ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

ΣΧΕΔΙΑΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΑΣΦΑΛΕΙΑΣ ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Application Threat Modelling

An approach for analyzing the security of an application.

It is a structured approach that enables you to identify, classify, rate, compare and prioritize the security risks associated with an application.

Inducing Application Threat Modeling into SDLC process has its advantages for the security of the entire project.

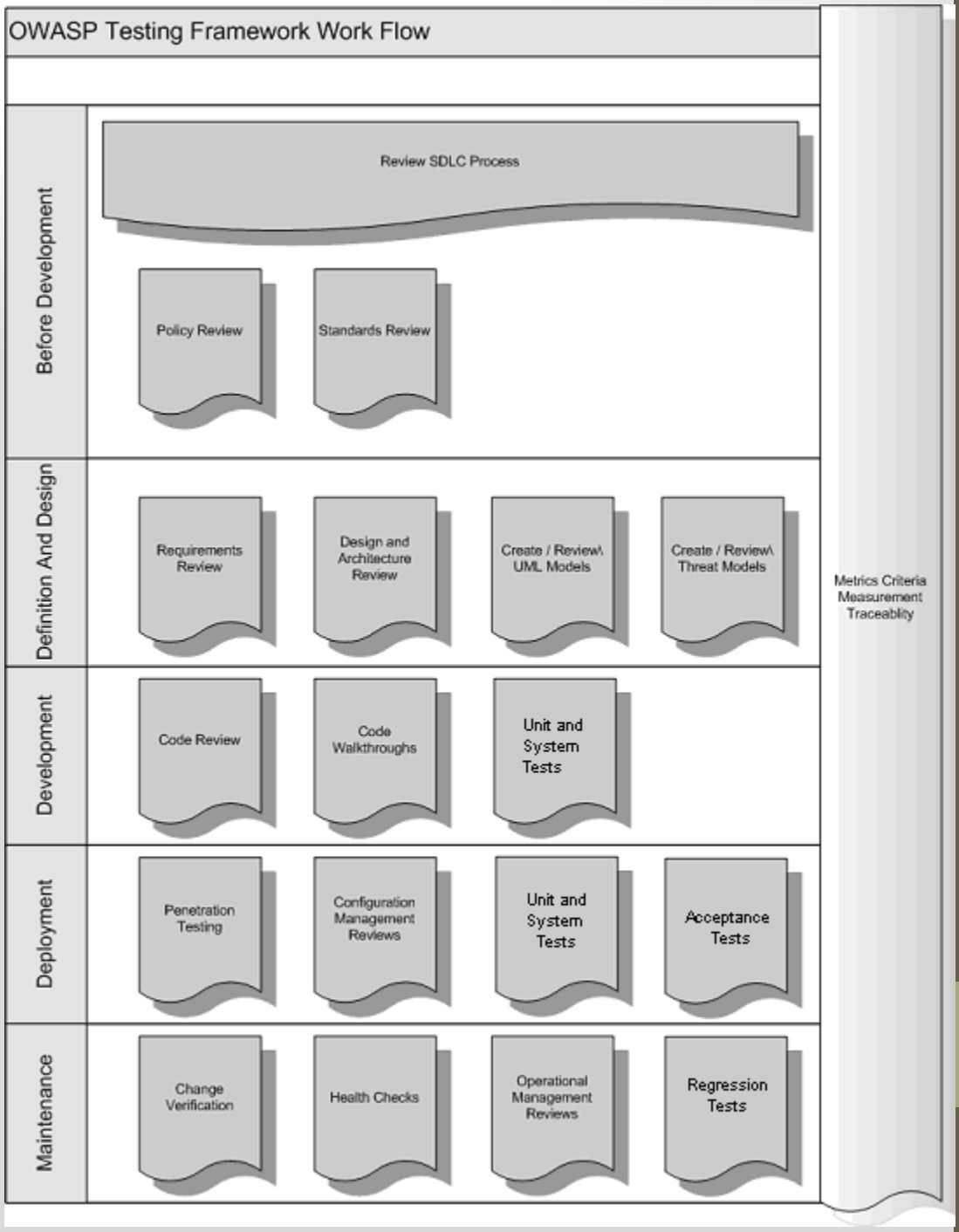
Threat Modeling Goals

- **GOAL I:** Systematically identify and rate the threats that are most likely to affect your application.
- **GOAL II:** Address threats with appropriate countermeasures

Threat Modeling Goals

To perform Application Threat Modeling use testing methodologies/techniques/frameworks/methods (e.g. OWASP testing framework) to identify, STRIDE methodology to Classify and DREAD methodology to rate, compare and prioritize risks, based on severity.

