MODULE 4 THREAT MODELLING AND PURSUIT OF SECURITY THREATS

Threat modeling is an approach for analyzing the security of an application. It is a structured approach that enables you to identify, quantify, and address the security risks associated with an application. Threat modeling is not an approach to reviewing code, but it does complement the security code review process. The inclusion of threat modeling in the SDLC can help to ensure that applications are being developed with security built-in from the very beginning.

The threat modeling process can be decomposed into 3 high level steps:

Step 1: Decompose the Application. The first step in the threat modeling process is concerned with gaining an understanding of the application and how it interacts with external entities. This involves creating use-cases to understand how the application is used, identifying entry points to see where a potential attacker could interact with the application, identifying assets i.e. items/areas that the attacker would be interested in, and identifying trust levels which represent the access rights that the application will grant to external entities. This information is documented in the Threat Model document and it is also used to produce data flow diagrams (DFDs) for the application. The DFDs show the different paths through the system, highlighting the privilege boundaries.

Step 2: Determine and rank threats. Critical to the identification of threats is using a threat categorization methodology. A threat categorization such as STRIDE can be used, or the Application Security Frame (ASF) that defines threat categories such as Auditing & Logging, Authentication, Authorization, Configuration Management, Data Protection in Storage and Transit, Data Validation, Exception Management. The goal of the threat categorization is to help identify threats both from the attacker (STRIDE) and the defensive perspective (ASF). DFDs produced in step 1 help to identify the potential threat targets from the attacker's perspective, such as data sources, processes, data flows, and interactions with users. These threats can be identified further as the roots for threat trees; there is one tree for each threat goal. From the defensive perspective, ASF categorization helps to identify the threats as weaknesses of security controls for such threats. Common threat-lists with examples can help in the identification of such threats. Use and abuse cases can illustrate how existing protective measures could be bypassed, or where a lack of such protection exists. The determination of the security risk for each threat can be determined using a value-based risk model such as DREAD or a less subjective qualitative risk model based upon general risk factors (e.g. likelihood and impact).

Step 3: Determine countermeasures and mitigation. A lack of protection against a threat might indicate a vulnerability whose risk exposure could be mitigated with the implementation of a countermeasure. Such countermeasures can be identified using threat-countermeasure mapping lists. Once a risk ranking is assigned to the threats, it is possible to sort threats from the highest to the lowest risk, and prioritize the mitigation effort, such as by responding to such threats by applying the identified countermeasures. The risk mitigation strategy might involve evaluating these threats from the business impact that they pose and reducing the risk. Other options might include taking the risk, assuming the business impact is acceptable because of compensating controls, informing the user of the threat, removing the risk posed by the threat completely, or the least preferable option, that is, to do nothing.

Each of the above steps are documented as they are carried out. The resulting document is the threat model for the application.

Decompose the Application

The goal of this step is to gain an understanding of the application and how it interacts with external entities. This goal is achieved by information gathering and documentation. The information gathering process is carried out using a clearly defined structure, which ensures the correct information is collected. This structure also defines how the information should be documented to produce the Threat Model.

Threat Model Information

The first item in the threat model is the information relating to the threat model. This must include the following:

1. **Application Name** - The name of the application.
2. **Application Version** - The version of the application.
3. **Description** - A high level description of the application.
4. **Document Owner** - The owner of the threat modeling document.
5. **Participants** - The participants involved in the threat modeling process for this application.
6. **Reviewer** - The reviewer(s) of the threat model.

External Dependencies

External dependencies are items external to the code of the application that may pose a threat to the application. These items are typically still within the control of the organization, but possibly not within the control of the development team. The first area to look at when investigating external dependencies is how the application will be deployed in a production environment, and what are the requirements surrounding this. This involves looking at how the application is or is not intended to be run. For example if the application is expected to be run on a server that has been hardened to the organization's hardening standard and it is expected to sit behind a firewall, then this information should be documented in the external dependencies section. External dependencies should be documented as follows:

1. **ID** - A unique ID assigned to the external dependency.
2. **Description** - A textual description of the external dependency.

