

A Collaborative Platform Featuring Visibility, Tracking, Monitoring and Awareness for Building Security In.

H. M. K. K. B. Herath*, G. D. S. P. Wimalaratne

University of Colombo School of Computing, 35 Reid Ave, Colombo 00700, Sri Lanka.

* Corresponding author. Tel.: +94-771584589; email: kavindakosala@gmail.com

Manuscript submitted July 01, 2018; accepted September 10, 2018.

doi: 10.17706/ijcce.2018.7.4.145-166

Abstract: Software developed referring to a poor design often causes the introduction of security issues which could spread into other phases of the Software Development Life Cycle if not address in the initial stages. This could lead to major security breaches and loss of valuable assets to the consumers. Identifying and fixing security issues as early as possible in a software product is the most cost-effective way of implementing software security. This research proposes a proactive approach to build security into the product itself with the aid of a new tool developed as a proof of concept. The proposed semi-automatic tool will address limitations in current approaches to secure software engineering when developing a software product by providing visibility, tracking, awareness, and progress monitoring. Additionally Developers, Architects, QA, BA, and Management, as well as the Users, can participate in the Threat Modeling and architectural security analysis contributing their input for Security Engineering with the support provided by the tool as an interactive platform, a knowledge base and as an integration platform. The Microsoft Threat Modeling Tool is being used to generate the threat models. The tool extracts threat model information and produces detailed mitigations using known vulnerability databases and classification techniques. Developers can better understand the potential threats, vulnerabilities when coding and integration functionality with a Project Management Tool can provide visibility and tracking of Building Security In throughout SDLC.

Key words: Threat modeling, building security In., architectural risk analysis, defence in depth, collaborative platform, secure software engineering, application security.

1. Introduction

Implementing a proper Secure Software Development Life Cycle which incorporate security has become a major turning point in a successful software application. The recent boom in the internet infrastructure and web application technologies and other software products have raised concerns over the security of software applications. The main reason behind this is that the software inherently carries sensitive information [1] and getting compromised means a huge loss of valuable assets for both software vendors and the consumers [2]. In the traditional Software Development Life Cycle, security is only focused on the production deployment/release phase and adding a security patch or an update to an identified bug or a vulnerability is a relatively costly thing compared to applying that during the development phase or design level [3]. It mostly concerned about the security of infrastructure and networks and address software security with the use of Antivirus Software, Firewalls, Intruder Detection Systems, and Operating Systems Security to harden the application environment [4]. With this approach, the security robustness of the

application itself is not considered and usually, software vendors release patches based on later findings of known security bugs, pen-testing results to harden the software with a new release version as a reactive countermeasure after catastrophic outcomes [4], [5]. This approach also recognized as Application Security approach by world-renowned Internet Security expert Garry McGraw [3]. As a solution to these aforementioned problems in the current Software Development Life Cycle phases, stakeholders started to find new ways of incorporating security into software development throughout all of its SDLC phases and to effectively develop a secure software code by focusing on Secure Architecture/ Design and Secure Implementation of software [3].

Exploits or security breaches in a software which lives inside a security-hardened environment are most of the time a result of a vulnerability or a bug exists in the application itself. These defects come into existence due to the Architecture/ Design flaws and later propagate to development phase as implementation bugs [4]. A security bug can be identified as defect occurs in the implementation phase whereas a security flaw can be identified as a defect in a software Architecture/ Design or a wrong realization of application behavior and the requirement [6]. An Architectural/ Design defect may cause a chain reaction to produce more security bugs vulnerabilities in a software system throughout the SDLC since they are the root artifacts for a software implementation [6], [7]. Moreover, stats show that 50% of the security problems found in software are caused by defects which can be found in the design artifacts [3]. Finding suitable methods to identify security flaws/vulnerabilities in a design artifact is a key thing to overcome these aforementioned problems. Threat modeling address this requirement where it does an architectural risk analysis based on architectural and design artifacts [3], [8].

This research is focusing on improving the incorporation of software security aspect with SDLC in its Design/ Implementation phases through threat modeling software design and to enforce and track Build Security In process throughout SDLC providing visibility to all the stakeholders of a Software product. A semi-automatic tool is being developed to support these requirements. An experienced Developer / Lead or an Architect need to create the threat model accordingly with the application data flow using the Microsoft Threat Modeling Tool (MSTMT). This threat model along with the final report from MSTMT will be used as inputs to the KOSALAK framework which extract threat model information and generates mitigation techniques accordingly. Inconclusive threats produce by current threat modeling tools tend to demoralized the threat modeling process and the effectiveness and there is a huge void in the area of communication and providing visibility to all the entities involved in the SDLC. The KOSALAK framework will target addressing these areas by providing visibility of threat modeling of software design to Developers, QAs, Architects, Management and Client and also refine threat modeling output by associating STRIDE technique [3], [7], [9] OWASP top 10 threats [10], CWE/SANS top 25 [11] and CAPEC [12] to eliminate inconclusive threats to reduce the effort and cost needed in the SDLC for mitigation.

2. Related Work

Software security is the science of engineering software in order to function correctly under malicious attacks [3]. This focuses on solutions to define and establish a solid development paradigm which includes practices, processes, tools, benchmarks, test suits, etc to target the reduce numbers and severity of vulnerabilities in software and identify and manage the security flaws throughout the development instead of at maintenance [13], [14]. Software vendors usually provide security patches or updates to resolve identified security vulnerabilities but it is a less practical in most of the situations where a security patch is given prior to it has been exploited by an attacker [3], [4] and could lead to compromised systems. Web applications are ranked higher than the most when it comes to security breaches due to their nature and statistics shows that most of these exploitation carried out due to vulnerabilities exist in the application

itself rather than due to network or operating system security [15], [16].

The internet security pioneer Garry McGraw who introduced the concept of Build Security In states seven touch-points and their interrelationship between software artifacts and the processes and methodologies that needs to follow in order to develop a software securely [3]. Fig. 1 shows that this process is conducted in an iterative manner where security-related activities can be cycled multiple times in a single phase of SDLC [3]. Each insight gain relating to software security within a single cycle is fed back to next cycle to improve the building security in within that stage. This carries on through all the stages of SDLC thus, the integration of touch points with the SDLC can work seamlessly with any software development model like Agile, Waterfall, Evolutionary, etc. [3].

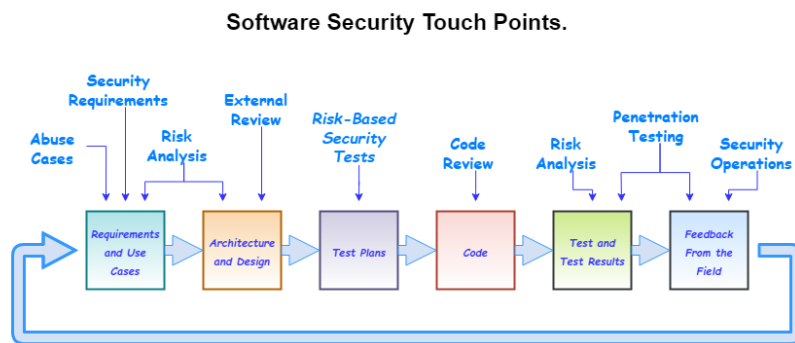


Fig. 1. Software security touch points [3].

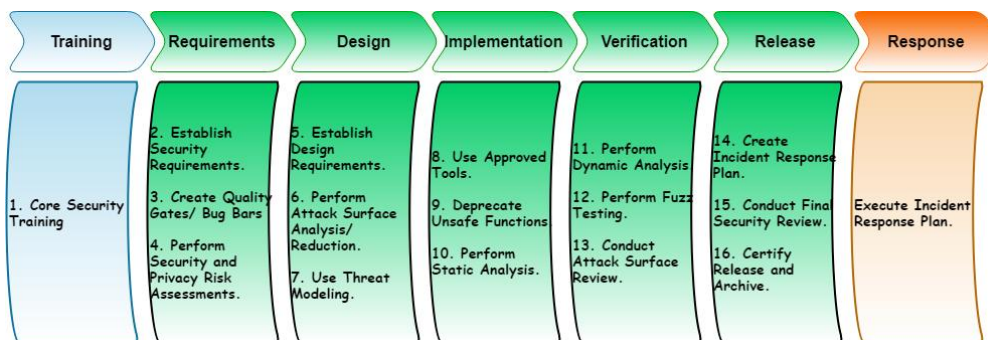


Fig. 2. Microsoft security development lifecycle [4].

Microsoft Corporation has also started their own development process after Bill Gates Memo in 2002 to incorporate security into their applications called Microsoft Security Development Lifecycle. This was implemented under the Microsoft Trustworthy Computing Initiative. Fig. 2 shows key aspects of the Microsoft Security Development Lifecycle, they are focusing on People, Process and Technology to address security issues [3], [4], [7]. Each person involved in a software development are being trained to give them a better understanding and awareness of software security and how to build secure applications [3], [4], [7]. Software development process within Microsoft Corporation has been also enhanced to incorporate security and privacy as mandatory in designing, developing and testing following a holistic and practical approach leading to reduced number and severity of vulnerabilities in Microsoft products [3], [4], [7]. Microsoft has also improved the technology stack with introducing new tools to be used within SDL and only allowing approved tools for development and testing as well as establishing guidelines on deprecating unsafe functions, methods and software components and by conducting code reviews focusing security aspect of the application [3], [4], [7]. Threat modeling comes in the Design phase of the Microsoft Security Development Lifecycle focusing improved security in software design artifacts. This is conducted with the

aid of Microsoft Threat Modeling Tool by analyzing Level-0, Level-1 DFD derived using design artifacts [4], [7]. Applying a structured approach to threat scenarios during design helps a team more effectively, productively and less expensively identify security vulnerabilities, determine risks from those threats, and establish appropriate mitigations [7].