Software Development Life Cycle (SDLC)



Security Development Lifecycle (SDL)

Main Phases

Phase I: Decompose the Application

- Application Architecture
- External Dependencies
- Trust Levels
- Entry Points
- Assets
- Data Flow Diagrams

Phase II: Determine and rank threats.

- Threat Identification
- Threat Categorization
- Security Controls
- Threat Analysis
- Ranking of Threats (DREAD)

Phase III: Determine countermeasures and mitigation

- Countermeasure Identification
- Mitigation Strategies

Phase I: Decompose the Application

- **Goal**: 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.

Step 1.1. Application Architecture

- Goal: Identification of the basic entities of the application

Examples:

- Application Servers
- Database Servers
- Application Technologies
- Application version
- Application Owner

Step 1.2. External Dependencies

- **Goal**: Identification of any external dependencies of the application

- Examples:
  - Web Servers (e.g. Apache)
  - Connection between database and web server
  - Firewall
  - Other applications

Step 1.3. Trust Levels

- **Goal**: Identify the appropriate access rights of the application entities (actors)
- "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 examples

ID	Name	Description
1	Anonymous Web User	A user who has connected to the college library website but has not provided valid credentials.
2	User with Valid Login Credentials	A user who has connected to the college library website and has logged in using valid login credentials.
3	User with Invalid Login Credentials	A user who has connected to the college library website and is attempting to log in using invalid login credentials.
4	Librarian	The librarian can create users on the library website and view their personal information.
5	Database Server Administrator	The database server administrator has read and write access to the database that is used by the website.
6	Website Administrator	The Website administrator can configure the website.
7	Database Read User	The database user account used to access the database for read access.
8	Database Read/Write User	The database user account used to access the database for read and write access.

Step 1.4. Entry Points

- **Goal**: Identify the entry points through which a potential attacker could interact with the application or gain access to data
- Examples:
- **HTTPS Port**
- **Main page**: The splash page for the website is the entry point for all users.
- **Login Function**: The login function accepts user supplied credentials and compares them with those in the database.
- **Search Entry Page**: This functionality allows users to enter a search query.

Step 1.5. Assets

- **Goal**: identify the critical assets (physical and abstract assets). Application information that the attacker is interested in

- Examples:
- Availability of the application
- User Login Details
- User Personal Data
- Access to the Database Server
- Login Session
- Reputation of the organization

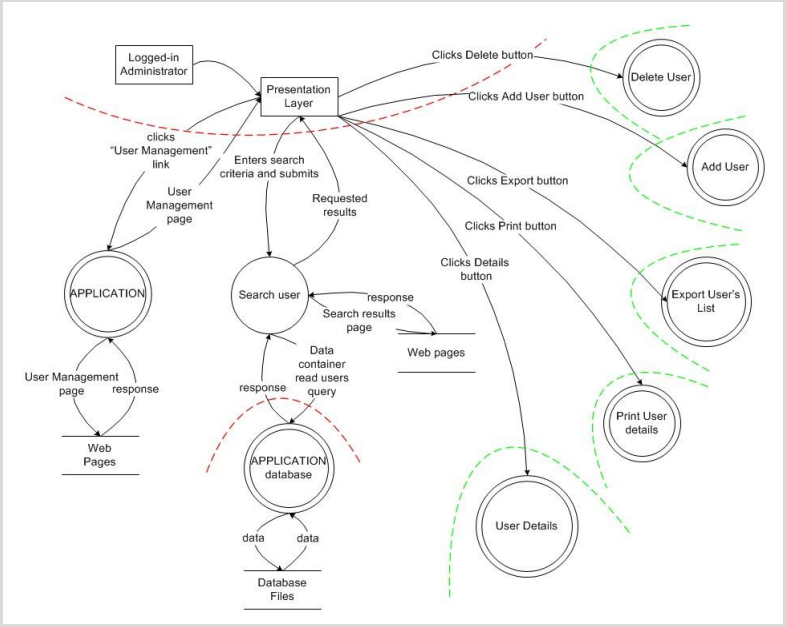
Step 1.6. Data Flow Diagrams

- **Goal**: better understand the application processes and workflow
- Basic DFDs shapes:
  - **External Entity:** user (actor), other application
  - **Process:** basic application process that handles data (reads or writes data)
  - **Multiple process**: this shape represents a collection of sub processes
  - **Data Store**: The data store shape is used to represent locations where data is stored.
  - **Data Flow:** This shape represents data movement within the application
  - **Privilege Boundary:** The privilege boundary shape is used to represent the change of privilege levels as the data flows through the application.