Entry Points

Entry points define the interfaces through which potential attackers can interact with the application or supply it with data. In order for a potential attacker to attack an application, entry points must exist. Entry points in an application can be layered, for example each web page in a web application may contain multiple entry points. Entry points should be documented as follows:

1. **ID** - A unique ID assigned to the entry point. This will be used to cross reference the entry point with any threats or vulnerabilities that are identified. In the case of layer entry points, a major. minor notation should be used.
2. **Name** - A descriptive name identifying the entry point and its purpose.
3. **Description** - A textual description detailing the interaction or processing that occurs at the entry point.
4. **Trust Levels** - The level of access required at the entry point is documented here. These will be cross referenced with the trusts levels defined later in the document.

Assets

The system must have something that the attacker is interested in; these items/areas of interest are defined as assets. Assets are essentially threat targets, i.e. they are the reason threats will exist. Assets can be both physical assets and abstract assets. For example, an asset of an application might be a list of clients and their personal information; this is a physical asset. An abstract asset might be the reputation of an organization. Assets are documented in the threat model as follows:

1. **ID** - A unique ID is assigned to identify each asset. This will be used to cross reference the asset with any threats or vulnerabilities that are identified.
2. **Name** - A descriptive name that clearly identifies the asset.
3. **Description** - A textual description of what the asset is and why it needs to be protected.

4. **Trust Levels** - The level of access required to access the entry point is documented here. These will be cross referenced with the trust levels defined in the next step.

Trust Levels

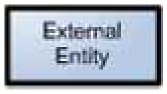
Trust levels represent the access rights that the application will grant to external entities. The trust levels are cross referenced with the entry points and assets. This allows us to define the access rights or privileges required at each entry point, and those required to interact with each asset. Trust levels are documented in the threat model as follows:

1. **ID** - A unique number is assigned to each trust level. This is used to cross reference the trust level with the entry points and assets.
2. **Name** - A descriptive name that allows you to identify the external entities that have been granted this trust level.
3. **Description** - A textual description of the trust level detailing the external entity who has been granted the trust level.

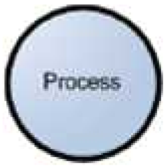
Data Flow Diagrams

All of the information collected allows us to accurately model the application through the use of Data Flow Diagrams (DFDs). The DFDs will allow us to gain a better understanding of the application by providing a visual representation of how the application processes data. The focus of the DFDs is on how data moves through the application and what happens to the data as it moves. DFDs are hierarchical in structure, so they can be used to decompose the application into subsystems and lower-level subsystems. The high level DFD will allow us to clarify the scope of the application being modeled. The lower level iterations will allow us to focus on the specific processes involved when processing specific data. There are a number of symbols that are used in DFDs for threat modeling. These are described below:

External Entity
The external entity shape is used to represent any entity outside the application that interacts with the application via an entry point.

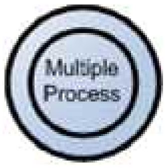


Process
The process shape represents a task that handles data within the application. The task may process the data or perform an action based on the data.



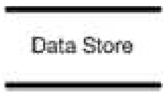
Multiple Process

The multiple process shape is used to present a collection of sub processes. The multiple process can be broken down into its sub processes in another DFD.



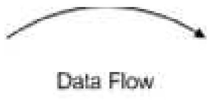
Data Store

The data store shape is used to represent locations where data is stored. Data stores do not modify the data, they only store data.



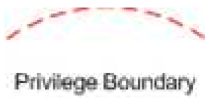
Data Flow

The data flow shape represents data movement within the application. The direction of the data movement is represented by the arrow.



