A Hybrid Approach to Threat Modelling

Author: Sriram Krishnan, sriram.krishnan@in.pega.com

Date: 25-Febraury-2017

Abstract

Threat Modelling is considered the fundamental approach in identifying security weakness in software applications during the design phase in Software Development Lifecycle process. Various techniques have been published for performing threat modelling including STRIDE, Attack Tree, and Attack Library. Organizations tend to lean towards a single technique to perform their modelling exercise. Each of these techniques is weighed down by limitations, hence when implemented individually impacts the effectiveness and comprehensiveness of the exercise. However, in order to achieve meaningful output it is imperative to use each of these techniques appropriately to the corresponding activity in the threat modelling exercise. This paper analyses the various limitations in each of these techniques and presents a hybrid model that eliminates these limitations by adopting a structured approach, capturing optimum details, and representing the data in an intelligible way.

1. Introduction

The current business world is Information Centric. Information is critical, information is money and information is at large. All the more, it is transacted over the internet. It requires no additional motivation for any group to obtain the information to monetise or use for other purposes that would benefit them. It goes without saying that it is the inherent responsibility of an organization to provide a secure operating environment for their customers and employees to protect their interest. At this state it is an eternal obligation of any organization to look at security just not as a function but a key business driver.

Threat modelling is the fundamental building block for building secure software. Unless one understands the threats that they are exposed to in a structured way, it will not be possible to build a secure operative environment and software. It is needless to say that threats grow along with evolution of technology and delivery models. A SANS survey (2015 State of Application Security: Closing the Gap [1]) indicates that threat assessment (which can also be referred to as threat modelling) is the second leading application security practise (next to penetration testing) for building secure web applications. Thus threat modelling is a pro-active security practice that organizations should adopt.

2. Threat Modelling

2.1 An Overview

Let us assume that an organization is tasked to build a payment processing application. This application is required to integrate with external applications, transmit and store sensitive data such as customer's SSN, credit card and debit card details. One of the top priorities for the organization is to bake security into the application. Some of the questions that need to be addressed by the security office upfront are:

- Where to start?
- How to identify the threats?
- What are the targets of an attacker?
- Who could attack and how can they attack?

- How should the organization defend?

When any organization plans to build a new application, add features to an existing application or change their delivery model (move to cloud), their primary objective should be to build a secure design for the application. Threat Modelling is a technique that helps achieve this objective in a simplified and structured way.

Following diagram depicts in simple terms what threat modelling is:

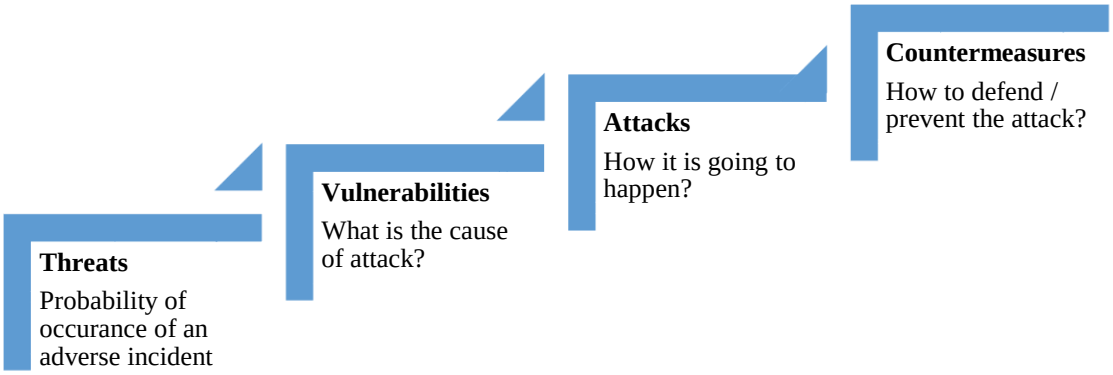


Figure 1 – Threat Modelling

In a threat modelling exercise we enumerate the threats, which could be caused by a vulnerability, which could be realized through an attack, which could be mitigated via countermeasure.

2.2 Types of Threat Modelling

Threat modelling is a structured procedure for identifying and categorizing threats, and enumerating threat scenarios require in-depth understanding of the architecture and underlying technology stack.

At a broad level there are 3 types of threat modelling techniques:

- STRIDE – Spoofing, Tampering, Repudiation, Information Disclosure, Denial-of-Service, and Elevation of Privilege
- Attack Trees
- Attack Libraries

2.2.1 STRIDE

The STRIDE approach to threat modelling was invented by Loren Kohnfelder and Praerit Garg in 1999 [2]. This technique helps in the enumeration of threats based on attack properties. For each of these attack properties there is set of security themes violated as illustrated in the following table:

Attack Property	Security Theme
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial-of-Service	Availability
Elevation of Privilege	Authorization

Table 1 – STRIDE Attack Properties

Threats are enumerated by considering each attack property and its corresponding impacted security theme. Consider a Software as a Service (SaaS) application that does pay-roll processing for various organizations. The application transmits and stores sensitive details such as employee's salary data. The system also integrates with its customer's network for authentication and obtaining Human Resource Management System (HRMS) data for payment processing. Below table provides an example of threat analysis for this application using STRIDE:

Attack Property	Threat Scenarios	Security Theme
Spoofing	An attacker can hijack a session id of a user and submit request to obtain the user's payroll details.	Authentication
Tampering	An attacker can intercept and modify salary data as it is transmitted via SSL v2 which uses weak algorithms for encryption.	Integrity
Repudiation	An attacker who also happens to be an employee can modify their own salary details by capturing and replaying the original request submitted by the organization.	Non-Repudiation

Information Disclosure	An attacker can obtain salary data upon access to database as they are stored as plain text in the database.	Confidentiality
Denial-of-Service	An attacker programmatically sends a large number of HTTP GET / POST requests designed to consume significant amount of the server resources and result in denial-of-service condition.	Availability
Elevation of Privilege	Application is vulnerable to insecure direct object reference which allows an attacker to manipulate the parameter value that directly refers to resources that can only be accessed by accounts with administrative privilege.	Authorization

Table 2 – Threat Enumeration using STRIDE

Please note that the above table provides an example of how to use STRIDE to enumerate threats but does not contain exhaustive set of threats given the context. Each attack property is mapped to the application functionality to identify the threats. In a real-world scenario the entire ecosystem of application and its technology stack should also be taken into consideration during threat enumeration.

