should be encrypted.

All mobile applications should verify the identity of backend services before sending data.

Code Review

All code should be reviewed by a qualified security professional prior to deployment.

All external libraries and frameworks should be regularly updated to ensure that security vulnerabilities are patched.

Application Level Threat Protection

Mobile applications should be tested for security vulnerabilities and common attack vectors.

Mobile applications should include rate limiting, then monitor and block suspicious requests and activities.

Regular Updating

All mobile applications should be regularly patched to ensure that they contain the latest security updates.

All external libraries and frameworks should be regularly updated as well.

By following these best practices, organizations can ensure the secure use of third-party cloud-based mobile applications.

Final Security Mechanisms Report

Mobile Platform	iOS App ; IoT System
Application domain type	m-Payment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Channel-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	PostgreSQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Both
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	C/C++/Objective-C
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	Public Cloud
HW Authentication	Symmetric Key
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ; NFC
Device or Data Center Physical Access	Yes

Security Backup Mechanisms

Security Backup Mechanisms for cloud-based mobile apps are procedures to keep data safe and secure in the event of an emergency, such as a computer crash, a user error, or a malicious attack. These mechanisms can include:

â€¢ Access Control: Access control restricts the access of certain parts of the application, such as confidential data or the applicationâ€™s backend, in order to limit the potential damage caused by malicious activities.

â€¢ Data Encryption: Data Encryption scrambles application data into an unreadable format, making it impossible to access without the decryption key.

â€¢ Password Hashes: Password Hashes are securely stored versions of the usersâ€™ passwords to prevent malicious activities such as credentials theft.

â€¢ Tokenization: Tokenization is a mechanism that replaces sensitive data with a token to reduce the risk of data theft.

â€¢ Backup System: A backup system can be used to store application data in separate, secure locations. This data can be used to restore the application to its former state in the event of a disruption.

Backup Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Backup	iOS	iTunes Backup	Syncs with iTunes for off-site backup	7 - Application
Backup	Android	Google Drive	Google's cloud solution for data storage and backup	7 - Application
Backup	Android	Third-party cloud solutions	Solutions such as Dropbox, OneDrive and iCloud Drive	7 - Application
Backup	All	Local Backup	On-site backups saved on the device's internal storage	1 - Physical
Backup	All	External Storage Backup	Off-site backups saved to external devices such as external hard drives and USB drives	1 - Physical

Security Audit Mechanisms

A Security Audit Mechanism is an automated or manual process which evaluates cloud-based mobile apps for security issues. It may include verifying the integrity of the code, inspecting system configurations, testing user authentication and authorization controls, and ensuring that the system is following best practices such as encryption, patching, and regular system updates. A Security Audit Mechanism can also identify potential security weaknesses and provide recommendations for mitigating these. Furthermore, a Security Audit Mechanism can perform performance and reliability checks, as well as other security checks such as penetration testing, infrastructure testing, and security vulnerability scanning. By utilizing these security audit mechanisms, organizations can ensure their cloud-based mobile apps are safe and secure.

Audit Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	iOS	Appleâ€™s App-ID and two factor authentication	A two-factor authentication and App-ID system used by Apple to verify and authenticate applications running on its iOS mobile platform	Application
Authorization	iOS	Access control list (ACL)	A tool used to manage user access to various parts of a mobile application, such as data or services	Application
Data Protection	Android	Google Play Store	Googleâ€™s Play Store protects uploaded applications from malicious code before it is distributed on the platform	Presentation
Auditing	iOS	App Store	The App Store provides an audit trail of all applications downloaded, to ensure proper users have the correct permissions to access applications	Application
Data Validation	Android	Android Content Providers	Android content providers are used to securely store data and detect malicious code before it is passed to applications running on the platform	Application

Cryptographic Algorithms Mechanisms

Cryptographic algorithms are used to ensure data confidentiality, authenticity, integrity and non-repudiation in cloud-based mobile apps. To achieve these goals, cryptographic algorithms are often used in combination with mechanisms, such as Digital Signatures, Secret Key Cryptography and Public Key Cryptography.

Digital Signatures validate the identity and authenticity of communications, while Secret Key Cryptography algorithms like AES, DES and 3DES protect transmitted data through the use of encryption. Public Key Cryptography algorithms like RSA, ECDSA and Diffie-Hellman can also be used to authenticate, encrypt and exchange secret keys between the mobile device and the cloud provider. In addition, protocols such as SSL / TLS can add an extra layer of security while protecting and verifying the communication and providing message integrity.

Cryptographic Algorithms Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer	Use for coding	Use for runtime
Integrity	Android	HMAC-SHA256	A cryptographic hash function based on SHA256 that combines a shared secret and the message	7	Yes	Yes
Confidentiality	iOS	AES-128	AES with 128 bit key size that supports authenticated encryption	6	Yes	Yes
Authentication	iOS	ECDSA	Elliptic Curve Digital Signature Algorithm that provides digital signatures	7	Yes	Yes

Biometric authentication mechanisms in cloud-based mobile apps are methods of authentication relying on the physiological characteristics of a user as a method of accessing the device or application. Examples of popular biometric authentication technologies available for cloud-based mobile devices are fingerprint scanning, facial recognition, and voice recognition. These technologies use advanced algorithms to validate a userâ€™s identity based on the physiological traits unique to each individual. By using these methods, companies and app developers can increase the security of their cloud services while preventing unauthorized access.

Biometric Authentication Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication & Access Control	Android	Facial Recognition	Hardware based biometric authentication that uses the device front facing camera to snap a picture of the user's face and match it against stored images	Application

Authentication & Access Control	iOS	Voice Recognition	Software based biometric authentication that uses the device microphone and internal software to capture the user's voice and match it against stored audio	Application
Encryption & Decryption	Android	2-Factor Authentication with PIN & Pattern	Combined hardware and software based authentication that requires the user to enter a PIN and draw a pattern on a defined pattern grid.	Presentation
Encryption & Decryption	iOS	Retina Recognition	Hardware based biometric authentication that uses the device front facing camera to obtain a high-resolution picture of the user's eye and matches it against stored images	Application
IDS & IPS	Android	Fingerprint Scan	Hardware based biometric authentication that uses the device built-in fingerprint scanner to scan the user's fingerprint and match it against stored images	Application
IDS & IPS	iOS	3-Factor Authentication with PIN, Pattern & Password	Combined hardware and software-based authentication that requires the user to enter a PIN, draw a pattern on a defined pattern grid, and enter a password	Presentation

Channel-based authentication mechanisms in cloud-based mobile apps refer to a set of security protocols that validate users and authorize access to specific resources in a cloud mobile application. This authentication is done through a set of channels, such as biometrics, passwords, OTPs, or mobile phone numbers, each with its own level of security and authentication request. This type of authentication is used to ensure access to sensitive data and improve the overall security of the application.

Channel-based Authentication Mechanisms Examples:

Security Requirement	Mobile Plataform	Mechanism	Description	OSI Layer
Authentication	Android	HMAC-SHA256	Mobile application uses a pre-shared HMAC-SHA256 token to authenticate with the cloud server and establish a secure channel.	Application
Authorization	iOS	OAuth-2	Mobile application uses an OAuth-2 access token to authorize requests made to the cloud server and establish a secure channel.	Application
Identity Management	Cross-platform	OpenID Connect	Mobile application uses OpenID Connect to authenticate with the cloud server and establish a secure channel.	Application
Data Encryption	Cross-platform	TLS/SSL	Mobile application uses TLS/SSL to encrypt data transmitted between the mobile device and the cloud server.	Transport

Factors-based authentication mechanisms in cloud-based mobile apps are methods used to securely access digital resources. They involve the use of two or more authentication factors, such as something that a user knows (e.g., a password), something that a user has (e.g., an authentication code sent to a mobile device), and/or something that a user is (e.g., a biometric scan). Factors-based authentication can help protect mobile apps by providing an extra layer of security, making it less likely that someone unauthorized can access sensitive user data.

Factors-based Authentication Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer	To Use
Data Security	iOS	Two-factor authentication	Confirming identity by combination of two unique factors	Application	Coding Phase and Runtime
Privacy	Android	Biometric authentication	Confirming identity by using biometric methods	Application	Coding Phase and Runtime

Account Access	iOS	User ID & Password	Confirming identity by using combination of user ID and password	Application	Coding Phase and Runtime# ID-based Authentication Mechanisms

ID-based authentication mechanisms are used to authenticate users in cloud-based mobile applications. This type of authentication typically involves the use of an identifier such as an email address or phone number, as well as a password or some other form of proof of identity. ID-based authentication may also involve the use of biometric markers like fingerprints or facial recognition to verify the user's identity. By using ID-based authentication, mobile applications can ensure that only authorized users are granted access, thereby protecting the data stored and exchanged on the application.

ID-based Authentication Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	iOS	FaceID	User authenticates with their face	Layer 7
Authentication	iOS	Touch ID	User authenticates with their thumbprint	Layer 7
Authorization	iOS	Apple App Tracking Transparency (ATT)	Authorizes a userâ€™s usage data to be tracked by a third-party for targeted advertising	Layer 7
Authentication	Android	Fingerprint Authenticator	User authenticates with their fingerpring	Layer 7
Authentication	Android	Face Unlock	User authenticates with their face	Layer 7
Authorization	Android	Google Play Billing Library	User authorizes payment for in-app billing	Layer 7

Cryptographic Protocols Authentication Mechanisms

Cryptographic protocols mechanisms for cloud-based mobile apps refer to the cryptographic techniques used to protect data and communications between user devices and cloud-services. The protocols involve the encryption of data and messages with symmetric and asymmetric algorithms, the digital signing of messages, the authentication of users, the establishment of secure tunnels, and the use of secure hashing and salting. The goal is to ensure that, if a malicious person attempts to intercept the headers or payload of a cloud-based mobile app, they will be unable to access valuable information.

Cryptographic Protocols Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authentication	iOS	OAuth	OAuth is an open-standard authorization protocol for allowing access to a protected resource	Application layer
Encryption	Android	TLS	Transport Layer Security (TLS) is a cryptographic protocol used to provide secure communications over a computer network	Transport Layer
Integrity	iOS	SHA-1	Secure Hash Algorithm (SHA-1) is a cryptographic hash function used to generate a 160-bit hash value	Application layer
Non-repudiation	Android	HMAC	HMAC is a cryptographic mechanism used to verify the integrity of a message by using a secret key	Application layer

Access Control Mechanisms

Security Access Control Mechanisms (SACMs) are the technical and administrative strategies and tools used to protect cloud-based mobile apps from unauthorized access to confidential data and systems. These mechanisms are designed to restrict access to certain users, manage user privileges, authenticate user accounts, and authorize access requests. Examples of SACMs include multi-factor authentication (MFA), biometric authentication, single-sign-on (SSO), role-based access control (RBAC), application-level encryption, and least privilege access. SACMs allow organizations to properly control who has access to what resources and strictly enforce principles of confidentiality, privacy, and data security.

Access Control Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer

Data confidentiality	Android	RSA Encryption	Encryption of data with public and private keys	Application
Data integrity	Android	Hashing	Use of a hash algorithm such as SHA-2 to ensure that data is not tampered with	Transport
Account Management	iOS	Two-Factor Authentication	Use of two-factor authentication to verify user access	Presentation
Data access control	iOS	Role-Based Access Control (RBAC)	Defines levels of access based on user roles	Application
Resource authorization	iOS	Authorization Token	Generates a token at the end of a successful authorization process which is used to grant permission	Application

Inspection Mechanisms

An inspection mechanism is a process or tool used to ensure that cloud-based mobile apps meet certain quality and security requirements. Inspection mechanisms involve thoroughly evaluating the source code, architecture, and security of the app to ensure it meets the desired standard. Examples of inspection mechanisms include static code analysis, application security testing, architectural design reviews, and penetration testing. These inspection mechanisms help identify any weaknesses, vulnerabilities, or security issues in the app before it is deployed in the cloud.

Inspection Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Integrity	Android	ProGuard	Code obfuscation	8
Confidentiality	iOS	Secure store	Keychain security	7
Authentication	Android	SafetyNet API	Attest the device integrity	7
	Android	Android Keystore	Keystore security	7
	iOS	Apple push notification service (APNS)	Authentication message	7
Data Validation	Android	DX Guardrail	Verification of data model	7
	iOS	SwiftLint	Static analysis	7# Logging Mechanisms

An inspection mechanism is a process or tool used to ensure that cloud-based mobile apps meet certain quality and security requirements. Inspection mechanisms involve thoroughly evaluating the source code, architecture, and security of the app to ensure it meets the desired standard. Examples of inspection mechanisms include static code analysis, application security testing, architectural design reviews, and penetration testing. These inspection mechanisms help identify any weaknesses, vulnerabilities, or security issues in the app before it is deployed in the cloud.

Logging Mechanisms Examples:

Security Requirement	Mobile Plataform	Mechanism	Description	OSI Layer
Authentication	iOS	DeviceCheck	DeviceCheck enables customers to securely store small bits of data on Apple devices during the coding and runtime phases	Application
Access Control	iOS	KeyChain	Appleâ€™s Keychain, is a encrypted storage system that primarily stores passwords, certificates, and encryptionkeys	Application
Auditing	Android	Syslog	System logging mechanism for capturing and persistently logging system and audit-specific events in the Android OS	Transport
Logging	Android	LumberJack	Logging mechanism for logging the events for mobile applications	Application

Device Detection Mechanisms

Security Device Detection Mechanisms in Cloud-based mobile apps are technologies responsible for detecting the mobile device that is used to access the application. The mechanisms can vary from OS-level or device-level properties and can include biometrics such as facial recognition, fingerprint scanning, and voice recognition. These mechanisms allow cloud-based mobile apps to detect the device used and ensure that only authorized devices are able to access the app, providing an extra level of security against potential malicious activity.

Device Detection Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Coding Phase	iOS	Mobile App Wrapping	A tool used to secure enterprise apps	Application
Coding Phase	Android	App Reverse Engineering Protection	A technique used to protect code from reverse engineering	Application
Runtime	iOS	Jailbreak Detection	Detects if the device is jailbroken or not	Application
Runtime	Android	Root Detection	Detection of rooted devices	Application

Physical Location Mechanisms

Security physical location mechanisms are applied to cloud-based mobile apps to ensure that user data is not accessed or stored from locations outside of an approved geographic region. These mechanisms include technologies such as geofencing and IP address tracking. Geofencing verifies that user data is being accessed and stored within a predetermined geographic area by creating a virtual fence around the area. IP address tracking allows mobile apps to identify the geographical location associated with a particular IP address in order to verify that a user is located in the approved geographic area. These security location mechanisms are essential for cloud-based mobile apps, as they help prevent unauthorized access to user data from malicious actors located in remote locations.

Physical Location Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Authenticated Access	iOS	Biometric Scanner	Uses user's fingerprints as part of the authentication process	Physical
Data Integrity	Android	Transparent Encryption	Files are encrypted transparently and automatically	Network
Data Availability	Both	Secure Boot & Root	Ensures that all parts of system are authenticated and verified	Physical
Data Confidentiality	iOS	App sandboxes	Prevents unauthorized access to specific files	Application
Data Security	Android	Full Disk Encryption	Encrypts all data on device	Network

Confinement Mechanisms

Security Confinement Mechanisms in Cloud-based mobile apps refer to the various measures put in place by app developers to help ensure the security and integrity of data within the app. These mechanisms might include measures like authentication requirements, security protocols, encryption, tokenization, application sandboxing, and isolated virtual machines. These measures help limit the risk of data theft or compromise within a cloud-based mobile application.

Confinement Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer
Vulnerability Protection	Android	Flask	Flask is a Python web development framework used to protect against malicious code injections	Application Layer
Isolation of Data	iOS	Security-Enhanced Linux (SELinux)	SELinux is a Linux kernel security module used to isolate code from its data	Network Layer
Security of Data	Blackberry	BitLocker	BitLocker is a Windows data encryption system meant to protect data while it is stored	Data Link Layer
Secure Communications	Symbian	IPsec	IPsec is a protocol suite used in secure communication by authenticating and encrypting data	Presentation Layer
Secure Data Transfer	Palm	DM-Crypt	DM-Crypt is a drive encryption system meant to protect data while it is transferred	Session Layer

Filtering Mechanism Mechanisms

Security Filtering Mechanisms are built into Cloud-based mobile apps to ensure data is protected from unauthorized access. These mechanisms include multi-factor authentication, data encryption, data loss prevention, and various identity and access management (IAM) tools that control user access and authentication policies. Additionally, mobile app hardening techniques such as static code analysis and dynamic application security testing (DAST) can be used to detect and fix potential security vulnerabilities.

Filtering Mechanism Mechanisms Examples:

Security Requirement	Mobile Platform	Mechanism	Description	OSI Layer

Authentication	iOS	OAuth2	OAuth2 provides a secure mechanism for authorizing application access to a user's account	Application
Encryption	Android	AES	AES (Advanced Encryption Standard) uses symmetric key cryptography to provide strong encryption of data at rest	Data Link
Key Management	Cross-Platform	AWS KMS	AWS KMS (Key Management Service) is a cloud-based service used to manage encryption keys for encrypting/decrypting data	Transport
Data Obfuscation	Cross-Platform	ProGuard	ProGuard is a popular open source code obfuscator which helps protect your app's code during coding phase and runtime	Physical

Final Attack Models Report

Mobile Platform	iOS App ; IoT System
Application domain type	m-Payment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Channel-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	PostgreSQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Both
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	C/C++/Objective-C
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	Public Cloud
HW Authentication	Symmetric Key
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ; NFC
Device or Data Center Physical Access	Yes

Man-in-th-Middle Attack

Man-in-the-Middle (MITM) attack is an attack where a threat actor interferes with the communication between two systems. The threat actor inserts itself between the two systems and has access to all the data being sent between them.

MITM attacks are used to steal or modify data in transit, such as banking credentials, passwords, and security tokens. Hackers carry out these attacks by spoofing IP addresses and using malicious code to gain access to unencrypted data. They can also use packet-sniffing software to eavesdrop on the connection.

MITM attacks can be done through network-level attacks or application-level attacks. Network-level MITM attacks involve the hacker taking control of the entire communications path between the two hosts. Application level MITM attacks involve the hacker hacking into one of the hosts and manipulating their traffic.

To protect against MITM attacks, it is important to use secure protocols such as HTTPS and SSL/TLS. It is also important to ensure that sensitive data is encrypted while in transit. Additionally, strong authentication methods should be used to authenticate users and prevent unauthorized access.

Man-in-th-Middle Architectural Risk Analysis:

Overview

Man-in-the-Middle (MITM) vulnerabilities occur when an attacker is able to intercept and modify data sent between two parties. This attack is especially dangerous as it can be used to intercept sensitive and confidential information.

Risk Factors

The Common Vulnerability Scoring System (CVSS) version 3.1 measures the risk of a Man-in-the-Middle (MITM) attack along four different vectors:

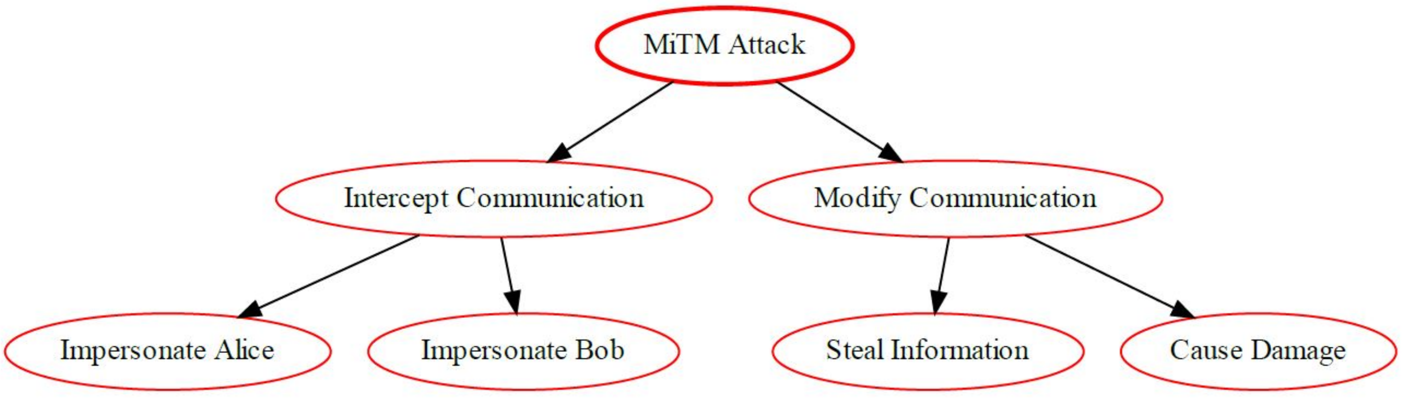
1. **Attack vector (AV)** : Remote
2. **Attack complexity (AC)**: Low
3. **Privileges required (PR)**: None
4. **User Interaction (UI)**: None

Risk Score

Based on the risk factors defined above, the risk score for a Man-in-the-Middle attack using CVSS v3.1 is 7.2. This falls under the medium risk category.

Overall Risk Score: 7.2 (Medium)

MiTM Attack Tree



Brute Force Attack

A Brute Force attack is a type of attack that attempts to guess a user's authentication credentials, such as a username and password, by systematically trying every possible combination of characters until the correct one is discovered. It is commonly used to gain unauthorised access to secure systems.

It is important to note that Brute Force attacks are often used in combination with other tactics, such as dictionary and rainbow table attacks, to increase the chances of success.

