2018 15th International ISC (Iranian Society of Cryptology) Conference on Information Security and Cryptology (ISCISC)
August 28-29, 2018; Shahid Rajaee Teacher Training University - Tehran, Iran

Threat Extraction Method Based on UML Software Description

Masoumeh Zeinali
Academic Complex of Electrical and Computer Engineering
Malek-Ashtar University of Technology
Tehran, Iran
Zeinali@mut.ac.ir

Mohammad Ali Hadavi
Academic Complex of Electrical and Computer Engineering
Malek-Ashtar University of Technology
Tehran, Iran
Hadavi@mut.ac.ir

***Abstract*—Threat modeling is one of the best practices to secure software development. A primary challenge for using this practice is how to extract threats. Existing threat extraction methods to this purpose are mainly based on penetration tests or vulnerability databases. This imposes a non-automated time-consuming process, which fully relies on the human knowledge and expertise. In this paper, a method is presented, which can extract the threats to a software system based on the existing description of the software behavior. We elaborately describe software behavior with sequence diagrams enriched by security relevant attributes. To enrich a sequence diagram, some attributes and their associated values are added to the diagram elements and the communication between them. We have also developed a threat knowledge base from reliable sources such as CWE and CAPEC lists. Every threat in the knowledge base is described according to its occurrence conditions in the software. To extract threats of a software system, the enriched sequence diagrams describing the software behavior are matched with the threat rules in our knowledge base using a simple inference process. Results in a set of potential threats for the software system. The proposed method is applied on a software application to extract its threats. Our case study indicates the effectiveness of the proposed method compared to other existing methods.**

Keywords—Software security, Threat modeling, Sequence diagram, Extracting threats

I. INTRODUCTION

Nowadays, the important role of software systems in our life is undeniable. Lack of attention to software security requirements during its development put the final product at security risks. That is, the security of software systems depends on the software design process and the integration of security features [1].

Microsoft Secure Software Development Lifecycle (MS SDL) [2], McGraw model [3], and CLASP[i] [4] are the most prominent processes of secure software development. The processes cover a wide range of activities in the software development lifecycle. Threat modeling has a pivotal role in these processes. This indicates the importance of threat modeling as a best practice in secure development or security evaluation of software systems.

Threat modeling is aimed at identifying and extracting security threats, which in turn leads to identifying security countermeasures to reduce threat risks. On the other hand, threat modeling can also be used by security analysts to evaluate system security in order to measure the level of resistance of the system under evaluation against the threats. Therefore, threat extraction is the basis of secure design as well as of a sound security evaluation.

Threat extraction is mostly based on human expertise, relying upon previous experiences including attack patterns and reported software weaknesses. Therefore, the process is non-automated and it completely depends on human knowledge and skills. This usually results in inaccurate results, which are not exactly the same in separate executions of the process by different experts. Moreover, such a process is not acceptable for developers who generally consider security as an obstacle to the agility of the development process. On the contrary, if threat modeling is automatically integrated within development processes by using common design artifacts, it facilitates security provision by software developers who are not necessarily security experts.

UML is a modeling language used extensively in the software development process. If threat extraction of a software application modeled with UML is also based on UML, the relationship between design and security will be more apparent. In this way, UML-based approaches have the chance of being directly used in the secure software development process [5]. Although there are some proposals based on the mentioned approach [6, 7, 8, 9, 5, 10], they have weaknesses in describing software with security attitude, resulting in defective threat identification.

The purpose of this paper is to propose a repeatable method, which can accurately extract threats of a software system based on an enriched UML-based software description. Because many of the threats are rooted in software behavior, the sequence diagram as a behavioral model is chosen to describe the software. Sequence diagram is enriched with attributes that are the root causes of software threats (Section III, part A). Then, software threats are modeled with the sequence diagram according to the attacker behavior (Section III, part B). A Threat is inferred for a software system when it is matched

978-1-5386-7582-3/18/$31.00 ©2018 IEEE

with the specifications in a sequence diagram (Section III, part C). The proposed method is evaluated using a case study (Section IV). The evaluation results indicate the effectiveness of this approach assuming correct descriptions of software as well as of threats. In the next section, related works to this field are reviewed.