STRIDE approach has two variants: STRIDE-per-Element and STRIDE-per-Interaction.

STRIDE-per-Element [3] [4]**:** STRIDE-per-element strives to achieve defence in depth. Table 2 provides a simplistic way to represent threat scenarios based on the attack properties. However, development teams typically sketch architecture diagrams or Data Flow Diagrams (DFD) based on the nature of the application being developed. These diagrams (when developed in an appropriate manner) would serve as snapshot of the application's ecosystem, indicating the different elements that interact with the application. Elements in a DFD are as follows:

External Entity: Represents the source from which the data is sent to the application or destination to which the application sends the data.

Data Flow: Represents the data flow from and to the application.

Data Store: Represents the data at rest (database).

Process / Business Logic: Represents an action, activity or logic that transforms or operates on the data.

This technique should be used when the team is motivated to find additional or intricate threats (on top of already identified threats) specific to these elements. This method could be perceived as one which is tightly-coupled with such elements. An organization should identify elements that need further threat analysis. Continuing with the payroll application example, consider the following DFD:

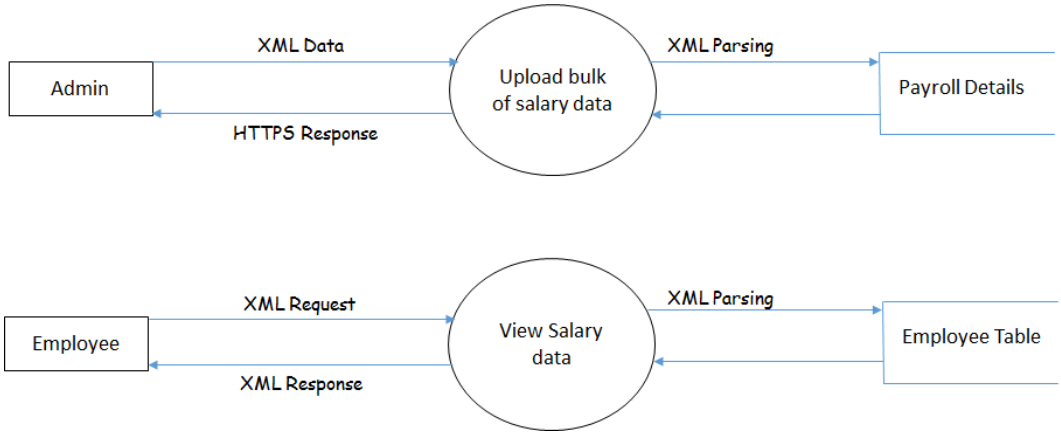


Figure 2 – DFD Diagram Payroll Application

The above diagram illustrates two basic functions of the application:

- Admin uploads bulk of employee salary data
- Employee requests to view his salary details

From the DFD it is evident that confidential data is transacted between user (admin and employee) and application, and that XML is used for transporting the data. Also, the application parses XML data before storing the values in the payroll details table. Another key point to note is that XML can present an attractive target for adversaries as it has been widely used and susceptible to different types of attack. Thus data flow becomes one of the elements that need additional focus. To break down, let us consider the following two points to decide whether an element represented in the DFD requires additional focus:

- Criticality of the function from the perspective of confidentiality, integrity, and availability (business risk should also be taken into consideration)

- Degree of weakness or strength and the options available in the underlying technology to protect itself from security attacks

Below table represents threat enumeration by adopting STRIDE-per-element:

Elements	Threat Scenario	S	T	R	I	D	E
Data Flow	An attacker will include nested entity expansion to produce amplified XML output that will crash processor's CPU/memory (***XML Entity Expansion Attack***)					×	
Data Flow	An attacked will include an URI in the entity that will contain malicious code to read sensitive data from the application server (***XML External Entity Injection Attack***)				×		
Data Flow	An attacker is able to forge or alter the data by inserting malicious element in the XML (***XML Signature Wrapping Attack***)			×			
Data Flow	An attacker is able to forge or alter XML data by removing the Signature element (***XML Signature Exclusion Attack***)	×					×

Table 3 – STRIDE-per-Element

Since the focus is related to data flow via XML, various threats relating to XML are enumerated so that defence in depth can be achieved.

STRIDE-per-Interaction [5]: STRIDE-per-Interaction is the other variant of STRIDE that attempts to further simplify threat modelling. This does not mean that STRIDE-per-Element is complicated or STRIDE-per-Interaction will allow threat modellers to identify more threats. All techniques can be used in conjunction to arrive at the best possible result. The Approach to Threat Modelling section will cover this aspect of threat modelling.

A dictionary meaning of Interaction is a mutual or reciprocal action. In the context of threat modelling, this represents the flow of data between entities involved in the application. Understanding each interaction and application flow provides much needed insight for threat modellers to enumerate threats applicable to that interaction. Thus addressing threats for each application workflow serves as an enabler in developing a more secure application. Such an approach will deeply benefit the

software developers as the threats will be documented as scenarios specific to that interaction. Continuing with the same example, let us further simplify the threat model:

Elements	Interaction	Threat Scenario	S	T	R	I	D	E
Data Flow	Admin -> Application Interface for bulk upload	An attacker can send malicious XML document that would contain nested entities expansion to produce amplified XML output that will crash processor's CPU/memory (***XML Entity Expansion Attack***), as the application allows the use of DTD.					×	
Data Flow	Employee -> Application interface to view employee details	An attacker can exclude the XML signature from the message to send arbitrary request and invoke functions for obtaining salary details of the employees.	×					×

Table 4 – STRIDE-per-Interaction

In this example, a column named “Interaction” is added to the threat modelling table. This enables threat modellers to document the specific interaction of the overall functional flow and facilitates representation of the threats in much more eloquent way. It is possible to capture “what could go wrong” in that specific interaction and arrive at effective countermeasures.