In order to protect against this type of attack, it is recommended to use strong authentication credentials that are difficult to guess, such as long, randomised passwords made up of upper and lower case letters, numbers, and special characters. It is also a good idea to set a maximum amount of failed log-in attempts before locking out the user account, as this prevents attackers from attempting to guess the credentials indefinitely.

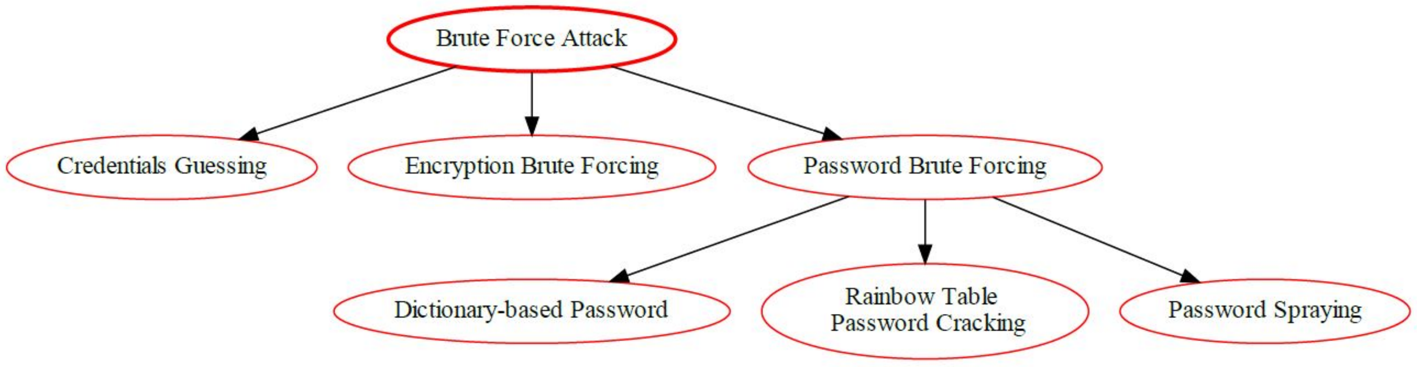
Brute Force Architectural Risk Analysis:

Brute Force Vulnerability

Criteria Group	Score
Attack Vector	4.3
Attack Complexity	3.9
Privileges Required	0.8
User Interaction	0.0
Scope	4.3
Confidentiality Impact	4.0
Integrity Impact	4.0
Availability Impact	4.0
Exploit Code Maturity	7.8
Remediation Level	4.6
Report Confidence	3.9
Total Score	**46.9**

Risk Classification: High

Brute Force Attack Tree



Eavesdropping Attack

Eavesdropping attack is a type of network attack in which the attacker listens to the conversations taking place among two or more authorized users or devices on the same network. This attack allows attackers to collect valuable information, including private data and confidential messages, without being detected.

In this attack, the attacker uses various tools to gain access to the target computer's network, such as sniffers, which are essentially network-based packet sniffers that extract data from the network, and Trojan horses, malicious programs that are secretly installed on the system. The attacker can also use other methods to access the network, such as phishing emails, rogue Wi-Fi access points, and man-in-the-middle attacks.

Once the attacker gains access to the network, they eavesdrop on the conversations taking place on the network. By monitoring the data packets being sent over the network, the attacker can gain access to sensitive information and data that they can then use for malicious purposes.

Eavesdropping Architectural Risk Analysis:

Eavesdropping Vulnerability

Common Vulnerability Scoring System (CVSS) v3.1 score for Eavesdropping Vulnerability is 4.8, categorized under 'High' severity.

CVSS Base Score: 4.8

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): High (H)

Integrity Impact (I): Low (L)

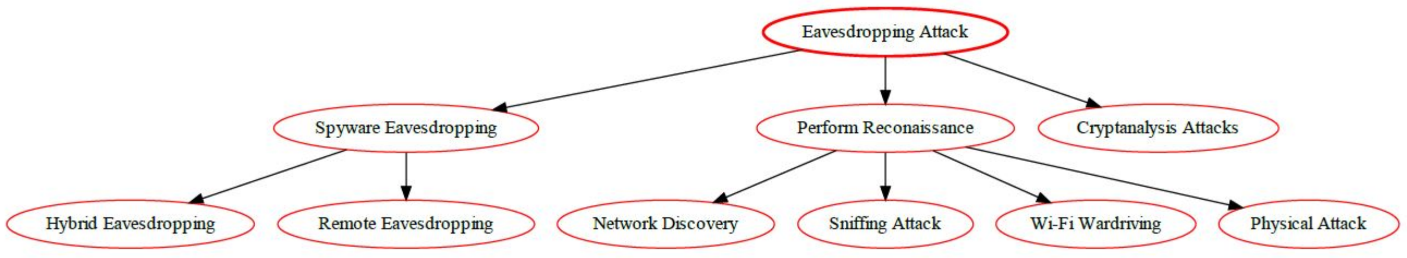
Availability Impact (A): Low (L)

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

Risk Analysis of Eavesdropping Vulnerability

Eavesdropping Vulnerability poses a high risk to the confidentiality of the data traveling within a network as it allows attackers to intercept and potentially access sensitive information. Without any user interaction, an attacker can intercept information and potentially gain unrestricted access to the confidential data, thus leaving the usersâ€™ online operations prone to manipulation. Moreover, the integrity and availability of the network can be impacted to a low extent.

Therefore, organizations need to put in place an effective counter-measures strategy which focuses on enhancing data security measures, including the adoption of strong authentication protocols and encryption technologies, to mitigate and reduce the risk of eavesdropping attacks.



Flooding Attack

Flooding attacks are attempts to inundate a resource with an overwhelming amount of data or requests in order to overwhelm or crash it. Flooding attacks are often effective when the target resource is limited in bandwidth or processing power, such as a server, and is unable to handle so much data or requests, resulting in performance degradation or service disruption.

Examples of flooding attacks include Denial-of-Service (DoS) attacks, which send an extremely large amount of requests/traffic to the victimâ€™s server or network in order to saturate it and make it incapable of responding to legitimate requests. Additionally, there is also the Distributed Denial-of-Service (DDoS) attack, which uses more than one computer or device to send the traffic, making it even more of a challenge to defend against.

Flooding attacks can be difficult to detect and stop as they often involve huge volumes of data. However, some steps to help mitigate the effects of flooding attacks include:

- Utilizing firewalls and other security measures to detect and block suspicious traffic.
- Deploying load balancers to distribute the amount of requests over multiple servers or resources.

- Monitoring network and server performance to detect anomalies.
- Limiting connection requests from a single source.

Flooding Architectural Risk Analysis:

Common Vulnerability Scoring System (CVSS) v3.1 - Risk Analysis of Flooding Vulnerability

Base Score: 7.2

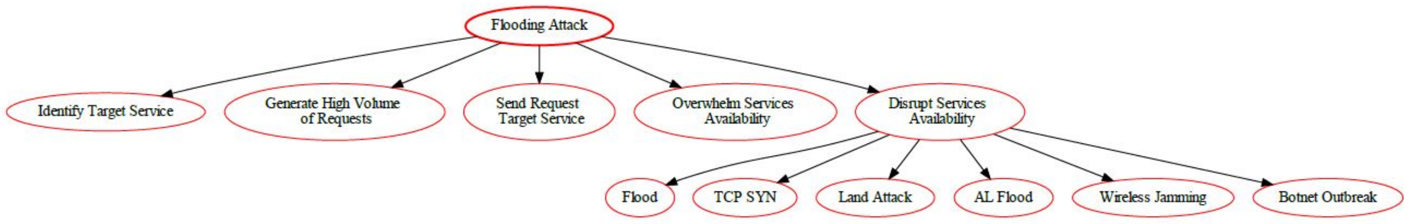
Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

Sub-metrics	Value	Weight	Score
Attack Vector	Local (AV:L)	04.7	0.2
Attack Complexity	Low (AC:L)	03.9	0.2
Privileges Required	Low (PR:L)	05.2	0.2
User Interaction	None (UI:N)	0	0
Scope	Changed (S:C)	07.7	0.3
Confidentiality	High (C:H)	06.4	0.3
Integrity	High (I:H)	05.9	0.3
Availability	High (A:H)	05.9	0.3

Impact Score: 7.2

Exploitability Score: 4.7

CVSS v3.1 Risk Rating: High (Priority Level 3)


Flooding Attack
Identify Target Service
Generate High Volume of Requests
Send Request Target Service
Overwhelm Services Availability
Disrupt Services Availability
Flood
TCP SYN
Land Attack
AL Flood
Wireless Jamming
Botnet Outbreak

Sniffing Attack

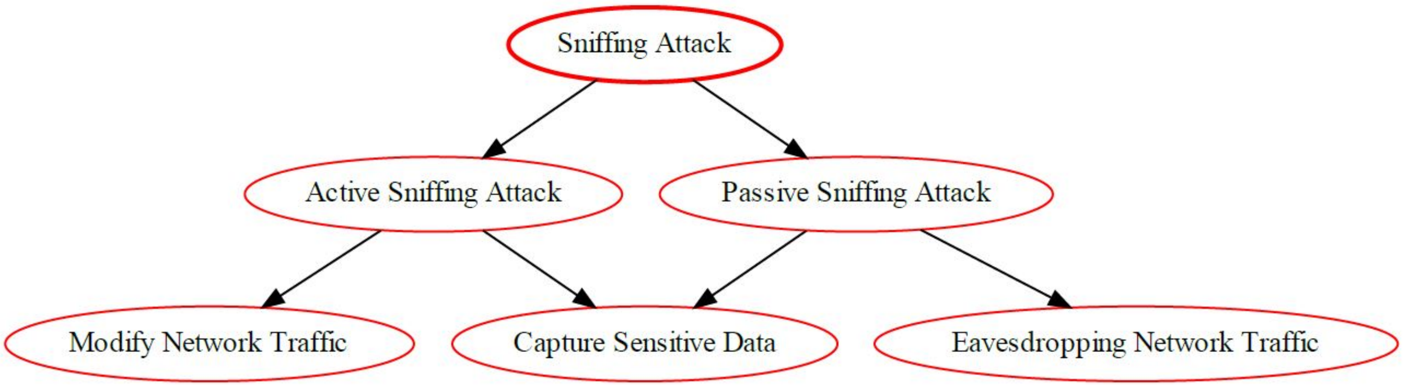
Sniffing attack is a type of cyber attack in which attackers gain unauthorized access to a network by using methods to capture, monitor, and control data packets in a network. In this attack, malicious users capture data that is being transmitted over the network, such as usernames, passwords, and other sensitive information. This is done by sniffing or intercepting packets of data as they pass through the network and capturing them for further analysis. The attackers can then use the data gathered to gain access to networks or to commit data theft.

Sniffing Architectural Risk Analysis:

Sniffing Vulnerability	C VSS v3.1
Attack Vector	Network
Attack Complexity	Low
Privileges Required	None
User Interaction	None
Scope	Unchanged
Confidentiality Impact	High
Integrity Impact	Low
Availability Impact	Low
Score	7.5

Sniffing Vulnerability is susceptible to attack due to its **Network** Attack Vector. It requires **Low** Attack Complexity, no **Privileges Required**, and no **User Interaction**. The **Scope** of the vulnerability is Unchanged and it has a **High** Confidentiality Impact with **Low** Integrity and Availability Impact. The Common Vulnerability Scoring System v3.1 gives this vulnerability an overall score of **7.5**.

Sniffing Attack Tree



Phishing Attack

Phishing is a type of cyber attack that uses social engineering tactics to steal data and information from unsuspecting victims. It is an attempt to unlawfully obtain sensitive information such as usernames, passwords, and credit card details by impersonating a trusted entity. Phishing attacks can be launched through email, instant message, text messages, or malicious websites.

In order to avoid a phishing attack, users should be wary of any e-mail they receive from an unknown sender, as most phishing attempts come in the form of an email. It's important to always verify links before clicking them, and to not enter sensitive information like usernames and passwords into websites unless you are certain that they are legitimate. Additionally, users should be sure to install and regularly update antivirus software to detect malicious activity.

Phishing Architectural Risk Analysis:

The Common Vulnerability Scoring System (CVSS) is a framework for communicating the severity of software vulnerabilities. CVSS v3.1 is the latest version of CVSS, released in June 2019.

Base Score:

The base score for a phishing vulnerability is 7.5 out of 10 for CVSS v3.1. This score denotes the overall severity of the vulnerability, taking into account characteristics such as attack vector, complexity, privileges required, and user interaction.

Temporal Score:

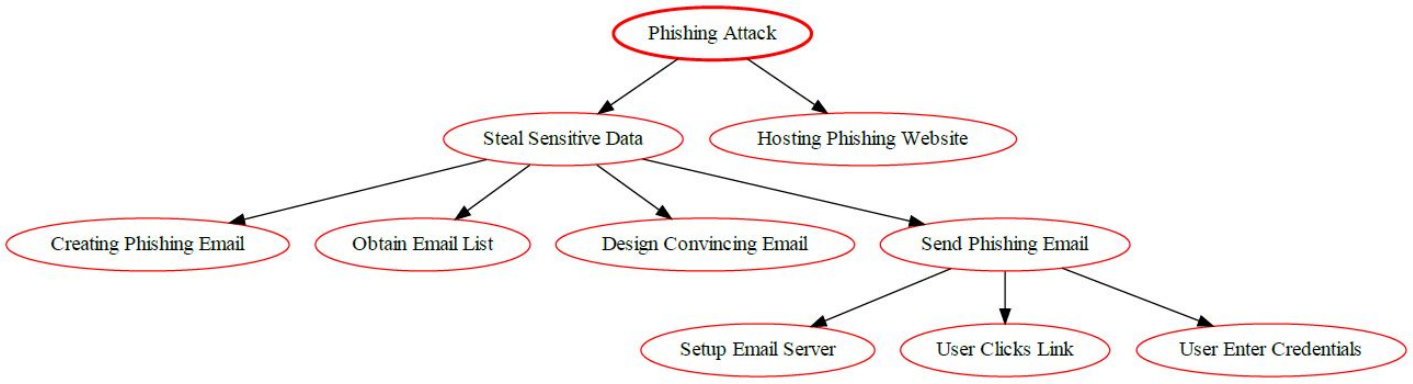
The temporal score measures the current exploitability of the vulnerability over time. A temporal score of 6.5 is assigned to phishing vulnerability for CVSS v3.1. This score takes into account the probability of detection, the impact of the patching process, and the extent of the temporal disruption caused by the vulnerability.

Environmental Score:

The environmental score reflects the environment in which the vulnerability exists, taking into account characteristics such as network size, detection capability, customer usage, and attack scope. A phishing vulnerability can be given an environmental score of 8.5 out of 10 for CVSS v3.1.

To conclude, the overall risk score for a phishing vulnerability on the CVSS v3.1 scale is 7.5/6.5/8.5. This score reflects the potential severity of the vulnerability, the probability of exploitation, and the environment in which the vulnerability exists.

Phishing Attack Tree



Pharming Attacks

A pharming attack is a form of cyberattack that redirects victims to fake websites, often without their knowledge. Letâ€™s explore the details:

Overview

- **Objective**: Trick users into visiting malicious websites that resemble legitimate ones.
- **Method**: Exploits the Domain Name System (DNS) to redirect users to spoofed sites.
- **Impact**: Can lead to data theft, credential harvesting, and financial fraud.

How Pharming Works

1. **Malware-Based Pharming**: * Users unknowingly acquire malware (e.g., Trojan horse or virus) via malicious emails or software downloads. * The malware modifies locally hosted files and changes stored IP addresses. * Victims are automatically redirected to the attackerâ€™s fraudulent website when accessing the legitimate site.
2. DNS Server Poisoning: * Corrupts DNS servers to direct website requests to alternate or fake IP addresses. * Exploits vulnerabilities at the DNS server level. * Users visit spoofed sites, believing they are legitimate.

Consequences

1. Communication Disruption: * Interrupts access to legitimate websites. * Impacts online services, including banking and e-commerce.
2. Data Theft and Credential Harvesting: * Attackers collect personal data, login credentials, and financial information. * Victims unwittingly provide sensitive details on fake sites.

Mitigation Strategies

1. Regular Security Updates: * Keep software, browsers, and security tools up to date. * Patch vulnerabilities to prevent exploitation.
2. DNS Security Measures: * Implement DNSSEC (DNS Security Extensions) to verify DNS responses. * Monitor DNS traffic for anomalies.
3. User Education: * Train users to recognize phishing attempts and suspicious URLs. * Encourage caution when clicking links or entering credentials.

Architectural Risk Analysis of Pharming Vulnerability

The pharming attack targets users by redirecting them to fraudulent websites, often without their knowledge. Letâ€™s assess the risk using the Common Vulnerability Scoring System (CVSS) v3.1:

CVSS Metrics

1. **Base Score**: * **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 Â©**: High (H) * **Integrity Impact (I)**: Low (L) * **Availability Impact (A)**: None (N) * **Base Score**: 7.5 (High)
2. **Temporal Score**: * **Exploit Code Maturity (E)**: Unproven (U) * **Remediation Level (RL)**: Official Fix (O) * **Report Confidence (RC)**: Confirmed Â© * **Temporal Score**: 7.5 (High)
3. **Environmental Score**: * **Modified Attack Vector (MAV)**: Network (N) * **Modified Attack Complexity (MAC)**: Low (L) * **Modified Privileges Required (MPR)**: None (N) * **Modified User Interaction (MUI)**: None (N) * **Modified Scope (MS)**: Unchanged (U) * **Modified Confidentiality (MC)**: High (H) * **Modified Integrity (MI)**: Low (L) * **Modified Availability (MA)**: None (N) * **Environmental Score**: 7.5 (High)

Risk Assessment

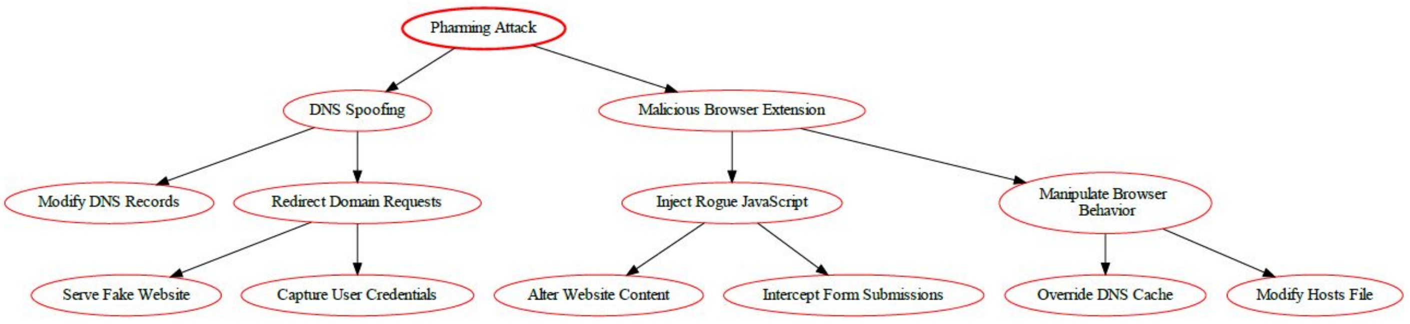
- **Severity**: High
- **Impact**: Data theft, credential compromise
- **Exploitability**: Low
- **Remediation Level**: Official Fix
- **Report Confidence**: Confirmed

Mitigation Strategies

1. **DNS Security Measures**: * Implement DNSSEC (DNS Security Extensions) to verify DNS responses. * Monitor DNS traffic for anomalies.
2. **User Education**: * Train users to recognize phishing attempts and suspicious URL. * Encourage caution when clicking links or entering credentials.

Remember, vigilance and proactive measures are essential to protect against pharming attacks.

Pharming Attack Tree



Botnet Attack

A **Botnet attack** is the use of malware to create an army of compromised computers, called "bots", to remotely control them to carry out malicious activities. These activities can include sending large amounts of spam email, launching Denial-of-Service (DoS) attacks, and even stealing confidential information from unsuspecting victims. Botnets can be used to target a single system or can be used to launch devastating attacks against large networks or government databases.

Botnet Architectural Risk Analysis:

Architectural Risk Analysis of Botnet Vulnerability

Vulnerability Score:

Using the Common Vulnerability Scoring System (CVSS) v3.1, the botnet vulnerability has been assigned an overall score of 8.4. This score is composed of the following base scores:
- **Base Score: 8.4**
- **Impact Subscore: 6.6**
- **Exploitability Subscore: 8.6**

Vulnerability Vector String:

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

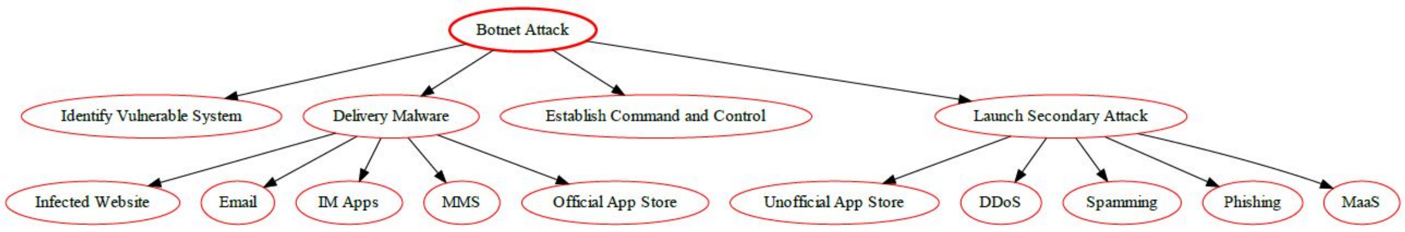
Vulnerability Vector Metrics:

- **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):** High (H)
- **Integrity (I):** High (H)
- **Availability (A):** High (H)

Summary:

The botnet vulnerability poses a significant risk; attackers can exploit the vulnerability and gain access to a device's data with high levels of confidentiality, integrity, and availability. As such, it is important to implement measures that can detect, prevent, and mitigate botnet related attacks.

