Software Security Analysis & Thread Modeling

AUBURN UNIVERSITY

Security Focus Areas

- **CIA Principles**
  - **Confidentiality -** Data is only available to the people intended to access it.
  - **Integrity-** Data and system resources are only changed in appropriate ways by appropriate people.
  - **Availability -** Systems are ready when needed and perform acceptably.
- **Authentication -** The identity of users is established (or you're willing to accept anonymous users).
- **Authorization -** Users are explicitly allowed or denied access to resources.
- **Nonrepudiation**- Users can't perform an action and later deny performing it.

Threat Modeling

- Think like an Adversary – deconstruct a software application into the components and look for potential exploits and weaknesses
- Threat modeling goal is to gain an understanding of the software application by decomposing it and understanding how it interacts with external entities.
- Typical thread modeling steps-
  1. Understand application architecture
  2. Identify System components – data stores, web server, access, protocols, network access, external dependencies, hosting, data classification, user interface
  3. Identify Known Threat – OWasp for web applications, datastore access connection & access, source code implementation
     i. Lenses to use- User (privileged, non-privileged, Integration, Administration,
  4. Identify Vulnerabilities based on previous steps

Key Security Design Principles

- **Open design** - Assume the attackers have the sources and the specs.
- **Fail-safe defaults** - Fail closed; no single point of failure.   Unless explicit access is given to a system function or data, it should be denied access
- **Least privilege** - No more privileges than what is needed.
- **Economy of mechanism** - KISS principle (Keep it simple, stupid). Security mechanisms should be as simple as possible. Fewer error possibilities and the testing process is less complex
- **Separation of privileges** - For elevated functions require multiple permissions. E.g Banks require two people to approve cash withdrawals over $X amount
- **Complete Mediation** - Check authorization every time, don’t assume privilege continues after checking once.
- **Least common mechanism -** Be aware of resources (websites) that are shared and restrict access to authorized or good actors.

Threat Analysis Models

- Using industry standard models ensures a comprehensive analysis of an application.
- Microsoft STRIDE model-

Area	Example	Security Control
Spoofing	Impersonating a valid user. Most basic is stealing username/password to get through	**Authentication**
Tampering	changing data both at rest (disk, database, memory, etc.) or in transit (http, ftp, etc.)	**Integrity**
Repudiation	not providing sufficient logging and traceability to verify or audit sensitive system activity	**Nonrepudiation**
Information Disclosure	data leakage to an unauthorized individual or containing system information	**Confidentiality**
Denial of service	denying access to a system to users or business functions.	**Availability**
Elevation of privilege	gaining privileged access allowing ability to execute system functions or obtain data	**Authorization**

Threat Modeling

Common Threat Analysis Areas

- Input validation
- Authentication
- Authorization
- Configuration management
- Sensitive data
- Session management
- Cryptography
- Exception management
- Parameter manipulation
- Audit
- Logging

Input Validation

All user input should be considered untrusted and should be validated before being used in software.

- Is all input data validated?
  - Length, type, bounds, etc.
- Could an attacker inject commands or malicious data into the application?
- Is data validated as it is passed between separate trust boundaries (by the recipient entry point)?
- Can data in the database be trusted?
- Would you prefer whitelisting or blacklisting of user input?
  - Allowed characters/not allowed characters

Authentication

All of user or APIs interactions with overall system should be validated through authentication. None of the services and functionality should be available without validating external entity is legitimate. Once authenticated the next step takes us into **authorization**.

- Are credentials secured if they are passed over the network?
- Are strong account policies used?
- Are strong passwords enforced?
- Are you using certificates?
- Are there any wild card certificates in use?
- Are password verifiers (using one-way hashes) used for user passwords?
- How do system components authenticate to each other
  - Database to application; API to Server; System to system
- During boot process for the service/application, how do system components authenticate to each other?
- Are keys used for authentication instead of password?
- Is multi-factor authentication (MFA) necessary?

Authorization

Users should only be allowed to access system functionality through fined grained permissions. Authorization defines the controls a system uses to grant permissions to individual entities or groups associated with individual entities.

- What gatekeepers are used at the entry points of the application?
- How is authorization enforced at the database?
- Is a defense-in-depth strategy used?
- Do you fail securely and only allow access upon successful confirmation of credentials?

Configuration

Configuration management enables us to harden software, systems, services and devices and lock them down thus reducing risk to the environment. Components of configuration management include hardening standards and guidelines, reviewing application dependencies on services, looking at user and administrator interfaces, security change management and so on.

- What administration interfaces does the application support?
- How are they secured?
- How is remote administration secured?
- What configuration stores are used and how are they secured?
- Have hardening standards been developed for the software stack (OS, DB, Application)?
- Does software system provide a way to detect variances from approved security configuration changes?
- Do all groups (IT, QA, Engineering, Operations) only use approved (golden master) software images for different components such as OS, DB, Web, and Application servers?
- Do approved images are used across entire lifecycle from development to deployment

Sensitive Data

This aspect deals with awareness around type of data handled by application and systems. In many cases, we have found that developers and operations teams are not aware or educated enough on type of data their application will handle (either by design or mistake) and if protection is enough for data elements.

