A STUDY ON EFFECTIVENESS OF SOFTWARE VULNERABILITY ASSESSMENT FOR COMPONENT-BASED SOFTWARE DEVELOPMENT

K.L. Dasun

138205B

Degree of Master of Science/Master of Engineering

Department of Computer Science and Engineering

University of Moratuwa
Sri Lanka

December 2016

A STUDY ON EFFECTIVENESS OF SOFTWARE VULNERABILITY ASSESSMENT FOR COMPONENT-BASED SOFTWARE DEVELOPMENT

K.L. Dasun

138205B

Thesis/Dissertation submitted in partial fulfilment of the requirements for the degree Master of Science/Master of Engineering in Computer Science and Engineering

Department of Computer Science and Engineering

University of Moratuwa

Sri Lanka

December 2016

Declaration

I declare that this is my own work and this dissertation does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Furthermore, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

…… …….……..

K.L.Dasun Date

The above candidate has carried out research for the Masters Dissertation under my supervision.

…… ……..

Dr. Chandana Gamage Date

(Research Supervisor)

Abstract

Security is an essential aspect for software development as many critical and vital functions, systems and services are now controlled by software. Operating systems to middleware to applications, integrated systems to embedded systems to firmware, and networks of all sizes and complexities are now controlled and managed by software. Thus, assurance of security in such software and thereby the protection of sensitive data is essential.

Due to the complexity, scalability and maintainability factors, the software industry is moving rapidly towards component-based systems development where various artefacts are integrated to achieve a variety of functionality. This integration occurs in different phases in the life cycle of a system and usually at a rapid pace. Therefore, it is doubtful if the correct level of emphasis is placed in the development process to assure the security of composing a system with such diverse components, even if they have a high level of security individually.

While there are many tools to test the potential for exploitation of vulnerabilities in software systems, these tools are most often optimized to test certain application scenarios, development phases, and specific software categories or methodologies. Therefore, with the increasing use of composed development of software systems and also the expansion in the tools and techniques available for software vulnerability exploitation, it is vital to evaluate the effectiveness of existing vulnerability assessment scheme on composed software development. This research is focused on determining the direction for improved effectiveness of software vulnerability tools in the composed system development paradigm.

Acknowledgements

I would like to express my special appreciation and thanks to my supervisor Dr Chandana Gamage, you have been a tremendous mentor to me. I would like to thank you for guiding me through your experience to make this research more worthwhile.

I would also like to take this opportunity to thank Dr Malaka Walpola, Dr Shehan Perera for guiding us in Research Seminar lecture sessions and for the extended support and kindness granted to us. At last but not least, I would thank all the academic staff members for helping, guiding, encouraging us and disseminating knowledge throughout the program.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Description
SDLC	Software Development Life cycle
COTS	Commercial Off The Shelf
FOSS	Free and Open Source Software
CBSE	Component Based Software Engineering
CBSD	Component Base Software Development
SAST	Static Application Security Testing
DAST	Dynamic Application Security Testing
IAST	Interactive application Security Testing
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TPR	True Positive Rate
FPR	False Positive Rate
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration

1. Introduction

Security is an essential aspect since from the first appearance of human society. Therefore, security concepts like confidentiality, integrity, and authenticity, as well as means of achieving them through encryption, decryption and, secure communication have become buzz words. The greater accessibility of computers to people and the widespread use of software applications for nearly every aspect of our daily lives have significantly increased the necessity of computer security. Even with such demand and the focus on security from the inception of computer systems development, the level of maturity we have achieved in software security is alarmingly low and is exemplified by the continuing exposes on security breaches and losses due to the exploitation of software vulnerabilities. Even though the software is mainly responsible for security vulnerabilities in modern systems, it only became a hot topic as a significant research area only in the late ‘90s [1].

Most of the vulnerability assessments are focused on detecting vulnerabilities in the deployed environment. Therefore, those assessments become overall system vulnerabilities rather than of the software of the particular system, which might be too late to detect and fix. Therefore, identification of the vulnerability of software is essential when it is in the early stages of the development.

1.1 Software Security

The software has become a necessity in day to day life and visible in every aspect of our work areas such as operating systems to middleware, embedded systems, firmware, and networking; almost everything runs on software directly or with the assistance of it to a certain extent. As we belong to a world which is dominated by information; software controls most mission-critical, life-threatening systems. Software protects most sensitive information, and moreover, the securing of the software itself is also done by software.

The software can be vulnerable due to several reasons such as complexity of the code and structure, deficiencies in development methodologies as well as developer capabilities and non-compliance to standards.With time, software becomes more complex and difficult to comprehend by humans without looking in depth, and that leads to a need for special tool support for debugging and obtaining the details required to fix problems with software. Even though in the past software had been created from scratch in one piece, nowadays most software contains many pre-built components or are composed of several parts of integrated modules. Therefore, developers who developed the software may not even have the necessary knowledge about the individual security of the used components and the impact of integrating code in terms of both static and dynamic aspects. When the source is available, it is an amenity to assess the security, but it is often hazardous as the components or software is only available in binary format.