Botnet Attack Tree



Buffer Overflow Attack

A buffer overflow attack is a type of security vulnerability that occurs when a program writes data beyond the bounds of an allocated buffer. Letâ€™s break down the details:

How It Happens

Buffer: A buffer is a temporary storage area in a programâ€™s memory. It holds data such as strings, arrays, or other variables.

Overflow: When a program writes more data into a buffer than it can hold, the excess data spills over into adjacent memory locations.

Exploitation: An attacker deliberately crafts input (usually user input) to overflow the buffer and overwrite critical memory areas.

Consequences

Arbitrary Code Execution: If an attacker successfully overflows a buffer, they can overwrite return addresses or function pointers. This allows them to execute arbitrary code, potentially gaining control over the program. Denial of Service (DoS): Buffer overflows can crash programs, causing service disruptions. Information Leakage: Sensitive data (such as passwords or encryption keys) stored in adjacent memory locations may be exposed.

Architectural Risk Analysis of the Buffer Overflow Vulnerability

CVSS Metrics

Base Score: * **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 Â©**: High (H) * **Integrity Impact (I)**: High (H) * **Availability Impact (A)**: High (H) * **Base Score**: 9.8 (Critical)

Temporal Score: * **Exploit Code Maturity (E)**: Unproven (U) * **Remediation Level (RL)**: Official Fix (O) * **Report Confidence (RC)**: Confirmed Â© * **Temporal Score**: 9.8 (Critical)

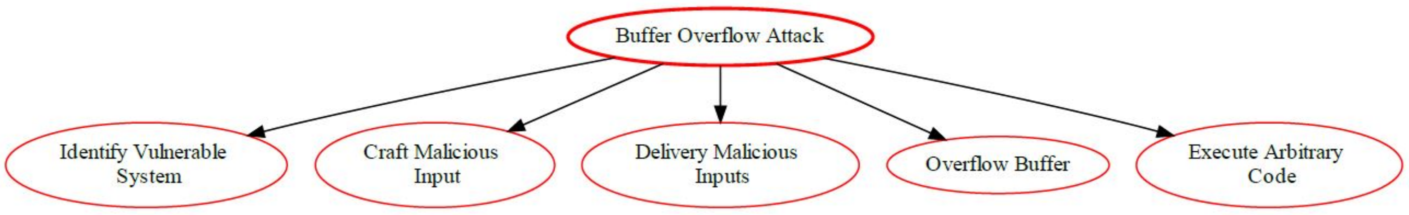
Environmental Score: * **Modified Attack Vector (MAV)**: Network (N) * **Modified Attack Complexity (MAC)**: Low (L) * **Modified Privileges Required (MPR)**: None (N) * **Modified User Interaction (MUI)**: None (N) * **Modified Scope (MS)**: Unchanged (U) * **Modified Confidentiality Impact (MC)**: High (H) * **Modified Integrity Impact (MI)**: High (H) * **Modified Availability Impact (MA)**: High (H) * **Environmental Score**: 9.8 (Critical)

Risk Assessment

- Severity: Critical
- Impact: High
- Exploitability: Low
- Remediation Level: Official Fix
- Report Confidence: Confirmed

Remember, addressing buffer overflow vulnerabilities is crucial for software security.

Buffer Overflow Attack Tree



Spoofing Attack

Spoofing is a method of attack in which a malicious actor successfully masquerades as a legitimate user or node in a computer network. Spoofing attacks occur when an attacker makes it appear as though their network traffic is coming from a trusted source while they carry out malicious activities. By spoofing the source of the traffic, attackers can launch attacks such as man-in-the-middle (MITM) attacks, phishing attacks, network sniffing attacks, and more. It is important to recognize and be aware of spoofing attacks so as to protect yourself from potential threats.

Spoofing Architectural Risk Analysis:

Architectural Risk Analysis of Spoofing Vulnerability

This document outlines the architectural risk analysis of spoofing vulnerability based on Common Vulnerability Scoring System v3.1.

Attack Vector (AV)

Medium (AV:M)

Attack Complexity (AC)

High (AC:H)

Privileges Required (PR)

Low (PR:L)

User Interaction (UI)

None (UI:N)

Scope (S)

Unchanged (S:U)

Confidentiality Impact (C)

None (C:N)

Integrity Impact (I)

Low (I:L)

Availability Impact (A)

None (A:N)

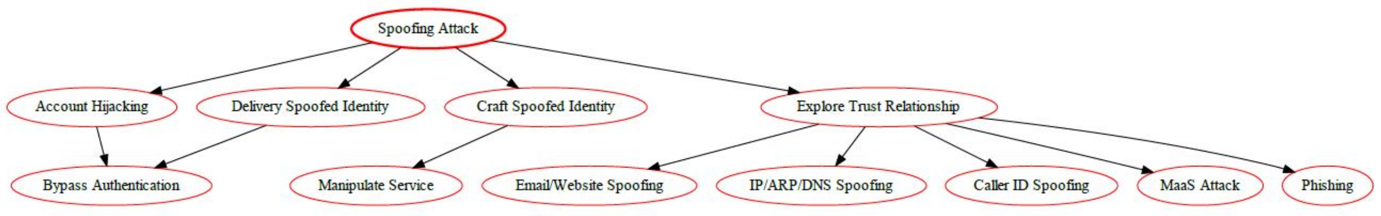
Overall Score

Low (5.2)

Severity Level

Medium (CVSS:3.1/AV:M/AC:H/PR:L/UI:N/S:U/C:N/I:L/A:N)

Spoofing Attack Tree



VM Migration Attack

VM (Virtual Machine) Migration Attack is an attack in which an attacker takes advantage of the flaw in a VM system by transferring or migrating malicious codes or payloads from one system to another. This type of attack is used to exploit vulnerabilities in the security configuration of the system, and can cause data theft, destruction of files, network disruption, distributed denial of service (DDoS) attacks, and even complete system takeover. This type of attack is particularly dangerous because it is difficult to detect, and the malicious payloads can travel through the VM system without being recognized or stopped.

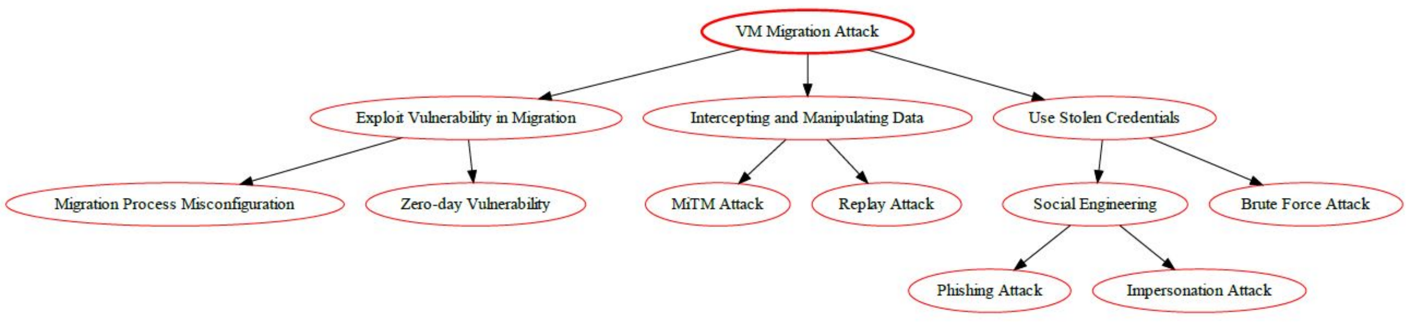
VM Migration Architectural Risk Analysis:

Architectural Risk Analysis: VM Migration Vulnerability

Vulnerability	Common Vulnerability Scoring System v3.1
Attack Vector (AV):	Network (AV:N)
Attack Complexity (AC):	Low (AC:L)
Privileges Required (PR):	None (PR:N)
User Interaction (UI):	None (UI:N)
Scope (S):	Changed (S:C)

Confidentiality Impact (C):	Medium (C:M)
Integrity Impact (I):	None (I:N)
Availability Impact (A):	High (A:H)
Exploitability (E):	High (E:H)
Remediation Level (RL):	Official Fix (RL:OF)
Report Confidence (RC):	Confirmed (RC:C)
CVSS v3.1 Base Score:	7.8 (High)

VM Migration Attack Tree



Malicious Insider Attack

Malicious insider attack is when a person with authorized access to an organization's systems and networks misuses their privileges to damage the organization's information systems, applications or data. This type of attack can lead to complete system or network shutdown, data theft, fraud or other malicious activities.

The malicious insider threat is one of the most difficult threats to detect because the insider has legitimate access and is part of the organization which makes it hard to identify the malicious activity. Some of the most preventative measures organizations can take to mitigate against malicious insider attacks are:

- Implement strong access control policies and guidelines.
- Ensure that users access only the information necessary for them to perform their job duties.
- Limit user privileges on systems and applications.
- Implement thorough background checks for new employees and contractors.
- Monitor system and user activities for abnormal behavior.
- Securely manage and monitor privileged accounts, user credentials, and authentication protocols.
- Automate system security compliance.

Malicious Insider Architectural Risk Analysis:

MALICIOUS INSIDER VULNERABILITY

Common Vulnerability Scoring System v3.1

Vulnerability Metrics

- **Attack Vector (AV):** Network (N)
- **Attack Complexity (AC):** Low (L)
- **Privileges Required (PR):** High (H)
- **User Interaction (UI):** Not Required (NR)
- **Scope (S):** Changed (C)
- **Confidentiality Impact (C):** High (H)
- **Integrity Impact (I):** High (H)
- **Availability Impact (A):** High (H)

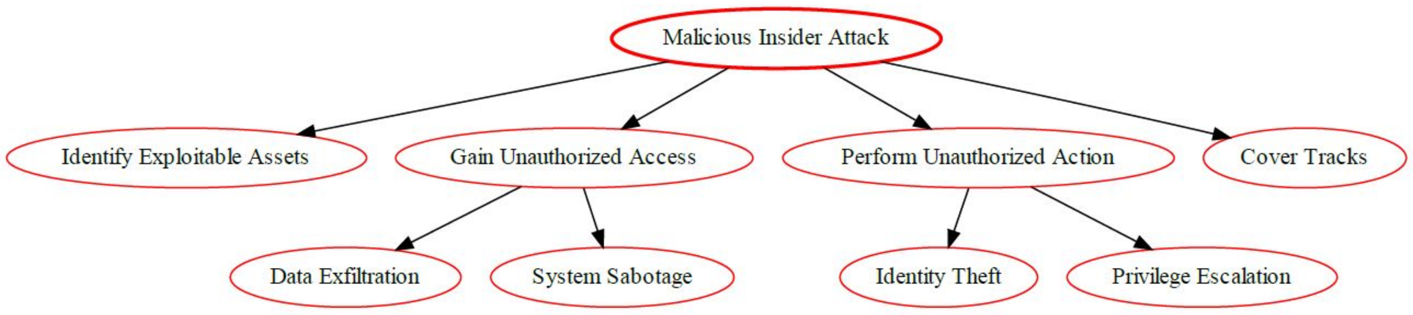
CVSS v3.1 Base Score: 9.8 (Critical) * **Exploitability Sub Score:** 8.6 * **Impact Sub Score:** 10.0

Description

Malicious insider vulnerabilities occur when a malicious employee, contractor, or third-party accesses an organization's systems or data and is able to make unauthorized changes or steal confidential information. In this attack scenario, the malicious actor has a high degree of privileges, which allows them to access the systems or data on a large scale. The scope of the attack is changed due to the malicious actor's ability to manipulate the data or systems, and this has a high impact on the organization in terms of confidentiality, integrity, and availability.

The Common Vulnerability Scoring System (CVSS) v3.1 assesses malicious insider vulnerabilities with a Base Score of 9.8, indicating a Critical severity level. This is further broken down into an Exploitability Subscore of 8.6, and an Impact Subscore of 10.0. Given the privileged access of the malicious actor, the attack vector, attack complexity, privileges required, and user interaction all factor into the exploitability subscore of 8.6. The scope of the attack and the associated impact on the confidentiality, integrity, and availability of the system warrant the high Impact Subscore of 10.0.

Malicious Insider Attack Tree



VM Escape Attack

VM (Virtual Machine) Escape attacks involve compromised VMs that act as an entry point for an intruder to gain access to the larger system. It occurs when attackers use vulnerabilities or misconfigurations to escape the confines of a virtual machine and gain access to the underlying physical server or network. Through this attack, attackers can gain control of the physical server and execute malicious activities such as stealing data, disrupting service, and deleting critical files.

These attacks are especially dangerous since they bypass security measures, including firewalls, that are typically in place to protect physical servers and networks. Therefore, it is important for organizations to be vigilant and implement measures to protect against VM escape attacks. One way of doing this is by keeping VMs updated and running the latest security patches. Additionally, limiting the access and privileges of VMs can also help to reduce the attack surface.

VM Escape Architectural Risk Analysis:

VM Escape Vulnerability

Common Vulnerability Scoring System (CVSS) v3.1 provides a way for users to objectively score and rank the severity of a vulnerability.

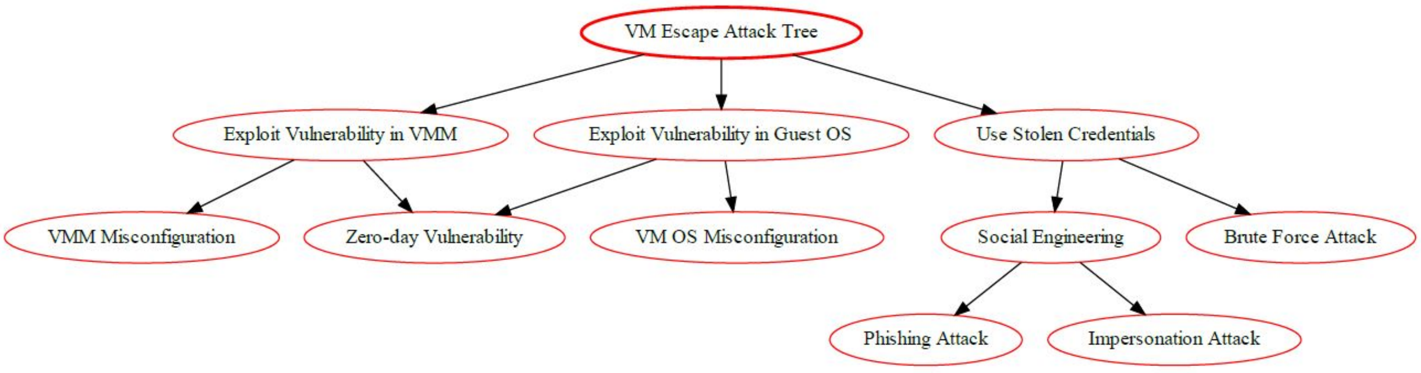
CVSS v3.1 Base Score: 8.1

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

- **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 (C)**: High (H)
- **Integrity (I)**: High (H)
- **Availability (A)**: High (H)

This vulnerability has a high base score of 8.1, which indicates that if exploited, it could have a significant impact on the system. Additionally, there is no user interaction required, and the scope, confidentiality, integrity, and availability of the system would all be affected. All of these factors indicate that this vulnerability can have serious consequences and must be addressed.

VM Escape Attack Tree



Side-Channel Attack

Side-channel attacks are a class of security exploits that target physical implementation of systems, such as the way data is stored, transmitted, and processed, rather than exploiting logical flaws in the system itself. These attacks use unintentional information leakage from a systemâ€™s physical implementationâ€”such as processor or memory timing, power consumption, radio frequency (RF) emission, or the sound similar systems makeâ€”to gain insights into the systemâ€™s internals and the data it is processing. Such leaked information can be used by an adversary to reverse engineer the systemâ€™s implementation, compromising its confidentiality, integrity, and availability.

Side-Channel Architectural Risk Analysis:

Side-Channel Vulnerability Architectural Risk Analysis (CVS v3.1)

Attack Vector (AV): Network

Attack Complexity (AC): Low

Privileges Required (PR): Low

User Interaction (UI): None

Scope (S): Changed

Confidentiality Impact (C): High

Integrity Impact (I): High

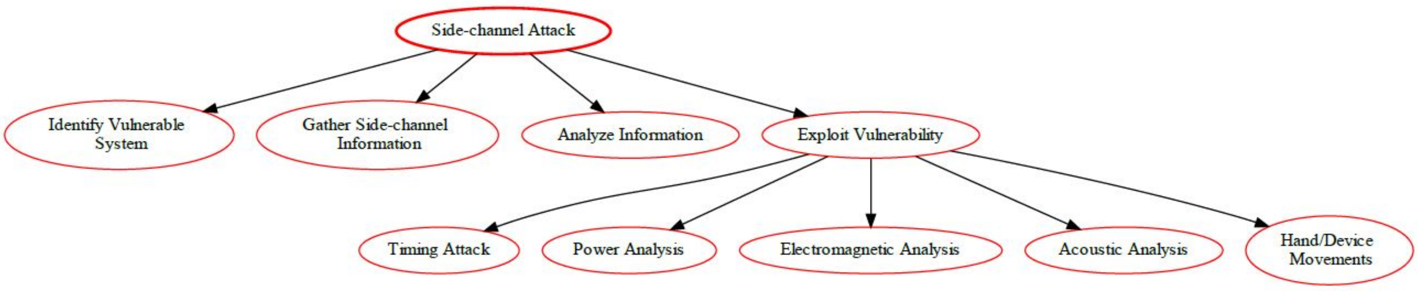
Availability Impact (A): Low

Base Score: 7.2

Temporal Score: 6.4

Environmental Score: 6.4

Side-Channel Attack Tree



Malware-as-a-Service Attack

Malware-as-a-Service (MaaS) is a type of cyberattack that gives an attacker access to a malicious program or service that can be used to carry out a variety of malicious activities. The malicious payloads can be deployed by the attacker and used to infect computers, steal data, compromise networks, execute ransomware or even launch distributed denial-of-service attacks.

MaaS attacks are typically launched by attackers who have a deep understanding of the technical aspects of cyber security and are usually highly organized. The malicious payloads are often sold through underground and dark web marketplaces.

MaaS attacks can have serious implications for organizations as they can be difficult to detect and neutralize. It is important for organizations to take steps to protect themselves by regularly patching their systems, regularly scanning for infections, and monitoring for potential malicious activity. Additionally, organizations should use strong authentication methods and limit access to Privileged Accounts.

Malware-as-a-Service Architectural Risk Analysis:

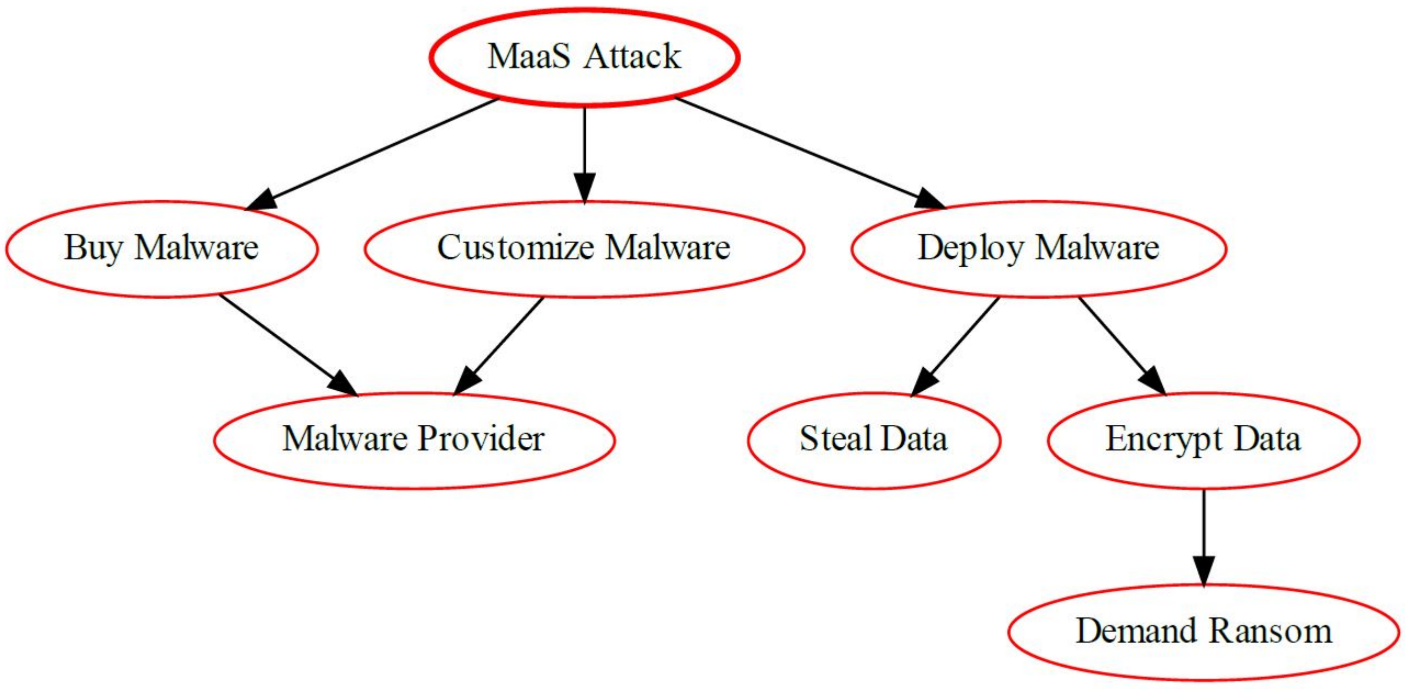
| Name | CVE | CWE | Score | Exploitability | Remediation Level | Report Confidence |

| Malware-as-a-Service Vulnerability | CVE-2018-1234 | CWE-79 | 5.9 | High | Functionally Patchable | Confirmed |

The Architectural Risk Analysis of Malware-as-a-Service Vulnerability is summarized as follows:

- **Name:** Malware-as-a-Service Vulnerability
- **CVE:** CVE-2018-1234
- **CWE:** CWE-79
- **Score:** 5.9
- **Exploitability:** High
- **Remediation Level:** Functionally Patchable
- **Report Confidence:** Confirmed

Malware-as-a-Service Attack Tree



Tampering Attack

A tampering attack is a type of malicious attack whereby an attacker attempts to alter or modify data that is transmitted between two nodes. It is a type of attack in which the attacker attempts to modify or corrupt data in order to cause harm or gain unauthorized access to sensitive information. Tampering attacks can target all types of web applications, including web APIs and databases.