DFD Example

- **"Search and View user list" byApplication Admin Use Case**

1. Application Admin clicks "User Management" link
2. The User Management page is displayed
3. Application Admin enters search criteria and click "Submit"
4. Application Admin is able to view the users list based on the requested search criteria
5. Application Admin is able to click the "Details" button for a specific user (in order to see the selected user details)
6. Application Admin is able to click the "print" button (in order to print the list)
7. Application Admin is able to click the "Export" button (in order to export the list)
8. Application Admin is able to click the "Add" button (in order to add a newuser)
9. Application Admin is able to click the "Delete" button for a specific user (in order to delete the user)



Phase II: Determine and rank threats.

- **Goal:** Identify and ranking threats

- Identify Threats
- Identify Possible Weaknesses
- Identify Security Controls
- Ranking Threats

Step 2.1. Threat Identification

- How can the malicious actor use or manipulate the asset to:
  - Modify or control the application?
  - Retrieve information within the application?
  - Manipulate information within the application?
  - Cause the application to fail or become unusable?
  - Gain additional rights?
- Can a malicious actor access the application asset:
  - Without being audited?
  - And skip any access control checks?
  - And appear to be another user?

Source: SANS Institute - Threat Modeling: A Process To Ensure Application Security

Step 2.2. Threat Categorization

- Categorizing threats makes it easier to understand what the threats allow an attacker to do and aids in assigning priority
- The threat categorization is performed based on the STRIDE model:
  - **Spoofing**: Threat action aimed to illegally access and use another user's credentials, such as username and password
  - **Tampering:** Threat action aimed to maliciously change/modify persistent data, such as persistent data in a database, and the alteration of data in transit between two computers over an open network, such as the Internet
  - **Repudiation:** Threat action aimed to perform illegal operations in a system that lacks the ability to trace the prohibited operations
  - **Information Disclosure:** Threat action to read a file that one was not granted access to, or to read data in transit
  - **Denial of Service:** Threat aimed to deny access to valid users, such as by making a web server temporarily unavailable or unusable
  - **Elevation of privilege**: Threat aimed to gain privileged access to resources for gaining unauthorized access to information or to compromise a system

Threats by Application Vulnerability Category

Category	Threats
Input validation	Buffer overflow; cross-site scripting; SQL injection; canonicalization
Authentication	Network eavesdropping; brute force attacks;dictionary attacks; cookie replay; credential theft
Authorization	Elevation of privilege; disclosure of confidential data; data tampering; luring attacks
Configuration management	Unauthorized access to administration interfaces; unauthorized access to configuration stores; retrieval of clear text configuration data; lack of individual accountability; over-privileged process and service accounts
Sensitive data	Access sensitive data in storage; network eavesdropping; data tampering
Session management	Session hijacking; session replay; man in the middle
Cryptography	Poor key generation or key management; weak or custom encryption
Parameter manipulation	Query string manipulation; form field manipulation; cookie manipulation; HTTP header manipulation
Exception management	Information disclosure; denial of service
Auditing and logging	User denies performing an operation; attacker exploits an application without trace; attacker covers his or her tracks

Step 2.3. Security Controls

- The primary goal of the code review is to ensure that appropriate controls are in place and work properly in order to mitigate the identified threats
- Detailed check list with security controls should be prepared by taking into account the ASVS (Application Security Verification Standard)

ASVS (Application Security Verification Standard) Domains

- Secure Software Development Lifecycle
- Authentication Architecture
- Access Control Architecture
- Input and Output Architecture
- Cryptographic Architecture
- Errors, Logging and Auditing Architecture
- Data Protection and Privacy Architecture
- Communications Architecture
- Malicious Software Architecture
- Business Logic Architecture
- Configuration Architecture

Secure Software Development Lifecycle

- Verify the use of a secure software development lifecycle that addresses security in all stages of development.
- Verify the use of threat modeling for every design change or sprint planning to identify threats, plan for countermeasures, facilitate appropriate risk responses, and guide security testing.
- Verify documentation and justification of all the application's trust boundaries, components, and significant data flows.
- Verify definition and security analysis of the application's high-level architecture and all connected remote services
- Verify implementation of centralized, simple (economy of design), vetted, secure, and reusable security controls to avoid duplicate, missing, ineffective, or insecure controls
- Verify availability of a secure coding checklist, security requirements, guideline, or policy to all developers and testers.