Another advantage of this technique is it permits grouping of threats for which the countermeasures are similar (many to one) – this is immensely beneficial for the developer community. For example, the countermeasures prescribed for XML Entity Expansion Attack is also applicable for XML Entity Injection Attack. This attack occurs when the XML message contains a reference to an external entity which can execute an arbitrary function to obtain confidential data from the server. Therefore, the XML processor should be configured to use only a local static DTD and disallow any declared DTD in the XML document. Grouping of threats with similar countermeasures helps developers to plan and prioritize the efforts during their implementation. This simplifies the whole exercise both from threat enumeration and mitigation standpoints.

Limitations of STRIDE: The STRIDE technique helps enumerate threats relating to the elements and functional flow of an application; however the essence of 'how to mitigate' the threats cannot be addressed. To achieve completeness, threat modelling must identify and design strong countermeasures for the identified threats. Taking this a step further, documenting the appropriate implementation countermeasures (secure coding guidelines) is not a feature of this technique.

2.2.2 Attack Tree

Attack tree is a conceptual representation of possible attacks against an application through which threats are ascertained. According to Bruce Schneier, "Attack trees provide a formal, methodical way of describing the security of systems, based on varying attacks. Basically, the attacks against a system are represented in a tree structure, with the goal as the root node and different ways of achieving that goal as leaf nodes" [6] [7]. It is a model that enables threat analysis from an attacker's perspective.

In the attack tree, the ultimate goal of the attack is the root node, while the children and leaf nodes represent the sub goals. In the tree, nodes can be either represented as "AND" or "OR" nodes. Let us consider the following diagram:

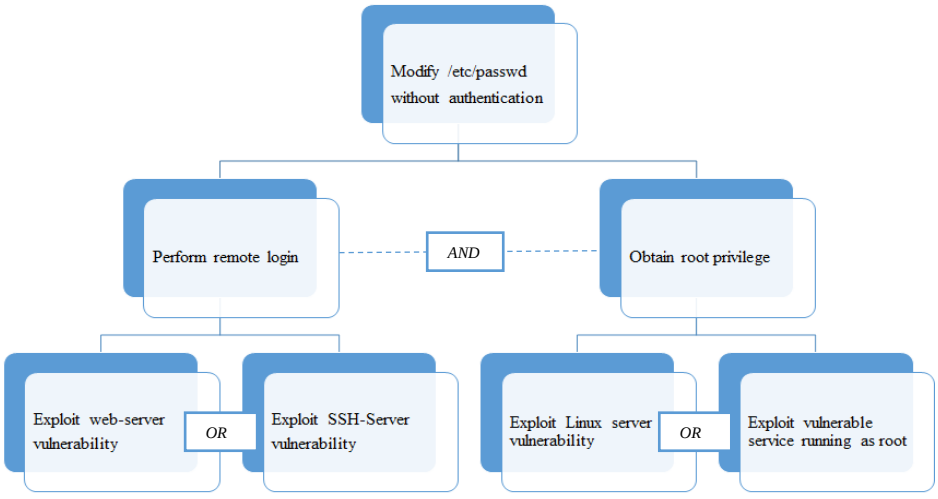


Figure 3 – Attack Tree

- "AND" represents different or multiple steps required to achieve the goal. In figure 3 the goal is to modify the /etc/passwd file. Without legitimate access to the server

an attacker must gain access to the server (remote login) AND escalate to root privilege.

- "OR" represents different ways to achieve the goal. In figure 3, an attacker can either exploit the vulnerability of the webserver OR SSH service to establish remote login or access to the server. Similarly, to escalate to root privilege, an attacker can either exploit linux-kernal vulnerability OR a vulnerable service that runs as root.

Once the possible attacks against an application have been modelled in a tree structure, one can assign attributes to those attacks. Following are the attributes that can be applied:

- Probability of an attack with a standard categorization - for example High, Medium, Low
- Cost of an attack considering the tools / software required to perform the attack
- Competency required to perform the attack – for example, script kiddie, basic working knowledge, expert or specialist
- Impact to business – for example, reputation damage, financial loss, non-compliance to regulatory requirements and privacy violation

Other than considering the attacker's goal, an attack tree can be built based on:

- Attack patterns such as injection attacks, Man-in-the-Middle attacks, Denial of Service, and Advanced Persistent Attack (Malware).
- Attacks specific to protocols - application security researcher Ivan Ristic performed an interesting threat model exercise focusing on the SSL protocol [8].

Building An Attack Tree: Following are the 3 simple steps to construct an attack tree:

STEP-1: Identify the goals – each goal can be a separate attack tree, in case of large attack vector even sub-goals can be represented in separate tree structures.

STEP-2: Identify the various categories of attacks required to accomplish the goals.

STEP-3: If a generic attack tree library exists, it can be plugged into the attack tree being constructed.