Threat Modeling

Threat modeling comes as an architectural risk analysis methodology based on software design documents. It provides a means to identify, classify, quantify and address security flaws in a software design [3], [4]. Since design artifacts are the root ingredients in a software system it is vital to understand about the probable security risks that can arise from design and address those early as possible eliminating half of the security vulnerabilities found in a software application [3], [5]. In most situation, Data Flow Diagrams are much more convenient and result driven if used for threat modeling the software design. A DFD overview of the system is created referencing design of the software following a three-step process which includes Attack resistance, Ambiguity analysis, and Weakness analysis [17]. Having DFDs in Level-0, Level-1 is sufficient to represent a threat model [17] and to identify the architectural security flaws in the design. Following attackers approach and by checking security constraints and by assigning security characteristics to components can generate a reasoning mechanism to identify most probable threats to a software component [18]. Then the attack resistance step carried out to generate an attack checklist to understand known attacks. After determining the threats associated with these attacks they can be categorized using Microsoft STRIDE category model [3]. Then the ambiguity analysis is conducted to remove false positives from the identified threats by examining the application's area of potential vulnerabilities. Apart from threat modeling, there are mainly two other approaches to do architectural risk analysis namely Trust modeling and Data Sensitivity modeling. Trust modeling is carried out to identify the trust boundaries for software components and data and Data Sensitivity modeling is conducted to identify the privacy and sensitivity of application data. Combining all these approaches can improve the accuracy of the final outcome. Threat modeling can be broken down into three main steps. The first step is focused on decomposing and gaining insight about the application and its behavior and the interactions it makes with other components, external/internal entities, etc. This allows to identify entry points/exit points of the application and determine where a potential attacker can interact with the system. During the second step, threat is identified, categorized and ranked to decide priorities. This can be done either using attackers perspective (STRIDE) [3] or using defensive perspective (ASF) [3]. Upon categorization and ranking threats, as the third step suitable mitigation and countermeasures can be decided. These countermeasures can be identified using threat-countermeasure mapping lists and with the help of ranking can decide the severity and the effort needed to resolve or implement the mitigation based on business requirements and the impact they pose.

3. Solution Design/Implementation

The research focuses on finding solutions to limitations in present secure software engineering process by providing a methodology to add visibility to Threat Modeling process and provide tracking and progress monitoring of Building Security In by developing a tool which could generate comprehensive mitigations and best practices for identified threats, predict inconclusive threats (false positives) and to enforce and educate developers on best practices to avoid known security vulnerabilities when coding, plus integration with the SDLC management tools. Based on findings from literature survey it has been identified that the MSTMT is the best feasible tool to conduct the architectural risk analysis due to reasons like its being a popular free tool and openly available to download as well as the simplicity and user-friendliness. MSTMT threat model has been taken in as the input for the framework. The system architecture consists of

components such as Threat Model Parser Engine, Knowledge Base, Mitigation Generator Engine, Inference Engine, User Interaction - Web View Engine, External Integration, and Publisher Engine. The mentioned components of the system and the methodology will be explained in detail in the following subsequent sections. The new design and architecture will address the drawbacks and limitations in the current approaches in a proactive manner to enhance the Secure Software Engineering paradigm.

Support of integration with other tools has been limited only to Microsoft Threat Modeling Tool and the JIRA project management tool. MSTMT will provide the input information for the framework to work and the JIRA application will be the end point of the KOSALAK frameworks output. Generation of architectural threats will only be limited to the capabilities of MSTMT and its knowledge base. The intended users and the beneficiaries of the framework are not limited to technical people and the nontechnical individuals also can use certain features of the system. Software developers, Architects, Security Engineers can use the MSTMT to generate an appropriate threat model after analyzing the software project architectural and design artifacts using Level-0 or Level-1 DFDs. The literature review indicated that the data contained in Level-0, Level-1 has been identified as sufficient to identify the architecture level security flaws. The “tm7” file and the “htm” file which can be generated by MSTMT will contain the threat model information in respect to STRIDE categorization needed to feed to the KOSALAK framework to generate mitigation techniques and best practices and identify and filter low priority threats. Identification and filtering out low priority threats will be based on OWASP top 10 and CWE/SANS top 25 and the generated final output can be published into JIRA project management tool using REST API in a semi-automatic manner since the verification of false positives needs human interaction in the present implementation. Fig. 3 depicts the use case diagram derived according to the above requirements of the framework.

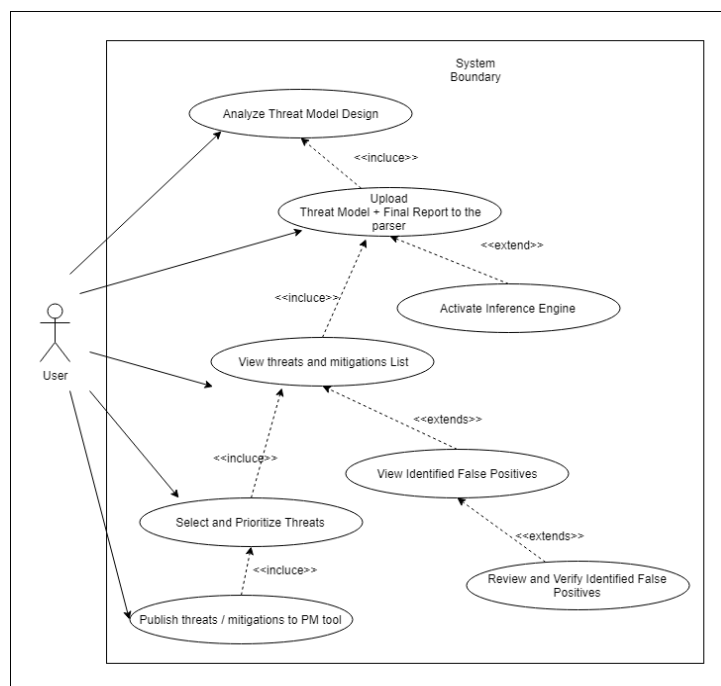


Fig. 3. Use cases.

3.1. System Architecture

The KOSALAK framework presents the proof of concept of this research and works as a semi-automated, standalone, web application which will take an MSTMT threat model and the corresponding final report as inputs and will parse those to extract useful threat model information. Extracted threat model information

will then be passed to the Inference Engine to generate mitigation techniques, best practices and to narrow down inconclusive threats which have been identified with respect to OWASP top 10, CWE/SANS top 25 and CAPEC and then the final output will be displayed in the User Interaction Web View. Users can view the threats and countermeasures in a user-friendly and interactive way enabling them to present and discuss the architectural risk analysis related information and can further conclude the false positive (inconclusive threats) with human intelligence. In the final stage, the user can submit the prioritized threats and mitigation information automatically to JIRA project management tool to provide visibility, tracking, and monitoring of Build Security Into all the stakeholders across the SDLC. The Reason for eliminating false positives is to improve the reliability of the threat modeling process so the unnecessary expenditure of effort and time can be saved without wasting on unlikely issues thus adding benefit to software institutions who actively use threat modeling. By being able to integrate with a project management tool enforces the threat modeling process by adding the issues to the project backlog and it will be easy to prioritize and later worked on by assigning to individuals when developing a software component. Fig. 4 demonstrate the process flow of the framework in contrast.

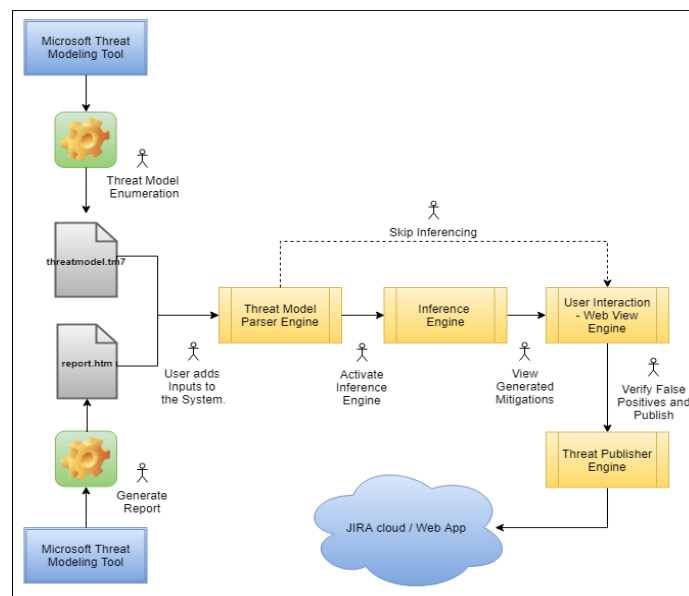


Fig. 4. Process flow.