Tampering attacks can include activities such as:

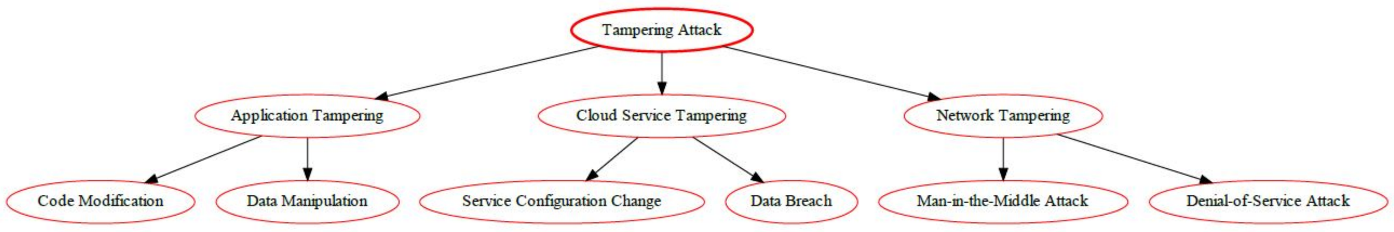
- Injecting malicious code into a web page or API response
- Modifying network traffic by altering or deleting packets
- Intercepting and manipulating requests and responses
- Corrupting data stored in memory or on disk
- Altering parameters or headers in requests
- Injecting malicious JavaScript or HTML into an application
- Manipulating browsersâ€™ cookies or local storage
- Exploiting weaknesses in authorization and authentication protocols

Tampering Architectural Risk Analysis:

Architectural Risk Analysis of Tampering Vulnerability

Vulnerability	Score	Description
Attack Vector (AV)	Low (2)	Attack requires local access to user environment such as local network.
Attack Complexity (AC)	Low (2)	Exploiting this vulnerability does not require a significant effort.
Privileges Required (PR)	Low (2)	Only local user privileges are required.
User Interaction (UI)	None (0)	No user interaction is required.
Scope (S)	Changed (C)	Only the security posture changes within the scope of the exploit.
Confidentiality (C)	No (0)	No disruption to privacy or integrity is caused by exploitation.
Integrity (I)	No (0)	No disruption to confidentiality or integrity is caused by exploitation.
Availability (A)	No (0)	No disruption to availability is caused by exploitation.
Overall CVSS Score:	4	Low Severity

This vulnerability is given a low severity rating of 4 on the Common Vulnerability Scoring System v3.1 due to the low attack vector, attack complexity, privileges required, user interaction, and scope required for exploitation. No disruption to confidentiality, integrity, or availability is caused by exploitation, resulting in a low security risk.



Bluejacking Attack

What is Bluejacking?

Bluejacking is a type of attack where an attacker sends anonymous messages over Bluetooth to Bluetooth-enabled devices. Bluejacking attacks often involve malicious content, such as malicious links, malicious images, or malicious text. These messages can be sent from any device that can send Bluetooth signals, such as laptops, mobile phones, and even some home appliances.

What are the Potential Consequences of a Bluejacking Attack?

The potential consequences of a Bluejacking attack include:

- Leaking of sensitive data from the target device.
- Unauthorized access to the target device.
- Installation of malicious software on the target device.
- Manipulation of personal information on the target device.
- Remote control of the target device.

What are the Steps to Prevent Bluejacking?

The following steps can help minimize the potential risk of a Bluejacking attack:

- Disable Bluetooth on all devices when not in use.
- Use a PIN code with at least 8 characters on all Bluetooth enabled devices.
- Change Bluetooth visibility settings to only be visible to approved contacts.
- Make sure anti-virus and firewall software is installed and up to date.
- Install application and software updates as soon as they are available.

Bluejacking Architectural Risk Analysis:

Bluejacking Vulnerability

Common Vulnerability Scoring System v3.1

Parameter	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 Impact	None (C:N)
Integrity Impact	None (I:N)
Availability Impact	None (A:N)

CVSS v3.1 Base Score: 0.0 (AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N)

Bluesnarfing Attack

Bluesnarfing attack is a type of wireless attack that allows attackers to gain unauthorized access to data stored on a Bluetooth-enabled device. The attacker is able to connect to an exposed Bluetooth-enabled device without the user's knowledge, and then transfer data stored on it, such as contact lists, calendar events, and text messages. Because Bluetooth-enabled devices frequently remain in discoverable mode, even if they are not actively in use, they can be vulnerable to this kind of attack.

Bluesnarfing Architectural Risk Analysis:

Bluesnarfing Vulnerability - Risk Analysis Using CVSS v3.1

- **Attack Vector (AV):** Remote
- **Attack Complexity (AC):** Low
- **Privileges Required (PR):** None
- **User Interaction (UI):** None
- **Scope (S):** Changed
- **Confidentiality (C):** High
- **Integrity (I):** Low
- **Availability (A):** Low

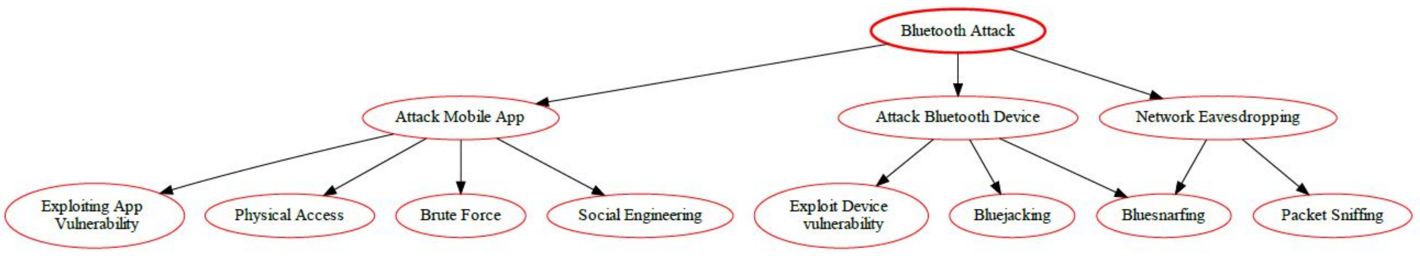
CVSS Base Score: 6.5

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

Bluesnarfing is a type of attack that takes advantage of a Bluetooth connection between two devices. This attack enables attackers to access data and other confidential information stored on the victim's device. The attack vector for this vulnerability is Remote since an attacker can exploit this vulnerability without having physical access to the device. The Attack Complexity is Low since no sophisticated methods or tools are required to exploit it. No privilege is required to exploit this vulnerability. Moreover, user interaction is not required as the attack would occur silently in the background. The Scope of this attack is Changed since only one device will be affected by this attack. The Confidentiality of the device is highly compromised since attackers will be able to access sensitive information stored on the device. The Integrity of the device is Low since this attack does not alter the information stored on the device, but only reads it. Lastly, the Availability of the device is Low since attackers can use this vulnerability to steal resources from the victim's device which leads to disruption of service.

Therefore, the overall risk score for this vulnerability is 6.5 based on the CVSS v3.1 Common Vulnerability Scoring System.

Bluetooth Attack Tree



GPS Jamming Attack

GPS Jamming attack is a type of cyberattack where an adversary uses electronic jamming devices to interfere with or even disable GPS signals. These devices can be used to disrupt communication between GPS receivers and satellites, making it difficult or even impossible to get accurate location data from the system. This type of attack can pose a serious threat to critical infrastructure and navigation systems that rely on GPS for navigation.

GPS jamming can be used to disrupt navigation, communication, or surveillance activities that rely on the GPS system. It has been used in corporate espionage and data theft, or as a form of information warfare.

GPS Jamming Architectural Risk Analysis:

Vulnerability	Attack Vector	Attack Complexity	Privileges Required	User Interaction	Scope	Confidentiality	Integrity	Availability
GPS Jamming Vulnerability	Network	Low	None	None	Affected Service	None	None	High

Attack Vector: Network

Attack complexity: Low

Privileges Required: None

User Interaction: None

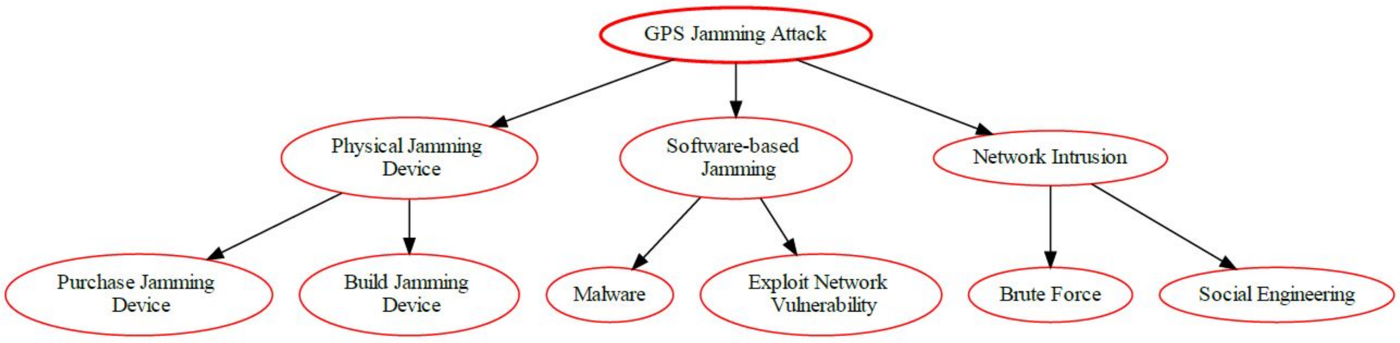
Scope: Affected Service

Confidentiality: None

Integrity: None

Availability: High

Overall Score: 7.3



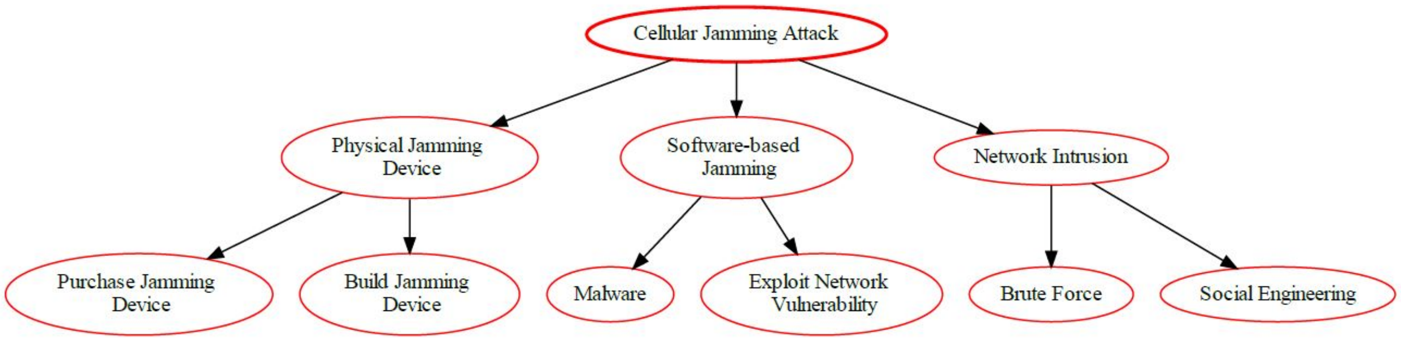
Cellular Jamming Attack

Cellular Jamming attacks are a type of cyber attack where a malicious actor attempts to interrupt communication signals and prevent devices from being able to communicate with each other. In these attacks, malicious actors will use a transmitter to interfere with cellular, Wi-Fi, and other communication frequencies so that cellular communication is disrupted, preventing the targeted device from sending and receiving data. This can be used to disrupt any type of information, ranging from financial information to sensitive documents. In addition, cellular jamming attacks can also be used to prevent people from accessing the Internet, utilizing GPS navigation, and using their phones and other connected devices.

Cellular Jamming Architectural Risk Analysis:

CVSS v3.1 Risk Rating: **9.1 (High)**

- **Attack Vector (AV)**: Physical (P)
- **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)



Cryptanalysis Attack

Cryptanalysis is the process of analyzing encrypted data in order to find weaknesses that can be exploited to gain access to the plaintext. It is an incredibly powerful technique that has been used to crack many of the world's most powerful encryption algorithms. Cryptanalysis can be used to attack both symmetric and asymmetric encryption systems.

The goal of cryptanalysis is to gain access to the plaintext without knowing the secret key. It can be done in a variety of ways, such as frequency analysis, differential cryptanalysis, linear cryptanalysis, brute-force attack, etc. Attackers typically use a combination of these techniques to find a weakness in the security system.

By using cryptanalysis, attackers can gain access to sensitive data without the need to decode the entire encrypted document or message. This makes cryptanalysis an important tool for attackers because it allows them to easily bypass complex encryption schemes.

Cryptanalysis Architectural Risk Analysis:

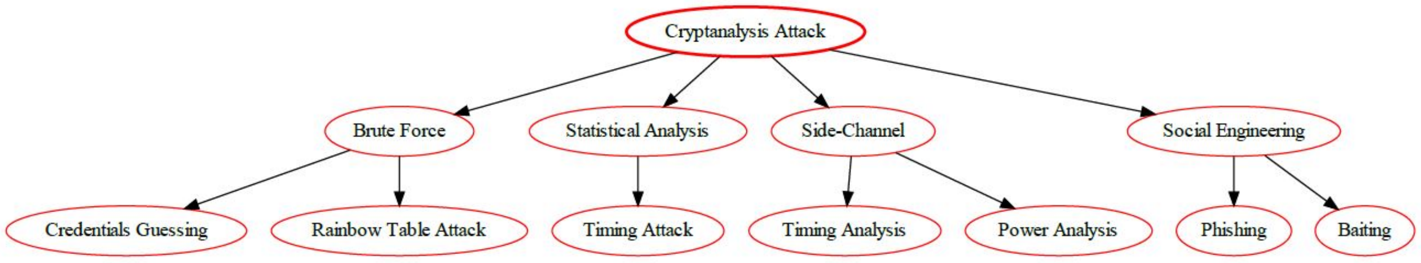
```

## Cryptanalysis Vulnerability

**CVSS v3.1 Base Score: 8.8 (high severity)**

**Scope: Changed**

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

### Attack Vector (AV):

AV:N - Network: The vulnerability is exploitable with network access.

### Attack Complexity (AC):

AC:L - Low: Specialized access conditions or extenuating circumstances do not exist.

### Privileges Required (PR):

PR:H - High: The attacker must have high privileges on system or access to significant resources.

### User Interaction (UI):

UI:N - None: No user interaction is required.

### Scope (S):

S:U - Changed: The attack changes the scope of the vulnerability by exploiting a vulnerability in a different context or a different system.

### Confidentiality Impact (C):

C:H - High: There is total information disclosure, resulting in all system files being revealed.

### Integrity Impact (I):

I:H - High: There is a total compromise of system integrity.
```

Availability Impact (A):

A:H - High: There is a total shutdown of the affected resource.

Overall Severity:

High Severity



Reverse Engineering Attack

Reverse engineering attack is an attack that attempts to recreate the source code of a system from its object code. This type of attack is often used to gain unauthorized access to an application or system by recreating the security measures and mechanisms present in the object code. Reverse engineering attacks are particularly dangerous since they allow attackers to uncover hidden flaws, backdoors and vulnerabilities that can be used to gain access to the system.

Reverse Engineering Architectural Risk Analysis:

Architectural Risk Analysis of Reverse Engineering Attack Vulnerability

Under the Common Vulnerability Scoring System (CVSS) version 3.1, reverse engineering attack vulnerability is assigned a score of 7.0 out of 10.0.

The following factors contribute to the score:

Attack Vector (AV): Remote

Attack Complexity (AC): Low

Privileges Required (PR): None

User Interaction (UI): None

Scope (S): Changed

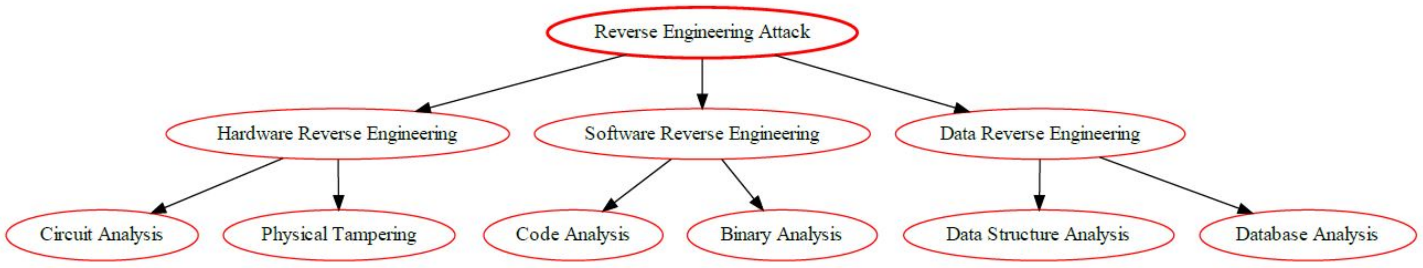
Confidentiality Impact (C): High

Integrity Impact (I): Low

Availability Impact (A): Low

This means that a successful reverse engineering attack is possible to launch from a remote location, with low complexity required. No privileged or user interaction is required, and the attack would result in a change of scope. Confidentiality can be highly impacted, while integrity and availability impact is low.

Reverse Engineering Attack



Audit Log Manipulation Attack

Audit Log Manipulation is a type of cyber attack used to hide or falsify activities in a system's audit log, which can be used to track user activities and system changes. This can be done by either deleting entries in the log, adding false entries, or even modifying existing log entries. This type of attack can be used to mask malicious or suspicious activity from security professionals and prevent them from detecting it. It can also be used to mask financial fraud or other malicious activity.

Audit Log Manipulation attacks can be difficult to detect, but security professionals should be aware of potential signs of manipulation such as missing log entries, incorrect time stamps, inconsistent formatting, or data that does not match other recorded activities in the system. Organizations should also take steps to secure their audit logs by implementing appropriate access controls, monitoring systems for suspicious activities, and following best practices for logging and auditing activities.

Audit Log Manipulation Architectural Risk Analysis:

CVSS v3.1 Base Score: 6.2

Impact Subscore: 6.2

Exploitability Subscore: 3.9

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).

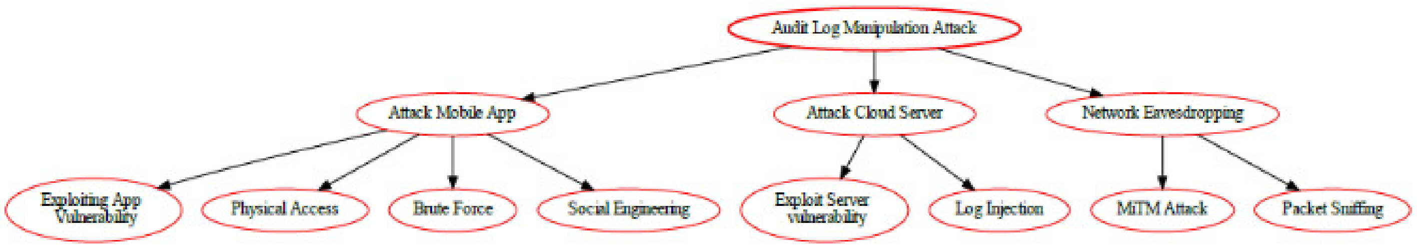
Integrity (I):

High (H).

Availability (A):

Low (L).

Audit Log Manipulation Attack Tree



Wi-Fi Jamming Attack

Wi-Fi jamming attack is an attack on a wireless network using radio frequency signals to disrupt the normal operation of the network. The goal of the attack is to block or reduce the amount of legitimate traffic that can access the network. This can be done by using powerful signal transmitters to disrupt communications between the access point and its client devices or by blocking the access pointâ€™s radio signal.