Authentication Architecture

- Verify the use of unique or special low-privilege operating system accounts for all application components, services, and servers.
- Verify that communications between application components, including APIs, middleware and data layers, are authenticated. Components should have the least necessary privileges needed.
- Verify that the application uses a single vetted authentication mechanism that is known to be secure, can be extended to include strong authentication, and has sufficient logging and monitoring to detect account abuse or breaches.
- Verify that all authentication pathways and identity management APIs implement consistent authentication security control strength, such that there are no weaker alternatives per the risk of the application.

Access Control Architecture

- Verify that trusted enforcement points, such as access control gateways, servers, and serverless functions, enforce access controls.
- Verify the application uses a single and well-vetted access control mechanism for accessing protected data and resources. All requests must pass through this single mechanism to avoid copy and paste or insecure alternative paths.
- Verify that attribute or feature-based access control is used whereby the code checks the user's authorization for a feature/data item rather than just their role. Permissions should still be allocated using roles

Input and Output Architecture

- Verify that input and output requirements clearly define how to handle and process data based on type, content, and applicable laws, regulations, and other policy compliance.
- Verify that serialization is not used when communicating with untrusted clients. If this is not possible, ensure that adequate integrity controls (and possibly encryption if sensitive data is sent) are enforced to prevent deserialization attacks including object injection.
- Verify that input validation is enforced on a trusted service layer

Cryptographic Architecture

- Verify that there is an explicit policy for management of cryptographic keys and that a cryptographic key lifecycle follows a key management standard.
- Verify that consumers of cryptographic services protect key material and other secrets by using key vaults or API based alternatives.
- Verify that all keys and passwords are replaceable and are part of a well-defined process to re-encrypt sensitive data.
- Verify that the architecture treats client-side secrets--such as symmetric keys, passwords, or API tokens--as insecure and never uses them to protect or access sensitive data.

Errors, Logging and Auditing Architecture

- Verify that a common logging format and approach is used across the system
- Verify that logs are securely transmitted to a preferably remote system for analysis, detection, alerting, and escalation

Data Protection and Privacy Architecture

- Verify that all sensitive data is identified and classified into protection levels.
- Verify that all protection levels have an associated set of protection requirements, such as encryption requirements, integrity requirements, retention, privacy and other confidentiality requirements, and that these are applied in the architecture.

Communications Architecture

- Verify the application encrypts communications between components, particularly when these components are in different containers, systems, sites, or cloud providers.
- Verify that application components verify the authenticity of each side in a communication link to prevent person-in-the-middle attacks.

Malicious Software Architecture

- Verify that a source code control system is in use, with procedures to ensure that check-ins are accompanied by issues or change tickets. The source code control system should have access control and identifiable users to allow traceability of any changes.

Business Logic Architecture

- Verify the definition and documentation of all application components in terms of the business or security functions they provide.
- Verify that all high-value business logic flows, including authentication, session management and access control, do not share unsynchronized state
- Verify that all high-value business logic flows, including authentication, session management and access control are thread safe and resistant to time-of-check and time-of-use race conditions

Configuration Architecture

- Verify the segregation of components of differing trust levels through well-defined security controls, firewall rules, API gateways, reverse proxies, cloud-based security groups, or similar mechanisms
- Verify that binary signatures, trusted connections, and verified endpoints are used to deploy binaries to remote devices.
- Verify that the build pipeline warns of out-of-date or insecure components and takes appropriate actions.
- Verify that the build pipeline contains a build step to automatically build and verify the secure deployment of the application, particularly if the application infrastructure is software defined, such as cloud environment build scripts.
- Verify that application deployments adequately sandbox, containerize and/or isolate at the network level to delay and deter attackers from attacking other applications, especially when they are performing sensitive or dangerous actions such as deserialization.