A published vulnerability or a threat in the PM tool will display the severity or priority of the issue with respect to the design and the description of the issue along with mitigation techniques and best practices with examples and the relation and the point it was extracted from threat model design. This way the developers who were assigned to the issue and other interested parties can interpret the issue's origin and can come up with a more solid way of handling the issue. Main components of the system can be categorized into five main groups. The Presentation Layer will represent the components which are interacting with the user. The Parser Module is an abstract implementation of different parsers supported for different types of threat modeling tools and other inputs allowing the system to be scalable with multiple threat modeling tools in the future. Currently, the parser for Microsoft Threat Modeling Tool is implemented and integrated. Core Logic Layer represents the decision making and knowledge modeling mechanism to generate mitigation techniques and best practices for identified threats and to identify false positives. Data Layer represents the persistent database and its components as well as lexical DB which is used for semantic inferring. Publisher Module is an abstract implementation that supports integration with different PM tools with support for scalability in future developments. Currently, integration support for

JIRA project management tool has been implemented. In Fig. 5 below shows, the components of the framework and the concrete implementations have been marked in solid color and the future extensions in dotted line.

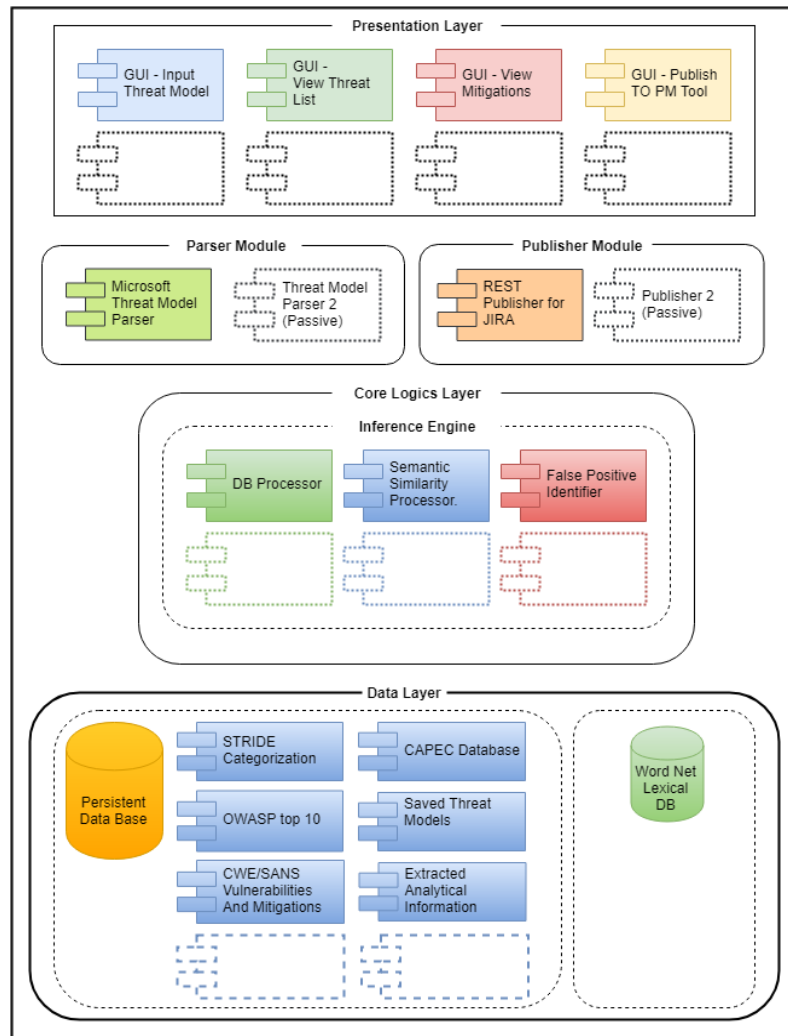


Fig. 5. Components.

3.2. Presentation Layer

This layer will logically group application modules related to UI/UX together to provide Abstraction.

3.2.1. GUI input threat model

This component in the Presentation layer represents the GUI sub-component which allows the user to set the input “tm7” file and the “htm” file to the system for parsing. It will ask the user to point to the files in an opened file explorer window and a checkbox to tick to activate false positive inference. If the inference engine is enabled it will generate the mitigations and best practices and mark identified false positives for the user to verify.

3.2.2. GUI view threat list

This component in the Presentation layer represents the GUI sub-component which allows the user to view the extracted threat model information and the false positives from a given threat model file.

3.2.3. GUI view mitigation

This component in the presentation layer represents the GUI sub-component which allows the user to

view mitigations and best practices generated by the inference engine for a selected threat. Once clicked on the threat the view will expand and display the mitigations and the best practices.

3.3. Threat Model Parser Interface

This layer will logically group application modules related to parsing inputs like files, web service messages, etc to the system together to provide Abstraction.

3.3.1. Microsoft threat model parser

This component in the Threat Model Parser Interface section is a concrete implementation of a threat model parser of type Microsoft threat modeling tools threat model and it can extract threat model information from a given "*.tm7" file and a given "htm" file for further processing.

3.3.2. Threat model parser 2 (passive)

This is an abstract representation of a threat model parser which can be implemented in a future development to support a different type of threat model providing scalability.

3.4. Core Logic Layer

This layer will logically group application modules related to core functions of the system together to provide Abstraction.

3.4.1. DB processor

This component in the Core Logic Engine will implement the back-end logic related to database communications. The implementation regarding database processing with persistent database and the lexical DB is implemented in the "tmm.corelogic.db" package facilitating long-term storage support accumulating all the analyzed threat models.

3.4.2. Inference engine

This layer will logically group application modules related to inference functions of the system together to provide Abstraction. The "tmm.corelogic.inference" package will contain the logic relating to inferences from threat models making the association with different threat categorization and mitigation techniques. Here the implemented logic is to match the semantic similarity of threats obtained from MSTMT against OWASPT Top 10, CWE/SANS Top 25, CAPEC vulnerability databases.

Semantic Similarity Processor

Here each threat description has been evaluated to find the highest matching threat from the collected vulnerability databases. With a given threshold value and any two strings above that value with semantic similarity matching is considered as a candidate threat to produce and extract mitigation and best practices. When calculating the similarity of the semantics word by word comparison of each string has been evaluated. The collective output of the words in each sentences is considered as the final value. Wu - Palmer semantic similarity matching algorithm has been used with the WS4J library in combination with Word.net lexical DB. Below code snippets shows the key methods of the semantic similarity calculations.

```
public double calculateWS4JSimilarity(String phrase1,String phrase2) {
    String[] words1 = phrase1.split("\\W+");
    String[] words2 = phrase2.split("\\W+");
    double total = 0.0;
    int count = 0;
    for(int i=0; i<words1.length; i++){
        for(int j=0; j<words2.length; j++){
            double distance = calculateWS4JSimilarityWord(words1[i], words2[j]);
            if(distance==Double.MAX_VALUE) {
                distance = 5;
            }
            total +=distance;
        }
    }
}
```



```

        count++;
    }
}
return total;
}

public double calculateWS4JSimilarityWord(String word1, String word2) {
    RelatednessCalculator lesk = new Lesk(db);
    POS posWord1= POS.n;
    POS posWord2= POS.n;
    double maxScore = 0;

    WS4JConfiguration.getInstance().setMFS(true);

    List<Concept> synsets1 = (List<Concept>) db.getAllConcepts(word1, posWord1.name());
    List<Concept> synsets2 = (List<Concept>) db.getAllConcepts(word2, posWord2.name());

    for (Concept synset1: synsets1) {
        for (Concept synset2: synsets2) {
            Relatedness relatedness = lesk.calcRelatednessOfSynset(synset1, synset2);
            double score = relatedness.getScore();
            if (score > maxScore) {
                maxScore = score;
            }
        }
    }
    if (maxScore == -1D) {
        maxScore = 0.0;
    }
    return maxScore;
}

private static RelatednessCalculator[] rcs = { new HirstStOnge(db),
        new LeacockChodorow(db), new Lesk(db), new WuPalmer(db),
        new Resnik(db), new JiangConrath(db), new Lin(db), new Path(db)
};

private static double compute(String word1, String word2) {
    WS4JConfiguration.getInstance().setMFS(true);
    return new WuPalmer(db).calcRelatednessOfWords(word1, word2);
}

```

3.5. Data Layer

3.5.1. Persistence database

Future references on information regarding past analyzed threat models will be an added advantage if the machine learning techniques to be applied in the future to improve the accuracy and the usability of the application thus, the persistent DB will provide long-term storage support of analyzed threats and generated analytical information. This DB also contains the vulnerability databases of CWE, CAPEC, OWASP top 10, SANS Top 25 along with mitigations for each threat.

3.5.2. Lexical DB

Lexical DB contains mappings of semantic similarities of words in English language and is being used to generate the semantic similarities of sentences to match MSTMT threats against vulnerability databases.

3.6. Publisher Interface

This layer will logically group application modules related to outputs of the system like files, web service messages, etc together to provide Abstraction.

3.6.1. REST publisher for JIRA

This component in the Publisher interface is a concrete implementation of REST API to integrate the system with JIRA Project Management tool. It can publish issues, threats to the JIRA from an extracted and refined threat list to provide visibility and tracking for identified issues during SDLC.