Also, the software could become vulnerable due to the methodology through which it has been developed or built. Most software methodologies do not put much emphasis into security, and most often security will not be checked on early phases of the software development life cycle (SDLC). Some software is tested for security only after its production is completed. Another reason for software to become vulnerable is the usage of unethical development procedures and non-standardized tools. Even though standard certified tools have been used, it may not have been used in the recommended way resulting in insecure software. Lack of adherence to best practices and weaknesses in essential knowledge in the development, especially in terms of security is also a significant factor to software to become vulnerable.

1.2 Security of the Component-Based Systems

Either with in-house modules or with third-party libraries, present-day software is developed with interconnected components as composed systems. There are many components in the market from open source to proprietary which is glued together with different technologies and under various architecture patterns.

Due to the heterogeneous nature of the components, the vividness of the deployment environment and diverse interaction patterns, security of such a composed system may be defined by the weakest component of the system [2]. In order to ensure the security of a composed system, it is vital to realise the security characterisation of the individual component as well as their interactions and eventually the security of the software as a whole.

1.3 Methods of Software Vulnerability Assessment

During the software development lifecycle, maintaining software security plays a significant role in almost every system. Therefore, the various methods of security assurance exist to support different lifecycle phases.

In the initial requirements gathering and design phase, manual review of the process and threat modelling can be used to declare the security objectives of systems. Afterwards, static code analysis assists the developers to identify code-level security vulnerabilities parallel to the functionality development. Penetration testing, fault injection and several other black box testing schemes provide the assurance of the security of the systems in the testing phase. Vulnerability scanners and binary analysis tools may be used to ensure security in the overall system in the deployment and execution stage.

1.4 Motivation for the Research

As a common practice, we find that different tools have been used to test the security of individual components and specific systems in different stages of software development. Therefore, measuring the security level of such a composed system has become a challenging task. Due to the increasing trend and unavoidable nature of composed systems usage in the industry and exponential growth of the cybercrimes, has resulted in evaluating the security of a composed system as an essential and critical activity. Although a wide range of tools is available to test different aspects of software security, suitability, and effectiveness, the assistance of those tools in the area of the

component-based system is not well studied. Hence, analysing such tools against the composed system is beneficial for today's software industry.

1.5 Research Problem

Heterogeneous nature of component-based systems and modern Agile development methodologies used to engineer such composed systems has resulted in making the measuring of the level of security of such a system to be complicated. Also, evaluating the effectiveness of vulnerability assessment tools on such component-based systems too has become tedious. Hence, this research will address the effectiveness of vulnerability assessment tools on component-based software systems.

1.6 Benefits of the Research

This study will evaluate different vulnerability assessment tools against component-based systems and provide an analysis of the effectiveness of selected tools which will be a baseline for tool selection and also a methodology for selecting security assessment tools for component-based software development. Further, it will provide guidelines on deficiencies in existing studies in this area and suggest future research directions for this area.

2. Literature Review

2.1 Composed Systems

Software component and component-based development became a buzz in the software industry and academia in the late ’90s. Several terms and definitions exist to refer to the same concept. Such terms like components, COTS (Commercial off the Shelf), FOSS (Free and Open Source Software) refer to some software that can be developed and might be used independently and more importantly through the interaction it can make larger or complicated software. Also, software systems that are built with such components are often refered to as composed systems or component-based software systems. The methodology and the practices with this refer to Component-Based Software Engineering (CBSE) or Component Based Software Development (CBSD). The following section shall describe aforesaid composed systems and the security aspects of such systems respectively.

Hopkins Component Primer [3] places emphasis on well-defined interfaces which separate them from their implementation as essential to the success of components to be loosely coupled. The author further claims that component-based development represents a milestone in the maturation of software engineering. Additionally, the reusability and maintainability are the critical engineering principles which motivated the CBSE.

2.1.1 Component

As mentioned above, there are many definitions for a component which express the same characteristics in different viewpoints. In Szyperski’s [4] definition “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties”. Also, D’Souza and Wills [5] define a component as “A coherent package of software artefacts that can be independently developed and delivered as a unit and that can be composed, unchanged, with other

components to build something larger". Combining both definitions Hopkins [3] provides an overview definition as "A software component is a physical packaging of executable software with a well-defined and published interface". Therefore, the component can be identified as a basic unit that can be built and maintained independently to compose massive software which interacts with each other through an interface.

Meijler and Nierstrasz [6] describe components are the next level of object-oriented development (OOP). Components address some limitation of OOP and extend their reusability with frameworks. Further, the authors claim the fast time to market reliability, maintainability, adaptability, heterogeneity, and division of labour are the motives for the existence of the components and its rapid popularity.

2.1.2 Component Interface

In a composed system, integration among the components is the crucial factor for success. Moreover, the key to integration is an interface that separates and hides the complexity of implementation. Therefore, a framework for identifying the features of interfaces is fundamental to understand the composed systems. Han [7] provides a framework for characterising software interfaces. Figure 2.1 shows the framework which considers the several aspects of the interface.