II. RELATED WORK

Existing threat modeling methods can be divided into two categories: (1) software-centric (2) attacker-centric. Software-centric threat modeling, starts with the design of the system. It involves application decomposition and profiling, identifying threats and mitigation strategies. The software-centric threat modeling may start with DFD[ii] or UML diagrams [11].

A. Software-centric threat modeling approaches

Few research has been reported to extract threats based on software descriptions. Microsoft proposed a method for threat modeling of web applications [6]. A tool is also available to automate this method [12]. First, the data flow diagram is given as input to the tool. Then, threats according to STRIDE[iii] category are assigned to the interactions [13]. Safi [7] has also proposed a DFD based approach to extract software threats. She extends software DFD with some attributes and their corresponding values. The threat are expressed using the attributes and their values as well. Frydman *et al.* [8] present AUTSEC (Automated Security Expert Consultant) that performs threat modeling based on the DFD whose elements are standardized. The output of the method is a list of threats along with their countermeasures. AUTSEC is designed to be compatible with Microsoft's threat modeling tool.

Johnstone [9] uses a case study in his research to describe the effect of using activity diagram as an alternative to the DFD in the Microsoft threat modeling approach. Although both approaches have similar functionality for threat modeling, the activity diagram performs a more complete description of the system since it is more expressive. Wang *et al.* [5] propose a security test method to detect the system's undesirable behavior at runtime. In their method, the threat scenario is shown with the sequence diagram. Each sequence diagram represents an event sequence that should not occur during the execution of the system. At first, the penetration test is applied to detect the threat effects on the software. The effects of the implementation are matched with the threat effects. If the effects of execution are an example of the effect of a threat, the security violation is reported. Then measures should be taken to reduce the risk of the threats in the system. This method depends on the penetration test, so the whole possible scenarios may not be evaluated and consequently, all threats cannot be extracted.

Yee *et al.* [10] present an automatic way to extract threats. Sequence and deployment diagrams are used to describe the software. In their method, information about the software system, such as assets, dependencies, and security assumptions is gathered, as well as the UML based software description. The diagrams together with a set of facts are given as inputs to an expert system. The expert system analyzes the facts and returns the threat list. This method, in addition to the UML diagrams, requires extra information, which is not well-defined. Moreover, software is not described with a security attitude, so a few threats are finally identified.

B. Attacker-centric threat modeling approaches

Attacker-centric threat modeling starts with the attacker objectives, motivation and capabilities. This is another type of threat modelling technique available to identify threats by looking from an attacker's perspective. The list of vulnerability and attack library is a collection of attacks for finding threats against the application being developed. The most common resources in this regard are CWE [14], OWASP [15], and CAPEC [16]. However, this approach seems to be more common for the implementation phase rather than the design stage, unless combined with other approaches of threat modeling.

Attack tree, misuse case, and abuse case [17] are among other methods for threat modeling. It should be noted that these approaches are mainly used to represent threats [7], which helps the development team to look at the software from the attacker's perspective and anticipate a proper reaction for unusual events. These methods are unrepeatable, time-consuming, and more importantly, reliant on the human knowledge and experience.

III. PROPOSED METHOD

The proposed method for extracting threats to a software system is comprised of the following steps, each of which is described in detail in the subsequent sections.

1. Software description with enriched sequence diagram
2. Threat description and developing a threat knowledge base
3. Extracting software threats

Fig. 1, shows a schematic view of the relationship between components in the presented method.

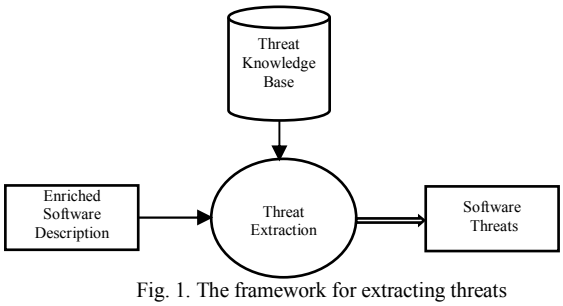

Fig. 1. The framework for extracting threats

A. Software description with enriched sequence diagram

In the proposed method, software threats are described by defining the conditions that must be met in software for a threat to occur. Therefore, the first step is to model the software with the purpose of extracting threats.