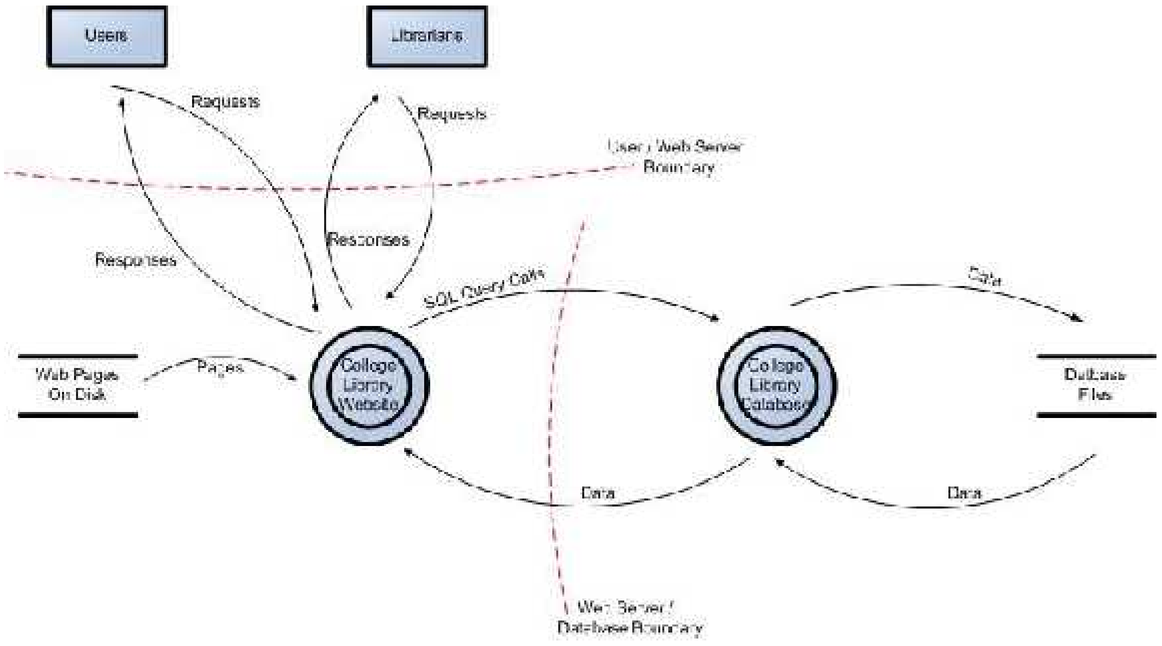
Privilege Boundary

The privilege boundary shape is used to represent the change of privilege levels as the data flows through the application.



Example

Data Flow Diagram for the College Library Website



Determine and Rank Threats

Threat Categorization

The first step in the determination of threats is adopting a threat categorization. A threat categorization provides a set of threat categories with corresponding examples so that threats can be systematically identified in the application in a structured and repeatable manner.

STRIDE

A threat categorization such as STRIDE is useful in the identification of threats by classifying attacker goals such as:

- Spoofing
- Tampering
- Repudiation
- Information Disclosure
- Denial of Service
- Elevation of Privilege.

Security Controls

Once the basic threat agents and business impacts are understood, the review team should try to identify the set of controls that could prevent these threat agents from causing those impacts. The primary focus of the code review should be to ensure that these security controls are in place, that they work properly, and that they are correctly invoked in all the necessary places. The checklist below can help to ensure that all the likely risks have been considered.

Authentication:

- Ensure all internal and external connections (user and entity) go through an appropriate and adequate form of authentication. Be assured that this control cannot be bypassed.
- Ensure all pages enforce the requirement for authentication.
- Ensure that whenever authentication credentials or any other sensitive information is passed, only accept the information via the HTTP “POST” method and will not accept it via the HTTP “GET” method.
- Any page deemed by the business or the development team as being outside the scope of authentication should be reviewed in order to assess any possibility of security breach.
- Ensure that authentication credentials do not traverse the wire in clear text form.
- Ensure development/debug backdoors are not present in production code.