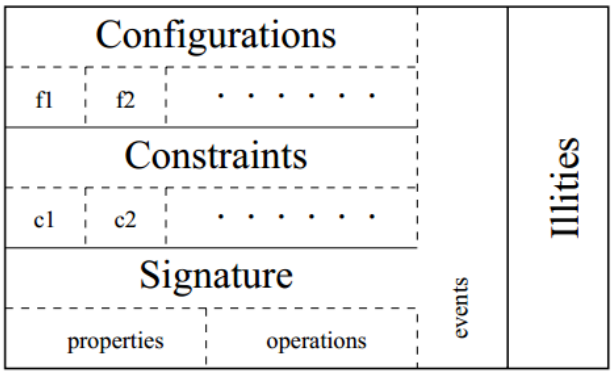

Figure 2.1: Structure of component interface [7]

In the bottom is the signature of the component which consists of properties, operations, and events. Then, the constraints let the interface to more restrictive and well-defined. Altogether signature and constraints feature about the capabilities of the component, and the configuration represents usage scenarios which identify the roles and the usage contexts to the specific component and the scenario. Therefore, there will be different configurations concerning the usage scenarios, which allows the component to been customised and reuse. Also, the non-functional attributes help to assess non-functional qualities of the components.

2.1.3 Component Composition

Although there are many components developed and made available to use, composing those heterogynous components into an effective system is not a trivial task. Although COTS or components reduce the expenses of development and maintenance, more often becomes a nightmare to integrate. A significant reason for this is that software is developed as a standalone application and always run with the aligned assumption with its environment [8]. Table 2.1 lists several advantages and disadvantages of cots by Boehm and Abts [9]. Further, it describes how the vendor behaviour, overwhelmed expectations, interoperability, and product evolution can cumbersome in the development process.

Table 2.1: Advantages and disadvantages of COTS components [9]

Advantages	**Disadvantageous**
Immediately available; earlier payback	Licensing, intellectual property procurement delays
Avoids expensive development	Up-front license fees
Avoids expensive maintenance	Recurring maintenance fees
Predictable, confirmable license fees and performance	Reliability often unknown or inadequate; scale difficult to change
Rich functionality	Too-rich functionality compromises usability, Performance.
Broadly used, mature technologies	Constraints on functionality, efficiency
Frequent upgrades often anticipate the organisation's needs	No control over upgrades and maintenance
Dedicated support organization	Dependence on vendor
Hardware/software independence	Integration not always trivial; incompatibilities among vendors
Tracks technology trends	Synchronising multiple-vendor upgrades

SEI technical concepts of CBSE describe the interaction in terms of compositional forms and binding time perspectives [10]. In a CBSE framework and components are the two main entities. Based on the components and frameworks in terms of interactions, there are six possible combinations in Compositional Forms. There are three main categories as follows.

- Component –Component
- Framework-Component
- Framework-Framework

Also, the above three major categories can extend to another three more special cases [10]. Table 2.2 illustrates the compositional forms with existing component models.

Table 2.2: Compositional forms of component models [10]

Compositional Form	**EJB**	**COM+**	**Java Beans**	**Water Beans**	**OMG/Orbos**
Component Deployment	✓	✓	✓	✓	✓
Framework Deployment	future (container contract)		✓ (JVM plug-in)		✓ (portable object adapter)
Simple Composition			✓	✓	
Heterogeneous Composition	✓ (IIOP)				✓ (IIOP)
Framework Extension		future (policy objects)			✓
Component (Sub)Assembly					

As the above table explains not only what is composed is significant, it is vital to know how it composed as well. The component is said as composed when a resource of one component is accessible for another. For resource binding, many methodologies are available that can be spread through the development time to runtime. According to the time of the resource is binding, components are categorised as early binding and late binding. However, late binding has always been preferred since it reduced the restrictions on development time.

2.1.4 Component Composition Patterns

When it comes to the composition of components, patterns have been introduced to make things easier for software designers and developers. Thus, to cope with hidden dependencies, complex interactions, and ambiguous design; Eskelin [11] describes few patterns to get the composition done seamlessly.

- ABSTRACT INTERACTIONS

  By defining each other's implementation on themselves, allows the components to communicate and interact without depending on the environment but, through an abstract interface. Java components add listeners in abstract interactions pattern.

- COMPONENT BUS

  Component Bus removes the interdependencies of two components. All the components connected to information BUS which will manage the communication and the routing among the components Enterprise Service Bus and similar implementations fall into this category**.**

- COMPONENT GLUE

  The Functionality of an adaptor between the two different components or as a mediator for components is handed over to script in this scenario. Only another component will be used to replace the script when it cannot meet the full requirements. For example, JINI uses this kind of script code to download and deploy a service.

- THIRD-PARTY BINDING

  In third-party binding, any interaction between two components is removed from a third-party component. Therefore, if any change that we have required in terms of interaction in a third-party component without affecting the two components involved in the interaction.

- CONSUMER-PRODUCER

  For the consumers, components will get one interface formatted similarly from the producer component which connects several various provider components. JNDI provider is such interface that exposes set of different services to its consumers.

"Inversion of Control" has been a typical pattern to assemble components nowadays in the community which also known as dependency injection. Dependency Injection allows lightweight containers to be assembled as a set of different components into a cohesive application [12]. Many books and research literature has been written on specific implementation on this pattern [13] [14] [15].

2.2 Issues and Problems of Composed Systems

2.2.1 Maximizing the Reusability

Reusability of software components and time to market are the main intentions that drive the composed system to the main software stream. However, on the other hand, it has also become a challenging task due to several reasons. There are many methods, technologies, models and framework available in CBSE [16]. Therefore, choosing the correct model or the technology and to what extent the components should be used to maximise the reuse, are questions that go hand in hand.