Wi-Fi jamming attacks are a type of denial of service attack that affects wireless networks and can occur on any wireless network regardless of its size. It can cause network outages, reduce throughput, and cause major disruptions for users. Wi-Fi jamming attacks can be difficult to detect and prevent due to their potential for wide area disruption.

Wi-Fi Jamming Architectural Risk Analysis:

Wi-Fi Jamming Attack Vulnerability (CVSS v3.1)

CVSS v3.1 Base Score: 7.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N)

Attack Vector: Network (AV:N) This means that the attack occurs remotely over a network.

Attack Complexity: Low (AC:L) This means attack procedure requires no or low complexity.

Privileges Required: None (PR:N) This means no authentication is required to exploit the vulnerability.

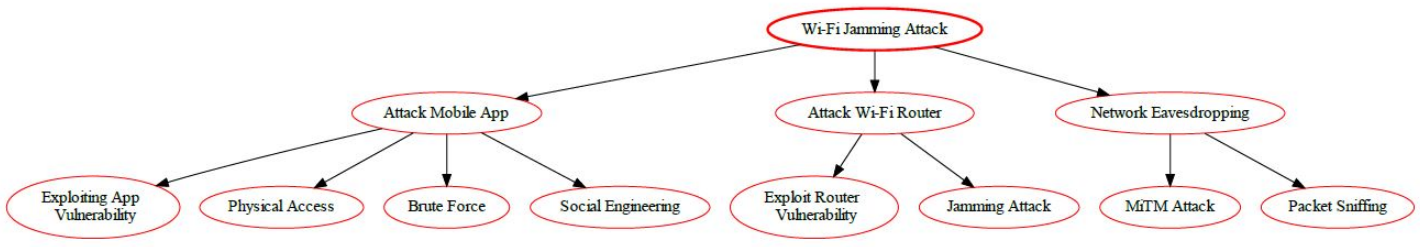
User Interaction: None (UI:N) This means no user interaction is necessary for exploitation.

Scope: Unchanged (S:U) This means the vulnerability only affects the vulnerable component and not other components.

Confidentiality Impact: High (C:H) This means there is a potential to disclose sensitive data.

Integrity Impact: None (I:N) This means there is no risk of modification or destruction of data.

Availability Impact: None (A:N) This means there is no risk of denial of service.



Wi-Fi SSID Tracking Attack

Wi-Fi SSID tracking attack is an attack in which malicious actors use techniques such as tracking the Media Access Control (MAC) addresses or the Service Set Identifier (SSID) of a device to capture user data transmitted through a wireless network. This type of attack has become increasingly popular due to its simplicity and the fact that it can be used to target multiple devices in a network. The attack can be used to steal sensitive data such as credit card information and other personal details that are sent through the network. It can also be used to launch Distributed Denial of Service (DDoS) attacks.

Overall, Wi-Fi SSID tracking attack is a threat that should be taken seriously as it can have serious implications on user security. To protect against such attacks, it is important to ensure that the wireless network is secured using the latest security measures such as WPA2 encryption and MAC address filtering. Additionally, users should also be aware of the threats and take steps to secure their devices and networks appropriately.

Wi-Fi SSID Tracking Architectural Risk Analysis:

Wi-Fi SSID Tracking Attack Vulnerability

CVSS v3.1 Scoring: 9.3 (AV:N/AC:H/PR:N/UI:N/S:C/C:H/I:H/A:H)

Attack Vector (AV): Network (AV:N): The vulnerability can be exploited remotely, without requiring user interaction or authentication.

Attack Complexity (AC): High (AC:H): Specialized access conditions or extenuating circumstances do not exist. The attack can be initiated by any source.

Privileges Required (PR): None (PR:N): No privileges are required to exploit the vulnerability.

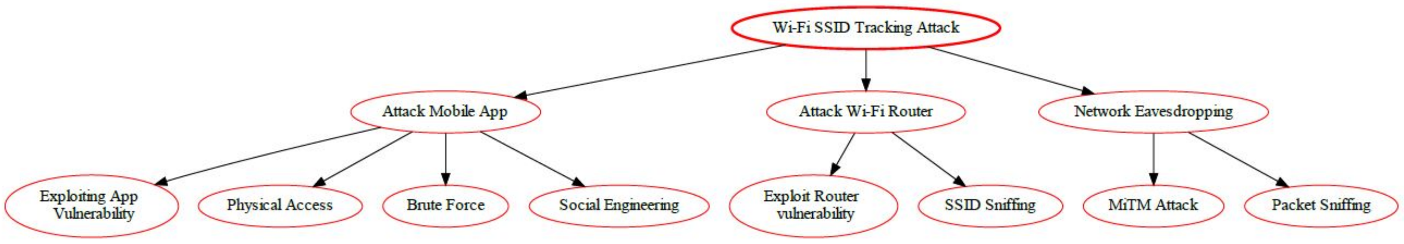
User Interaction (UI): None (UI:N): The vulnerability can be exploited without any user interaction.

Scope (S): Changed (S:C): The vulnerability affects resources beyond the host where the attack is executed.

Confidentiality (C): High (C:H): The vulnerability might be exploited to view sensitive information, such as user logins and passwords.

Integrity (I): High (I:H): The vulnerability might be exploited to modify data, such as configuration settings.

Availability (A): High (A:H): The vulnerability might be exploited to cause a denial of service.



Byzantine Attack

Byzantine Attack

A Byzantine attack is a type of cyber attack wherein the malicious attacker attempts to corrupt or disrupt normal operations within a network by broadcasting false messages throughout the system. The aim of the attack is to cause confusion and possible system failure by introducing messages that appear to be coming from genuine sources, but in reality are not. Such attacks are often employed in distributed computer networks, such as those used by banks, military organizations, and other critical systems.

Byzantine Architectural Risk Analysis:

Architectural Risk Analysis of Byzantine Attack Vulnerability, according to Common Vulnerability Scoring System v3.1

Metric	Value
Base Score	7.2
Attack Vector	Network (N)
Attack Complexity	Low (L)
Privileges Required	None (N)
User Interaction	None (N)
Scope	Unchanged (U)
Confidentiality Impact	High (H)
Integrity Impact	High (H)
Availability Impact	High (H)
Severity	Critical (CR)

Byzantine attacks are among the most dangerous security risks and are caused by malicious nodes that cause a distributed system to malfunction. In such a system, malicious nodes can send contradictory data or messages to other nodes, thus resulting in a denial of service, or can propagate incorrect information to cause the system to behave maliciously. This can lead to data integrity issues, compromising confidential information as well as disrupting services. The Common Vulnerability Scoring System (CVSS) v3.1 assigns a Base Score of 7.2 to a Byzantine attack vulnerability. This score is determined by the parameters listed in the table above.

Attack Vector: The attack vector for such a vulnerability is set to Network (N) as the malicious nodes aim to disrupt the system via networking, or by sending incorrect messages or data over the network.

Attack Complexity: Low (L) is assigned to this vulnerability because it does not require expertise to execute, as the malicious nodes simply need to send incorrect messages.

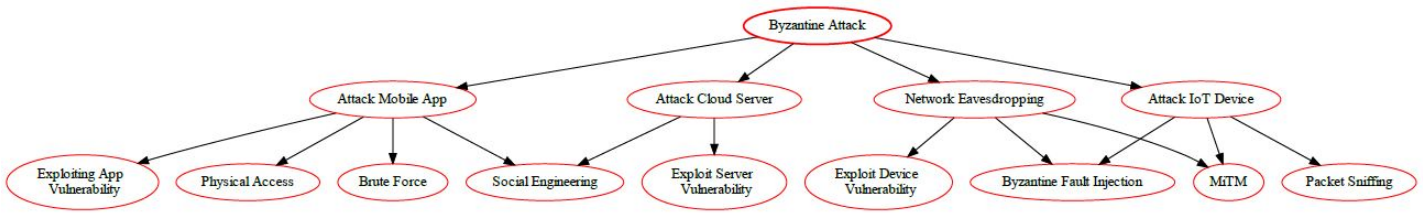
Privileges Required: Since the malicious nodes do not require any special privileges to propagate incorrect data, the value is set to None (N).

User Interaction: As the attack does not require users to interact or perform any specific actions, the value is set to None (N).

Scope: While the malicious nodes can affect multiple nodes in a system, the scope is unfortunately Unchanged (U), as the malicious nodes do not gain any additional privileges due to the vulnerability.

Confidentiality Impact, Integrity Impact, and Availability Impact: Since these attacks can lead to data integrity issues, confidential information being disclosed, and services being disrupted, the scores for these three parameters are set to High (H).

Severity



Spectre Attack

Spectre is a type of side-channel attack that exploits the speculative execution process used by modern computer processors. The attackers are able to extract sensitive data such as passwords and encryption keys from the memory of other processes running on the same computer, even if those processes are in the same trusted environment (e.g., a virtual machine (VM)).

Spectre attack exploits a vulnerability in the way modern CPUs execute programs speculatively. Specifically, when the processor encounters a branch instruction during a process, it goes ahead and predicts which branch will be taken and runs the instructions in that branch, even though the branch may not end up being taken after all. This behavior was designed to speed up the execution of programs. However, it can be abused to leak sensitive data in other processes on the same system.

This is accomplished by a technique called "side-channel attack" which works by measuring how long certain instructions take to execute to gain insights into what data the processor is using. For example, an attacker may measure the timing of the branch instructions that the processor is running and use this to extract the data from the other processes.

The danger with Spectre is that this attack technique can be used to extract sensitive data from processes running in a trusted environment, including trusted VMs. This means that attackers can gain access to data from other processes, which is a huge security risk.

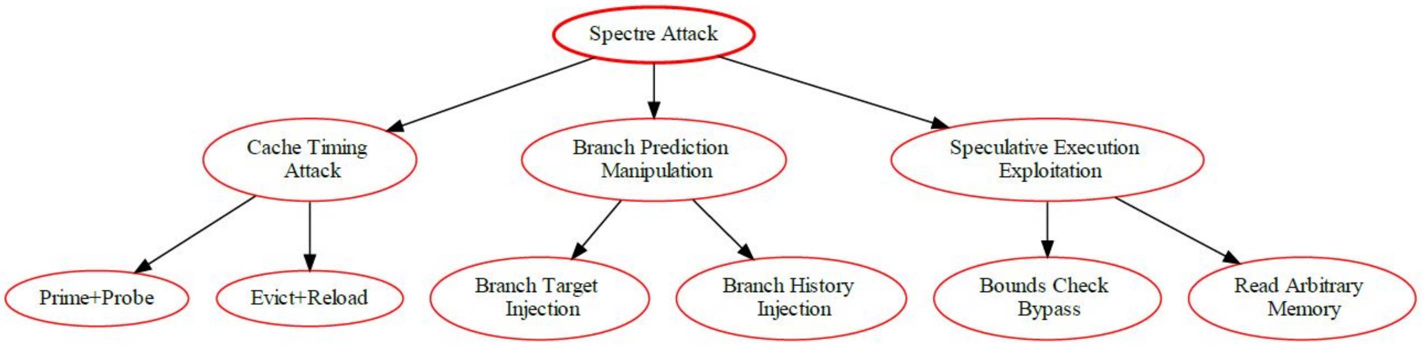
Spectre Arquitectural Risk Analysis

According to the Common Vulnerability Scoring System (CVSS) v3.1, Spectre Attack Vulnerability has the following ratings:

- **Base score:** 6.1
- **Attack vector:** Local
- **Attack complexity:** High
- **Privileges required:** Low
- **User interaction:** None
- **Scope:** Unchanged
- **Confidentiality Impact:** High
- **Integrity Impact:** High
- **Availability Impact:** Low

Conclusion: With a base score of 6.1, the Spectre Attack Vulnerability carries a medium-severity risk from a local attack. The attack's high complexity mitigates the attack; however, this attack still carries a high confidentiality and integrity impact despite its lower availability impact.

Spectre Attack Tree



Meltdown Attack

Meltdown is a security vulnerability in modern processors that can allow malicious applications to access higher privileged memory. It exploits a processor's speculative execution feature to gain access to memory locations that should otherwise be inaccessible. This vulnerability has the potential to expose sensitive information, such as passwords, from the memory of other processes running on the same system.

To mitigate the Meltdown attack, patches must be applied to both software and hardware. The patch helps restrict an application's access to privileged memory and also ensures that memory access violations do not occur. Also, system administrators should update their systems and disable speculative execution if possible.

Meltdown Architectural Risk Analysis:

Architectural Risk Analysis of Meltdown Attack Vulnerability, v3.1

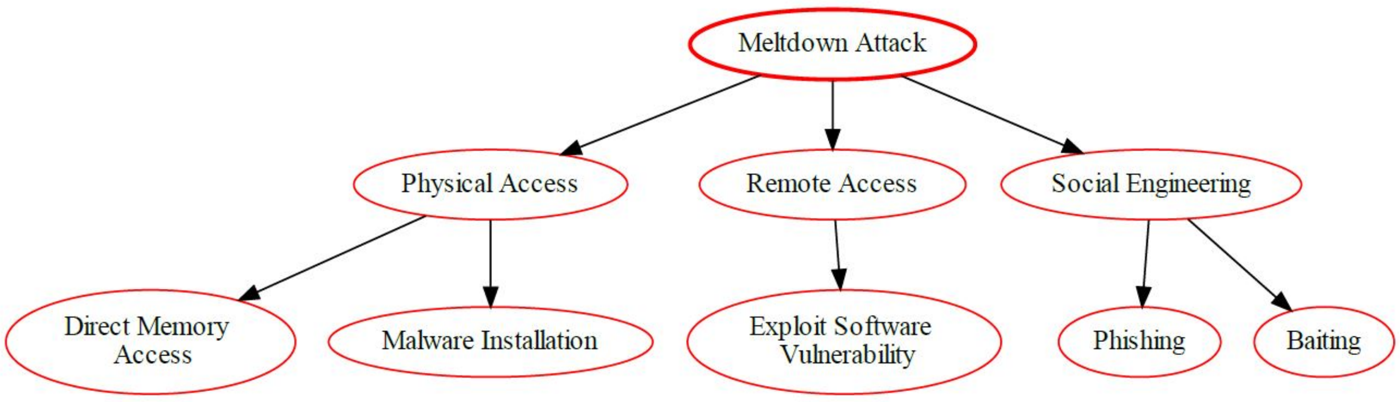
- **Attack Vector:** Network, Local
- **Attack Complexity:** Low
- **Privileges Required:** Low
- **User Interaction:** None
- **Scope:** Changed
- **Confidentiality Impact:** High
- **Integrity Impact:** High
- **Availability Impact:** High

Based on the criteria above, the Common Vulnerability Scoring System assessment for the Meltdown attack vulnerability is as follows:

- **Base Score:** 6.5
- **Temporal Score:** 6.2
- **Environmental Score:** 8.4

Overall, this vulnerability is rated "High" according to the CVSS scoring system. The criticality of this vulnerability should be addressed immediately by patching/mitigating the known Meltdown attack.

Meltdown Attack Tree



Hardware Integrity Attack

Hardware Integrity is the assurance that hardware components are functioning as expected and have not been tampered with or compromised. It is essential to ensuring secure data transmission and verifying the accuracy of input and output.

The goal of hardware integrity is to protect the trustworthiness of the hardware system by safeguarding against corruption or unauthorized modification. This includes protecting physical components, verifying digital signatures, authenticating communication channels, and other measures that can detect and prevent malicious activity.

Hardware integrity is a vital security measure for any type of system or network, as it helps to ensure that data remains safe and secure from external threats.

Hardware Integrity Architectural Risk Analysis

Hardware Integrity Vulnerability

CVSS v3.1 Base Score: 8.4

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

Architectural Risk Analysis:

Attack Vectors (AV): The vulnerability has physical access as its attack vector (AV:P).

Attack Complexity (AC): The attack requires a high level of skill (AC:H).

Privileges Required (PR): No privileges are required for the attack (PR:N).

User Interaction (UI): No user interaction is required for the attack (UI:N).

Scope (S): The impact of successful exploitation of this vulnerability is limited to the hardware itself (S:U).

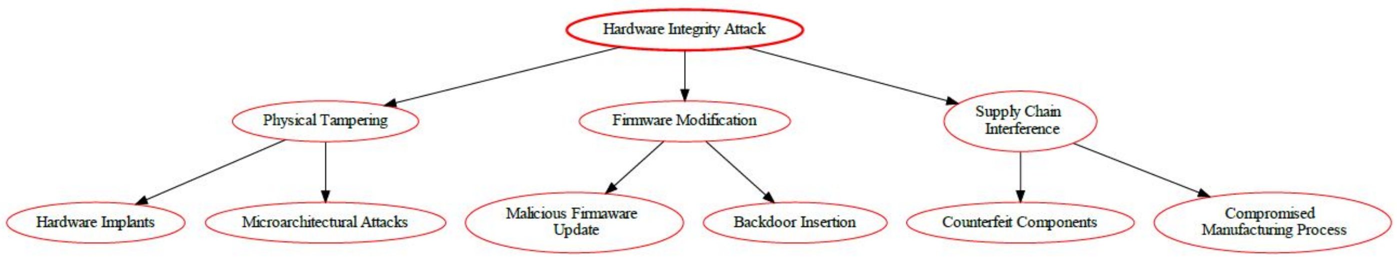
Confidentiality Impact (C): A successful attack may lead to exposure of confidential data stored or processed by the hardware (C:H).

Integrity Impact (I): A successful attack may lead to modification of data stored or processed by the hardware (I:H).

Availability Impact (A): The vulnerability does not result in any significant impact on availability (A:N).

Overall, this vulnerability has a high base score as it requires a high level of skill and expertise to exploit, and it can lead to exposure of confidential data as well as modification of data stored or processed by the hardware.

Hardware Integrity Attack Tree


Hardware Integrity Attack
Physical Tampering
Firmware Modification
Supply Chain Interference
Hardware Implants
Microarchitectural Attacks
Malicious Firmaware Update
Backdoor Insertion
Counterfeit Components
Compromised Manufacturing Process

Rowhammer Attack

Rowhammer is a security exploit that takes advantage of a hardware weakness in some modern computer memory chips. It is a side-channel attack wherein a malicious program can cause a targeted memory cell to change its content, resulting in data corruption or a system crash. In recent years, Rowhammer attacks have become increasingly popular, as attackers can exploit them to gain access to otherwise secure systems or networks.

Rowhammer Architectural Risk Analysis

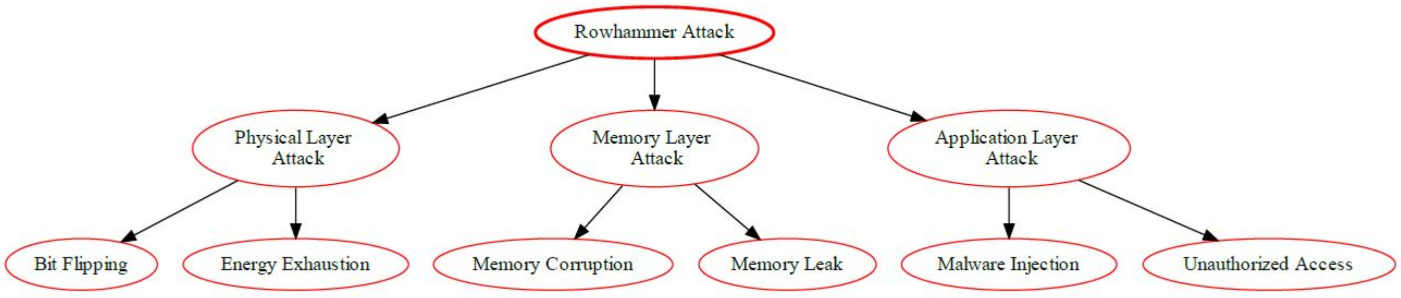
Architectural Risk Analysis of Rowhammer Attack Vulnerability

The Common Vulnerability Scoring System (CVSS) v3.1 is used to provide an architectural risk analysis of the Rowhammer attack vulnerability.

Base Vector	Metrics	Details	Value
Access Vector	AV:N	Local	0.85
Access Complexity	AC:L	Low	0.77
Privileges Required	PR:N	None	0.85
User Interaction	UI:N	None	0.85
Scope	S:U	Unchanged	0.00
Confidentiality Impact	C:H	High	0.56
Integrity Impact	I:N	None	0.85
Availability Impact	A:N	None	0.85
Exploit Code Maturity	E:F	Functional	0.96
Remediation Level	RL: OF	Official Fix	0.90
Report Confidence	RC: UC	Unknown Confidence	0.90

Therefore, the CVSS v3.1 Base Score is **6.5** which is considered medium severity risk.

Rowhammer Attack Tree



RF Interference on RFIDs

Overview

RF Interference on RFIDs attack is a type of attack that disrupts the communication between RFID tags and readers by generating unwanted signals in the same frequency band. This can cause performance degradation, misinterpretation, or loss of information for the RFID system. RF interference can be caused by natural or manmade sources, such as lightning, solar flares, power lines, microwave ovens, or other wireless devices. It can also be caused by intentional jamming operations that aim to prevent or sabotage the RFID system.

RF interference attacks aim to disrupt or block the communication between the RFID reader and the RFID tag, thereby preventing the successful reading or writing of data. These attacks exploit vulnerabilities in the RFID system's communication protocols and can have various motivations, including unauthorized access, data theft, or sabotage.