Consider a scenario where attacks against administrative credentials and functions have to be explored. This being the overall goal, it can be further broken down into two separate sub-goals:

- To obtain admin credentials
- Perform administrative function without appropriate authorization

Following representations depict the attack tree for these sub-goals:

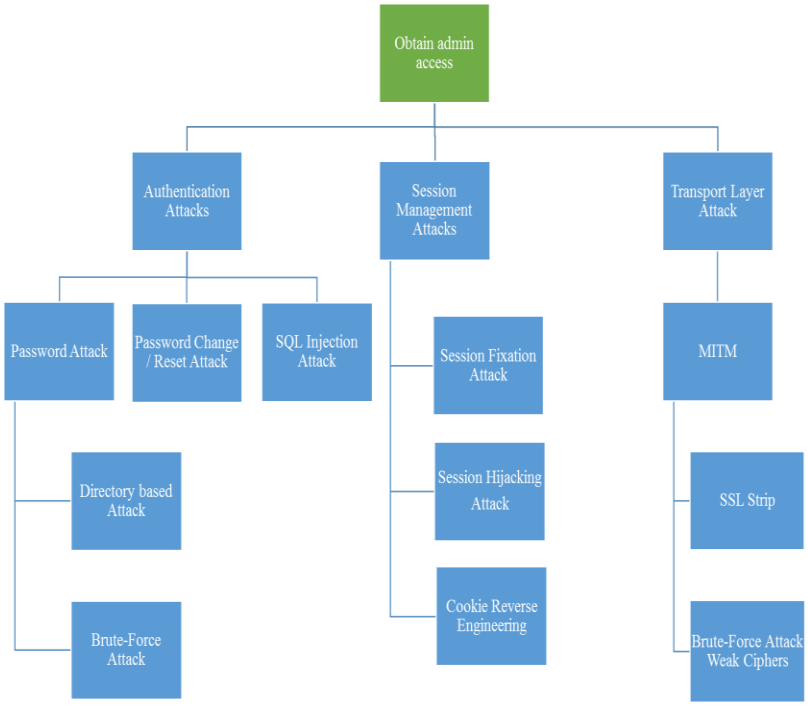


Figure 4 – Attack Tree to obtain Admin credentials

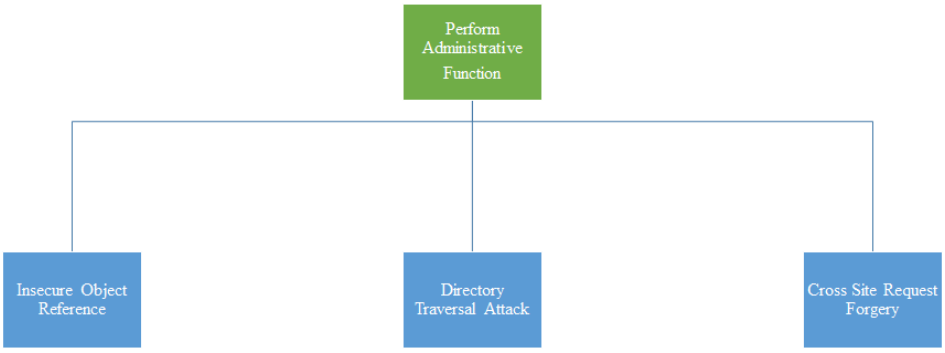


Figure 5 – Attack Tree to obtain Administrative Functions

Limitations of Attack Tree: Upon identification of the attacks to accomplish the goal, attributes of the attacks have to be considered. As mentioned earlier, details such as probability of the attack, cost of the attack, and countermeasures have to be documented to achieve the completeness of the exercise. However, it is impossible to capture all those details, and adding such information will complicate the tree structure at the expense of readability. Instead these attacks have to be assigned a reference number and their attributes must be documented against their corresponding attack in a separate table. Moreover, this model is suited to providing a high-level representation of the attacks, but does not suit modelling of threats at a more granular level. The attack tree when used independently as a threat modelling technique does not yield the best result.

2.2.3 Attack Library

Attack Library is a collection of attacks for finding threats against the application being developed [9]. This is another type of threat modelling technique available to identify threats by looking from an attacker's perspective. The idea is to provide as much details as possible for an attack type (for example code injection) to help threat modellers or the developer community to understand the landscape of threats. Any threat modelling technique adopting the attacker's perspective is more of a checklist model, i.e. traverse the library of attacks applicable in the context of the application, analyse whether the threats are handled, and identify countermeasures.

An attack library can be constructed based on the following:

- List of possible attacks against the application
- List of potential vulnerabilities in the application
- List of software weakness (programming errors) introduced in the application

Before proceeding, let's review the difference between a software vulnerability and a weakness. Software weaknesses can be viewed as programming errors that may lead to potential vulnerabilities in the application, while a software vulnerability is a flaw in the application that can be used by an attacker to gain access or deny service. Hence both of these aspects should be considered in the attack library.

Organizations can either develop their own library or leverage formal lists or dictionaries published by the security community or consortium such as Open Web Application Security Project (OWASP), Common Weakness and Enumeration (CWE), and Common Attack Pattern Enumeration and Classification (CAPEC). Let us look at each of these libraries:

OWASP [10]: Open Web Application Security Project is an open community dedicated to enabling organizations to conceive, develop, acquire, operate, and maintain applications that can be trusted. They create best practices methodologies, documentation, articles, and tools to ensure development of secure applications. Periodically they release a list of the top ten web application vulnerabilities to educate developers and security professionals about the most important web application security weakness and its consequences. For each of these vulnerabilities, OWASP provides detailed threat agents, attack vectors, security weaknesses, technical impact, business impact, countermeasures, and examples of attack scenarios. OWASP proposed security practices and methodologies are widely used for web application projects, hence it is be suitable to an exercise such as threat modelling.