2.2.2 Quality of the Components

CBSE is a phenomenon due to the advantages it produces, but at the same time, it can be a catastrophe if it has been composed by using less reliable and low-quality ingredients. Given a component, it might have known and unknown issues, but the system that composed from it can present unexpected side effects and unknown consequences. Hence, assuring the quality of such components or system plays a vital role. The industry often uses integration testing, regression testing and load and performance testing against the composed systems. Many models and frameworks exist in the academic researches and practices for component-based systems [17].

Also, testing such a system is chaotic due to several factors. One of the major reason is the unavailability of code and often executed in black box testing.

Moreover, such black box testing could not provide the required confidence or guarantee. Low adequacy of the testing is also another factor, and often it is due to the disability of interoperability testing with traditional methods [16]. Another factor which results in poor quality is the lack of debugging ability in the code.

2.2.3 Standards and Certifications

There should be good standards and certification for components both favouring the supplier and the customer. There are many standards for software, and few exist for software components and reuse. IEEE 610.12, IEEE 730, IEEE 830, and BS 7925-2 are some available standards for software component and reuse. However, those standards do not cover every aspect of CBSE. Thus, there should be more future work on this.

Unavailability of proper certification methodology is another issue which becomes the blocker for quality assurance. As mentioned in above that assurance of quality is a hectic task in component-based systems, therefore, developing components to standard and providing such a certification from the third-party is essential to improve the quality and reduce the cost of testing in CBSE. Certification authority for validating and certify components provides the necessary infrastructure to continue the proprietary nature of the COTS business model. Further, if any certification exists, that will reveal the level of security, and it increases the confidence of the customer.

2.2.4 Component Search and Repository

Another issue that CBSE face was finding the correct components when required. Finding out a good, trustable repository is a hard task and even discovering good components sometimes become a nightmare due to lack of non-availability of standards for the repositories. Further, for in-house development, a good repository to

store and maintain the components is essential. Alnusair, A. and Tian Zhao [18] provide and ontology-based Component search to describe, retrieve and explore components using source code knowledge. Source Code Representation Ontology (SCRO) captures the relationship among the source code artefacts based on concepts like encapsulation, inheritance, method-overloading, and method-overriding as well as method signature information. In ComRE, the implementation is deployed as a plugin to eclipse which bootstrap all the information of the workbench.

Further, there are several repositories for open source projects like git [19], SourceForge [20] which are based on the availability of source code.
Sonaytype Nexus [21] and Artifactory [22] provide a component repository, and it is more focused on organisation perspective component storage. Their central implementation also focuses on FOSS components, and it is in a more technical perspective than business oriented. Thus, internet search providers are the available option that someone can use at the moment for search necessary software components.

2.2.5 Other Issues

Khan et al. [2] explain another set of issues that inherent in component-based systems. Functional differences are one of the significant problems in component-based development, where newly developed components and existing components hardly ever fitted together, and more often, a component that is going to develop new should be adjusted according to the existing components to match with existing functionalities.

When communication between each other's in the component's language, the way it has been written matters intensely and often if producer component is written in a different language than the consuming component, consuming component need to add more adaptable code to handle the language problems and communication problems.
Components were built targeting the different environments such as the operating system, specific CPU architecture, networking protocols and other software components. Therefore, if any requirement is raised to implement in a different environment, often lead to problems and need to change the original component

configurations. Also, components were built focusing on several factors in its operating environments such as a number of concurrent users, the capacity of the available resources, and the amount of data it handles. Thus, any change of such parameter may result in unexpected behaviour of the component. Countries and regions have different data format for date, time, currency, number format, etcetera. Therefore, the software component which is written targeting a set of data format will not always be usable with other regions.

2.3 Security of Composed Systems

Software components need to adjust to the relevant environment and requirements. Even though there is a need for external protection to the components, composers most of the time fail to implement that due to the binary black box nature of the component. Hence, Khan et al. [23] express that security of a component should be treated differently than the application security because of the distributed nature of the components in the heterogeneous environment. Further, they categorize the security properties of a component in two broader categories as

1. Nonfunctional security properties (NFS)
2. Properties as security functions (SF).

NFS is codified and embedded inside the component whereas, SF can be implemented externally as separate functions. Identifying both NFS and SF of a component is very valuable especially before selecting a component to integrate into the system. However, adding a strong external SF is effortless if the NFS is weak. For example, adding strong encryption function is useless if inbuilt properties of components had security flaws. Thus, the internal computing properties will define the ultimate level of the security of a component.

2.3.1 Scenario Based Component Security

Security of a component cannot be decided by a component alone since it is profoundly affected by the user context. There are two types of security mechanisms as implemented by underlying infrastructures such as protocols, connectors and much lower level system applications such as Operating Systems and hardware mechanisms, and the other is security implied by components with their internal security features. Hence, security of a component substantially is influenced by component deployment infrastructure and the use case scenario of the component. In other words, security of a component in a system is dependent on the scenario it has been driven. Thus, Khan and Han [24] characterise the security of a component into the following formula.

a_Security_funcion(S, O, K, D)

where S is the identity of a component in a hypothetical scenario, O is an arbitrary operation set which executed by component S, K is the security attribute set used by the member S to operate O. D is the data or information set belonging to the component.