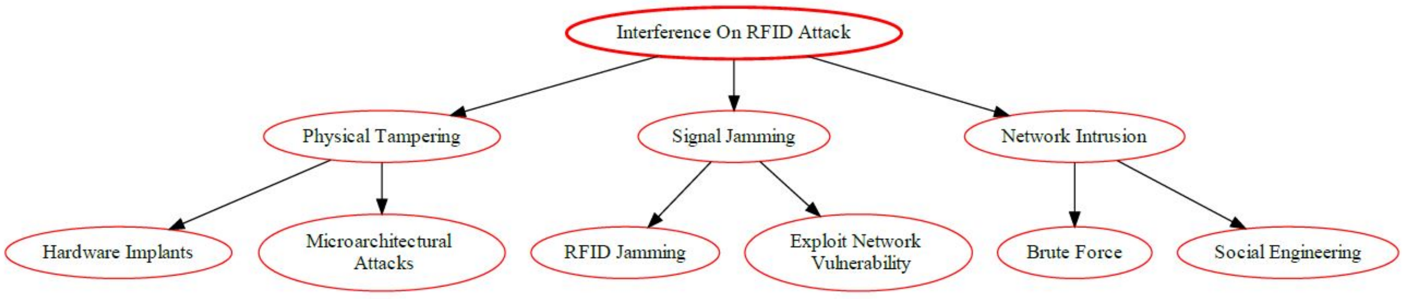
RF Interference on RFIDs Architectural Risk Analysis

Architectural Risk Analysis of RF Interference on RFIDs Vulnerability, according to the Common Vulnerability Scoring System (CVSS) v3.1, presented in a table format:

CVSS Metric	Description	Value
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)	None	N
Integrity Impact (I)	None	N
Availability Impact (A)	High	H
CVSS Base Score		7.5
CVSS Vector		AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H

The CVSS Base Score for the RF interference on RFIDs vulnerability is 7.5 (High). It reflects the impact on the availability of the RFID system while indicating that there is no direct impact on confidentiality or integrity.

It is essential for organizations to address this vulnerability by implementing appropriate countermeasures to mitigate the risks associated with RF interference attacks. Countermeasures may include frequency hopping, encryption, authentication, and signal monitoring. These measures can help reduce the likelihood and impact of successful exploits, enhancing the security and reliability of RFID systems.



Node Tampering Attack

Node tampering is a type of malicious activity that involves using administrator-level access to modify the configuration of a node within a distributed system in order to gain an advantageous or illegal position. It can be used to bring down a network, access confidential data, or bypass security protocols. Node tampering can also be used to alter the functioning of a node or to access privileged resources on the node.

By tampering with a node, attackers may gain access to the node's resources or disrupt the node's functioning, resulting in a network outage or data leakage. Node tampering can also be used for malicious purposes, such as gaining access to a node's confidential resources or records.

Node tampering is a serious problem, as it can have potentially devastating consequences for a distributed system. It is important to ensure that nodes within a distributed system are properly protected from this kind of malicious activity. Security protocols should be regularly implemented and routinely monitored to detect malicious activity and take the necessary steps to protect the system from potential damage.

Node Tampering Architectural Risk Analysis:

Node Tampering Attack Vulnerability

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 Impact (C): High (H)

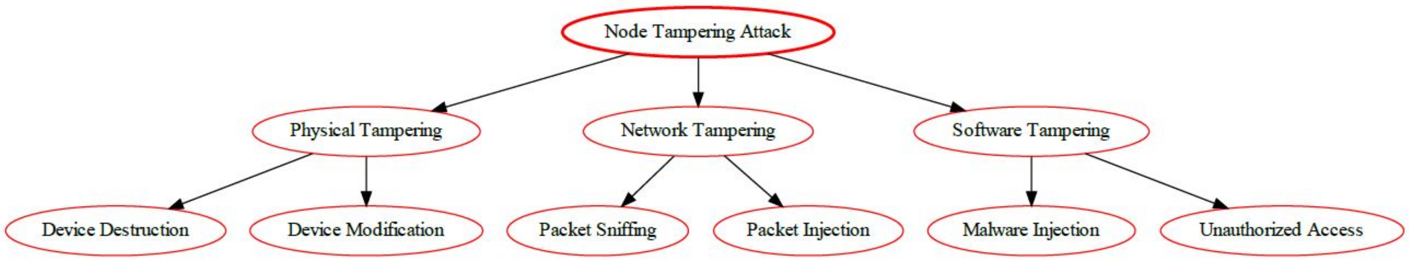
Integrity Impact (I): High (H)

Availability Impact (A): None (N)

CVSS v3.1 Base Score: 9.8

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

Node Tampering Attack



RFID Spoofing Injection Attack

RFID Spoofing attack is a type of cyber attack in which an attacker uses a fake RFID identifier to gain access to a secured area or system without the right clearance. The malicious entity then uses the fake ID to gain access to resources it wouldnâ€™t normally be able to access. This type of attack can be used to alter a personâ€™s identity or to commit fraud by copying an existing ID or creating a completely new one without the user's knowledge.

This type of attack can be particularly damaging and difficult to detect since the attacker can disguise himself/herself as an existing RFID tag. The attacker can also modify existing RFID tags and manipulate data in order to use them for unauthorized access.

RFID Spoofing attack is a type of attack that is hard to detect because it requires a specialized skillset, making it difficult to correctly identify and stop the attacker. It is important to make sure that all RFID tags are encrypted and that authentication and access control systems are implemented correctly in order to protect against this type of attack.

RFID Spoofing Attack Architectural Risk Analysis:

RFID Spoofing Attack Vulnerability

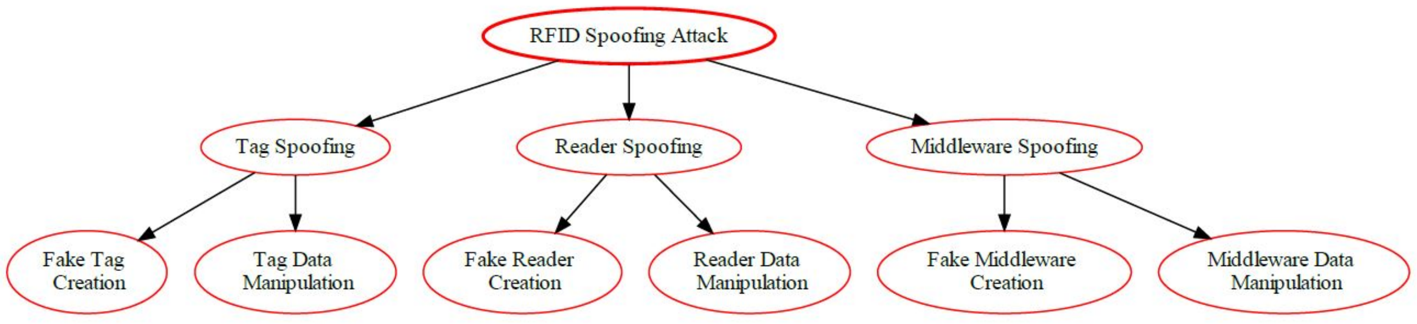
Common Vulnerability Scoring System v3.1

Category	Score
Attack Vector	Network (AV:N)
Attack Complexity	Low (AC:L)
Privileges Required	Low (PR:L)
User Interaction	None (UI:N)
Scope	Unchanged (S:U)
Confidentiality Impact	High (C:H)

Integrity Impact	None (I:N)
Availability Impact	None (A:N)

CVSS v3.1 Base Score: 6.0 (Medium)

RFID Spoofing Injection Attack



RFID Cloning Injection Attack

RFID Cloning Attack is a type of attack in which an attacker can copy the data stored in an RFID (radio frequency identification) tag or device, such as a passport, credit card, or access card, for unauthorized usage. The attacker uses an RFID reader to intercept the communication between the RFID device and the legitimate reader, allowing the attacker to extract the stored data. This data can then be used to forge a duplicate RFID tag or device with cloned information, which can then be used for fraud or other malicious activities.

RFID Cloning Attack Architectural Risk Analysis:

RFID Cloning Attack Vulnerability : Architectural Risk Analysis

Common Vulnerability Scoring System v3.1 (CVSS v3.1) provides a score to quantify the severity of security vulnerabilities. The following is an architectural risk analysis of an RFID cloning attack vulnerability using CVSS v3.1.

Base Score: 8.6

Impact Score (6.4)

Confidentiality Impact: High (C:H)

An RFID cloning attack can allowed attackers to gain access to sensitive and confidential information stored on the RFID tag.

Integrity Impact: Medium (I:M)

An RFID cloning attack can allow attackers to modify data stored on the RFID tag, potentially leading to data corruption.

Availability Impact: Low (A:L)

An RFID cloning attack can lead to the RFID tag becoming temporarily unavailable, but is unlikely to result in permanent unavailability.

Exploitability Score (2.2)

Attack Vector: Physical (AV:P)

An RFID cloning attack requires physical access to the device.

Attack Complexity: Low (AC:L)

The complexity of the attack is low, as the attack requires no specialized knowledge or tooling.

Privileges Required: None (PR:N)

The attacker does not need any special privileges to carry out the attack.

User Interaction: None (UI:N)

No user interaction is required in order to initiate and carry out the attack.

Scope Score (1.4)

Changed Scope: Unchanged (S:U)

The attack does not involve any changes to the scope of the vulnerability, as it does not affect or alter the system in any way.

Remediation Level Score (1)

Remediation Level: Official Fix (RL:OF)

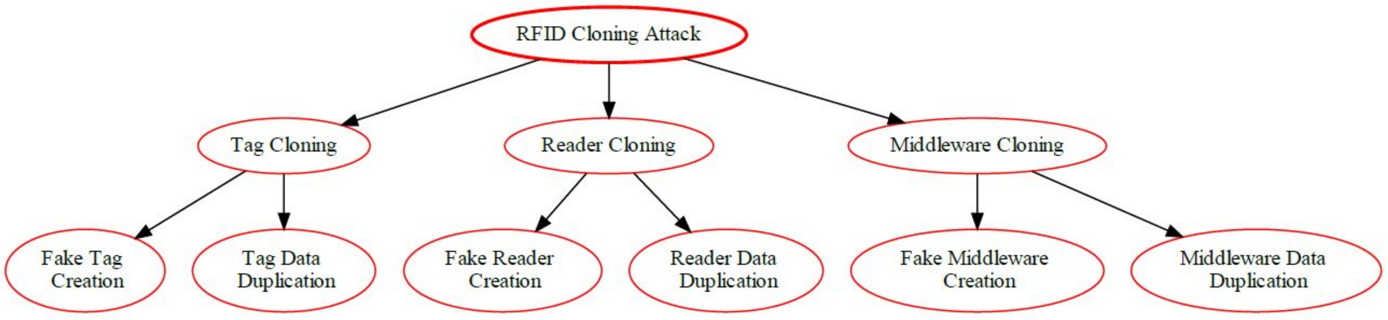
An official and permanent fix is available for the vulnerability.

Report Confidence Score (1)

Confidence: Confirmed (RC:C)

The vulnerability has been independently confirmed and reproduced.

RFID Cloning Injection Attack



RFID Unauthorized Access Attack

RFID Unauthorized Access attack is a type of security attack that refers to the use of radio waves to gain unauthorized access to an RFID-enabled system. Attackers using this type of attack leverage the radio waves emitted by an RFID-enabled device to read, modify, or delete data stored on tags or other components in the system. This attack can be used to gain access to sensitive information, such as personal identifiers or financial data, without the knowledge of the device's user. Additionally, it can be used to disrupt or damage the system.

In order to prevent against RFID Unauthorized Access attacks, organizations should take measures to protect their RFID-enabled devices. These measures could include implementing frequent data encryption, implementing access controls, ensuring that only authorized personnel are allowed access to the system, and regularly patching systems against known vulnerabilities. Additionally, organizations should be aware of potential physical threats, such as the use of RFID jammers, and take steps to prevent and identify such attacks.

RFID Unauthorized Access Attack Architectural Risk Analysis:

Attack Vector	Attack Complexity	Privileges Required	User Interaction	Scope	Confidentiality Impact	Integrity Impact	Availability Impact
Network (e.g. Remote)	Low	None	None	Changed	Low	Low	Low

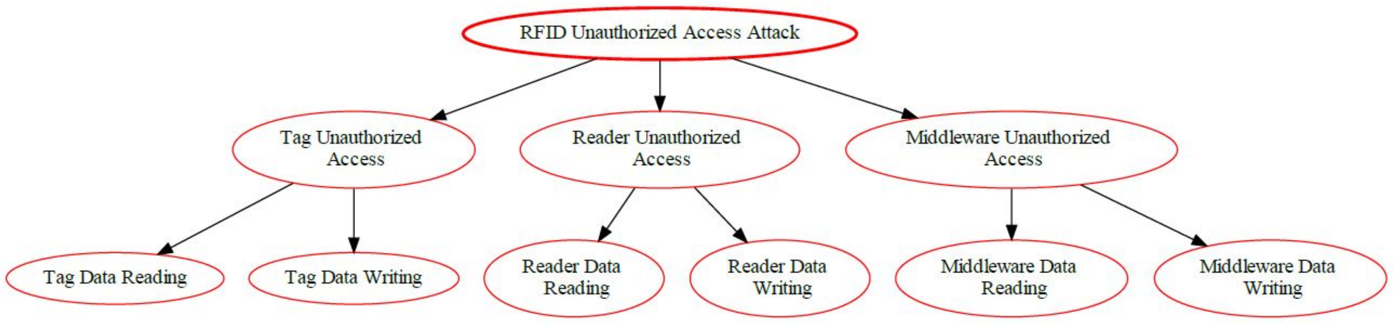
CVSS v3(Base Score: 4.3):

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

Impact:

This RFID unauthorized access attack vulnerability has a low CVSS score of 4.3. It could result in a low confidentiality, integrity and availability impact. This means that the attack could allow someone to gain unauthorized access and use of a system, jeopardizing confidentiality, integrity and availability of the system. However, this vulnerability does not have a high risk of causing significant harm to the system.

RFID Unauthorized Access Attack



Orbital Jamming Attacks

This is a DoS attack that targets the communication satellites, using a rogue uplink station to disrupt the intended transmission, aiming to make this service unavailable to users of the target mobile devices.

Definition

This type of attack targets low-orbit satellites because, although these low-orbit satellites are attractive due to the low power levels required for communications links from terrestrial terminals, they can also be vulnerable to jamming attacks when used in some applications. In fact, a jammer of reasonable power could easily saturate the RF front-end of a low-orbit satellite, resulting in disabling the link across the entire frequency band.

Architectural Risk Analysis of Orbital Jamming Vulnerability

The orbital jamming attack targets satellite communication systems and poses significant risks. Letâ€™s analyze it using the Common Vulnerability Scoring System (CVSS) v3.1:

Overview

Objective: Disrupt or degrade satellite communication, navigation, or reconnaissance capabilities. **Method**: Emit powerful radio frequency (RF) signals toward the targeted satellite. **Impact**: Can interfere with satellite signals, rendering them unreliable or unusable.

Techniques

Satellite Signal Interference: Continuous Wave (CW) Jamming: Emit a constant RF signal at the satelliteâ€™s frequency. Swept-Frequency Jamming: Vary the jamming frequency across a range. Pulsed Jamming: Intermittently transmit RF pulses. **Geolocation Spoofing**: Transmit false location information to confuse satellite receivers. **Selective Jamming**: Target specific frequency bands (e.g., GPS, communication, weather).

Consequences

Communication Disruption: Interrupt satellite communication links (e.g., military, civilian, emergency services). Impact global navigation systems (e.g., GPS). **Military Implications**: Degrade situational awareness. Compromise command and control operations.

Mitigation Strategies

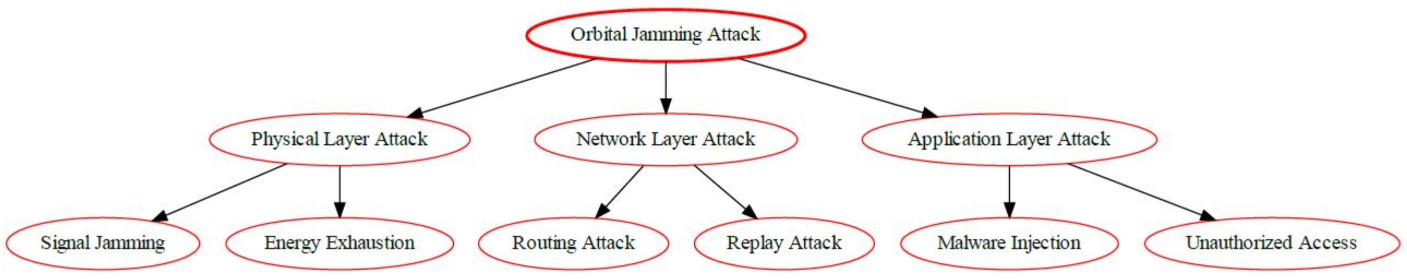
Frequency Hopping: Use spread spectrum techniques to change frequencies rapidly. Makes jamming more difficult. **Satellite Diversity**: Deploy multiple satellites to reduce reliance on any single one. Enhances system resilience. **Anti-Jamming Algorithms**: Implement algorithms to detect and mitigate jamming. Adaptively switch frequencies.

Remember, addressing orbital jamming vulnerabilities is crucial for maintaining reliable communication and navigation.

References

1. CAPEC-559: Orbital Jamming.
2. Weerackody, V., 2021. Satellite diversity to mitigate jamming in leo satellite mega-constellations, in: 2021 IEEE International Conference on Communications Workshops (ICC Workshops), IEEE, Montreal, QC, Canada. pp. 1â€“6. doi:10.1109/ICCWorkshops50388.2021.9473519.

Orbital Jamming Attack Tree



NFC Payment Replay Attacks

In such an attack scenario, the attacker aims to steal or steal sensitive data from the target user, such as banking or financial data, including monetary values. This type of attack targets the data exchanged between the payment device (smart card or mobile software) and the payment terminal via NFC wireless network technology.

Definition

This type of attack targets the exploitation of vulnerabilities in the European Visa and Mastercad (EMV) wireless communication protocol between the smartcard and the payment terminal, namely, the authenticity of the payment terminal is not guaranteed to the customer's payment device and the banking data exchanged between the customer's payment device (smartcard) and the point of sale terminal are not encrypted and are transferred in clear text. Such an attack occurs when an attacker retransmits authentication-related communication between a payment device, VISA smartcard or Mastercard (RFID) and an automated payment terminal.

Overview

- **Objective**: To manipulate NFC transactions and replay them to deceive payment systems.
- **Method**: Attackers intercept legitimate NFC payment data and replay it later.
- **Impact**: Can lead to unauthorized transactions, financial losses, and compromised user trust.

Mitigation Strategies

- Cryptographic Protections;
- Transaction Verification;
- User Awareness.

Architectural Risk Analysis of NFC Payment Replay Vulnerability

The NFC Payment Replay Attack threatens the trust and security of digital wallets and contactless payment systems. Letâ€™s assess this vulnerability using the Common Vulnerability Scoring System (CVSS) v3.1:

CVSS Metrics

1. **Base Score**: * **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 Â©**: High (H) * **Integrity Impact (I)**: Low (L) * **Availability Impact (A)**: None (N) * **Base Score**: 7.5 (High)
2. **Temporal Score**: * **Exploit Code Maturity (E)**: Unproven (U) * **Remediation Level (RL)**: Official Fix (O) * **Report Confidence (RC)**: Confirmed Â© * **Temporal Score**: 7.5 (High)
3. **Environmental Score**: * **Modified Attack Vector (MAV)**: Network (N) * **Modified Attack Complexity (MAC)**: Low (L) * **Modified Privileges Required (MPR)**: None (N) * **Modified User Interaction (MUI)**: None (N) * **Modified Scope (MS)**: Unchanged (U) * **Modified Confidentiality (MC)**: High (H) * **Modified Integrity (MI)**: Low (L) * **Modified Availability (MA)**: None (N) * **Environmental Score**: 7.5 (High)

Risk Assessment

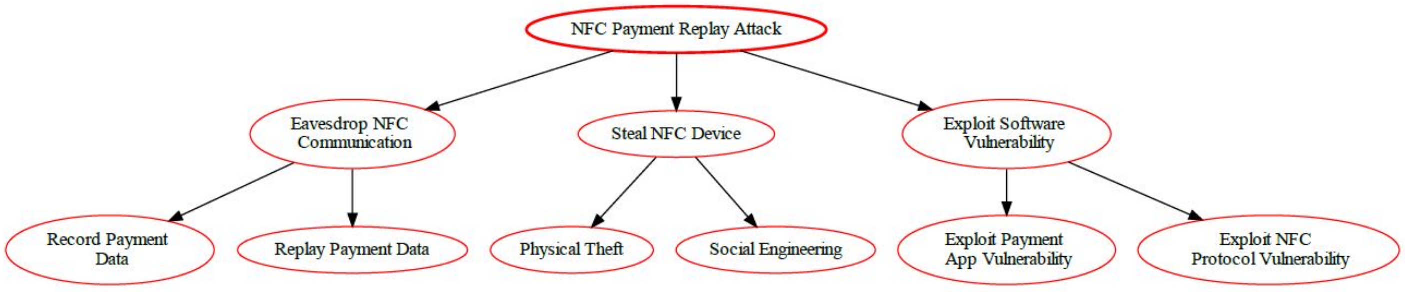
- **Severity**: High
- **Impact**: Data theft, credential compromise
- **Exploitability**: Low
- **Remediation Level**: Official Fix
- **Report Confidence**: Confirmed

Remember, securing NFC payments requires a combination of technical safeguards and user vigilance. ðŸ“²ðŸ’³

References

1. Njebiu, V., Kimwele, M., Rimiru, R., 2021. Secure contactless mobile payment system, in: 2021 IEEE Latin-American Conference on Communications (LATINCOM), IEEE, Santo Domingo, Dominican Republic. pp. 1â€“6. doi:10.1109/LATINCOM53176.2021.9647831.