CAPEC [11]: It is a publicly available, community-developed list of common attack patterns along with a comprehensive schema and classification taxonomy. Attack patterns are descriptions of common methods for exploiting software systems. It describes how an adversary can attack the vulnerable system and common techniques used to tackle these challenges. These patterns also help categorise the attacks and teach the development community to better understand and effectively defend against the attacks. CAPEC are well structured with a total of 504 attack patterns grouped into 16 mechanism of attacks (as per version 2.8). This is available in this link https://capec.mitre.org/data/definitions/1000.html

CAPEC provides exhaustive details about the attacks : summary of attack, attack execution flow, attack prerequisites, typical severity, typical likelihood of exploit, methods of attack, examples – instances, attacker skills or knowledge required, resources required, probing technique, indicators-warnings of attack, solution and mitigation, attack motivation – consequences, injection vector, payload, activation

zone, payload activation impact, security requirements, CIA impact, and technical context. Although this list is rich with information, how much of these details are necessary for performing an effective threat modelling exercise?

Any organization or team new to threat modelling can be overwhelmed with information and miss out on important aspects to consider. Even experienced threat modellers may find this method more time consuming and thus adversely affecting productivity.

CWE [12]: It is a formal list or dictionary of common software weaknesses that can occur in software architecture, design, code or implementation that may lead to exploitable security vulnerabilities. In common terms, CWE has identified critical programming errors that may lead to software vulnerabilities. CWE serves as a standard measuring parameter for software security tools targeting these weaknesses. The purpose is to provide a common baseline standard for weakness identification, mitigation, and prevention efforts. As per version 2.9 there are in total 1004 CWEs which can be grouped based on various criteria. Each of the primary clusters have secondary clusters. Like CAPEC, the primary cluster is to categorize software weakness for better understanding. Details provided in each CWE include: description, applicable platform, common consequences, demonstrative example, observed examples, and related attack patterns.

Procedure of using Attack Library technique in a threat modelling exercise is simple:

STEP 1: Build an attack library or identify an existing library that can provide information about attack patterns.
STEP 2: Identify the area (based on the development scope or project) for which threat modelling exercise is applicable.
STEP 3: Review the application that was developed against each of the attack / vulnerability / weakness in the attack library
STEP 4: Document the potential threats and countermeasures.

Let us consider a scenario where a threat modelling exercise using Attack Library technique is performed for an application. Key focus areas relating to web application security, at a broad-level, can be categorized into the following:

- authentication
- authorization
- cryptography
- session management
- data validation
- secure configuration
- logging
- error handling

Each of the above categories is considered in relation to the relevant references in the attack library. The following represents the references in the library for Authentication:

Weakness	Vulnerability	Attack
○ CWE-287: Improper Authentication ○ CWE-288: Authentication Bypass Using an Alternate Path or Channel ○ CWE-289: Authentication Bypass by Alternate ○ CWE-290: Authentication Bypass by Spoofing ○ CWE-307: Improper Restriction of Excessive Authentication Attempts ○ CWE-308: Use of Single-factor Authentication	○ A2-Broken Authentication and Session Management	○ CAPEC-114: Authentication Abuse ○ CAPEC-115: Authentication Bypass ○ CAPEC-49: Password Brute Forcing ○ CAPEC-302: Authentication Bypass by Assumed-Immutable Data ○ CWE-305: Authentication Bypass by Primary Weakness ○ CWE-308: Use of Single-factor Authentication

By creating a library for each of the application security schemes, it is possible to iterate each of these entries against the application design and identify issues. The coverage and depth of the attack library should depend on the context and criticality of the application. For example, a multi-factor authentication mechanism should be considered for an internet banking application, but the same may not be required in an online accommodation / travel application.

Limitations of Attack Library: There are some significant limitations in this technique:

- Suitable as a check-list model, hence cannot be applied during the design phase.

- Information rich and needs referencing which may lead to time consuming and a tedious exercise (trade-off between exhaustive details vs productivity).
- Countermeasures suggested in the attack library (OWASP, CAPEC, CWE) are abstract or at high-level, and may not align to programming language or framework used in the organization.
- This technique helps to think from attacker's perspective, but poses a challenge to structure a way to address the defence against these attacks.

2.3 Hybrid Model – An Effective Approach to Threat Modelling

As discussed in the previous sections, there are limitations when each of these threat modelling techniques are implemented independently. When implemented as separate techniques, some of the key aspects required for threat modelling may be missed, thus impacting the productivity and comprehensiveness of the exercise. Therefore the optimal approach is to adopt a hybrid model drawing from the best of each of these techniques.

To perform a productive and comprehensive threat modelling, include the following three aspects:

- Structured approach
- Optimum detail
- Readability

Structured Approach: A successful process must be schematised and should adopt a procedure. It enables establishment of the objective, capture of appropriate details, and implementation of the agreed procedure to achieve the set objective. For example, a software development team adopts a suitable software development lifecycle model such as agile or waterfall so that objectives are clearly understood and translated into desired software products in a timely manner. Similarly, threat modelling should embrace a structured approach so that desired outcomes can be achieved from the exercise.

Optimum Detail: Providing information that can be easily interpreted and acted upon serves as a critical factor to the successful outcome of a process. The consumers

of this exercise are mostly software developers / architects / testers who might not necessarily be security experts. Hence providing either excessive information or minimal details will not only impact the result of the exercise but also the security of the software application. Therefore publishing the optimum amount of detail will immensely contribute to the successful outcome of the exercise.