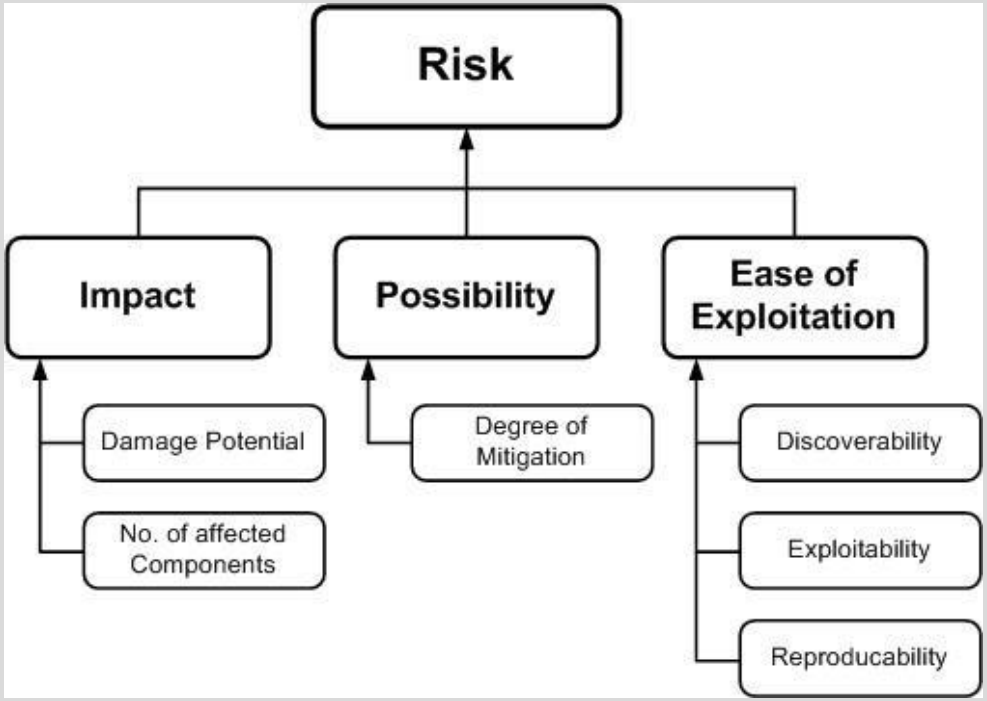
Step 2.4. Threat Analysis

- Threat trees should be prepared for each identified threat
- Identification of vulnerabilities for each threat

Vulnerabilities examples:

- SQL injection
- Weak encryption algorithms
- Cross Site Scripting / HTML injection
- Browser caches sensitive information
- Lack of password complexity enforcement.
- Failure to validate cookie input.

Step 2.5. Ranking of Threats (DREAD)



Step 2.5. Ranking of Threats (DREAD)

- **Damage**: How big would the damage be if the attack succeeded?
- **Reproducibility**: How easy is it to reproduce an attack to work?
- **Exploitability**: How much time, effort, and expertise is needed to exploit the threat?
- **Affected Users**: If a threat were exploited, what percentage of users would be affected?
- **Discoverability**: How easy is it for an attacker to discover this threat?

Damage Potential

If a threat exploit occurs, how much damage will be caused?

0 = Nothing
3 = Individual user data is compromised, affected or availability denied
5 = All individual tenant data is compromised, affected or availability denied
7 = All tenant data is compromised, affected or availability denied
7 = Availability of a specific cloud controller components/service is denied
8 = Availability of all cloud controller components is denied
9 = Underlying cloud management and infrastructure data is compromised or affected
10 = Complete system or data destruction, failure or compromise

Reproducible

How easy is it to reproduce the threat exploit?

0 = Very hard or impossible, even for administrators. The vulnerability is unstable and statistically unlikey to be reliably exploited
5 = One or two steps required, tooling / scripting readily available
10 = Unauthenticated users can trivially and reliably exploit using only a web browser

Exploitability

What is needed to exploit this threat?

0 = N/A We assert that every vulnerability is exploitable, given time and effort. All scores should be 1-10
1 = Even with direct knowledge of the vulnerability we do not see a viable path for exploitation
2 = Advanced techniques required, custom tooling. Only exploitable by authenticated users
5 = Exploit is available/understood, usable with only moderate skill by authenticated users
7 = Exploit is available/understood, usable by non-authenticated users
10 = Trivial - just a web browser

Affected Users

How many users will be affected?