In terms of practical approach for this, a scenario is presented by message communication protocols and architectural descriptions and from, required and ensured security properties of the individual operations, specific threats and the associated security policies and functionality has been identified.

2.3.2 Characterization of Component Security

A formal model has been introduced to identify and quantify the comprehensive list of security properties embedded with the services that a component provides [25]. In the scheme, security class contains a collection of a set of security objectives related to the class, set of security functions and entity and action used by the security functions. There are two entities as subject and object, and those entities and actions are taken as a predicate to express overall security of a function.

Each function is associated with a rating based on the strengths and weakness of the function in the particular context of the application. The accumulated rating of all functions of a particular security objective treated as ultimate strength of that objective and similarly accumulated values of security objectives would treat as the ultimate strength of that particular security class. Figure 2.2 displays the simple signature of the security class.

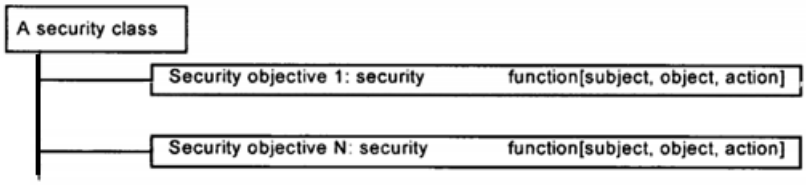

Figure 2.2: The Structure of the characterisation scheme [25]

When a component is developed, the above scheme may apply by the developer and should be attached that to the component interface to runtime access for the contracting client component. This will help to evaluate the security of a component and take necessary action to mitigate any security risks.

There are many systems which are composed of using components available on the internet. The main drawback is the absence of guarantee or certification of trustability of such components, i.e., unavailability of security characteristics will lead to ignore the component completely or to risk the security of the overall system.

Khan et al. [26] provide a methodology to publish trust-related security properties of a component in a machine-readable way, which provide certification for the component. First, the security characterisation of atomic components has been found, and then the security characteristics at the component level have been certified. Afterwards, check the compatibility of those security properties between components in the contract level and then determine the overall security of the final system considering the system level contracts. Figure 2.3 shows the process of the software development life cycle.

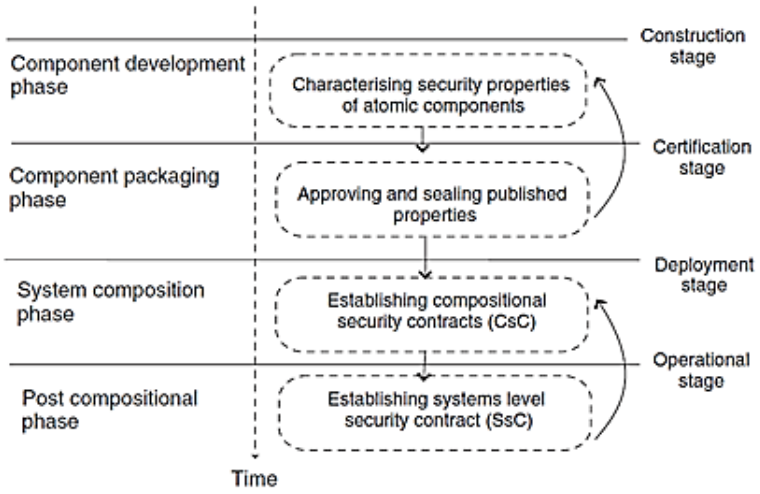

Figure 2.3: A Security characterisation process framework [26]

Component level security is static unless the component is modified. However, compositional and system level security is dynamic since the properties will vary depending on the individual composition.

Security properties are categorised as ensured, and required security properties and security knowledge base has been used to store the security characteristics. Compositional security contacts (CsC) is decided by adherence of required and ensured security characteristic matching and system level security contract (ScC) is determined by CsCs.

2.4 Measuring the Security of Component-Based Systems

Khan and Han [27] introduce an assessment scheme to calculate a numeric score for a component's security for a given software application or system which represent the relative strength of security properties of the given component. Figure 2.4 describes the structure of the scheme. In the given scheme, system security requirement gathered for the considered system or application, and the candidate component will rate depending on the security properties it contains. Those properties can be component-specific security services, security classes, security objectives and security functions.

Finally, using the evaluation templates scheme will calculate the final score for the system.

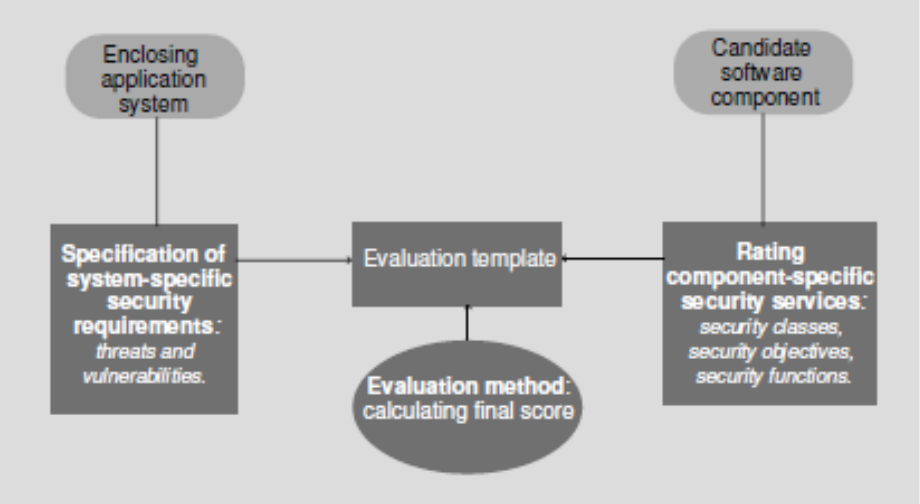

Figure 2.4: An Assessment scheme [27]