Because many of the threats are rooted in software behavior, the sequence diagram, which presents the behavior of the software system is chosen to describe the software. Using the sequence diagram can:

- Display the elements of the system and identify the threats associated with them.
- Identify the interaction of elements and the threats that may occur in these interactions.
- Identify threats associated to the type of transferred data between the elements.

To better describe the threat conditions in the software, the sequence diagram is enriched. In the enriched sequence diagram, each element has a list of attributes with specified values. On the one hand, by giving values to these attributes in the description of the software and on the other hand, by describing threats with the help of the attribute values, threat model is created. To determine the attributes and their values, software vulnerabilities and threats were thoroughly reviewed from different sources. The attributes are then assigned to the diagram elements including process, interface (actor), data store, flow, and the data transferred in the flow. Let us consider "data store", for example, as a threat root source in a software system. In this case, attributes such as the type of the data store (which can be either database or file) and the stored data on it (which can be either encrypted or plain), are indicative attributes for threat extraction. As another example, for the "Flow" itself, the communication protocol (HTTP, FTP, HTTPS, and etc.) and the path of communication (which can be either the Internet or LAN) are important attributes indicating some threats.

The "process", the "actor" and the "data store" are objects that interact with each other through flows. It should be noted that an actor is not directly connected with the process or other objects in a sequence diagram. The interaction between them is possible through an interface. Therefore, in the threat rules, an interface is considered as an actor. That is, the attributes of the actor are assigned to the interface. TABLE. 1, shows the list of attributes and their values for each element of the software sequence diagram. Fig. 2, shows a sample of the enriched sequence diagram for the use case of user registration in a software system.

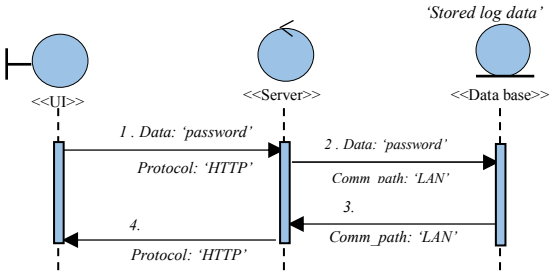

Fig. 2. Enriched sequence diagram for user registration

In the process of registering the user, the server and the database are interconnected. Log information and user information are stored on the data store. The user sends the registration information, including password to the server. Then, the data is transferred via a LAN to store on a database. At the end, a report is sent to the user about the success or failure of registration. As shown in Fig. 2, a part of this information is attached to the diagram as stereotype.

TABLE. 1. List of attributes and their values for diagram elements

		attributes	values
Object	***Process***	Position	***Server, Client***
		Type	***Open_source, COTS, Developed_by_the_team***
		Validated_Input	***Yes, No***
		Validated_Output	***Yes, No***
		Language	***C, C++, C#***
	Interface	Type	***Human, Software***
		Authenticated	***Yes, No***
	Data_store	Type	***File, Database***
		Stored_Log_Data	***Yes, No***
		Encrypted	***Yes, No***
	Flow	Comm_Path	***LAN, Internet***
		Protocol	***HTTP, HTTPS, IPSec, SMTP, SOAP***
		Need_Authentication	***Yes, No***
	Data	Type	***Password, Captcha_id, Time_stamp, One_time_Password, CSRF_token***
		Classified	***Yes, No***

Attributes and their values in Table. 1 have been defined in such a way that:

- With the help of these attributes, threats can be modeled. In other words, the attributes are the intersection part between the elements of the diagram and the threats.
- A software designer without security knowledge and experience should be able to set these attributes for software components.
- Attributes should be as generic as possible so that be applicable for more than a single threat.

Without above considerations, many attributes can be extracted for each diagram element, while their values cannot be straightforwardly specified by software designer.

B. Describing threats

With the selected attributes and their values associated to the elements of sequence diagram, the rules of each threat is determined. As noted in the previous part, the sequence diagram represents the interaction between objects. Each interaction consists of a source, destination, and message. Since threats are generally defined over the interactions, each threat rule can include four clauses, namely *Src*, *Dest*, *Data*, and *Flow*. The *Src* clause represents an object as the source of the interaction. It may be an interface, a process, or a data store. The *Src* clause includes attributes of the source of interaction whose values signify a threat. Similarly, the *Flow* clause is related to the attributes of communication, and the *Data* clause refers to the attributes of data being sent. The last clause, *Dest*, represents attributes of the interaction destination.