Readability: It is not adequate if the exercise has a structured approach and contains optimum details alone. Representation of data in the best possible way guarantees the completeness of the exercise. In software development, the best way to represent complex information or work flow within an application is via a data flow diagram (DFD). A flow chart may not suit this scenario. Likewise in threat modelling the data should be presented in a readable format, contributing to the overall simplification of the exercise. The threat modelling process can be represented as follows:

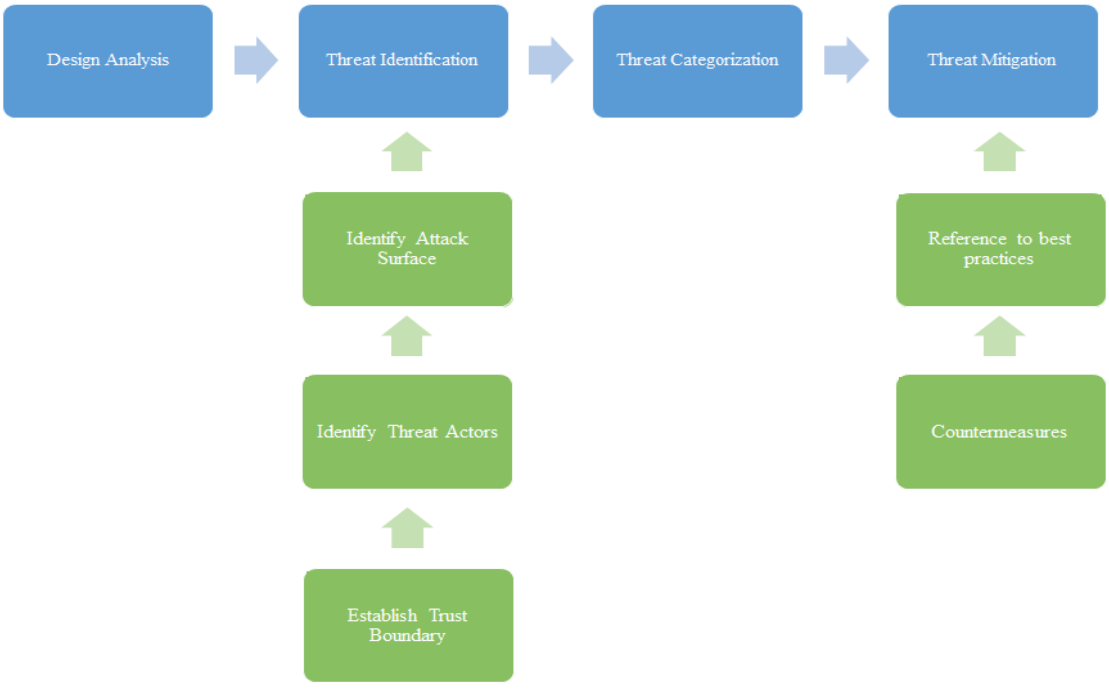


Figure 6 – Threat Modelling Process

- Design Analysis: The first step is the study of DFD or architecture diagrams to obtain knowledge about the data flow in the application component. If required the

areas that may require further consideration from the application security context should be marked.

- Threat Identification: To identify the threat or what could go wrong, the following elements are considered:
  - *Establish Trust Boundary*: Trust boundary is a line beyond which the web application will not have control over the data. This indicates that any data sent from elements beyond this line should always be validated before being processed. The objective of establishing a trust boundary is to define the potential threat actors i.e. users who can launch an attack against the application. In a diagram, the trust boundary is often depicted as dotted lines representing trust for components within the lines. Any component beyond this boundary should be treated as unknown and should be scrutinized.

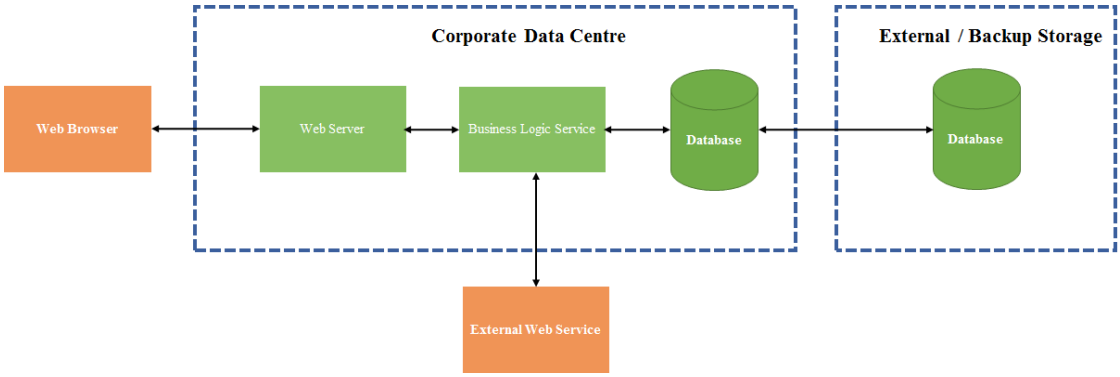


Figure 7 – Trust Boundary

  - *Identity Threat Actors:* Once a trust boundary is established the threat modellers will know the entities whose request or input should always be validated. Anyone who poses a threat to the web application can be classified as a threat actor. This can include legitimate users of the application or adversaries (not having approved access to the web application) on the internet trying to attack the web application. While analysing complex enterprise applications that contain multiples user roles with varying privileges, it is often essential to create a list of threat actors. Defining these threat actors is completely based on the criticality and business context of that web application.
  - *Identity Attack Surface:* Attackers may launch their payload from various entry points in the application and ultimately impact the business. Threat actors define

"who will attack" whereas attack surface indicates "from where will the attack originate". While examining the DFD, the points where data transacted from untrusted sources are identified and marked as potential entry points. For example, a web application authentication page where users submit credentials (username and password via input fields) to access protected resources. This can be an entry point for the attacker to conduct a brute-force attack or SQL injection attack via these input fields. Identifying the attack surface completes the process of threat enumeration, as one would have adequate information on the trust boundary, potential adversaries, and the point from where the attack will be launched.