0 = None
2.5 individual/employer that is already compromised.
6 = some users of individual or employer privileges, but not all.
8 = Administrative users
10 = All users

Discoverability

How easy is it to discover this threat?

0 = Very hard to impossible to detect even given access to source code and privilege access to running systems
5 = Can figure it out by guessing or by monitoring network traces
9 = Details of faults like this are already in the public domain and can be easily discovered using a search engine
10 = The information is visible in the web browser address bar or in a form

Rate, Compare and Prioritize Threats

DREAD score:

(Damage + Reproducibility + Exploitability + Affected Users + Discoverability) / 5 = RISK

0-3 as 'Trivial, fix in next release“,
4-7 as “Important, fix as a priority“,
8-10 may be "Critical, fix immediately".

Phase III: Determine countermeasures and mitigation

- **Goal:** Identify the appropriate countermeasure and determine the effective mitigation strategies

Threat	Countermeasures
Spoofing user identity	Use strong authentication. Do not store secrets (for example, passwords) in plaintext. Do not pass credentials in plaintext over the wire. Protect authentication cookies with Secure Sockets Layer (SSL).
Tampering with data	Use data hashing and signing. Use digital signatures. Use strong authorization. Use tamper-resistant protocols across communication links. Secure communication links with protocols that provide message integrity.
Repudiation	Create secure audit trails. Use digital signatures.
Information disclosure	Use strong authorization. Use strong encryption. Secure communication links with protocols that provide message confidentiality. Do not store secrets (for example, passwords) in plaintext.
Denial of service	Use resource and bandwidth throttling techniques. Validate and filter input.
Elevation of privilege	Follow the principle of least privilege and use least privileged service accounts to run processes and access resources.

Step 3.1. Countermeasure Identification

- **Purpose:** Determine the protective measures in order to mitigate each identified threat

- **Countermeasures** are based on the ASVS (Application Security Verification Standard) requirements

- Three different choices are possible:
  - **Non mitigated threats:** Threats which have no countermeasures and represent vulnerabilities that can be fully exploited and cause an impact
  - **Partially mitigated threats:** Threats partially mitigated by one or more countermeasures which represent vulnerabilities that can only partially be exploited and cause a limited impact
  - **Fully mitigated threats:** These threats have appropriate countermeasures in place and do not expose vulnerability and cause impact

Step 3.2. Mitigation Strategies

- The decision of which strategy is most appropriate depends on:
  - the impact an exploitation of a threat can have,
  - the likelihood of its occurrence,
  - and the costs for transferring (i.e. costs for insurance) or avoiding (i.e. costs or losses due redesign) it.

- Decision is based on the risk a threat poses to the application

- **Possible mitigation Strategies**
  - **Do nothing**: no action
  - **Inform about the risk**: for example, warning user population about the risk
  - **Mitigate the risk**: for example, by putting countermeasures in place
  - **Accept the risk**: for example, after evaluating the impact of the exploitation (business impact)
  - **Transfer the risk**: for example, through contractual agreements and insurance
  - **Terminate the risk**: for example, shutdown, turn-off, unplug or decommission the asset

Useful tools

- SDL ThreatModeling:
- http://www.microsoft.com/en-us/download/details.aspx?id=2955

- Microsoft Threat Modeling Tool 2014:
- http://www.microsoft.com/en-us/download/details.aspx?id=42518

References

- https://www.owasp.org/index.php/Application_Threat_Modeling
- http://msdn.microsoft.com/en-us/library/ff649779.aspx
- http://msdn.microsoft.com/en-us/library/ff648866.aspx
- http://msdn.microsoft.com/en-us/library/ff647894.aspx
- https://www.sans.org/reading-room/whitepapers/securecode/threat-modeling-process-ensure-application-security-1646
- https://www.owasp.org/images/5/58/OWASP_ASVS_Version_2.pdf

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ
Τμήμα Πληροφορικής

ΣΧΕΔΙΑΣΗ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΑΣΦΑΛΕΙΑΣ ΠΡΟΗΓΜΕΝΑ ΣΥΣΤΗΜΑΤΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Thankyou