Prolog programming notation has been used to write threat rules. Each threat rule in Prolog, consists of two

sections; title and body. The phrase in the title is correct, provided that the expressions in the body are correct. Symbols ":-", ",", and ";" in the rules indicate "*if*", *"and"*, and "*or*", respectively.

Each rule of threat is generally written as follows.

Threat (Threat name):-

Src (Object (‘***Interface’*** ; ***’Process’*** ; ‘***Data_store***’), Attribute (‘***Value’***)**),**

Flow (Attribute (‘***Value’***)**),**

Data (Attribute (‘***Value’***)**),**

Dest (Object (‘***Interface’*** ; ***’Process’*** ; ‘***Data_store***’), Attribute (‘***Value’***)**)**

Threat scenarios and the information that is needed to formulate any threat have been thoroughly investigated. CWE, CAPEC, OWASP and experts' opinions are among those threat sources to create a threat knowledge base. At the time of writing this paper, the threat knowledge base includes 45 threats. In this section, sample rules for four threats are described. Due to space constraints, the rest of the threats are accessible in the appendix. It is worth mentioning that attributes and values that trigger a threat are attached to the sequence diagram with stereotypes.

Threat 1- Brute-force attack: It is an attempt to discover a password by systematically trying every possible combination of letters, numbers, and symbols until discovering the correct combination. Any web site requiring user authentication is a good target for the brute-force attack. If the CAPCHA code is used for authentication (with correct implementation), the software will be immune to this threat. A connection whose source is an anonymous user, message type is password, and the flow requires source authentication represents the user's login. Fig. 3, shows the sequence diagram for this threat.

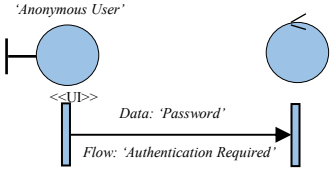

Fig. 3. Brute force threat sequence diagram

According to the above descriptions and sequence diagram in Fig. 3, the threat rule is as follows:

Threat (Brute_Force_Login):-

Src (Obj**ect** (‘***Interface’***), Type (‘***Human’***), Authenticated (‘***No’***))**,**

Flow (Need_Authentication (‘***Yes’***))**,**

Data (Type (‘***Password’***), Not (Type (‘***Capthca_id’***)))**

Threat 2- Cross-Site Request Forgery (CSRF): It is an attack that forces the victim to send a request to the web site trusting the victim. The use of challenge-response mechanisms is a countermeasure for CSRF attack. CAPTCHA code, if properly implemented, can be useful as well. Web sites can also use other mechanisms such as CSRF tokens to protect themselves against the attack. Therefore, any server accepting a request from an authenticated user or a process, without imposing any control over it, is vulnerable to CSRF. Fig. 4 (A) and Fig. 4 (B), illustrate the sequence diagram of these vulnerable communications. However, the software system may be connected to an external vulnerable server. For example, a CSRF attack may occur in an online shopping that is connected to a payment server, which appears in the sequence diagram as an actor. Therefore, an interaction with a software-type actor as the destination is vulnerable to CSRF. Fig. 4 (C), shows the sequence diagrams of this type of threat.

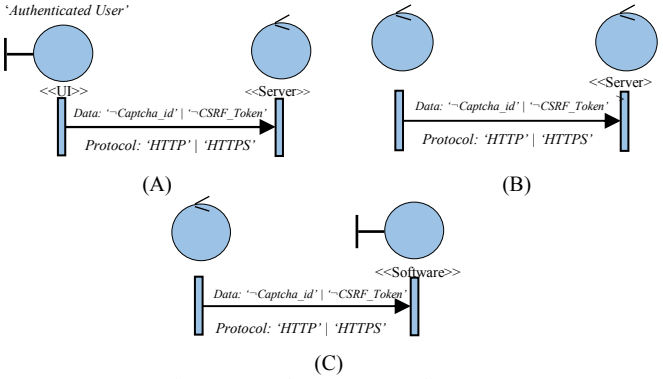

Fig. 4. CSRF threat sequence diagrams

According to the above description and Fig. 4, The CSRF occurs if the following conditions are hold:

If

1. The source of the connection is an **authenticated user** or a **process**
2. The protocol communication is **HTTP** or **HTTPS**
3. The sent data does not include the **captcha_id** or **CSRF token**
4. The destination is to be a **server** or an **actor** with **software** type

Then

There is a **CSRF** threat.