- Threat Categorization: The objective of threat categorization is to frame the countermeasures for the given scenario. There are bound to be challenges while categorizing, especially when threats overlap multiple categories. In such situations, mark the threats in the applicable categories, but use the primary attack objective to help identify the countermeasures. For example, consider a threat scenario where a session id is not well protected (uses weak hashing algorithms). An adversary attacks and obtains a valid session id, while either at rest (cached in web browser) or in transit, and then uses it to impersonate a legitimate user. A typical dilemma may be to categorize it as either information disclosure, spoofing, or both. In this case, the objective of the attack is to obtain the exposed session id while the result of the compromise may lead to spoofing. Because the attack objective relates to information disclosure, categorized the threat as information disclosure and document the associated countermeasure: create session ids using strong cryptographic algorithms and transmit values over secure communication channels.
- Threat Mitigation: The following points have to be addressed to arrive at an effective threat mitigation plan:
    - *Countermeasures*: This is a set of actions taken to defend the attack for the given threat scenario. In this section, threat modellers should provide both conceptual and technical details to secure the web application. For example, the countermeasure for SQL Injection Attack: using of parameterized queries, stored procedures, and whitelist input validation must be explained conceptually for the software developers to understand the approach for defending against such an attack.

- *Reference to Best Practices*: A guide for implementing the identified countermeasures should be presented as references to well-known standards such as OWASP and CWE or secure coding notes built in an organization that contain a summary about the attack and compliant code snippets, in the programming language used in the organization.

The following representation depicts mapping of the appropriate threat modelling technique to the threat modelling activity:

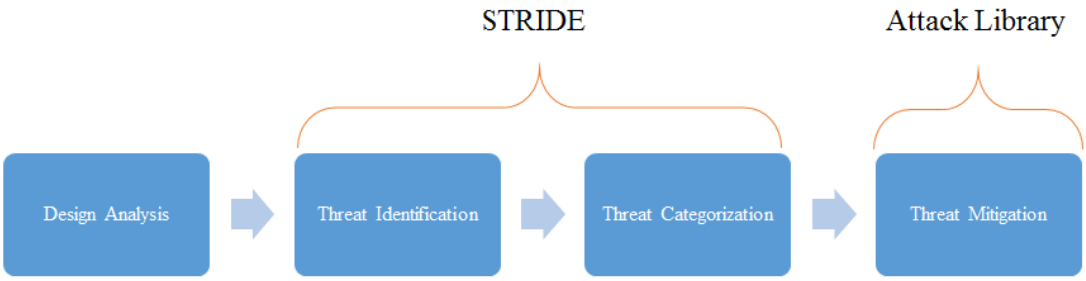


Figure 8 – Threat Modelling Hybrid Model

As indicated in the above diagram, the STRIDE technique should be used for threat identification and threat categorization, and attack library for threat mitigation. Attack tree should be used to provide an abstract view about attacks against a particular feature or component of an application. The following points provide the rationale behind this mapping:

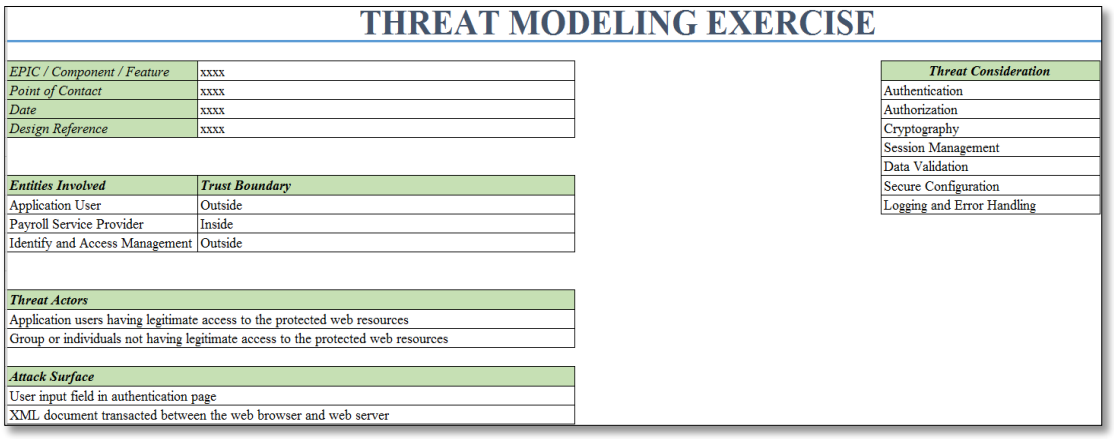
- STRIDE (per-element and per-interaction) will provide the required context, in terms of the elements involved and specific interaction or flow for a feature in the application, enabling threat modellers to cogitate upon the potential threats and develop threat scenarios.
- Threats are then categorized by considering each of the attack properties in STRIDE and marking the most appropriate one for the given threat scenario.
- Threat mitigation will be formulated by analysing the possible attacks for the threat scenario from rich information contained in the Attack library / reference.

The threat modelling exercise has to be documented so that software developers use the data while developing the application and also for future reference. Hence it is imperative to develop a suitable template to gather and represent the data by adopting

the hybrid model covering the key aspects of a structured approach, optimum details and readability. The template is divided into two parts:

- For capturing data for scoping
- For capturing data relating to activities in threat modelling

Template for capturing the scoping details:



THREAT MODELING EXERCISE

EPIC / Component / Feature	xxxx
Point of Contact	xxxx
Date	xxxx
Design Reference	xxxx

Entities Involved	***Trust Boundary***
Application User	Outside
Payroll Service Provider	Inside
Identify and Access Management	Outside

Threat Actors
Application users having legitimate access to the protected web resources
Group or individuals not having legitimate access to the protected web resources

Attack Surface
User input field in authentication page
XML document transacted between the web browser and web server

Threat Consideration
Authentication
Authorization
Cryptography
Session Management
Data Validation
Secure Configuration
Logging and Error Handling

Figure 9 – Threat Modelling Template for scoping

Table-1: The first four rows capture the summary about the specific area selected for threat modelling, details about the team and references to the design that will be analysed.

Table-2: Various users of the application and even external web services or systems interacting with the application are mentioned along with their scope to the trust boundary.

Table-3: Threat actors: potential threat agents or entities who can harm the application are documented.

Table-4: Attack surface: different entry points by which the threat actors can launch their payload and gain illegitimate access to the application.