The scheme is evolved using the ISO/IEC 15408 the Common Criteria for Information Technology Security Evaluation (CC) and the Multi-Element Component Comparison and Analysis (MECCA) model. The scheme will address the limitations of CC. CC is common for general applications and does not address specific security requirements of evaluators. Also, this will provide a preliminary assessment before the system is integrated. CC evaluation needs huge effort and incurs a significant cost which many companies could not afford.

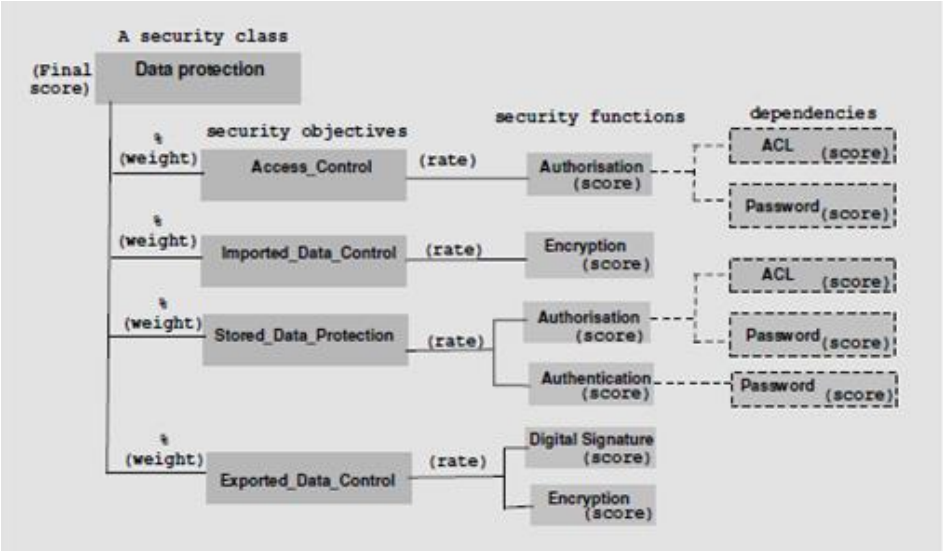

Figure 2.5: An evaluation template for the banking system [27]

As illustrated in figure 2.5, the evaluation method uses a percentage of weighting to the security objectives. A percentage weight is assigned to each security objective throughout a given class. Similarly, each class is also given a percentage weight comparative to the importance of other security classes. At the class level, the percentage of the weights of all classes would sum up to 100. Accumulated percentage weighting of the security objectives in a given class would always be 100. The percentage weighting is defined by the software engineer depending on the importance of the individual security objectives.

Busch et al. [28] extend the Palladio component model (PCM) to support security assessment using annotations and extended PCM used to calculate the security of CBSE (component security and mutual security interference) and getting different from other systems they have considered the attacker, the attacker's skills, and attacker scenarios, starting and aiming point of the system. Then, those are modelled to an analytical model using Semi-Markov process which will result in Mean time to security failure (MTTF). Figure 2.7 shows the proposed scheme.

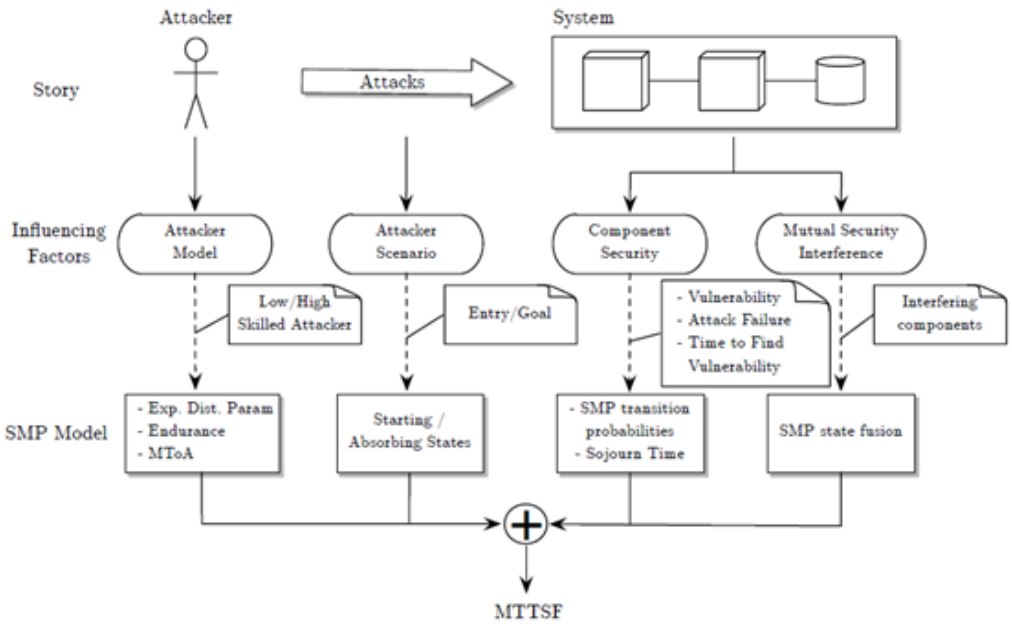

Figure 2.6: Security assessment approach [28]

Comparing the values of MTTSFs will allow software developers to assess the degree of security. Such methodology will help developers to consider the security considerations when they take architectural trade-off decisions which ultimately help to select, design and build proper systems with higher calibre.