Using Prolog notation, the CSRF threat rule is written as follows:

Threat (CSRF):-

Src (((Object (‘***Interface’***), Type (‘***Human’***), Authenticated (‘***Yes’***)); Object (‘***Process’***))**,**

Flow (Protocol (‘***HTTP’*** ; ‘***HTTPS’***))**,**

Data (Not (Type (‘***Captcha_id’*** , ‘***CSRF_token’***)))**,**

Dest ((Object (‘***Process’***), Position (‘***Server’***)); (Object (‘***Interface’***), Type (‘***Software’***)))

Threat 3- SQL injection: The attack is to inject a SQL query via the input data from the client to the software that allows the attacker to control the database. An interaction with a database destination is susceptible to the threat. Fig. 5, shows the sequence diagram for this threat.

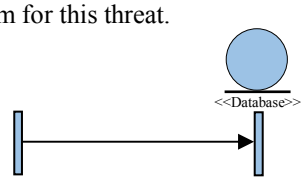

Fig. 5. SQL injection threat sequence diagram

According to the above descriptions and sequence diagram in Fig. 5, the threat rule is as follows:

Threat (SQL_Injection):-

Dest (Object (‘***Data_Store’***), Type (‘***Database’***))**

Threat 4- Error message with information: In applications, information about the configuration, internal operations, or privacy of users may be disclosed through unexpected error messages. An attacker can fetch sensitive information by sending different inputs and receiving error messages. Therefore, if the source of interaction is a process, which does not control its output, and the destination is an interface, there is a possibility of data leakage. Fig. 6, shows the sequence diagram for this threat.

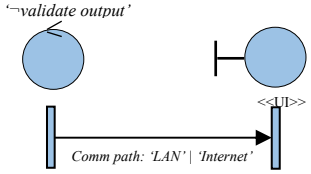

Fig. 6. Error message info threat sequence diagram

According to the above descriptions and sequence diagram in Fig. 6, the threat rule is as follows:

Threat (Error_Message_ Info):-

Src (Object (‘***Process’***), Validated_Output (‘***No’***)),**

Flow (Comm_Path (‘***LAN’*** ; ‘***Internet’***)),**

Dest (Object (‘***Interface’***), Type (‘***Human’***))**

C. Identify and extract software threats

To extract threats of a software system, the enriched sequence diagram describing the software behavior are matched with the threat rules in our threat knowledge base, relying on an inference process. At the end of this step, a list of all identified threats is presented as output, which is the result of matching the software description diagrams with the modeled threats. For example, in Fig. 2, which shows the enriched sequence diagram for user registration, the destination of the second interaction is database. This interaction is matched with the SQL injection model (Fig. 5). Therefore, the SQL injection is inferred and identified as a threat for the software system.

IV. CASE STUDY

The proposed method has been applied on an online shopping system having six major use cases to extract the threats. Fig. 7, shows the Use case diagram for this software. In order to evaluate the results of the proposed method, we also performed the method presented in [10] on the same system to compare the resulting threats. The method presented in [10] is the closest method to our proposal. For each use case, we drew an enriched sequence diagram through which the threats can be extracted. Due to page limit, we explain only the extracted threats from the user registration use case.

Fig. 7. Use case diagram of the shopping software system

Fig. 2, shows the enriched sequence diagram of the user registration operation. For the registration operation, 43 threats have been extracted in the above method. TABLE. 2, shows part of these threats. In the table, in addition to the name of the threat, we can see elements of the diagram that have contributed to the threat. These elements include the source, destination, interaction number and the type of data being sent in the interaction. The method presented in [10] also identifies nine threats to the registration use case.