- What sensitive data is handled by the application?
  - Compliance – PCI, HIPAA
  - Confidential – PII, Financial, Customer
- What regulatory/compliance requirements are applicable to data/ data elements?
- How is it secured over the network and in persistent stores? Is this good enough given legal/regulatory requirements?
- What type of encryption is used and how are encryption keys secured?
- Are sensitive data elements present in logs, source code or configuration (e.g., XML) files?
- Is production data being used in test or development envirionments

Session Management

Securely establishing and maintaining integrity of session is one of the key components of today's applications specifically web applications. Once user is authenticated, a session is established. This can result in multiple scenarios where session can be abused. Session management focuses on preventing such abuses.

- How are session cookies generated?
- How are they secured to prevent session hijacking?
- How is persistent session state secured?
- Where is session information stored? On server or the client side?
- How is session state secured as it crosses the network?
- How does the application authenticate with the session store?
- Are credentials passed over the wire and are they maintained by the application? If so, how are they secured?
- How are multiple sessions from a user/component handled?

Cryptography

This aspect deals with awareness around type of data handled by application and systems. In many cases, we have found that developers and operations teams are not aware or educated enough on type of data their application will handle (either by design or mistake) and if protection is enough for data elements.

- What is the problem cryptography is going to solve (confidentiality, integrity or both)?
- What algorithms and cryptographic techniques are used?
- Are there any proprietary or in-house algorithms used?
- How long are encryption keys and how are they secured?
- Does the application put its own encryption into action?
- How often are keys recycled? Are certificates checked for their validity? Are certificates checked against revocation lists?

Parameter Manipulation

Application often passes parameters to communicate with the other side. Parameters range from (not so important) iterators, variable names and values to session tokens. Man-in-the middle (MITM) attacks and deliberate parameter tampering makes it important for us to device mechanism to detect if parameters received are indeed safe and can be used as designed. Imperative here is on the receiver side—like input validation, do not blindly trust parameters.

- Does the application detect tampered parameters?
- Does the application rely on only client-side validation or there is server side validation as well?
- Does it validate all parameters in form fields, view state, cookie data, and HTTP headers?
- Are parameters directly used in database queries?
- Are parameters directly reflected back to the browser?

Exception Management

Gracefully handling error conditions and exceptions is critical to software applications. Often, developers miss such conditions or handle them incorrectly. Side effects from improper error handling/exception management range from denial of service or information leakage.

- How does the application handle error conditions?
- Is there a default catch for exceptions?
- Are exceptions ever allowed to propagate back to the client?
- Are generic error messages that do not contain exploitable information used?
- Do exceptions log any sensitive information to logs?
- Are built-in capabilities from programming languages used for this purpose or developers rely on in-house modules? Auditing and Logging Audit and logging is critical for multiple reasons.

Auditing & Logging

Audit and logging is critical for multiple reasons. Security being one of them, audit trail in case of legal issues being another. Operations/ debugging is often the driver for audit/logging though increasingly attention is being paid to security aspect as well.

- Does your application audit activity across all tiers on all servers?
- How are log files secured?
- Does application log any sensitive information (e.g. credentials, data elements, session tokens)?
- Are log files transported securely (e.g. TCP/TLS)?
- Is retention period clearly defined for log files? Does it align with regulatory and legal requirements?
- How often are logs rotated?
- Are trigger levels defined for certain types of events?

Mitigation Techniques

Mitigation Techniques & Methods

- **Input Validation**
  - Data type, format, length, and range checks are enforced.
  - All data sent from the client is validated.
- **Authentication**
  - Credentials and authentication tokens are protected with encryption in storage and transit.
  - Protocols are resistant to brute force, dictionary, and replay attacks.
  - Strong password policies are enforced.
  - Trusted server authentication is used instead of SQL authentication.
  - Passwords are stored with salted hashes.
  - Password resets do not reveal password hints and valid usernames.

Mitigation Techniques & Methods

- **Authorization**
  - Strong ACLs are used for enforcing authorized access to resources.
  - Role-based access controls are used to restrict access to specific operations.
  - The system follows the principle of least privilege for user and service accounts.
  - Privilege separation is correctly configured within the presentation, business, and data access layers.
- **Configuration management**
  - Least privileged processes are used and service accounts with no administration capability.
  - Auditing and logging of all administration activities is enabled..
  - Access to configuration files and administrator interfaces is restricted to administrators

Mitigation Techniques & Methods

- **Encryption**
  - Standard encryption algorithms and correct key sizes are used.
  - Hashed message authentication codes (HMACs) are used to protect data integrity.
  - Secrets (e.g., keys, confidential data) are cryptographically protected both in transport and in storage.
  - Built-in secure storage is used for protecting keys.
  - No credentials and sensitive data are sent in clear text over the wire.
- **Parameter validation**
  - No security decision is based on parameters (e.g., URL parameters) that can be manipulated.
  - Input filtering via white list validation is used.
  - Output encoding is used.