Nazir et al. [29] present an Analytic Network Process (ANP) based approach to evaluate components security which is based on ISO/IEC 27002 standards. Provided the weights and values from expert's opinions, given components will be compared using ANP to select the most secure component to be used.

2.5 Software Security Testing Methodologies

In the security Testing of software can be categorised mainly into three areas as a white box, grey box, and black box testing. White box testing is where the source code is available to analyse and test. Grey box testing is where both source code and executable artefacts are available and could be performed on in-house developed components as well as open source components. Black box testing is only the

executable, or the binary artefacts are available where propriety components and third-party libraries.

2.5.1 Security in the Requirement Gathering & Design Phase

For security assurance, we can perform threat modelling early in the development cycle to mitigate the security risks of software. Swiderski and Snyder [30] describe the importance of threat modelling as a specification in the requirement gathering phase for complex software systems. Furthermore, performing manual inspections and reviews to fixing the gaps in the process of software development is another significant task in the early stages of software development.

In the requirement gathering phase, most pragmatic practice is to follow security standards related to requirement gathering. ISO/IEC 15408 consists of three sections as general, components and assurance which ensures the security best practices [31] [32] [33].

2.5.2 Static Analysis & Code Reviews

Static code analysis is the primary strategy in white box security testing. Chess & McGraw [1] emphasise the significance of static analysing in the early phase of the software development and the impact of automated tools stressing where the quality of the rule set that the tools enforced plays critical role. Since the reviewing code for security is a slow and tedious task, static analysis tools are often used to review the code automatically. Sonarqube, a composed tool with Findbugs has been used for this purpose even though these tools have advantages and disadvantages to their individuality [34].

2.5.3 Fault Injections

Faults injection is a methodology for testing security of a system by simulating the faults on its execution environment. This is to stress the system where the fault is

injected into the system; it may behave strangely than it is in the normal state. There are two main branches in the fault injection as source code and binary. Source code injection is the tester that will determine the faults based on the information he can gather in the source code and instrumented the code with fault and observer the behaviour of the system.

On the other hand, binary injection is identifying the interaction methods of the system on its execution environment through system calls and remote procedure calls and access the surrounding environment resources of the program to simulate the attacker scenarios. Security of the system can be determined by the success ratio of such fault injections.

2.5.4 Dynamic Testing

Dynamic testing performed by the tester while the program is being executed. The dynamic test will monitor system memory functional behaviour, response time and performance features. Dynamic analysis enables to expose the vulnerabilities of software associated with user interactions, the configuration of the environment and its behaviours since the tests are carried against actual runtime or a simulated environment.

2.5.5 Binary Analysis

Binary analysis is checking the code in the machine code or binary levels. Often intermediate language such as Java Byte code analysis is analysed for procedures, instructions, registers, and memory addressing. Further, this can be categorised as static binary analysis where static analysis happens before the program run, and dynamic analysis is performed at a run time.

2.5.6 Penetration Testing

A penetration test is a strategy used to test the security of software by attacking the system with intruder's mindset. Furthermore, it can be performed manually or automated according to the security requirements of the considered system.

2.5.7 Vulnerability Scanning

Vulnerability Scanning is using one computer or program to detect and exploit the security weakness and flaws of other software and systems. Most of the vulnerability scanners work with vulnerability database which reports, publishes and categorises the security with specialists' opinion. Vividness in the software systems leads to having a different set of tools which specialised in specific areas such as network and application environments. However, because of the vividness of the systems that have been scanned by these vulnerability scanners, often the reports which generated by those scanners contains a lot of false positives.

3. Methodology

3.1 Introduction

There are many methodologies and tools in academia as well as in industry to evaluate software security. The task of security testing is divided mainly into two branches as manual and automated. Also, it can be categorised as white box testing and black box testing and less often as grey box testing. Due to severe competition in the market place as well as due to marketing strategies used by companies, currently software development is being done at a rapid pace using proprietary or open source components which are then composed to a complex system. Security of such a system will be determined by the individual security of the components, their interactions and integration mechanisms as well as the deployment or the execution environment security.

Even though there is a wide range of tools available to test software security, they are specialised to assess a specific method or area or particular type of vulnerabilities. Furthermore, those components are integrated into the system in different phases of the software development lifecycle where the phases are time-boxed to small iterative periods. Hence, it is critical to evaluate the applicability and effectiveness of the security tools in the context of it as a composed system with a diverse range of components and engineered with modern development methodologies. Thus, this study will evaluate the effectiveness of vulnerability assessment tools for component-based software systems.

3.2 Existing Experiment Efforts

There have been studies conducted more in a theoretical perspective which have considered security at the application level rather than at the development level. However, security analysis of composed systems focusing on the full SDLC and the software engineering perspective does not appear in published research at a level even

remotely comparable to the vast amount of research work targeting deployment stage and dynamic testing and also focusing prominently on web application features.