3.6.2. Publisher 2 (passive)

This is an abstract implementation of a integration with a different PM tool which can accommodate in a future development to provide scalability and support to different PM tools.

3.7. Tools and Technologies

- Programming Language : Java 8 EE.
- IDE : Eclipse Oxygen
- Application Server : Apache Tomcat 9.
- Build and Dependency Management : Apache Maven 3.
- Threat Modeling : Microsoft Threat Modeling Tool.
- Project Management Tool : Atlassian Jira.
- Core Framework : Spring MVC 4.1.0
- Integration API : REST
- Persistence DB : Postgres 10.0.
- Lexical DB : Word net lexical db 3.1.

3.7.1. Core libraries

Table 1. Core Libraries

Library	Version	Description
tomcat-servlet-api	9.0.2	The package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the run-time environment provided for an instance of such a class by a conforming servlet container
jackson-databind	2.4.1	Contains basic mapper (conversion) functionality that allows for converting between regular streaming json content and Java objects.
sitemesh	2.4.2	SiteMesh is a lightweight and flexible Java web application framework that applies the Gang of Four decorator pattern to allow a clean separation of content from presentation.
jsoup	1.8.3	Jsoup is an Open source Java HTML parser with CSS, DOM and jquery-like methods for easy data extraction.
json	20090211	JSON (JavaScript Object Notation) is a lightweight data-interchange format which is used to implement REST-API messages.
commons-codec	1.9	The Codec package from Apache Commons contains simple encoder and decoders for various formats such as Base64, UTF-8, Hexadecimal, etc.
json-simple	1.1.1	json-simple uses Map and List internally for JSON processing and can use for parsing JSON data as well as writing JSON to file.
Ws4j	0.1.0	WordNet Similarity for Java provides an API for several Semantic Relatedness/Similarity algorithms
commons-io	2.0.1	The Apache Commons IO library contains utility classes, stream implementations, file filters, file comparators, endian transformation

		classes, and much more
jersey-client	1.19.4	Jersey is an open source framework for developing RESTful Web Services.
spring-web-mvc	4.3.0	Spring mvc is a popular java framework for building scalable web applications.

Table 1 above lists down the core libraries used when implementing the framework. Extracted mitigations from inference engine will be passed to front-end as a formatted output. The Java EE Spring framework packages will facilitate the MVC architecture based web application implementation containing all the core functionalities of a web application. Detailed view of the web application in action will be discussed in the next chapter. The "tmm.tmp.*" packages will contain the logic and object implementation of MSTMT parser and HTML parser as well as publisher logic for the web service integrating JIRA. The database itself will act as the knowledge base containing all the information regarding threats, appropriate mitigations, best practices, ranking and priorities of threats, previous experiences, etc.

4. Testing and Evaluation

Three case studies have been used for the evaluation covering web applications with a sample library application, e-commerce portals with a sample e-commerce application and cloud-based network virtual appliances with Microsoft Azure HA NVD [19]. Formal verification of functional and non-functional requirements of the KOSALAK framework has been carried out with developer testing to ensure that the software behaves as intended in the requirements and design specifications. Unit testing has been done using JUnit framework to verify micro implementations and modules such as methods and functions. Integration testing has been conducted to test integration between threat model file and the parser, final report, parser and the web application and the parser and integration between KOSALAK web application and JIRA project management tool.



The results from each test have been manually and automatically validated and verified. The application is intended to function in real-time and a performance test is in accordance to verify that non-functional requirement. The user needs to have an adequate UI/UX to interact proactively with the tool and this has been tested accordingly. The application should not break when analyzing industrial scale threat models and the case study 3 covering real-world scenario of Microsoft Azure High Availability Network Virtual Appliance implementation using docker technology [19] has been used to test that and will be discussed in subsequent chapters in detail.

4.1. Evaluation Process

For the evaluation of the framework an MSTMT threat model file which contains a Level-0 or Level-0 DFD along with its final report has to be fed into the system. The tool then parse input files and extract threat model information. Extracted threats will then be mapped against OWASP Top 10 Security threats, CAPEC, CWE/SANS Top 25 threats to identify the high priority security threats using semantic similarity comparison techniques. Appropriate mitigations and the best practices will be generated for each threat from pre-defined templates. The user can validate the findings later then submit the selected threats and mitigations to the JIRA as the endpoint. The reason behind picking JIRA as the supported PM tool is that is has been a popular PM tool in the software institutions. If the Build Security In concept is to be enforced it must support existing SDLC tools. The MSTMT threat model is developed following STRIDE threat categorization technique. Evaluating the output with respect to different categorization technique will prove valuable to identify if there are false positive threats. The accuracy of this false positive identification purely relies on the semantic similarity between MSTMT threats and the OWASP top 10 and CWE/SANS top 25 threats descriptions. Generation of mitigations and best practices are selected from predefined

templates since the threat modeling is only focusing on known security vulnerabilities. ASF classification will look at generating mitigations from a defensive perspective and can be used to identify the appropriate security controls for a particular threat. OWASP top 10 security controls also proved valuable for the same concern since there also predefined mitigations and best practices have been illustrated. CWE/SANS also introduced countermeasures for known security threats, and these enumerations have also concerned when generating mitigation techniques within the application.

Three case studies have been identified covering popular software application development areas. A sample library application which will focus on general web applications has been selected as the first case study. In this threat model, a typical simple web application has been illustrated. An e-commerce portal has been selected as the second case study adding more advanced components and concepts to elaborate the effectiveness and the uses of the KOSALAK tool in general software engineering. Finally, as the third case study, a cloud-based infrastructure platform has been selected to cover the Dev-Ops, Networking, and Systems Engineering concepts using the framework. Threat models and the styling will vary based on the user who developed that and is highly people dependent. No common standard is practiced when creating threat models but, options available in MSTMT will set a boundary on threat model creation. Each threat model will carry the perspective and the impression that the user had on the software application when creating the threat model.

Below Fig. 6 illustrates the home screen of the KOSALAK threat model analyzer in while on the action. Input files for the framework can be selected from given file selection buttons. And by ticking the checkbox the user can activate the inference engine. Identified inconclusive threats or false positives will be marked with the  icon and by clicking on the  icon will allow the user to expand the information and see mitigations best practices, etc related to a particular threat.

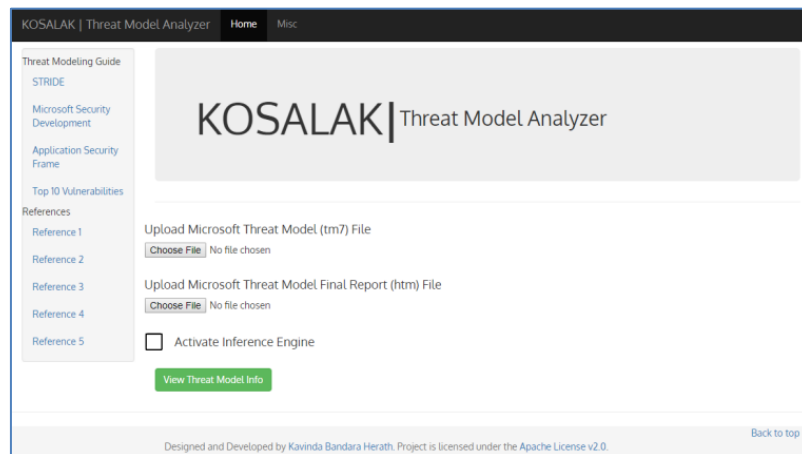


Fig. 6. KOSALAK - home screen.

4.2. Evaluation Results

All the mitigations have been presented in the same format where OWASP top 10 countermeasures have been presented on their web site [10]. In below case studies same views of mitigations have been used for OWASP Top 10 related threats and vulnerabilities. If a particular threat cannot match against OWASP top 10 vulnerability then it will check in subsequent vulnerability databases CWE [11] and CAPEC[12]. Top 5 rating threats will then get selected when generating mitigations and displayed using the same format. When creating the threat models for case studies user can set the status of threats as mitigated, or not applicable and those will not get loaded into the application.

4.2.1. Case study 1: Simple web-based library application

Threat model DFD developed using MSTMT for the architectural risk analysis of the web-based library application is illustrated in the below figure. MSTMT 2016 has generated a set of known potential threats associated with the drawn DFD following STRIDE categorization technique. Main components of the system and the connectivity, communication and the integration along with trust boundaries have been drawn using templates found in the MSTMT 2016. Below Fig. 7, and Fig. 8 will illustrate the output from the KOSALAK Threat Model Analyzer once this threat model has been loaded to the application. Visual representation of the threat model have been extracted from the “htm” file since the users need to visualize the threat model.