Authorization:

- Ensure that there are authorization mechanisms in place.
- Ensure that the application has clearly defined the user types and the rights of said users.
- Ensure there is a least privilege stance in operation.
- Ensure that the Authorization mechanisms work properly, fail securely, and cannot be circumvented.
- Ensure that authorization is checked on every request.
- Ensure development/debug backdoors are not present in production code.

Cookie Management:

- Ensure that sensitive information is not comprised.

- Ensure that unauthorized activities cannot take place via cookie manipulation.
- Ensure that proper encryption is in use.
- Ensure secure flag is set to prevent accidental transmission over “the wire” in a non-secure manner.
- Determine if all state transitions in the application code properly check for the cookies and enforce their use.
- Ensure the session data is being validated.
- Ensure cookies contain as little private information as possible.
- Ensure entire cookie is encrypted if sensitive data is persisted in the cookie.
- Define all cookies being used by the application, their name, and why they are needed.

Data/Input Validation:

- Ensure that a DV mechanism is present.
- Ensure all input that can (and will) be modified by a malicious user such as HTTP headers, input fields, hidden fields, drop down lists, and other web components are properly validated.
- Ensure that the proper length checks on all input exist.
- Ensure that all fields, cookies, http headers/bodies, and form fields are validated.
- Ensure that the data is well formed and contains only known good chars if possible.
- Ensure that the data validation occurs on the server side.
- Examine where data validation occurs and if a centralized model or decentralized model is used.
- Ensure there are no backdoors in the data validation model.
- **Golden Rule: All external input, no matter what it is, is examined and validated.**

Error Handling/Information leakage:

- Ensure that all method/function calls that return a value have proper error handling and return value checking.

- Ensure that exceptions and error conditions are properly handled.
- Ensure that no system errors can be returned to the user.
- Ensure that the application fails in a secure manner.
- Ensure resources are released if an error occurs.

Logging/Auditing:

- Ensure that no sensitive information is logged in the event of an error.
- Ensure the payload being logged is of a defined maximum length and that the logging mechanism enforces that length.
- Ensure no sensitive data can be logged; e.g. cookies, HTTP “GET” method, authentication credentials.
- Examine if the application will audit the actions being taken by the application on behalf of the client (particularly data manipulation/Create, Update, Delete (CUD) operations).
- Ensure successful and unsuccessful authentication is logged.
- Ensure application errors are logged.
- Examine the application for debug logging with the view to logging of sensitive data.

Cryptography:

- Ensure no sensitive data is transmitted in the clear, internally or externally.
- Ensure the application is implementing known good cryptographic methods.

Secure Code Environment:

- Examine the file structure. Are any components that should not be directly accessible available to the user?
- Examine all memory allocations/de-allocations.
- Examine the application for dynamic SQL and determine if it is vulnerable to injection.
- Examine the application for “main()” executable functions and debug harnesses/backdoors.

- Search for commented out code, commented out test code, which may contain sensitive information.
- Ensure all logical decisions have a default clause.
- Ensure no development environment kit is contained on the build directories.
- Search for any calls to the underlying operating system or file open calls and examine the error possibilities.

Session Management:

- Examine how and when a session is created for a user, unauthenticated and authenticated.
- Examine the session ID and verify if it is complex enough to fulfill requirements regarding strength.
- Examine how sessions are stored: e.g. in a database, in memory etc.
- Examine how the application tracks sessions.
- Determine the actions the application takes if an invalid session ID occurs.
- Examine session invalidation.
- Determine how multithreaded/multi-user session management is performed.
- Determine the session HTTP inactivity timeout.
- Determine how the log-out functionality functions.

WEB SERVER SECURITY

Introduction

Internet facing web servers are exposed to high security risks. We quite commonly see web servers being hacked (eg: malicious code being injected in website content), and then clients that are browsing the website are most likely to be transparently compromised (aka drive-by download). And there is also the denial of service risk, the information leakage risk, etc.