TABLE. 2. List of user registration threats

	Threat name	Source	Data	Interaction number	Destination
1.	Pass_Weak	UI	Password	1	
2.	XSS	UI		1	Server
3.	SQL_injection			2	DB
4.	Error_Message_Info	Server		4	UI
5.	Injection_ Log			2	DB
6.	Format_string	UI		1	P3
7.	SSRF	UI		1	P3
8.	Sniff_Data_Transit		classified	1	
9.	*Spoof_Datastore*			2	DB
10.	*Invalid_Redirects_Forwards*	UI		1	Server
11.	*Fingerprinting*	Server		4	UI
12.	*Integer_OverFlow*				Server
13.	*Server_Side_Injection*	UI		1	Server

Fig. 8, shows the total number of extracted threats for the system per use case. In the figure, the number of threats is specified in both methods.

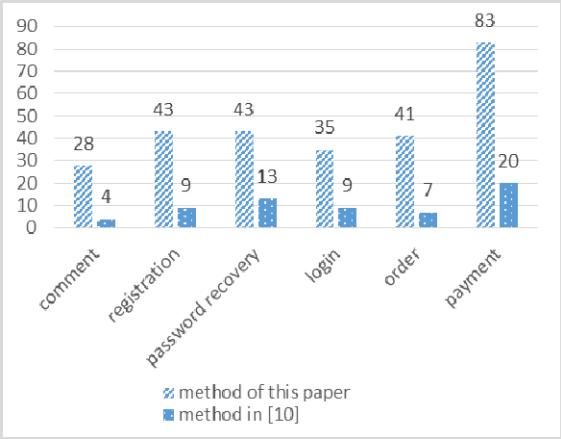

Fig. 8. The number of threats for both methods

As shown in Fig. 8, the threats identified in the proposed method are considerably more than those identified using the other method. Threats extracted by the method in [10] are limited to four types including man in the middle, Trojan horse, SQL injection, and denial of service. The proposed method addresses these types of threats more precisely. That is, man in the middle, Trojan horse and denial of service threats are divided into smaller threats. For example, denial of service threat may occur for many reasons, such as loss of data store or a weakness in a process source code. In fact, they are divided into atomic threats. The more specific threats are extracted the more comprehensible security countermeasures are chosen. The threats described in the proposed method are classified into 10 categories including security misconfiguration, sensitive data exposure, denial of service, weak authentication, weak access control, spoofing, inadequate validation, data manipulation, insecure session management, and insufficient logging.

V. CONCLUSION

In this paper, a method for extracting software threats that uses UML-based descriptions is presented. Considering that many of the threats are rooted in software behavior, sequence diagram, which is a behavioral model was used for software description. To infer threats from the software behavior, sequence diagram was enriched. In this way, some attributes and values are added to the elements of the diagram and their communications. A rule based threat repository was also developed with 45 threats defined in its current version. To extract software threats by this method, a sequence diagram is matched to a threats in the knowledge base to find out if the diagram satisfies the threat condition. The proposed method is more accurate in extracting threats due to the appropriate selection of attributes and their values with which sequence diagram has been enriched. In the future, we plan to automate threat extraction by implementing the inference engine inputting threat repository and sequence diagrams. In addition, there is the ability to expand the threat knowledge base by describing new threats.

REFERENCES

[1] H. shafigh, K. Asif, A. Shabir, R. Ghulam, I. Sajid, "Threat modelling methodologies: a survey", Science International.(Lahore), 2014. Vol. 26(4): p. 1607-1609.

[2] S. Lipner, "The trustworthy computing security development lifecycle", in Computer Security Applications Conference, 2004. P. 2-13.

[3] G. McGraw, "Software security: building security in", vol. 1, 2006: ISBN-10: 0-321-35670-5 Addison-Wesley Professional.

[4] "Category: CLASP Concepts- OWASP." [Online]. Available: https://www.owasp.org/index.php/CLASP_Concepts. [Accessed: 23-Feb-2018].

[5] L. Wang, , E. Wong, D. Xu, "A threat model driven approach for security testing", in Software Engineering for Secure Systems (SESS'07), ICSE Workshops 2007, Third International Workshop, p. 10

[6] J. Meier, A. Mackman, B. Wastell, "Threat modeling web applications", Microsoft Corporation company, 2005. [Online]. Available: https://msdn.microsoft.com/en-us/library/ff648006.aspx [Accessed: 23-Feb-2018].

[7] Mohadeseh, safi, "Proposing a method of software threat specification and creating a threat description repository", Information and Communications Security Institute, Malek-e-Ashtar University of Technology, tehran, 2014.