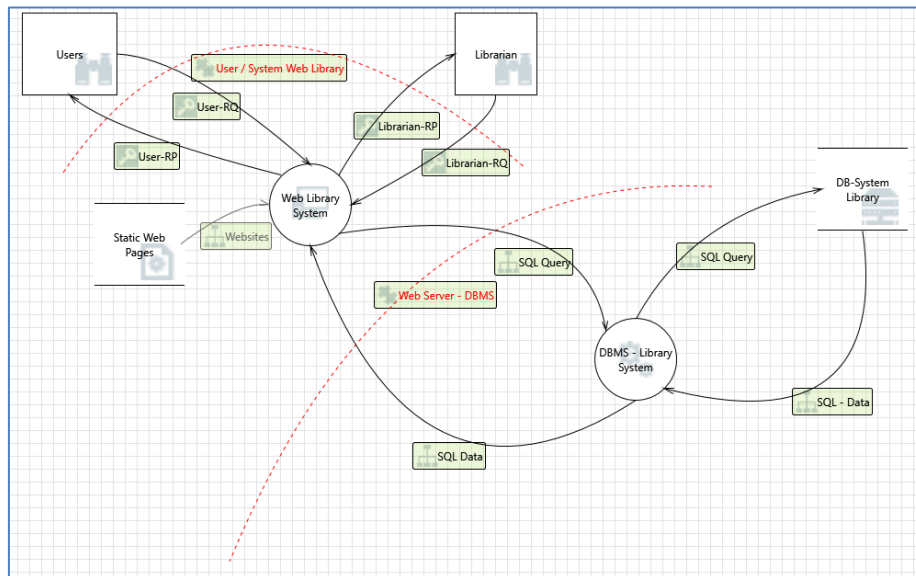


Fig. 7. Simple web based library system - threat model.

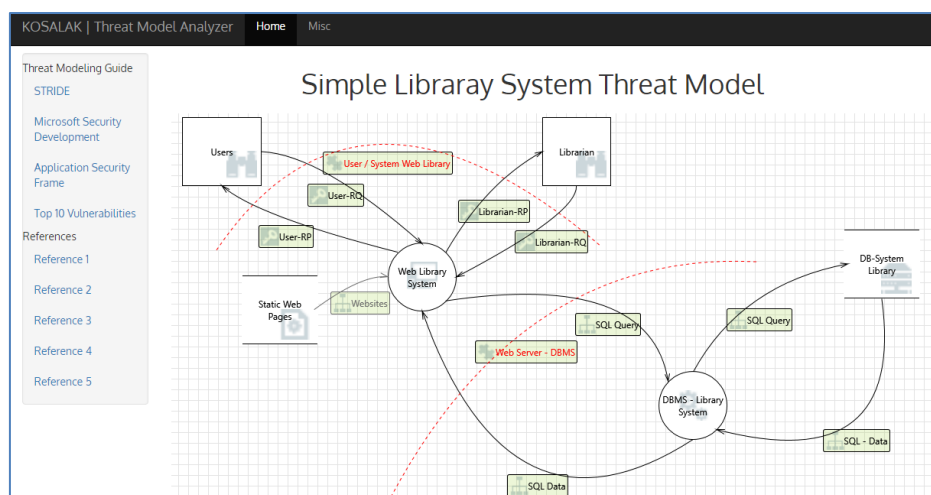




Fig. 8. Simple web based library system - loaded into KOSALAK.

Below snapshot taken in Fig. 9 will illustrate how the KOSALAK Threat Model Analyzer displays the extracted threats and the false positives. Marked rows with the  icon indicated potential false positives or low priority threats and by clicking on the  icon can expand the row to see the in-depth analysis of the threat and mitigations and best practices.

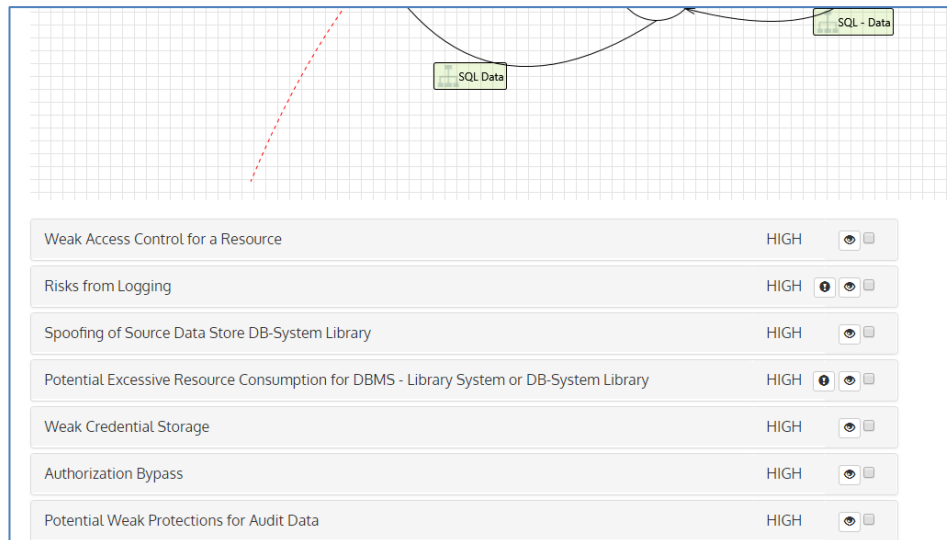


Fig. 9. Simple web based library system - loaded threat list.

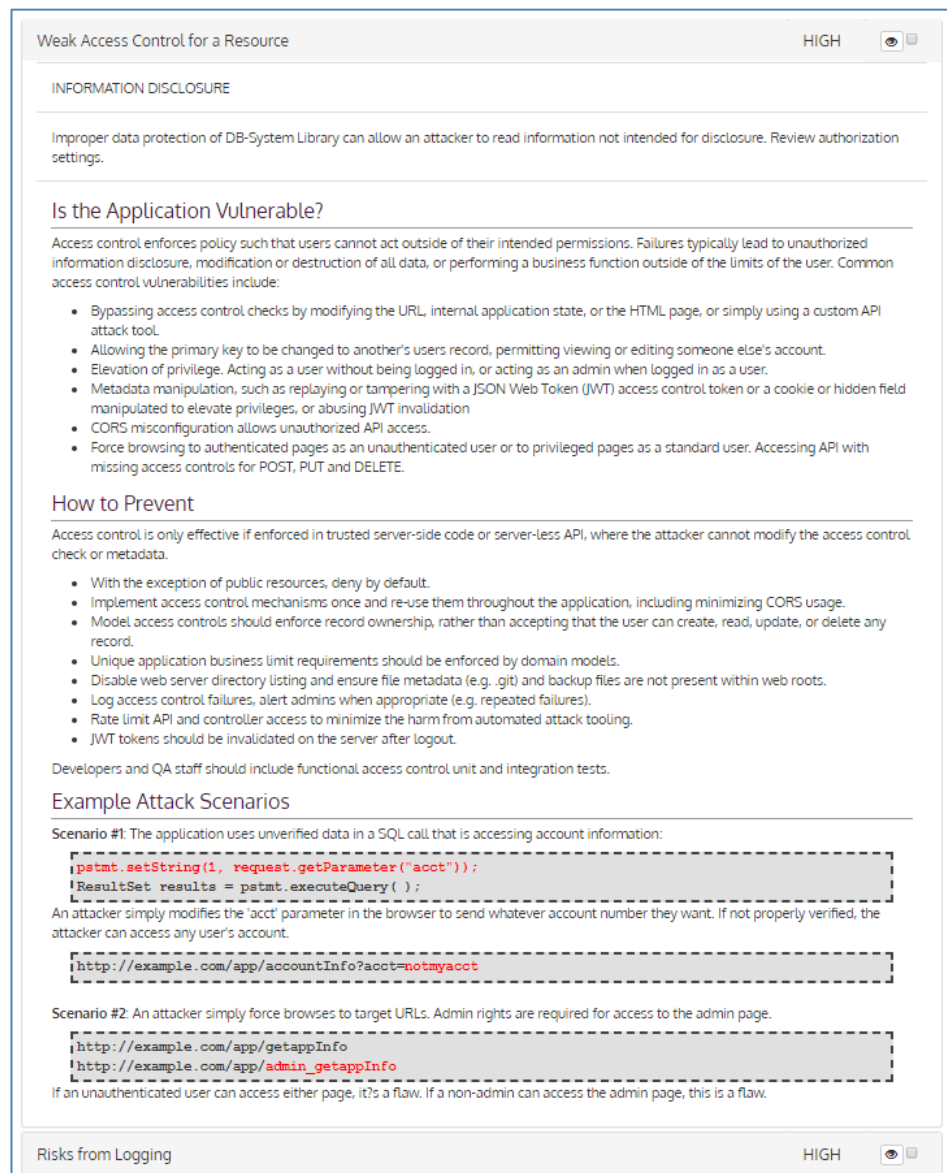


Fig. 10. Simple web based library system - loaded mitigation1.

Weak Credential Storage
HIGH

INFORMATION DISCLOSURE

Credentials held at the server are often disclosed or tampered with and credentials stored on the client are often stolen. For server side, consider storing a salted hash of the credentials instead of storing the credentials themselves. If this is not possible due to business requirements, be sure to encrypt the credentials before storage, using an SDL-approved mechanism. For client side, if storing credentials is required, encrypt them and protect the data store in which they're stored

Is the Application Vulnerable?

An application is vulnerable to attack when:

- A password is stored in plaintext in an application's properties or configuration file. A plaintext password in a configuration file allows anyone who can read the file access to the password-protected resource.
- Missing a security tactics during the architecture and design phase.
- Storing passwords in easily accessible locations.
- Failure to encrypt critical data.
- Insecure storage of keys, certificates, and passwords.
- Improper storage of secrets in memory.
- Poor choice of algorithm.
- Failure to include support for encryption key changes and other required maintenance procedures.