Mitigation Techniques & Methods

- **Exception management**
  - All exceptions are handled in a structured manner.
  - Privileges are restored to the appropriate level in case of errors and exceptions.
  - Error messages are scrubbed so that no sensitive information is revealed to the attacker.
- **Session management**
  - No sensitive information is stored in clear text in the cookie.
  - The contents of the authentication cookies is encrypted.
  - Cookies are configured to expire.
  - Sessions are resistant to replay attacks.
  - Secure communication channels are used to protect authentication cookies.
  - User is forced to re-authenticate when performing critical functions.
  - Sessions are expired at logout.

Mitigation Techniques & Methods

- **Auditing and logging**
  - Sensitive information (e.g., passwords, PII) is not logged. 2
  - . Access controls (e.g., ACLs) are enforced on log files to prevent unauthorized access.
  - Integrity controls (e.g., signatures) are enforced on log files to provide nonrepudiation.
  - Log files provide for audit trail for sensitive operations and logging of key events.
  - Auditing and logging is enabled across the tiers on multiple servers.

Sample Web Application

Example – Web Application

- **Input and Data Validation**
  - How do you know that the input your application receives is valid and safe? Input validation refers to how your application filters, scrubs, or rejects input before additional processing. Consider constraining input through entry points and encoding output through exit points.
  - Do you trust data from sources such as databases and file shares?
- **Authentication**
  - Who are you? Authentication is the process whereby an entity proves the identity of another entity, typically through credentials, such as a user name and password.
- **Authorization**
  - What can users do?
  - Authorization is how your application provides access controls for resources and operations.

Example – Web Application

- **Configuration Management**
  - Who does your application run as?
  - Which databases does it connect to?
  - How is your application administered?
  - How are these settings secured? Configuration management refers to how your application handles these operational issues.
- **Sensitive Data**
  - How does your application handle sensitive data? Sensitive data refers to how your application handles any data that must be protected either in memory, over the network, or in persistent stores.
- **Session Management**
  - How does your application handle and protect user sessions? A session refers to a series of related interactions between a user and your web application

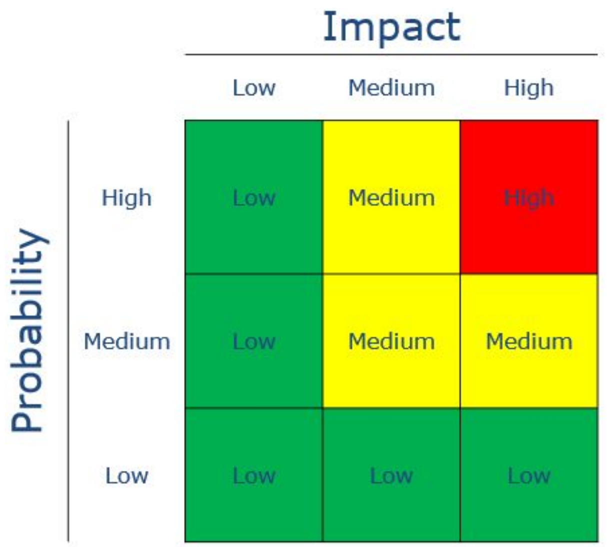
Example – Web Application

- **Cryptography**
  - How are you keeping secrets (confidentiality)? How are you tamper-proofing your data or libraries (integrity)? How are you providing seeds for random values that must be cryptographically strong? Cryptography refers to how your application enforces confidentiality and integrity.
- **Exception Management**
  - When a method call in your application fails, what does your application do? How much do you reveal?
  - Do you return friendly error information to end users?
  - Do you pass valuable exception information back to the caller? Does your application fail gracefully?
- **Auditing and Logging**
  - Who did what and when? Auditing and logging refer to how your application records security-related events

Security Risk Analysis

Risk Analysis

- Once potential vulnerabilities are identified it is important to measure the risk associated with each as it will drive level of mitigation, testing, and timing.
- There exists various risk formulas in industry to measure the risk and consequence of a particular vulnerability. Simplified the calculation is something like-
  - **Risk Value = Probability X Damage Potential**
- View in a Matrix form-
  - Low
  - Medium
  - High



		Impact		
		Low	Medium	High
Probability	High	Low	Medium	High
	Medium	Low	Medium	Medium
	Low	Low	Low	Low

Microsoft DREAD Risk Model

- **RISK** = (**D**amage + **R**eproducibility + **E**xploitability + **A**ffected Users + **D**iscoverability) / 5
  - Damage – Damage if vulnerability is exploited
  - Reproducibility – Is the vulnerability easy to exploit?
  - Exploitability – What skills/knowledge are needed to take advantage of exploit
  - Affected Users – How many users or how much data will be affected
  - Discoverability – Is the threat easy to discover?
- Example-
  - **Affected Users** on scale of 1-10-
    - 1 – no users
    - 5 – limited subset of users or data
    - 10 – entire user base or dataset

Summary

- Accurate Software Architectural & Decomposition
- Use of appropriate threat framework to ensure comprehensive analysis
- Identification of security threats
- Risk and impact analysis of identified threats.
- Implementation of appropriate mitigation