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/201

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.