How to Prevent

Preventing requires keeping data secure.

- Avoid storing passwords in easily accessible locations.
- To protect against cryptographic flaws is to minimize the use of encryption and only keep information that is absolutely necessary.
- If cryptography must be used, choose a library that has been exposed to public scrutiny and make sure that there are no open vulnerabilities.
- Be sure that secrets, such as keys, certificates, and passwords, are stored securely.
- To make it difficult for an attacker, the master secret should be split into at least two locations and assembled at runtime. Such locations might include a configuration file, an external server, or within the code itself.
- Use a cryptographically strong credential-specific salt.
- Considering the threats you plan to protect this data from (e.g., insider attack, external user), make sure you encrypt all such data at rest in a manner that defends against these threats.
- Ensure offsite backups are encrypted, but the keys are managed and backed up separately.

Example Attack Scenarios

Scenario #1: The following code reads a password from a properties file and uses the password to connect to a database.

```

...
Properties prop = new Properties();
prop.load(new FileInputStream("config.properties"));
String password = prop.getProperty("password");
DriverManager.getConnection(url, usr, password);
...

```

This code will run successfully, but anyone who has access to config.properties can read the value of password. If a devious employee has access to this information, they can use it to break into the system.

Scenario #2: An application encrypts credit cards in a database to prevent exposure to end users. However, the database is set to automatically decrypt queries against the credit card columns, allowing an SQL injection flaw to retrieve all the credit cards in plaintext. The system should have been configured to allow only back end applications to decrypt them, not the front end web application.

Scenario #3: A backup tape is made of encrypted health records, but the encryption key is on the same backup. The tape never arrives at the backup center.

Scenario #4: The password database uses unsalted hashes to store everyone's passwords. A file upload flaw allows an attacker to retrieve the password file. All the unsalted hashes can be brute forced in 4 weeks, while properly salted hashes would have taken over 3000 years.

Authorization Bypass
HIGH

Fig. 11. Simple web based library system - loaded mitigation2.

Ticking on the checkbox after reviewing and discussing with the team will mark that particular threat eligible to be exported to the JIRA along with all the information. Fig. 10 and Fig. 11 above is a snapshot taken to illustrates how the application shows a generated mitigation to a selected issue by clicking on the icon. Mitigations and best practices have been derived with respect to OWASP top 10 security controls for this particular threat. The full list of extracted include 56 potential threats for this threat model. Below shows one such threat where it illustrates the category of the threat, description along with additional

information, how to prevent such an incident, example attack scenarios. This way when the team evaluates each of these threats they can educate team members about the best practices and guidelines to follow when implementing the application.

4.2.2. Case study 2: Simple e-commerce web application

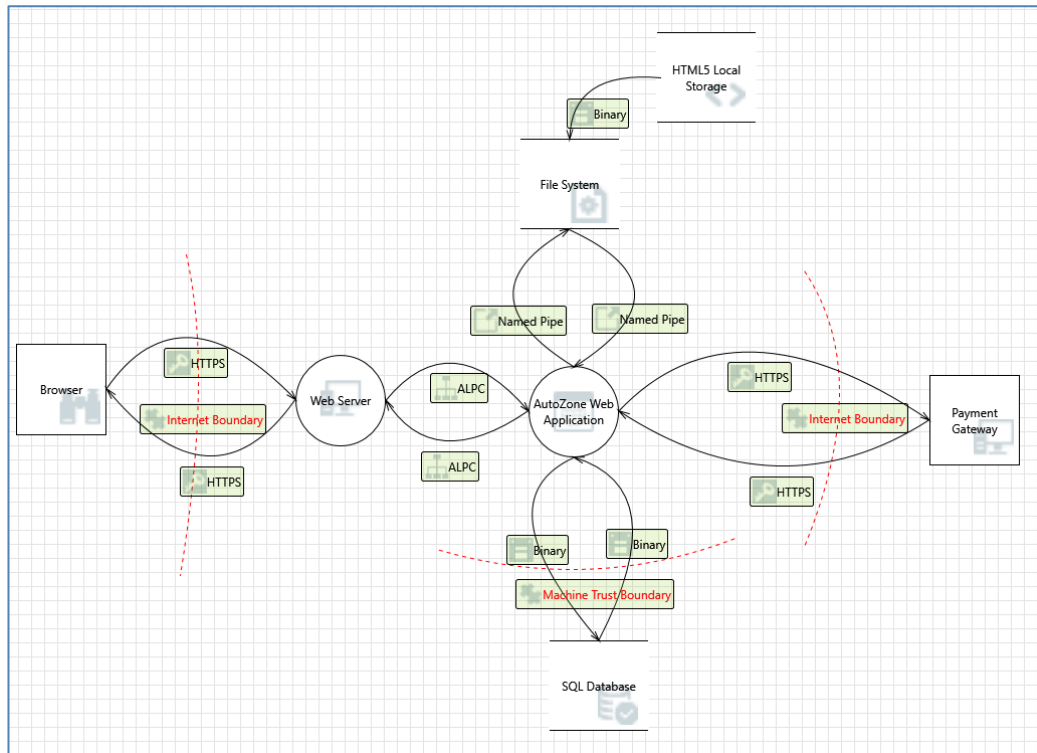


Fig. 12. Simple e-commerce web application - threat model.

Potential Data Repudiation by Web Server	HIGH		<input type="checkbox"/>
Cross Site Scripting	HIGH		<input type="checkbox"/>
Spoofing the Browser External Entity	HIGH		<input type="checkbox"/>
Elevation Using Impersonation	HIGH		<input type="checkbox"/>
Cross Site Scripting	HIGH		<input type="checkbox"/>
Elevation Using Impersonation	HIGH		<input type="checkbox"/>
Cross Site Scripting	HIGH		<input type="checkbox"/>
Web Server Process Memory Tampered	HIGH		<input type="checkbox"/>
Spoofing of Destination Data Store SQL Database	HIGH		<input type="checkbox"/>
Potential SQL Injection Vulnerability for SQL Database	HIGH		<input type="checkbox"/>
Potential Excessive Resource Consumption for AutoZone Web Application or SQL Database	HIGH		<input type="checkbox"/>
Spoofing of Source Data Store SQL Database	HIGH		<input type="checkbox"/>

Fig. 13. Simple e-commerce system - loaded threats.

For this case study also threat model DFD was developed using MSTMT for the architectural risk analysis

of the e-commerce web application and is illustrated in the above Fig. 12. MSTMT 2016 has generated a set of known potential threats associated with the drawn DFD following STRIDE categorization technique. Main components of the system and the connectivity, communication and the integration along with trust boundaries have been drawn using templates found in the MSTMT 2016. Fig. 13 will illustrate the threat list output from the KOSALAK Threat Model Analyzer once this threat model has been loaded to the application along with the report file. The full list of extracted include 57 potential threats for this threat model. As a sample below Fig. 14 displays a SQL injection-related threat and mitigations in detailed format after clicking on the relevant tab with the threat. This feature can be used to educate the team on how to follow the best practices to avoid these known security vulnerabilities.

Potential SQL Injection Vulnerability for SQL Database
HIGH

TAMPERING

SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. Any procedure that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. Even parameterized data can be manipulated by a skilled and determined attacker.

Is the Application Vulnerable?

An application is vulnerable to attack when:

- User-supplied data is not validated, filtered, or sanitized by the application.
- Dynamic queries or non-parameterized calls without context-aware escaping are used directly in the interpreter.
- Hostile data is used within object-relational mapping (ORM) search parameters to extract additional, sensitive records.
- Hostile data is directly used or concatenated, such that the SQL or command contains both structure and hostile data in dynamic queries, commands, or stored procedures.
- Some of the more common injections are SQL, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injection. The concept is identical among all interpreters. Source code review is the best method of detecting if applications are vulnerable to injections, closely followed by thorough automated testing of all parameters, headers, URL, cookies, JSON, SOAP, and XML data inputs. Organizations can include static source ([SAST](#)) and dynamic application test ([DAST](#)) tools into the CI/CD pipeline to identify newly introduced injection flaws prior to production deployment.

How to Prevent

Preventing injection requires keeping data separate from commands and queries.

- The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a parameterized interface, or migrate to use Object Relational Mapping Tools (ORMs).
Note: Even when parameterized, stored procedures can still introduce SQL injection if PL/SQL or T-SQL concatenates queries and data, or executes hostile data with EXECUTE IMMEDIATE or exec().
- Use positive or "whitelist" server-side input validation. This is not a complete defense as many applications require special characters, such as text areas or APIs for mobile applications.
- For any residual dynamic queries, escape special characters using the specific escape syntax for that interpreter.
Note: SQL structure such as table names, column names, and so on cannot be escaped, and thus user-supplied structure names are dangerous. This is a common issue in report-writing software.
- Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL injection.