Table-5: Threat consideration: threat modellers can highlight the schemes that are considered as part of this threat modelling exercise, as it provides a nice overview and understanding about areas covered in this exercise.

Template to capture the data pertaining to threat modelling:

										Attack Library		
Ref No.	Elements	Interaction	Threat Scenario	S	T	R	I	D	E	Countermeasures	CWE	CAPEC	OWASP Top Ten
TM-01	External Entity	*User Login Request -> Auth Service*	Attacker can inject malicous input strings so that the target application constructs SQL statements that allows the attacker to: 1) bypass the authentication and get access to the application 2) enumerate the databse schema or user details from the database		x		x			CM-1: Use of parameterized queries or stored procedures CM-2: White list input validation CM-3: Use of custom error page	CWE-89: Improper Neutralization of Special Elements used in an SQL Command	CAPEC-66: SQL Injection	A1-Injection
TM-02	Data Flow	Admin -> Application Interface for bulk upload	An attacker can send malicious XML document that would contain nested entities expansion to produce amplified XML output that will crash processor's CPU/memory (XML Entity Expansion Attack), as the application allows the use of DTD.					x		CM-4: Configure the XML parser to use local static DTD, and disallow any declared DTD included in the XML document. CM-5: Ensure XML document is validated for inclusion of any declared DTD before parsing the same.	CWE-776: Improper Restriction of Recursive Entity References in DTDs	CAPEC-197: XML Entity Expansion	

Col-1: Ref No: a nomenclature to uniquely identify a threat scenario. This can be subsequently mentioned in various stages in the SDLC process to ensure the threat has been adequately handled.

Col-2: Elements: indicates the element (external entity, data flow, data store, and business logic) to which the threat is mapped. This aids consideration of the threats specific to those elements.

Col-3: Interaction: The specific flow in the application to which the threat scenario will be addressed.

Col-4: Threat scenario: captures "what could go wrong" considering the element involved and specific flow in the application.

Col-5 to 10: STRIDE: these columns are used to categorize the identified threat. As mentioned earlier, the threat can be categorised in more than one attack property which will help in determining precise countermeasures.

Col-11: Countermeasures: conceptual details about the security controls that needs to be implemented to tackle the identified threat scenario. The countermeasures need to be effective in order to ensure that the threats are not realized.

Col-12 to 14: Attack library: illustrates the best practices by providing a description of the possible attacks, and secure coding practises to prevent the same. This serves as a guideline for software developers while implementing the countermeasure for the threat scenario. It is in the best interest of the organization to develop their own attack library or secure coding guidelines as it will be aligned to their technology stack (programming language and development frameworks). Solutions provided in CAPEC or CWE may be abstract and not in the programming language adopted by the organization. However the intention is to provide a direction for the software developers on secure coding practices which will result in developing secure applications.

3. Conclusion

In the face of increasing attacks at the application layer and enterprise applications moving towards the cloud, security is viewed as a key business requirement. Understanding the threat landscape is a perquisite for building secure applications. Threat modelling helps organizations realise their own threat landscape and detect and mitigate flaws early in the development process.

There are various techniques published for conducting a threat modelling exercise. The most common techniques such as STRIDE, Attack Tree, and Attack Library are discussed in this paper. Organizations tend to be prejudiced towards a particular technique while performing the exercise. However this approach will be detrimental to the objective of performing an effective and comprehensive exercise. The STRIDE technique may be good in enumerating the threats however does not aid in developing countermeasures / mitigation plan. Attack Tree provides an overview about the attack surface at some level of abstraction which results in not capturing data essential for understanding the threat scenario. Finally Attack Library may provide information about the attack vectors and be suitable as checklist model, it may not contribute to the completeness we expect in the exercise.

To reap the complete benefit from the exercise we have to utilize a combination of each of these techniques to perform the various activities in the threat modelling process. Critical success factors for the threat modelling exercise lies in adopting a structured approach, providing optimum details and readability. Hence this paper proposes a hybrid model that would implement the techniques that best suit each of the activities in threat modelling.

Acknowledgments

I wish to thank Mr. Marty Solomon - Senior Director Software Security Architecture and Mr. Tom Connor – Director Software Engineering, Pegasystems Inc. for providing their valuable feedback and being of support.

References

[1] Bird, J., Johnson, E., & Kim, F. (2015, May). *2015 State of Application Security: Closing the Gap*. Retrieved from https://www.sans.org/reading-room/whitepapers/analyst/2015-state-application-security-closing-gap-35942

[2] Shostack, A. (2014). In *Threat Modeling Designing for Security* (p. 61).

[3] Shostack, A. (2014). In *Threat Modeling Designing for Security* (p. 78).

[4] Osterman, L. (2007, September 10). *Threat Modeling Again, STRIDE per Element*. Retrieved from https://blogs.msdn.microsoft.com/larryosterman/2007/09/10/threat-modeling-again-stride-per-element/

[5] Shostack, A. (2014). In *Threat Modeling Designing for Security* (p. 80).

[6] Shostack, A. (2014). In *Threat Modeling Designing for Security* (p. 87).

[7] Schneier, B. (1999, December). *Attack Trees*. Retrieved from https://www.schneier.com/academic/archives/1999/12/attack_trees.html

[8] Ristic, I. (2009, September 09). *SSL Threat Model*. Retrieved from https://blog.ivanristic.com/2009/09/ssl-threat-model.html

[9] Shostack, A. (2014). In *Threat Modeling Desiging for Security* (p. 101).

[10] *OWASP Top 10 Vulnerabilities*. (2013). Retrieved from https://www.owasp.org/index.php/Top_10_2013-Top_10

[11] *CAPEC: Mechanisms of Attack*. (2015, December). Retrieved from https://capec.mitre.org/data/definitions/1000.html

[12] *CWE: Software Fault Pattern*. (December). Retrieved from 2015: https://cwe.mitre.org/data/graphs/888.html