Security to the web server needs to be divided in various sections, these sections are having different challenges which includes the zero day attacks, patching issues, infrastructure related problems.

Securing the Operating System

Web servers operate on a general-purpose operating system. Many security issues can be avoided if the operating systems underlying Web servers are configured appropriately. Default hardware and software configurations are typically set by vendors to emphasize features, functions, and ease of use at the expense of security.

Securely installing and configuring an Operating system

1. Patch and Upgrade Operating System
   All operating systems released today have some known vulnerabilities that should be corrected before using the operating system to host a Web server. The patching should be done through a process as follows:
   - Create and implement a patching process
   - Identify latest patches for the vulnerabilities in the wild
   - Mitigate vulnerabilities through some work around solution until patches are available
   - Install permanent fixes.

2. Remove or Disable Unnecessary Services and Applications
   A Web server should be on a dedicated, single-purpose host. Many operating systems are configured by default to provide a wider range of services and applications than required by a Web server; therefore, a Web administrator should configure the operating system to remove or disable unneeded services. Some common examples of services that should usually be disabled would include:
   - Windows Network Basic Input/Output System (NetBIOS), if not required
   - NFS, if not required „ File Transfer Protocol (FTP)
   - Berkeley “r” services (e.g., rlogin, rsh, rcp)
   - Telnet

- Network Information System (NIS)
- Simple Mail Transfer Protocol (SMTP)
- Compilers
- Software development tools

Removing unnecessary services and applications is preferable to simply disabling them through configuration settings, because attacks that attempt to alter settings and activate a disabled service cannot succeed when the functional components are completely removed.

When configuring the operating system, apply the principle "disable everything except that which is expressly permitted" – that is, disable or, preferably, remove all services and applications and then selectively enable those required by the Web server. If possible, install the minimal operating system configuration that is required for the Web server application. If the operating system installation system provides a "minimal installation" option, choose that because it will minimize the effort required to remove unnecessary services. Many uninstall scripts or programs do not completely remove all components of service; therefore, it is always better to avoid installing unnecessary services when possible.

The services enabled on a Web server will depend on the functions the organization wants the server to provide. Those services might include database protocols to access a database, file transfer protocols, and remote administration services. Each of these services, even though they may be required, comes with an increased risk to the server. Whether the risks outweigh the benefits is a decision each organization must make for itself.

3. Configuring Operating System User Authentication

   For Web servers, authorized users who can configure the system and initiate Web services are typically a small number of designated Web administrators and Webmasters. However, the users who can access the public Web server may range from unrestricted to restricted subsets of the Internet community. To enforce policy restrictions, if required, the Web administrator must configure the system to authenticate prospective users by requiring proof that each person is authorized for such access. Even though a Web server may allow unauthenticated access to most Web services, administrative and other types of specialized access should be limited to specific individuals and groups.

   To ensure the appropriate user authentication is in place, take the following steps:

   a) Remove or disable unneeded default accounts and groups

      The default configuration of the operating system often includes guest accounts (with and without passwords), administrator or root level accounts, and accounts associated with local and network services. The names and passwords for those accounts are well known. Remove or disable unnecessary accounts to eliminate their use by intruders, including guest accounts on computers containing sensitive information. If there is no requirement to retain a guest account or group, severely restrict its access and change the password in accordance with the organizational password policy.

b) Disable non-interactive accounts
Disable accounts (and the associated passwords) that need to exist but do not require an interactive login. For Unix systems, disable the login shell, or provide a login shell with NULL functionality (/bin/false).

c) Create the user groups.
Assign users to the appropriate groups. Then assign rights to the groups. This approach is preferable to assigning rights to individual users.

d) Create the user accounts
Identify who will be authorized to use each computer and its services. Create only the necessary accounts. Discourage or prohibit the use of shared accounts.