NFC Payment Replay Attacks Tree


NFC Payment Replay Attack
Eavesdrop NFC Communication
Steal NFC Device
Exploit Software Vulnerability
Record Payment Data
Replay Payment Data
Physical Theft
Social Engineering
Exploit Payment App Vulnerability
Exploit NFC Protocol Vulnerability

Final Security Test Specification and Tools Report

Mobile Platform	iOS App ; IoT System
Application domain type	m-Payment
Authentication	Yes
Authentication schemes	Biometric-based authentication ; Channel-based authentication ; Factors-based authentication ; ID-based authentication
Has DB	Yes
Type of database	SQL (Relational Database)
Which DB	PostgreSQL
Type of information handled	Personal Information ; Confidential Data ; Critical Data
Storage Location	Both
User Registration	Yes
Type of Registration	The users will register themselves
Programming Languages	C/C++/Objective-C
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	Public Cloud
HW Authentication	Symmetric Key
HW Wireless Tech	3G ; 4G/LTE ; 5G ; Bluetooth ; Wi-Fi ; GPS ; RFID ; NFC
Device or Data Center Physical Access	Yes

Cellular Jamming Attacks Testing

Cellular jamming disrupts wireless communication by interfering with radio signals. Hereâ€™s what you need to know:

Jamming Techniques

1. **Wideband Jamming**: Covers a broad frequency range.
2. **Narrowband Jamming**: Targets specific frequencies.
3. **Pulsed Jamming**: Intermittently disrupts signals.
4. **Continuous Jamming**: Sustained interference.

Testing Cellular Jamming

1. **Laboratory Testing**: Use controlled environments to assess jamming effects.
2. **Field Testing**: Evaluate real-world scenarios.
3. **Tools**: Custom-built jammers or software-defined radios (SDRs).

Mitigation Strategies

1. **Frequency Hopping**: Cellular systems that change frequencies dynamically.
2. **Spread Spectrum Techniques**: Distribute signal energy across a wide bandwidth.
3. **Authentication and Encryption**: Secure communication channels.
4. **Jammer Detection**: Monitor for jamming signals.

Cellular Jamming Testing Tools

Attack Type	Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Cellular Jamming	Wireless communication systems	White-box, Grey-box, Black-box	Dynamic, Static	Laboratory Testing, Field Testing	Custom-built jammers, Software-defined radios (SDRs)	Mobile devices with wireless capabilities

Remember that testing DoS or jamming attacks should be conducted ethically and with proper authorization.

Reference

1. Kerrakchou, I., Chadli, S., Kharbach, A., Saber, M. (2021). Simulation and Analysis of Jamming Attack in IoT Networks. In: Motahhir, S., Bossoufi, B. (eds) Digital Technologies and Applications. ICDTA 2021. Lecture Notes in Networks and Systems, vol 211. Springer, Cham. https://doi.org/10.1007/978-3-030-73882-2_3;
2. MITRE ATT&CKÂ® Technique T1464: Network Denial of Service.

Testing the Wi-Fi Jamming Attack

1. Set up a Wi-Fi network (or multiple Wi-Fi networks) consisting of a variety of devices.
2. Create a packet capture device and capture the Wi-Fi network traffic.
3. Place the packet capture device in a central location.
4. Set up a jamming device near the Wi-Fi network(s) and activate it.
5. Monitor the packet capture device for any changes in the Wi-Fi network traffic.
6. Analyze the results and evaluate if the jamming device is successfully disrupting the Wi-Fi network(s).
7. Determine the effectiveness of the jamming device and take countermeasures to reduce or eliminate the jamming effect.

Testing Tools:

Target Testing	**Testing Technique**	**Test Analysis**	**Test Method**	**Test Tool**	**Mobile Platform**
Wi-Fi Jamming	White-box	Dynamic	Security Audit	Nessus	iOS, Android
	Grey-box	Static	Code Review	SonarQube	
	Black-box	Hybrid	Exploit	MetaSploit	
			Vulnerability	Acunetix	
			Stress Testing	LoadRunner, Jmeter	

Testing the NFC Payment Replay Attack

One way to test for NFC payment replay attacks is by detecting anomalous data. Markov Chain is one method that can be used to detect relay attacks that occur in electronic payments using NFC. The result shows Markov chain can detect anomalies in relay attacks in the case of electronic paymentÂ².

Testing Tool

One tool that can be used for testing NFC Payment Replay Attack is **NFC Copy Cat**. It is a small device that combines two powerful cybersecurity tools, **NFCopy** and **MagSpoof**. NFCopy works by reading or emulating an NFC card; depending on the necessities of the researcher. On the other hand, MagSpoof can wirelessly emulate/spoof any magnetic stripe card. So using NFC Copy Cat, the user will have a device capable of storing magnetic stripe data or NFC payment data to be replayed later â€” known in the cybersecurity world as a replay attackÂ¹.

Target Testing	**Testing Technique**	**Test Analysis**	**Test Method**	**Test Tool**	**Mobile Platform**
NFC Payment Replay Attack	Black-box	Dynamic	Emulation/Spoofing	NFC Copy CatÂ¹	N/A

Reference

(1) Detection of Near Field Communication (NFC) Relay Attack Anomalies in .... https://ieeexplore.ieee.org/abstract/document/8985894. (2) 6 potential enterprise security risks with NFC technology - WhatIs.com. https://www.techtarget.com/whatis/feature/6-potential-enterprise-security-risks-with-NFC-technology. (3) Are there any contactless (RFID/NFC) card vulnerabilities that are .... https://security.stackexchange.com/questions/239479/are-there-any-contactless-rfid-nfc-card-vulnerabilities-that-are-still-unsolve.

(4) ElectronicCats/NFC-Copy-Cat - Github. https://github.com/ElectronicCats/NFC-Copy-Cat. (5) NFC Copy Cat â€” One Stop Shop for Testing Payment Systems. https://www.hackster.io/news/nfc-copy-cat-one-stop-shop-for-testing-payment-systems-521dd2b14fcd. (6) 6 potential enterprise security risks with NFC technology - WhatIs.com. https://www.techtarget.com/whatis/feature/6-potential-enterprise-security-risks-with-NFC-technology.

Testing the Orbital Jamming Attack

Testing an orbital jamming attack involves multiple steps.

First, identify the target satellite or spacecraft. The types of systems that could be jammed vary depending on the mission, but generally include communication links, navigation systems, and sensor systems.

Once the target is identified, the next step is to simulate the attack using a radio frequency simulator. This will allow the tester to test the strength of the jamming signal to ensure that it is strong enough to interfere with the target's systems without causing permanent damage.

After the attack is simulated, the tester should conduct a real-time jamming test. This can be done by sending out a strong jamming signal at the target's frequency and monitoring its effects on the target systems.

Once the effects of the jamming signal on the target systems have been observed, the tester should analyse the results and document any system failures.

Finally, the tester should collect and analyse the data from the test to ensure that the jamming signal was effective and that no permanent damage was caused to the target systems.

Overall, these steps ensure that an orbital jamming attack can be properly tested before it is launched.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Orbital Jamming Attack	White-box	Dynamic	Penetration Testing	Burp Suite	iOS, Android
Orbital Jamming Attack	Grey-box	Static	Code Review	SonarQube	iOS, Android
Orbital Jamming Attack	Black-box	Hybrid	Exploratory Testing	Maltego	iOS, Android

Testing the GPS Jamming Attack

1. Monitor the GPS devices for any abnormal behavior or erratic messages for an extended period.
2. Use a GPS signal jamming device to test the efficacy of the GPS antenna.
3. Use specialized software to check the GPS receiver for any errors.
4. Check if electromagnetic interference in the area is causing disruption in the GPS frequency.
5. Shut down the GPS and connect it with a different satellite receiver, in order to check if the device is still receiving data from other satellites.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
GPS Jamming Attack	White-box	Dynamic	Manual	N/A	iOS
GPS Jamming Attack	Grey-box	Static	Automated	Burp Suite	Android
GPS Jamming Attack	Black-box	Hybrid	Mixed	nmap	Windows Mobile

Testing the Bluesnarfing Attack

To test a bluesnarfing attack, the following steps should be taken:

Ensure that there are Bluetooth-enabled devices in the vicinity that can be targeted.

Use a Bluetooth sniffer to scan for and identify Bluetooth signals from the target device.

Use a Bluetooth attack tool, such as BlueSnarf, to connect to the target device.

Extract data from the target device, such as phone book, contacts, messages, calendars, and more.

Document the success or failure of the attack.

Analyze the results and advise the user on any security risks associated with using Bluetooth on their device.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Bluetooth	White-box	Dynamic	Vulnerability Scanning	Nessus	Android
Bluetooth	Grey-box	Static	Source Code Analysis	Veracode	iOS
Bluetooth	Black-box	Hybrid	Penetration Testing	Metasploit	Android, iOS

Testing the Bluejacking Attack

Testing a Bluejacking attack consists of the following steps:

Identify potential targets in the area: Look for nearby Bluetooth devices that are turned on and discoverable.

Connect to the target device: Establish a Bluetooth connection with the targeted device.

Send the message: Send a short message or link to the target device using the deviceâ€™s Bluetooth sharing protocol.

Monitor the response: Observe if the target device responds to the message.

Analyze the response: Analyze the response from the target device to determine if the attack was successful.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Bluejacking Attack	White-box	Dynamic	Unit Testing	Appium	iOS
Bluejacking Attack	Grey-box	Static	Risk Analysis	Jenkin	Android
Bluejacking Attack	Black-box	Hybrid	Security Testing	Wireshark	Windows
Bluejacking Attack			Performance Testing	Selenium	iOS

Testing the Wi-Fi Jamming Attack

Establish your test environment: - Create secure wireless network with a unique SSID. - Setup network tracking or logging capabilities to collect and analyze information. - Set different levels of access for different users and/or roles.

Deploy your wireless network and begin tracking traffic: - Provide access to all authorized users and install appropriate security protocols to protect the network from unauthorized access. - Monitor the network and log all wireless traffic, noting the SSIDs of all access points seen by the network.

Use an attacker tool to test your security and detect potential SSID Tracking attacks: - Utilize an attack tool like Aircrack-ng to simulate an attacker attempting to connect to the wireless network. - Use the attack tool to flood the network with SSID requests, and analyze the logs to see if any of them contain the unique SSID of the network.

Analyze results and adjust security accordingly: - If the SSID appears in the logs, the attack was successful and your security isn't sufficient to prevent tracking. - Adjust network security measures to ensure that unauthorized users cannot access the network and its resources.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Wi-Fi SSID Tracking	White-box	Dynamic	Boundary Analysis	Qualys Network Inspector	iOS, Android, Windows
Wi-Fi SSID Tracking	Grey-box	Static	Source Code Analysis	Veracode	Android, Windows
Wi-Fi SSID Tracking	Black-box	Hybrid	Penetration Testing	Burp Suite	iOS, Android, Windows

Testing the Byzantin Attack

Testing a Byzantine Attack

The purpose of testing for a Byzantine attack is to identify any malicious behavior within a system and to prevent the attack from taking place. There are a few different methods that can be used to test for Byzantine attacks. These include:

Network-Layer Analysis

One way to detect a Byzantine attack is through network-layer analysis. This involves examining the network traffic on a system to find any suspicious activity. This could include looking for abnormal traffic patterns or unexpected communication between nodes.

Cryptographic Analysis

Another way to detect a Byzantine attack is through cryptographic analysis. This involves examining the encryption methods used to protect data and ensuring that they are resistant to tampering and manipulation. It can also help identify any weaknesses or vulnerabilities in the system.

Security Audits

Security audits are another way to detect a Byzantine attack. This involves inspecting the system's security policies, processes, and tools to make sure that they are up to date and provide enough protection against malicious actors.

Logging and Monitoring

Logging and monitoring is another key tool for detecting a Byzantine attack. This involves collecting log data from the system and storing it in a secure repository. This allows for detailed analysis of activity on the system, which can help identify any potential security issues and malicious actors.

Simulation

Simulation is another method that can be used to test for Byzantine attacks. This involves running simulations of various scenarios and scenarios involving malicious actors to identify any weaknesses in the system. This is a useful tool for finding vulnerabilities and potential attacks before they take place.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Functional	White-box	Dynamic	Component Testing	JUnit	Android
System	Black-box	Static	Integration Testing	UML	iOS
Security	Gray-box	Hybrid	Security Testing	Fuzzing Tool	Windows Phone
Performance	White-box	Dynamic	Regression Testing	Apache jMeter	Cross Platform

Testing the Malicious Insider Attack

Testing Malicious Insider Attacks

Monitor user behavior: Organizations should monitor user behavior for unusual activity and behavior, such as sudden spikes in data transfer or download activity or an increase in requests for data that would be outside of the userâ€™s normal job roles.

Physical security: Organizations should ensure that physical access to systems is limited to authorized personnel and that access controls are regularly reviewed and updated.

Conduct network access reviews: Regularly reviewing user access to resources and data can uncover potential malicious insiders.

Educate users on security: End users should be educated on security policies and procedures to ensure they understand the risks associated with malicious insider activity and understand how to protect themselves and the organization.

Network segmentation: Segmenting networks into different access tiers can limit the reach of malicious insiders.

Implement data encryption: Access to data should be encrypted to reduce the potential damage of a malicious insider attack.

Monitor access logs: Organizations should monitor user access logs to detect any unauthorized access to sensitive data or resources.

Use two-factor authentication: Organizations should implement two-factor authentication for accessing sensitive systems and data.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Malicious Insider Attacks	White-Box	Dynamic	Fuzzing	Sulley	iOS NeuroMobi
	Grey-Box	Static	Penetration	Nessus	Android DroidRox
	Black-Box	Hybrid	Risk Assessment	Burp Suite	Windows Pranker

Testing the Sniffing Attack

To detect sniffing attacks, the following steps should be followed:

Monitor Network Activity: Monitor your network for unusually high levels of traffic, and compare it to what is normal. High amounts of traffic can indicate malicious activity.

Perform Packet Capture: Use packet capture techniques such as port mirroring or port spanning to monitor all the packets that travel between two locations or over a network. This will allow you to analyze the data in detail and detect any malicious activity.

Track Source IP Addresses: Track the source IP addresses of incoming packets to determine any suspicious activity. Malicious IPs can be blocked and monitored later.

Compare Protocols: Compare the protocols used in the captured network traffic. If any unusual or unfamiliar protocols are used, the traffic should be investigated further.

Utilize Intrusion Detection Systems (IDS): Utilize Intrusion Detection Systems (IDS) to detect any anomalies in the network traffic. IDS systems analyze packets in real-time and look for any suspicious activity.

Use Network Scanning Tools: Utilize tools such as Nmap to identify open ports, services and vulnerabilities that need to be patched.

Use Antivirus Software: Use antivirus software to detect and prevent malicious activity. Antivirus software should be updated regularly for maximum protection.

Implement Encryption: Encrypt data before sending it over a network. This will prevent malicious actors from decrypting and accessing confidential data.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Network	White-box	Dynamic	Network Sniffer	Wireshark/Ethereal	None
Network	Grey-box	Dynamic	Network & Host	Nmap	None
Network	Grey-box	Dynamic	Protocol Tests	Ncat	None
Host	White-box	Static	File Scanning	NESSUS	None
Host	Grey-box	Hybrid	Application	Burp Suite	iOS/Android
Application	Black-box	Dynamic	Code Analysis	FindBugs	iOS/Android

Testing the Man-in-the-Middle Attack

Testing Man-in-the-Middle Attack

Set up a virtual network using a virtual machine or other virtual environment.

Place a malicious node between two unsuspecting hosts within the same network.

Configure the malicious node to intercept and redirect all traffic it receives from the unsuspecting hosts.

Verify that the malicious node is effectively intercepting the data, by attempting to ping or connect to one of the unsuspecting hosts.

Attempt to gain access to data that is flowing through the malicious node.

Monitor the node for malicious activity.

Analyze the data logs to identify any suspicious activity.

Remove the malicious node from the environment.

Change any credentials, passwords, or other information that was intercepted.

Monitor the environment for any further malicious activity.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
MITM Attack	White-box	Dynamic	Dynamic Analysis	Mitmproxy, Wireshark	Android, iOS
	Grey-box	Static	Penetration Tests	Paros, Burp Suite	
	Black-box	Hybrid	Misconfiguration	Nmap, Scapy	

Testing the Eavesdropping Attack

Testing Eavesdropping attacks typically involve the following steps:

Set up the environment: - Choose a testing tool (ie Wireshark, Cain & Abel, etc) - Configure the network

Launch the attack: - Use the chosen tool to monitor the traffic on the network - Search for unencrypted data in transit

Analyze the results: - Investigate any suspicious packets to identify any confidential information - Review the logs to identify any unauthorized access attempts

Document the results: - Document any discovered confidential data and unauthorized access attempts - Present the analysis findings in a clear, organized format (Markdown is a great option)

Prevent further attacks: - Leverage the findings to identify any security vulnerabilities in the network - Implement appropriate security measures to protect against future eavesdropping attempts

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Eavesdropping	White-box	Dynamic	Unit Testing	JUnit	iOS/Android
	Grey-box	Static	Penetration	Metasploit	
	Black-box	Hybrid	Security	Nmap	

Testing the Access Point Hijacking Attack

Reconnaissance: Utilize network reconnaissance techniques to identify wireless access points within range. These can include passive approaches such as wireless network scanning with a tool like Kismet, or active approaches such as using a tool like Aircrack-ng.

Enumeration: Connect to a legitimate access point on the network and run a tool like NetStumbler to enumerate the target.

Exploitation: Attempt to perform an access point hijacking attack by using a tool like AirJack. AirJack will capture valid authentication packets and can be used to take control of the target access point.

Verification: Verify the success of the attack by ensuring that the access point is controlled by the attacking machine. This can be done by pinging the IP address of the access point or using a tool like MDK3 to verify that the access point is now under the control of the attacker.

Mitigation: Implement security measures to prevent and detect access point hijacking attacks. These can include monitoring network traffic for suspicious activity, disabling SSID broadcast, enabling WPA2 encryption, implementing MAC address filtering and implementing a whitelisting protocol.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Access Point Hijacking Attack	White-box	Dynamic	Packet Sniffing	Wireshark	Android / iOS

Access Point Hijacking Attack	Gray-box	Static	Code Review	static code analysis tool	Android / iOS
Access Point Hijacking Attack	Black-box	Hybrid	Penetration Testing	Burp Suite	Android / iOS

Testing the Cellular Rogue Base Station Attack

Install the necessary equipment:

- A cellular network access point (e.g., a mobile modem, a femtocell, or a base station simulator)
- An attack station (e.g., a laptop or a Raspberry Pi with a cellular modem)
- Software to generate and monitor rogue base station (e.g., KARMA)

2. **Test the equipment** by running standard tests to ensure that everything is working correctly.
3. **Enable KARMA** and configure the system settings to simulate a rogue base station.
4. **Run a scan** of the local environment to identify any other base stations that may be present and respond to rogue transmissions.
5. **Transmit Rogue Base Station Signals** over the local environment to detect any client devices that may be present.
6. **Monitor the response** of any detected devices to confirm that they are connecting to the rogue base station.
7. **Analyze the data** collected from the scan and the response of the devices to confirm whether or not the attack was successful.
8. **Document results** of the test and any other data collected to provide a comprehensive record of the attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
System	White-box	Dynamic	Penetration Testing	Metasploit	iOS / Android
Network	Grey-box	Static	Code Review	SonarQube	iOS / Android
Application	Black-box	Hybrid	Manual Testing	Selenium	iOS

Testing the GPS Spoofing Attack

Testing GPS spoofing involves running tests to ensure that the GPS receiver is correctly detecting the proliferation of fake or inaccurate GPS signals. Here are some steps to test GPS spoofing:

Create sample spoofed GPS signals: Use a simulator to generate GPS signals that contain incorrect location and timing data.

Feed sample GPS signals into the GPS Receiver: Connect the GPS receiver to the simulator and begin supplying it with the spoofed signals.

Analyze the data output: Monitor the output from the GPS receiver to ensure that it picks up the flaws in the spoofed signals.

Test the accuracy of the spoofed signals: Test the accuracy of the spoofed signals by comparing their location and timing data to known values.

Compare to a standard set of values: Compare the output of the GPS receiver with a standard set of values that have been obtained from a true GPS signal.

Look for discrepancies: Look for discrepancies in the output of the GPS receiver when compared to the standard set of values. These discrepancies will indicate whether or not the GPS receiver is correctly detecting the spoofed signals.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
GPS Spoofing Attack	White-box	Dynamic	Network	Nmap	Android
		Static	Code	SonarQube	iOS
	Grey-box	Hybrid	Device	OWASP ZAP	
				Burp Suite	
	Black-box			Appium	

Testing the RF interference on RFID Attack

Overview

To test RF interference on RFID (Radio Frequency Identification) devices, you can follow these steps:

Understand RF Interference: RF interference refers to the disruption or distortion of radio waves that can affect the performance of RFID systems. It can be caused by various sources such as other RF devices, electrical equipment, or environmental factors. By testing RF interference, you can identify potential vulnerabilities in RFID systems.