[8] M. Frydman, G. Ruiz, E. Heymann, E. Cesar, B.P. Miller, "Automating risk analysis of software design models", The Scientific World Journal, vol. 2014, p. 12, 2014.

[9] M.N. Johnstone, "Threat modelling with STRIDE and UML", Proceedings of the 8th Australian Information Security Management Conference, 2010, p. 18-27.

[10] G. Yee, , X. Xie, and S. Majumdar. "Automated threat identification for UML", in Security and Cryptography (SECRYPT), Proceedings of the 2010 International Conference on. 2010, p. 1-7.

[11] S.R. Satapathy, "Threat modeling in web applications", MTech thesis, Department of Computer Science and Engineering, National Institute of Technology Rourkela, India 2014.

[12] Microsoft's Threat Modeling Tool. [Online]. Available: https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool. [Accessed: 23-Feb-2018].

[13] Swiderski, F. and Snyder, W., 2004. Threat Modeling (Microsoft Professional), Vol. 7. Microsoft Press, ISBN-10: 0735619913

[14] "CWE - Documents." [Online]. Available: https://cwe.mitre.org/about/documents.html. [Accessed: 23-Feb-2018]

[15] "Open Web Application Security Project (OWASP)", Application Security Risks-2017, [Online]. Available: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf. [Accessed: 23-Feb-2018]

[16] "CAPEC" [Online]. Available: https://capec.mitre.org/index.html. [Accessed: 23-Feb-2018].

[17] Yuan. X, Nuakoh. E .B, Williams. I, Yu .H, "Developing Abuse Cases Based on Threat Modeling and Attack Patterns". Journal of Software (JSW), vol. 10(4), . 491-498, Doi: 10.17706, 2015.

[i] Comprehensive, Lightweight Application Security Process

[ii] Data Flow Diagram

[iii] Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege

…Appendix…

Due to space constraints, the rules for other threats are available in the following list:

Threat (Pass_Weak):-
Src (*Obj (‘Interface’), Type (‘Human’))****,*
Data (*Type (‘Password’))*

Threat (Session_Expiration):-
Src (*Object (‘Interface’), Type (‘Human’), Authenticated (‘Yes’))****,*
Flow (*Need_Authentication (‘Yes’))*

Threat (SSRF):-
Src (*(Object (‘Interface’), Type (‘Human’)); Object (‘Process’))****,*
Flow (*Comm_Path (‘LAN’ ; ‘Internet’))****,*
Dest (*Object (‘Process’), Position (‘Server’), Validated_Input (‘No’))*

Threat (Weak_Enc_Stored_Data):-
Src (*Object (’Data_store’), Encrypted (‘Yes’))****;*
Dest (*Object (’Data_store’), Encrypted (‘Yes’))*

Threat (Data_Store_Lost):-
Src (*Object (‘Data_store’))****;*
Dest (*Object (‘Data_store’))*

Threat (Insufficient_Log):-
Src (*Object (‘Process’))****,*

Threat (Misconfiguration):-
Src (*Object (‘Process’), Type(‘COTS’))*

Threat (Sniffing_Data_Transit):-
Data (*Classified (‘Yes’); Type (‘Password’))****,*
Flow (*Comm_Path (‘LAN’ ; ‘Internet’), Not (Protocol (‘IPsec’ ; ‘HTTPS’)))*

Threat (Replay):-
Src (*Object (‘Process’ ; ‘Interface’))****,*
Flow (*Comm_Path (‘LAN’ ; ‘Internet’))****,*
Data (*Not (Type (‘Time_stamp’ ; ‘One-time Password’)))*

Threat (XSS):-
Src (*(Object (‘Interface’), Type (‘Human’)); Object (‘Process’))****,*
Flow (*Comm_Path (‘LAN’ ; ‘Internet’))****,*
Dest (*(Object (‘Process’), Position(‘Server’), Validated_Input (‘No’)); (Object (‘Interface’), Type (‘Software’)))*

Threat (Captcha_Bad_Implementation):-
Src (*Object (‘Interface’), Type (‘Human’))****,*
Data (*Type (‘Captcha_id’))*

Threat (Unlimited_ Datastore_Allocation):-
Dest (*Object (‘Data_store’))*

Threat (Web_Root_Data):-
Src (*Object (‘Process’), Position (‘Server’))****,*
Flow (*Comm_Path (‘LAN’ ; ‘Internet’)),*
Dest (*Object (‘Interface’), Type (‘Human’))*