e) Check the organization's password policy
Set account passwords appropriately which should be based on Length, complexity, Aging, Reuse, Authority.

f) Configure computers to deny login after a small number of failed attempts
It is relatively easy for an unauthorized user to try to gain access to a computer by using automated software tools that attempt all passwords. If the operating system provides the capability, configure it to deny login after three failed attempts. Typically, the account is "locked out" for a period of time (such as 30 minutes) or until a user with appropriate authority reactivates it.

g) Install and configure other security mechanisms to strengthen authentication
If the information on the Web server requires it, consider using other authentication mechanisms such as tokens, client/server certificates, or one-time password systems. Although they can be more expensive and difficult to implement, they may be justified in some circumstances. When such authentication mechanisms and devices are used, the organization's policy should be reviewed to reflect in the way in which they are applied.

h) Security testing the OS
It's necessary to validate all the controls through periodic assessment like penetration testing and vulnerability assessment.

Securely installing and configuring the web server

1. Securely Installing the Web Server

The minimal amount of Web server services required and eliminate any known vulnerabilities through patches or upgrades. If the installation program installs any unnecessary applications, services, or scripts, they should be removed immediately once the installation process completes. During the installation of the Web server, the following steps should be performed:

a) Install the server software on a dedicated host
b) Install the minimal Internet services required
c) Apply any patches or upgrades to correct for known vulnerabilities
d) Create a dedicated physical disk or logical partition (separate from operating system and server application) for Web content
e) Remove or disable all services installed by the Web server application but not required (e.g., gopher, FTP, and remote administration)
f) From the Web server application root directory, remove all files that are not part of the Web site
g) Remove all sample documents, scripts, and executable code
h) Remove all vendor documentation from server
i) Apply appropriate security template or hardening script to server Reconfigure HTTP service banner (and others as required) NOT to report Web server and operating system type and version. (This can be accomplished in IIS using the Microsoft's free IIS Lockdown Tool and in Apache via the "ServerTokens" directive.)

2. Configuring Access Controls

Most Web server host operating systems provide a capability to specify access privileges individually for files, devices, and other computational resources on that host. Any information that the Web server can access using these controls can potentially be distributed to all users accessing the public Web site. The Web server software is likely to provide additional file, device, and resource access controls specific to its operation.

Web administrators should consider from two perspectives how best to configure these access controls to protect information stored on their public Web server:

a) Limit the access of the Web server software to a subset of computational resources
b) Limit the access of users through additional access controls enforced by the Web server, where more detailed levels of access control are required

Typical files to which access should be controlled are as follows:

- Application software and configuration files
- Files related directly to security mechanisms:
  - Password hash files and other files used in authentication

        - Files containing authorization information used in controlling access
        - Cryptographic key material used in confidentiality, integrity, and non-repudiation services.
    - Server log and system audit files
    - System software and configuration files.

3. Using File Integrity Checkers

A file integrity checker is an application that computes and stores a checksum for every guarded file and establishes a database of file checksums. It allows a system administrator to easily recognize changes to critical files, particularly unauthorized changes. Checksums should be recomputed regularly to test the current value against the stored value to identify any file modifications.

Although an integrity checker is a useful tool that does not require a high degree of human interaction, it needs to be used carefully to ensure that it is effective. To create the first reference database a file integrity checker requires a system that is known to be in a secure state.

Integrity checkers should be run nightly on a selection of system files that would be affected by a compromise. Integrity checkers should also be used when a compromise is suspected for determining the extent of possible damage. If an integrity checker detects unauthorized system file modifications, the possibility of a security incident should be considered and investigated according to the organization's incident response and reporting policy and procedures.

Securing Web Content

The two main components to Web security are the security of the underlying server application and operating systems, and the security of the actual content.

1. Publishing Information on Public Web Sites

Authentication and Encryption Technologies

Implementing a secure network for a web server

Administration of Webserver