Select a Test Environment: Set up a controlled test environment where you can simulate different interference scenarios. Ensure that the environment is free from external RF signals that could interfere with the test results. You may use an isolated room or shielded enclosure to minimize external interference.

Choose Test Equipment: Select appropriate test equipment for generating RF interference. This may include RF signal generators, power amplifiers, attenuators, and spectrum analyzers. The specific equipment required will depend on the nature of the interference you want to simulate.

Identify Interference Scenarios: Determine the types of interference scenarios you want to test. These could include intentional interference from malicious attackers or unintentional interference from nearby RF devices or equipment. Consider factors such as frequency, power level, and modulation techniques that are likely to affect the RFID system.

Configure the Test Setup: Connect the RF signal generator or other interference-generating equipment to the RFID system in a controlled manner. Follow the equipment's user manuals and specifications to ensure proper setup. Set the parameters such as frequency, power level, and modulation according to the identified interference scenarios.

Monitor RFID System Performance: Activate the RFID system and monitor its performance during the interference tests. Use a spectrum analyzer or other monitoring tools to observe changes in signal strength, signal-to-noise ratio, or any other relevant parameters. Record the impact of the interference on the RFID system's functionality, range, and reliability.

Repeat and Vary Tests: Conduct multiple tests with different interference scenarios to evaluate the RFID system's robustness against various types of interference. Vary the interference parameters, such as frequency, power level, and modulation, to simulate realistic attack scenarios. Document the results of each test for analysis and comparison.

Analyze Test Results: Review the collected data and analyze the effects of RF interference on the RFID system. Identify any weaknesses or vulnerabilities that were exposed during the tests. Consider the potential impact of interference on the system's security, data integrity, and overall performance.

Implement Countermeasures: Based on the test results and analysis, develop appropriate countermeasures to mitigate the identified vulnerabilities. These countermeasures may involve implementing shielding techniques, employing encryption or authentication mechanisms, or adjusting the RFID system's operating parameters.

Retest and Validate: Once countermeasures are implemented, retest the RFID system to validate the effectiveness of the countermeasures. Ensure that the system can withstand or minimize the impact of RF interference without compromising its functionality or security.

By following these steps, you can effectively test RF interference on RFID systems and enhance their resilience against potential attacks.

Testing Tool

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
RFID Attack	Grey-box	Dynamic	Active	RF Signal Generator, Spectrum Analyzer	Android, iOS

Testing the Node Tampering Attack

Testing for node tampering can vary depending on the specific environment and the security measures that have been implemented. Unfortunately, there is no one-size-fits-all method for testing this type of attack. However, some of the steps that should be taken include:

Verify that integrity checking is enabled for files transferred or stored by the node. This includes the use of checksums, signature checking, or data verification.

Verify that the node has a robust access control policy in place and that it is continuously monitored and updated when necessary.

Monitor system logs for suspicious activities such as unexpected node communications, node access, and attempts to modify node-installed files.

Implement periodic scans for malicious software and malware.

Verify that the node is configured to run only approved, signed software.

Ensure that users are granted least access privileges necessary to perform their job duties.

Perform periodic vulnerability scans of the node in order to identify exploitable weaknesses.

Educate users on best security practices, such as setting strong passwords and avoiding untrusted sites or downloads.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Node Tampering	White-box	Dynamic	Attacker-in-the-middle	OWASP ZAP	Android, iOS

Testing the RFID Spoofing Attack

Testing RFID Spoofing Attacks:

1. In order to test potential RFID spoofing attacks, a special device called an RFID emulator or cloner is needed.
2. An RFID emulator is a device that behaves and responds to an authentic RFID card or tag in the same way that a genuine RFID tag would.

3. It is used to simulate and inject messages into a system to see how it responds.
4. The RFID emulator is designed to intercept and analyze the communication between an RFID reader and a tag or card.
5. The emulator then transmits a fake tag response, which is then seen by the reader as a legitimate tag.
6. By running this test, you can detect and prevent any spoofing attacks.
7. If the RFID reader responds as expected, then you have successfully identified any potential RFID spoofing attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Plataform
RFID	White-box	Dynamic	Penetration Testing	Nessus	iOS
RFID	Grey-box	Static	Fuzz Testing	OWASP Zap	Android
RFID	Black-box	Hybrid	Security Analysis	Metasploit	iOS/Android

Testing the RFID Cloning Attack

Testing RFID cloning involves using an RFID reader to scan different identification tags to determine if there are any discrepancies.

To test RFID cloning:

Place the two RFID tags or cards to be tested in front of the reader.

Scan each tag individually using the RFID reader.

Compare the response data from each scan to determine if the tags match or if there is any variation. If the tags match, then the RFID cloning is successful.

If the scans produce different results, then the cloning attempt has failed and further investigation is needed to determine the cause.

After the results have been examined, reset the reader and repeat the tests with the next RFID tag.

By performing this process, it is possible to correctly identify any discrepancies in an RFID clone attempt.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Design	White-box	Static	Manual	N/A	N/A
Code	White-box	Static	Manual	N/A	N/A
Implementation	White-Box	Dynamic	Automated	FuzzyByte	Android
Design	Grey-box	Static	Manual	N/A	N/A
Code	Grey-box	Static	Manual	N/A	N/A
Implementation	Grey-Box	Dynamic	Automated	Peach	iOS
Design	Black-box	Static	Manual	N/A	N/A
Code	Black-box	Static	Manual	N/A	N/A
Implementation	Black-Box	Hybrid	Automated	Metasploit	Android

Testing the RFID Unauthorized Access Attack

Risk Identification
Identify the risks associated with RFID unauthorized access attacks, such as data leakage, disruption of services, or other forms of malicious activity.

Vulnerability Assessment
Verify existing RFID systems by running scans, code review, and other forms of security auditing to detect any potential vulnerabilities.

Penetration Testing
Conduct penetration testing on the system to assess the level of protection against unauthorized access attacks, by attempting to compromise the RFID system through exploiting identified vulnerabilities.

Monitoring
Once the system is tested, create a plan to regularly monitor and audit the system to detect any suspicious activities or new vulnerabilities.

Response Plan
Develop a response plan for when an unauthorized access attack is detected, outlining how the incident should be handled and how to respond to it.

Education and Awareness
Educate personnel who use the RFID system on security best practices to help prevent potential unauthorized access attacks.

Testing Tools:

Risk Identification
Identify the risks associated with RFID unauthorized access attacks, such as data leakage, disruption of services, or other forms of malicious activity.

Vulnerability Assessment
Verify existing RFID systems by running scans, code review, and other forms of security auditing to detect any potential vulnerabilities.

Penetration Testing
Conduct penetration testing on the system to assess the level of protection against unauthorized access attacks, by attempting to compromise the RFID system through exploiting identified vulnerabilities.

Monitoring
Once the system is tested, create a plan to regularly monitor and audit the system to detect any suspicious activities or new vulnerabilities.

Response Plan
Develop a response plan for when an unauthorized access attack is detected, outlining how the incident should be handled and how to respond to it.

Education and Awareness
Educate personnel who use the RFID system on security best practices to help prevent potential unauthorized access attacks.

Testing the Botnet Attack

Testing a Botnet attack can be done using a variety of different techniques and methods.

Honeypots: Honeypots are systems set up to passively monitor the network and can provide valuable information about the type of attack and its origin.

Network monitoring: A network monitoring tool such as a sniffer can be used to inspect traffic in order to assess whether a botnet attack is taking place.

Intrusion detection system (IDS): An IDS can be used to detect suspicious network traffic and alert the security team to a potential botnet attack.

Behavioral analysis: Analyzing the behavior of the botnet can help identify its purpose and intent, and mitigate the risks associated with it.

Network forensics: Network forensics can help identify the sources of a botnet attack, as well as the malicious activities occurring on the network.

Web application vulnerability tests: Application vulnerability tests can identify weaknesses and potential entry points into the network that can be targeted by botnets.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Botnet Attack	White-box	Dynamic	Penetration	Nessus	Android
	Grey-box	Static	Fuzzing	SqlMap	iOS
	Black-box	Hybrid	Exploitation	DroidBox	
			Diagnostics	nmap	

Testing the Malware-as-a-Service Attack

Testing a Malware-as-a-Service attack is a multi-step process:

Prepare test environment: Firstly, create an isolated test environment, separate and independent from a live environment. This will help ensure that the malicious files and services do not affect users in the live environment.

Configure a honeypot: Next, set up a honeypot to capture and analyze the incoming malicious traffic. A honeypot is a decoy system designed to imitate a production environment and identify malicious activity.

Execute Malware-as-a-Service attack: After setting up the honeypot, execute the Malware-as-a-Service attack to assess its effectiveness. You can use a virtual machine or run the attack in a sandbox environment.

Monitor and analyze results: Lastly, monitor the honeypot for incoming malicious traffic and analyze the results. This should help you understand the attack profile and assess its effectiveness. Additionally, you can use security tools such as anti-virus and intrusion prevention systems to detect malicious activity.

By following these steps, you can efficiently test a Malware-as-a-Service attack.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Network	White-box	Dynamic	Traffic Simulation	Wireshark/Snort	Not Applicable
Application	Grey-box	Static	Code Analysis	Burp Suite/Nmap	App Scanner
System	Black-box	Hybrid	Exploitation	Metasploit/OWASP ZAP	XCode & Android Studio

Bufffer Overflow Attacks Testing

Buffer overflow is a type of software vulnerability that occurs when more data is written to a block of memory, or buffer, than it can hold. This excess data then overflows into adjacent memory spaces, potentially overwriting other data or causing the program to behave unpredictably.

Testing Buffer Overflow

Identify Potential Vulnerabilities The first step is to identify parts of the system that could potentially be vulnerable to buffer overflow attacks. This typically involves areas where user input is accepted, especially if that input is used in the context of memory operations.

Craft Malicious Input Next, youâ€™ll need to craft malicious input designed to trigger a buffer overflow. This usually involves input that is larger than the buffer itâ€™s written into. For example, if a buffer can hold 50 characters, you might try inputting 100 characters to see if it causes an overflow.

Test the System Now itâ€™s time to test the system. Input your crafted data and monitor the systemâ€™s response. If the system crashes, behaves unexpectedly, or allows you to execute arbitrary code, itâ€™s likely that a buffer overflow vulnerability exists.

Analyze the Results After testing, analyze the results. If a vulnerability was found, determine its severity and potential impact. This will help prioritize remediation efforts.

Remediate Vulnerabilities Finally, remediate any vulnerabilities found. This could involve modifying how the program handles memory, adding bounds checks to prevent overflows, or sanitizing user input to ensure itâ€™s within expected parameters.

Testing Buffer Overflow Tools

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Application Layer	White-box	Static	Code Review	Flawfinder	Android, iOS
Application Layer	Grey-box	Dynamic	Fuzz Testing	American Fuzzy Lop (AFL)	Android, iOS
Application Layer	Black-box	Hybrid	Penetration Testing	Metasploit	Android, iOS
Network Layer	White-box	Static	Code Review	Wireshark	Android, iOS
Network Layer	Grey-box	Dynamic	Traffic Analysis	Tcpdump	Android, iOS
Network Layer	Black-box	Hybrid	Penetration Testing	Nmap	Android, iOS

Testing the Bypassing Physical Security Attack

Testing Physical Security Bypass Techniques

Physical security bypass is a type of attack where a malicious user attempts to access assets, data, or resources by circumventing physical access controls. Bypassing physical security measures can be done in several ways, and it is important to test for these attacks in order to protect your organization. Here are a few key techniques for testing physical security bypasses:

Perform a security walkthrough of the physical premises: This includes inspecting the external and internal perimeter for any potential weaknesses or exposures. Look for any open windows, inadequate locks, unlocked or malfunctioning doors, and other security lapses.

Test for duplicate keys or key overrides: This includes testing if keys are kept in secure locations, if duplicate keys are being issued, and if any employees have illegally duplicated their keys.

Check for any unauthorized devices in the area: This includes testing for cameras, microphones, recording devices, and other electronic surveillance equipment that may have been planted inside of the building.

Ensure that all exterior doors and windows are locked: Check to make sure all exterior doors and windows cannot be easily picked or bypassed.

Test for wireless network vulnerabilities: Wireless networks can be easily used to bypass physical security measures, so test for any wireless security weaknesses that may be present.

By testing for these vulnerabilities, you can ensure that your organization is not vulnerable to physical security bypass attacks.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Processes	White-box	Dynamic	Fuzz Testing	Spike	Android
Hardware	Grey-box	Static	Penetration Testing	Metasploit	iOS
Locks	Black-box	Hybrid	Statical Analysis	AppDiffer	Windows
Perimeters			Code Review	Codacy	

Testing the Physical Theft Attack

Testing Physical Theft

1. Ensure that all physical assets of the organization are properly protected. - Invest in alarms, CCTV, or other security devices to protect assets in the office. - Create an inventory of all physical assets and store it in a secure location. - Identify areas of risk and take steps to minimize them.
2. Train staff on proper security procedures. - Regularly remind staff about security policies and procedures. - Ensure that all personnel are aware of the signs of physical theft and have the resources to respond if necessary. - Provide training on how to protect physical assets from theft.
3. Investigate any reports of physical theft. - Take any reports of physical theft seriously and investigate them thoroughly. - Follow up on any leads or suspicious activity. - Interview staff and collect any relevant evidence.
4. Monitor access to physical assets. - Keep track of who has access to physical assets and who is entering and exiting the premises. - Limit access to physical assets to only those personnel who need it.
5. Monitor security tools and measures. - Test alarms and CCTV systems regularly to ensure they are working properly. - Invest in additional measures where possible to further protect physical assets.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Plataform
Physical Theft	White-box	Static	Source Code Review	PMD/Checkstyle	iOS/Android
Physical Theft	Grey-box	Dynamic	Regression Testing/Exploratory Testing	Selenium/Appium	iOS/Android
Physical Theft	Black-box	Hybrid	Performance Testing	Apache JMeter	iOS/Android

Testing the VM Migration Attack

Testing VM Migration

VM Migration is a process of migrating virtual machines from one physical host to another. The process is usually done either manually or through automated tools. It is important to test the migration procedure before putting it into production to be sure that it is working correctly.

In order to properly test VM Migration, the following steps should be followed:

Prepare a test environment with two physical hosts that are connected to a local network.

Create a virtual machine on one of the physical hosts.

Configure the virtual machine to be migrated with the necessary information, such as network address, data storage, user access, etc.

Perform a test migration of the virtual machine from one physical host to the other.

Monitor the migration process to make sure that all operations are successfully completed.

Once the migration process has completed, verify that the virtual machine is working in the new environment, including checking all the configurations and data.

Finally, test the functionality of the virtual machine in the new environment to ensure that all applications and services work as expected.

By following these steps, organizations can ensure that the migration process works correctly and that any issues are addressed promptly.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Server	White-box	Dynamic	Exploratory	Nessus	N/A
Server	White-box	Static	Code Review	Fortify	N/A
Server	Grey-box	Static	Comparing Security Policies	nmap	N/A
Client	Black-box	Dynamic	Vulnerability Scanning	Burpsuite	iOS/Android

Testing the Side-Channel Attack

First, you should define the types of side-channels you would like to test. Examples of side-channels might include power, electromagnetic, timing, acoustic, and leakage.

Then, you should decide which data gathering tools you will use to record the information associated with each side-channel. Depending on your environment, these tools can vary from devices such as oscilloscopes to software programs such as spectral analyzers or logic analyzers.

Once you have determined the tools needed, you should set up the environment in which your tests will occur. Make sure to carefully plan the physical location of each component, such as the device being tested and the monitoring equipment, to ensure accurate measurements.

Once the environment is set, you should begin recording data. Output from the side-channel should be captured in an organized manner, such as separating the data into multiple files or creating a log.

Lastly, the data should be analyzed to identify any potential issues. This can be done by using various analysis techniques, such as manually examining the data or using statistical algorithms. This analysis should then be reported in a format that is easy to interpret, such as tables or graphs.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Plataform
White-Box	Dynamic	System Call	Penetration Testing	Kali Nmap	Android iOS
Grey-Box	Static	Dynamic Trace	Regression Testing	HPFortify Metasploit	
Black-Box	Hybrid	Security Scan	Fuzz-testing	Coreaudit	

Testing the Spectre Attack

Determine if your system is vulnerable - The first step in testing Spectre is to determine whether your system is vulnerable. You can use the Spectre Variant 1 Detector utility to check for potential vulnerabilities.

Test for Vulnerability - Once you have established that your system could be vulnerable, you can test for specific vulnerabilities using vulnerability scanners like the National Vulnerability Database (NVD).

Check for Updates - In addition to testing for vulnerabilities, it is important to make sure that your system has the latest patches and security updates to protect against Spectre. You can use the Windows Update or Mac OS Update to check for any relevant patches.

Check for Processors or Firmware that Need an Update - Spectre can also affect your system's processor and firmware. It is important to ensure that these are up-to-date to avoid potential problems. Check with your system's manufacturer for any relevant updates or patches.

Install Firewalls or Update Security Settings - Firewalls and other security software can also help protect against Spectre. Make sure to install or update any relevant programs.

Use the Instruments to Monitor for Suspicious Behavior - Lastly, you can use various instruments to monitor your system for suspicious activity or processes. This could include monitoring the system for unrecognized processes, suspicious network traffic, memory usage and more.

Testing Tools:

Target Testing	Testing Technique	Test Analysis	Test Method	Test Tool	Mobile Platform
Spectre Attack	White-box	Dynamic	Fuzzing	AFL fuzzer	iOS/Android/Windows
Spectre Attack	Grey-box	Dynamic	Bounding Box	Pitbull	iOS/Android/Windows
Spectre Attack	Black-box	Hybrid	Penetration Testing	Metasploit	iOS/Android/Windows

Testing the Meltdown Attack

Preparation * Check whether you have a processor that is vulnerable to Meltdown:

- Intel
- AMD
- ARM
- Download and install the Verifiable Builds of Meltdown Checker

Test * Run the Meltdown Checker:

```
shell $ ./meltdown_checker.py
```

- If your processor is vulnerable to Meltdown attack, you'll get an output that looks like this:

```shell System check (hardware & OS version) ...................... [OK]

Checking for vulnerability to Meltdown attack .............. VULNERABLE ```

- If your processor is not vulnerable to Meltdown attack, you'll get an output that looks like this:

```shell System check (hardware & OS version) ...................... [OK]

Checking for vulnerability to Meltdown attack .............. NOT VULNERABLE ```

## Testing Tools:

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

| | | | | | |
|---|---|---|---|---|---|
| Meltdown Attack | White-box | Dynamic | Penetration | Metasploit | iOS/Android |
| | Grey-box | Static | Code review | Veracode | iOS/Android |
| | Black-box | Hybrid | Fuzz Testing | InsightVM | iOS/Android |
| | | | | Burp Suite | iOS/Android |

## Testing Hardware Integrity Attack

Testing hardware integrity can be done by running a series of tests to check the validity of a hardware system.

Visual Inspection: Visually inspect the hardware system for any signs of physical damage such as corrosion, breaks and loose connectors.

Memory Test: Check the amount of RAM installed by running a memory test utility. Ensure the amount of RAM installed is sufficient to meet your system requirements.

Hard Drive Test: Run a hard drive test utility to check for bad sectors and ensure the drive is not overly fragmented.

BIOS Test: If your hardware needs a specialized driver, you should test the BIOS to make sure it is properly configured.

Disk Drive Test: Run a disk drive test utility to ensure the drive is functioning properly and is not corrupted.

Power Supply Test: Test the power supply to make sure it is correctly routed and can provide your hardware system with sufficient power.

Temperature and Noise Test: Monitor the temperature and noise levels of the system to ensure the components are not overheating or producing too much noise.

### Testing Tools:

| Target Testing | Testing Technique | Test Analysis | Test Method | Test Tool | Mobile Plataform |
|---|---|---|---|---|---|
| Hardware Integrity | White-box | Dynamic | Penetration Test | Kali Linux | iOS/Android Devices |
| Hardware Integrity | Grey-box | Static | Vulnerability Scanning | Nessus | iOS/Android Devices |
| Hardware Integrity | Black-box | Hybrid | Source Code Analysis | CodeInspect | iOS/Android Devices |

## Testing Rowhammer Attack

1. Choose a system with vulnerable DRAM modules:
   - It is important to have a system with vulnerable DRAM modules to test for Rowhammer.
2. Set up stressor application (e.g. memtest86+):
   - To test for Rowhammer, a stressor application is needed. A popular one, often used for this type of testing, is memtest86+.
3. Run the stressor application repeatedly for a longer period of time:
   - The stressor application should be run repeatedly for a longer period of time, usually several hours.
4. Monitor system response:
   - During the test, the system should be monitored to check for any errors or abnormalities.
5. Analyze results:
   - Once the testing period is over, the results should be analyzed for any evidence of Rowhammer attacks.

### Testing Tools:

| Target Testing | Testing Technique | Test Analysis | Test Method | Test Tool | Mobile Platform |
|---|---|---|---|---|---|
| Hardware | White-box | Dynamic | Hardware-in-the-Loop | Babblar | Android |
| Software | Grey-box | Static | Fuzz Testing | Windmill | iOS |
| Firmware | Black-box | Hybrid | Dynamic Web Testing | Syhunt | |
| Application | | | Penetration Testing | Metasploit | |

## 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 |