Example Attack Scenarios

Scenario #1: An application uses untrusted data in the construction of the following **vulnerable** SQL call:

```
String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";
```

Scenario #2: Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query hqlQuery = session.createQuery("FROM accounts WHERE custID='" + request.getParameter("id") + "'");
```

In both cases, the attacker modifies the ?id? parameter value in their browser to send: ' or '1'=1. For example:

```
http://example.com/app/accountView?id=' or '1'=1
```

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

Fig. 14. Simple e-commerce system - loaded mitigation.

Selected threat and the best practices can be published to PM tool supported by ticking on the threat head and then clicking on the publish button at the bottom linking with a development item. The user assigned the item will be able to refer this information and will be enforced to follow the best practices when implementing the feature or the functionality. The management and leads can monitor the security building progress via the JIRA. Fig. 15 below is a snapshot taken to illustrates how the threat is presented in JIRA project Management tool's Dashboard once it is published.

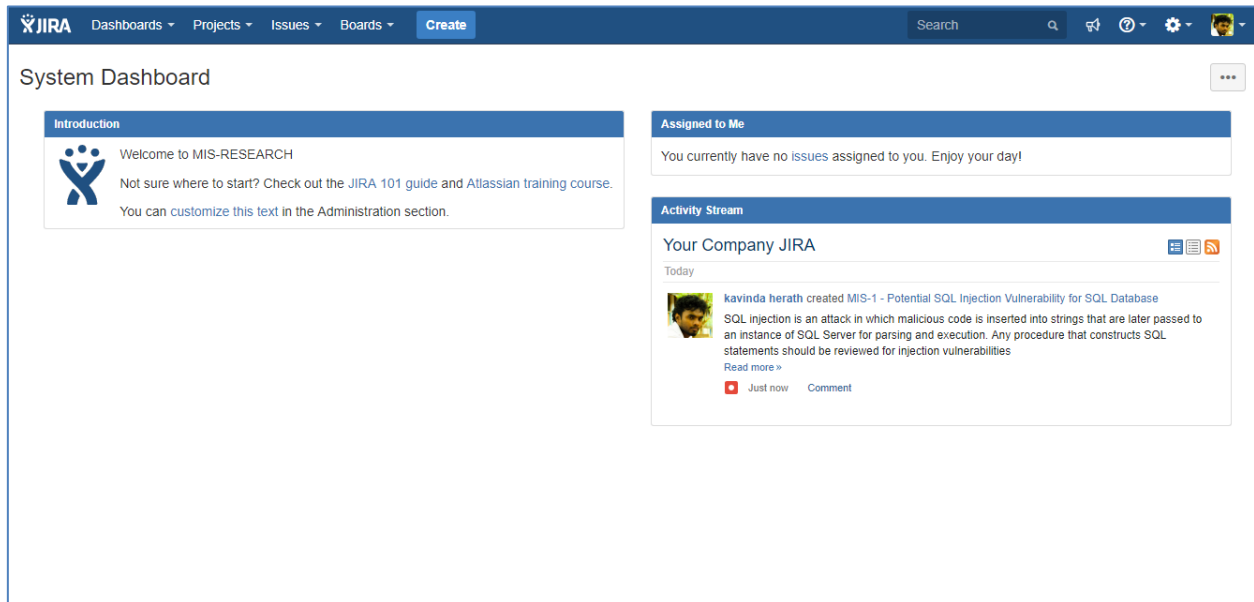


Fig. 15. JIRA - published threat and mitigation.

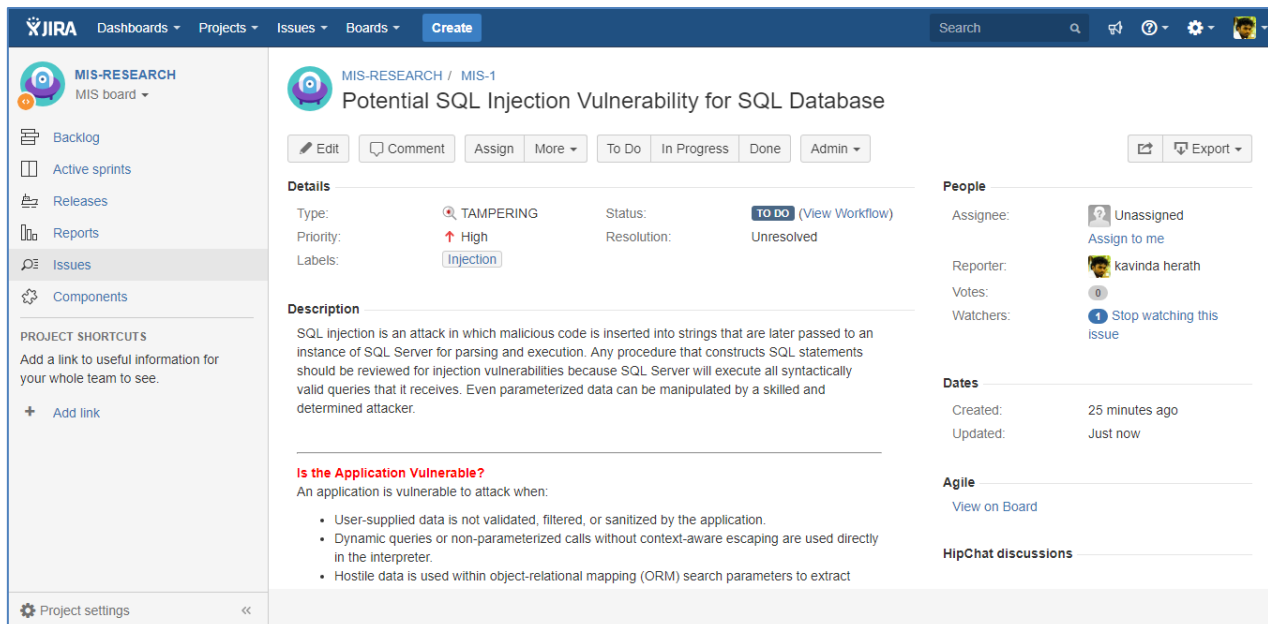


Fig. 16. JIRA - published threat and mitigation - detail.

Fig. 16 and Fig. 17 shows how the published threat is being available in a detailed view with all the information associated and the mitigations and the best practices generated. Any developer who's going to work on the particular item can gain insight on how to properly code to avoid known security bugs. Additionally these threats can be linked up with the development items to provide more visibility.

Example Attack Scenarios

Scenario #1:

An application uses untrusted data in the construction of the following **vulnerable** SQL call:

```
String query = "SELECT * FROM accounts WHERE custID=" + request.getParameter("id") + "";
```

Scenario #2:

Similarly, an application's blind trust in frameworks may result in queries that are still vulnerable, (e.g. Hibernate Query Language (HQL)):