Threat (Spoof_source):-
Src (*Object (‘Process’ ; ‘Interface’))****,*
Flow(*Not (Protocol (‘IPsec’)))*

Threat (Weak_Access_Control):-
Src (*Object (‘Data_store’))****,*
Data (*Classified (‘Yes’))*

Threat (Unlimited_ Memory _Allocation):-
Src (*Object (‘Process’))*

Threat (Unlimited_ Process_Allocation):-
Src (*Object (‘Process’))*

Threat (Unpatched_Component):-
Src (*Object (‘Process’), Type(‘COTS’))*

Threat (Unpatched_Opensource):-
Src (*Object (‘Process’), Type (‘Open_source’))*

Threat (Cleartext_Data_Storage):
Data (*Classified (‘Yes’); Type (‘Password’))****,*
Dest (*Object (‘Data_store’), Encrypted (‘No’))*

Threat (Injection_ SMTP_Header):-

Flow (*Protocol (‘SMTP’))*

Threat (Weak_Password_recovery):-

Data (*Type (‘Password’)),*

Dest (*Object (‘Interface’), Type (‘Human’))*

Threat (Spoof_Data_store):-

Src (*Object (‘Data_store’));*

Dest (*Object (‘Data_store’))*

Threat (Pass_Autofill):-

Src (*Object (‘Interface’), Type (‘Human’), Authenticated (‘No’)),*

Flow (*Need_Authentication (‘Yes’)),*

Data (*Type (‘Password’))*

Threat (Fingerprinting):-

Src (*Object (‘Process’)),*

Dest (*Object (‘Interface’), Type (‘Human’))*

Threat (Injection_Log):-

Dest (*Object (‘Data_store’), Stored_Log_Data (‘Yes’))*

Threat (Server_Side_Injection):-

Src (*(Object (‘Interface’), Type (‘Human’)); Object (‘Process’)),*

Flow (*Protocol (‘HTTP’)),*

Dest (*Object (‘Process’), Position (‘Server’), Validated_Input (‘No’))*

Threat (Spam):-

Flow (*Protocol (‘SMTP’)),*

Data (*Not (Type (‘Captcha_id’)))*

Threat (Integer_overflow):-

Dest (*Object (‘Process’), Validated_Input (‘No’))*

Threat (Session_Fixation):-

Src (*Object (‘Interface’), Type (‘Human’), Authenticated (‘Yes’)),*

Flow (*Need_Authentication (‘Yes’))*

Threat (*Buffer_OverFlow****):-***

Dest (*Object (‘Process’), Validated_Input (‘No’), language (‘C’ ; ‘C++’))*

Threat (IDOR_File):-

Dest (*Object (‘Data_store’), Type (‘File’))*

Threat (Encoding_Error):-

Src (*Object (‘Process’), Validated_Output (‘No’)),*

Dest (*Object (‘Interface’), Type(‘Human’))*

Threat (Format_string):-

Src (*Object (‘Interface’ ; ‘Process’)),*

Dest (*Object (‘Process’), Validated_Input (‘No’))*

Threat (Crash_Process):-

Dest (*Object (‘Process’))*

Threat (Spoof_Destination):-

Flow (*Not (Protocol (‘IPsec’ ; ‘HTTPS’))),*

Dest (*Object (‘Process’ ; ‘Interface’))*

Threat (Log_Info):-

Src (*Object (‘Data_store’), Stored_Log_Data (‘Yes’)),*

Threat (Invalid_Redirects_Forwards):-

Src (Object (‘Interface’), Type (‘Human’)),

Flow (Protocol (‘HTTP’)),

Dest (Object (‘Process’), Validated_Intput (‘No’))

Threat (Command_Injection):-

Src (Object (‘Interface’ ; ‘Process’)),

Dest (Object (‘Process’),Validated_Input (‘No’))

Threat (Session_Hijacking):-

Flow (Not (Protocol (‘HTTPS’ ; ‘IPsec’))),

Threat (Info_Exposure_WSDL):-

Src (*Object (‘Process’), Validated_Output (‘No’)),*

Flow (*Protocol (‘SOAP’))*