```
Query HQLQuery = session.createQuery("FROM accounts WHERE custID=" + request.getParameter("id") + "");
```

In both cases, the attacker modifies the ?id? parameter value in their browser to send: ' or '1'=1. For example:

<http://example.com/app/accountView?id=' or '1'=1>

This changes the meaning of both queries to return all the records from the accounts table. More dangerous attacks could modify or delete data, or even invoke stored procedures.

Attachments

Drop files to attach, or [browse](#).



simple-e-commerce-site.png
Just now 80 kB

Activity

All Comments Work Log History Activity

Fig. 17. JIRA - published threat and mitigation - detail.

4.2.3. Case study 3: Microsoft azure high availability network virtual appliance implementation using docker technology [19]

This case study is a industrial scale real world scenario based threat model DFD which was developed using MSTMT for the architectural risk analysis of the cloud based HA-NVD implementation and is illustrated in the below figure. MSTMT 2016 has generated a set of known potential threats associated with the drawn DFD following STRIDE categorization technique. Main components of the system and the connectivity, communication and the integration along with trust boundaries have been drawn using templates found in the MSTMT 2016. Fig. 18 illustrates the threat model associated with this case study.

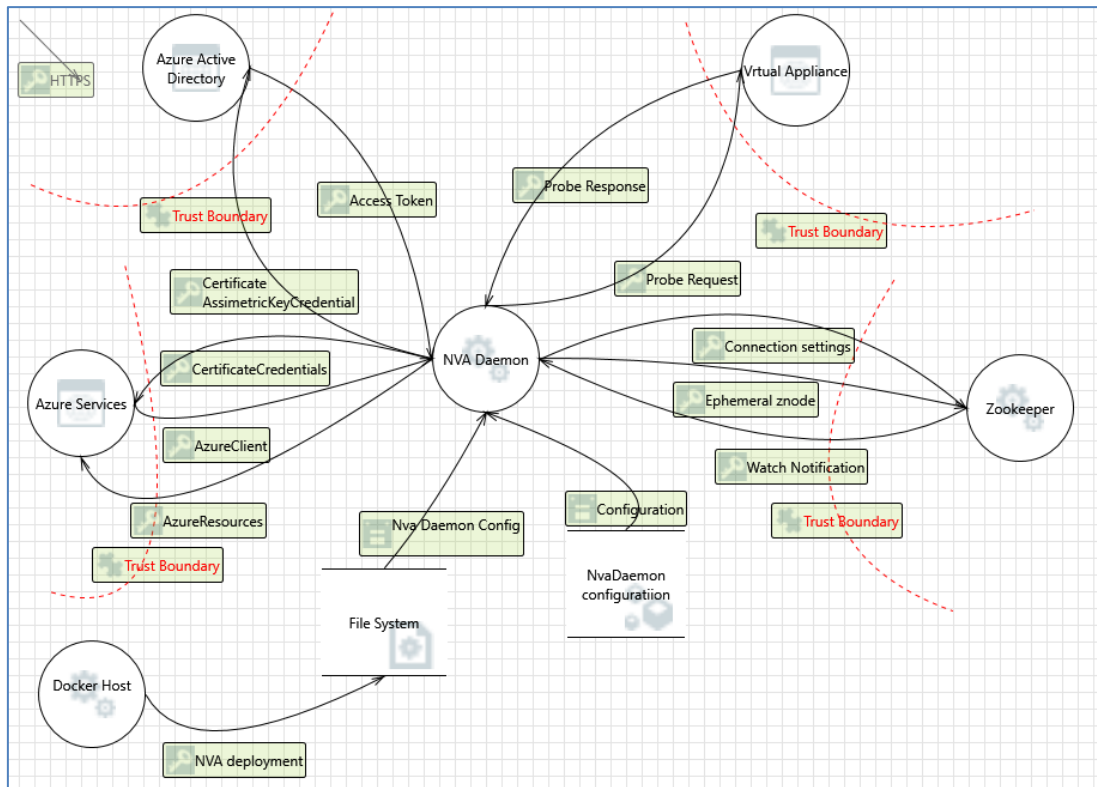


Fig. 18. HA-NVD system - threat model.

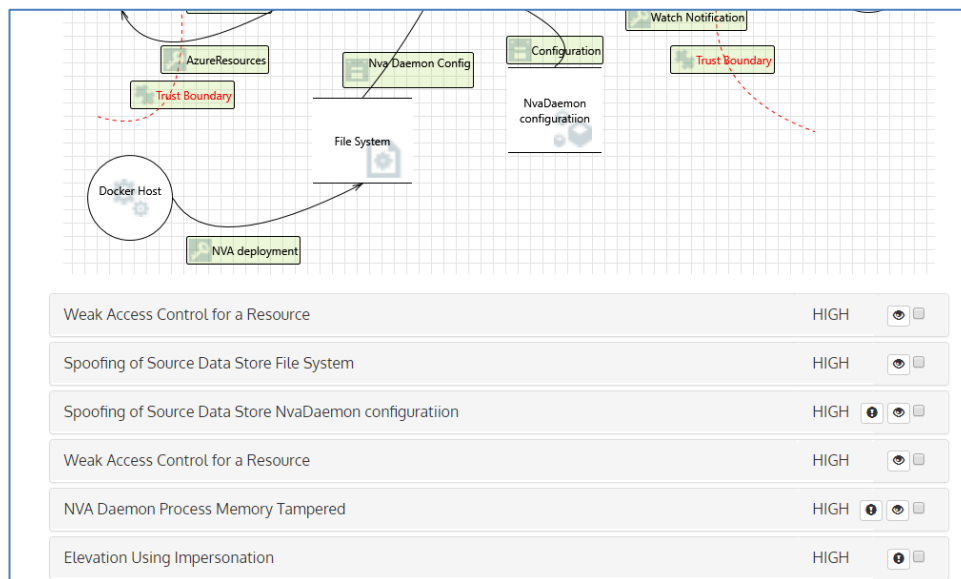


Fig. 19. HA-NVD system - loaded threats.

The full list of extracted include 91 potential threats for this threat model and can be seen in the snapshot taken in Fig. 19. The same level of application behavior can be observed in this case study and the application generates mitigations following the same format and mechanism used during previous two case studies.

5. Conclusion

The research proposes means and methods to improve the existing SDLC in terms of Secure Software Engineering. The particular field of research is focusing on deriving mitigations and best practices to

identified known security risks in a software design and to reduce the unwanted cost associated with low priority vulnerabilities and to provide visibility and guidance to all the stakeholders in SDLC on how to incorporate Build Security In to software products. A tool has been implemented as a proof of concept to assist the Developers, Leads, Architects, QA and Management on secure software development. The results and the accuracy are depended on many factors and the available knowledge in the field of threat modeling is proved to be inadequate in certain situations. The current dissertation will add valuable features to the field of threat modeling and secure software engineering which will provide more opportunities for the future growth of the field.

Mitigation techniques generated will give a helping hand to developers when coding since they can be aware of the best practices to avoid introducing known security vulnerabilities to the software products. QA will have a better insight on how to test the security of a software product when doing quality assurance analyst tasks focusing on software security. Management can have an insight about building security into the software application and can provide visibility, tracking and monitoring throughout SDLC and educating all stakeholders in an SDLC is also an additional advantage with the help of the KOSALAK framework.

References

- [1] Grechanik, M., McMillan, C., *et al.* (2014). Redacting sensitive information in software artifacts. *Proceedings of the 22nd International Conference on Program Comprehension, ICPC 2014* (pp. 314-325).
- [2] Armerding, T. (2018). The 17 biggest data breaches of the 21st century. Retrieved from the website: www.csoononline.com/article/2130877/data-breach/the-biggest-data-breaches-of-the-21st-century.html
- [3] McGraw, G. (2006). Software security: Building security In. *Proceedings of the 2006 17th International Symposium on Software Reliability Engineering* (p. 6).
- [4] Howard, S. L. M. (2006). *The Security Development Lifecycle* (pp. 5-11). Redmond, WA: Microsoft Press.
- [5] Bilge, L., & Dumitras, T. (2013). Before we knew it: An empirical study of zero-day attacks in the real world. *CCS*, 38(4), 6.
- [6] Arce, I., Clark-Fisher, K., Daswani, N., DelGrosso, J., Dhillon, D., Kern, C., Kohno, T., Landwehr, C., McGraw, G., Schoenfield, B., Seltzer, M., Spinellis, D., Tarandach, I., & West, J. (2014). Avoiding the top 10 software security design flaws. *IEEE Computer Societys Center for Secure Design (CSD)*.
- [7] Microsoft Corporation. (2004). Evolution of the Microsoft SDL - Microsoft. Retrieved from the website: <https://www.microsoft.com/en-us/SDL/Resources/evolution.aspx>
- [8] Kazman, R. (2016). A tool to address cybersecurity vulnerabilites through design. Software Engineering Institute, Carnegie Mellon University. Retrieved from the website: https://insights.sei.cmu.edu/sei_blog/2016/02/a-tool-to-address-cybersecurity-vulnerabilities-through-design.html
- [9] Ruffy, F., Hommel, W., & Von Eye, F. (2016). A STRIDE-based security architecture for software-defined networking. *Proceedings of ICN 2016: The Fifteenth International Conference on Networks* (pp. 95-100).
- [10] OWASP. (2017). Top 10-2017 application security risks. Retrieved from the website: www.owasp.org/index.php/Top_10-2017_Application_Security_Risks
- [11] MITRE. (2011). Common weakness enumeration. CWE - 2011 CWE/SANS top 25 most dangerous software errors. Retrieved from the website: <https://cwe.mitre.org/top25/>
- [12] MITRE. (2011). Common attack pattern enumeration and classification. CAPEC - CAPEC list version 2.11. Retrieved from the website: <https://capec.mitre.org/data/index.html>
- [13] International Organization for Standardization. (2011). Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

ISO/IEC 25010:2011.

- [14] Palmaers, T., & Distler, D. (2013). Implementing a vulnerability management process - SANS institute reading room. *White Papers*. Retrieved from the website: <https://www.sans.org/reading-room/whitepapers/threats/implementing-vulnerability-management-process-34180>
- [15] Avital, N., & Elul, N. (2017). The state of web application vulnerabilities in 2017. *Blog | Imperva*. Retrieved from the website: www.imperva.com/blog/2017/12/the-state-of-web-application-vulnerabilities-in-2017/
- [16] Curran, P. (2016). 3 web application security lessons from recent vulnerabilities and exploits. *Checkmarx*. Retrieved from the website: www.checkmarx.com/2016/11/13/3-web-application-security-lessons-recent-vulnerabilities-exploits
- [17] McGraw, G. (2009). Software security touchpoint: Architectural risk analysis. *Software Security Building Security In.: Addison Wesley Software Security Series*. Retrieved from the website: <https://www.cigital.com/presentations/ARA10.pdf>
- [18] Vanciu, L. R. (2014). *Static Extraction of Dataflow Communication for Security*. Ph.D. dissertation, Wayne State University, Michigan.
- [19] GitHub. High availability NVA on Microsoft azure. Retrieved from the website: <https://github.com/mspnp/ha-nva>



H. M. K. K. B. Herath received his bachelor's degree in computer science from University of Colombo School of Computing, Sri Lanka in 2015 and master's degree in information security from University of Colombo School of Computing, Sri Lanka in 2018 and work as a lead software engineer for an US based software engineering firm. His current research interests are information security and DevOps, high availability/traffic systems, cloud computing, machine learning, and human computer interaction.



G. D. S. P. Wimalaratne obtained his BSc special degree in computer science from the University of Colombo and his PhD in virtual environments from the University of Salford, United Kingdom in 2002. He is a senior member of IEEE and a member of Computer Society of Sri Lanka (CSSL). His research interests include virtual/augmented reality, secure software systems and assistive technology. He joined the academic staff of the University of Colombo in 1995 and is currently the head of the Department of Communication and Media Technologies at University of Colombo

School of Computing. He was the former director of the Career Guidance Unit at the University of Colombo. He has served as a council member of the Sri Lanka Association for the Software Industry (SLASI), vice chair of Institute of Electrical and Electronic Engineers (IEEE) Sri Lanka Section and a council member of the Computer Society of Sri Lanka (CSSL).