Antunes and Vieira [35] have experimented benchmarking the web services security detection against the available tools. They proposed a mechanism to rate the security tool with precision and recall measurement based on the true positives and the false positives reported by the tool. However, it is specific only to web services security.

WaveSep benchmark developed by Shay Chen [36] is another highly capable benchmark for testing of security tools. It is an open source project written in Java with support for a variety of test cases and publishes the score for a suite of tools on a website with respect to their performance. Juliet Test Suite is another Java-based benchmark tool with many test cases from the National Institute of Standards and Technology (NIST) [37]. However, these test suites primarily focus on DAST of web applications.

3.3 Goals and Motives of the Experiment

The study has evaluated the effectiveness of the tools on component-based systems considering the availability and applicability in different stages of software development. Also, the security tools are evaluated on the applicability and effectiveness for new development methodologies such as Agile where rapid and frequent release cycles been adopted in order to reduce the time to market. In the initial literature survey, it was found that not many methodologies or tools exist to cover the full lifecycle phases in software development which assures the security of component-based development. Hence, this study is focused on the tools along with the applicability of the life cycle phases consisting mainly of development, testing, and deployment. Furthermore, the study can be used as a baseline or methodology to evaluate the effectiveness of security tools.

3.4 Experiment Design

The experiment design aimed to evaluate the effectiveness of three main properties.

I. Effectiveness of the tools with respect to detecting the vulnerabilities and the accuracy of the reported vulnerability
II. Effectiveness of the tools in the development phases of the SDLC, development processes, and supportive tools
III. Supportiveness to the Agile development methodologies where rapid and frequent release cycles have been adopted

3.5.1 Test Data Selection

Having a comprehensive set of test cases is an essential factor in evaluating security tools. Creating such a set of test cases from scratch is a time-consuming task and need many person-hours to implement appropriately. Furthermore, this research is focused on the existing state of the art in security evaluation. Hence, it was decided to choose an existing benchmark tool as a set of test data.

When selecting the benchmark, the properties such as quality of the test data, the frequency of updates and activeness in development, measurement technique, ease of extendibility, ease of usage, supportiveness for different life cycle phases and component-based development have been considered as primary characteristics.

From the available benchmarks, Juliet Test Suite had not been updated recently and it is primarily focused on the Java web applications and Java applications targeting SAST methodology. Also, WaveSep is another good benchmark that focused more on the DAST methodology and favoured web applications. Therefore, it’s extensibility to other applications was not considered as the primary design goal of this research is to check the web application security.

OWASP Benchmark has been chosen due to the several key properties of the tool. Firstly, it is an active project in the community although it is still in the incubator level

project in OWASP. Although, currently it only has support for the web applications for DAST and SAST, the core design of the tool has been done in a way where it can be extendable to other types of applications as well as to support different stages of the software development process. Furthermore, it has comprehensive documentation and is considered as a utility tool that generates results in a more organized structure.

OWASP Benchmark

The OWASP Benchmark version 1.2, which is the currently available release, is used for this study [38]. A total number of 2740 test cases are included in the eleven CWE security vulnerability categories and web application as test cases, which include merged vulnerability strategies for SAST and DAST.
The execution resources for tests have been provided as it demands since the resources-wise performance is not considered in the experiment. Figure 3.1 shows the underlying mechanism of the OWASP benchmark.

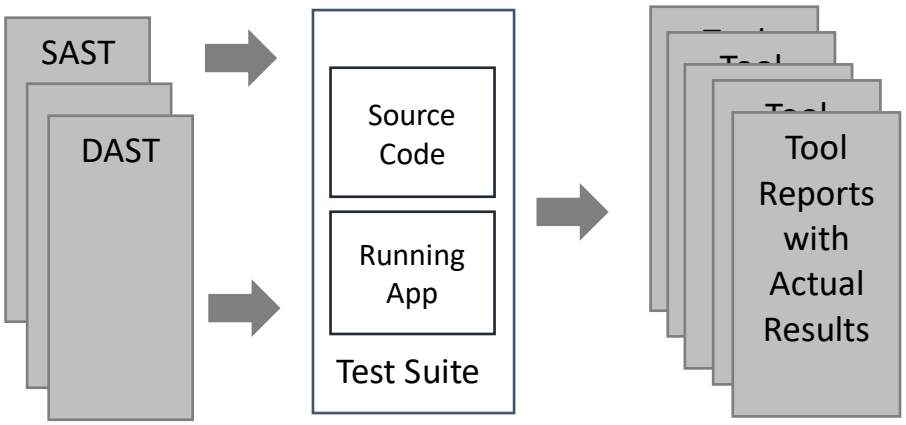

Figure 3.1: OWASP benchmark basic structure

Customized Test Cases

With the initial study, it was identified that the main limitations of the existing frameworks are the lack of support for different types of applications and the lack of adequate consideration for the software development process and the architectures as well as the methodologies. The main focus of the study is component-based software systems with latest methodologies like Agile. Thus, to have a measurement of security tools supportiveness in this aspect, a specific set of test cases been tested in the research using state-of-the-art component-based development tools and technologies.

The experiment has been done simulating the real development environment, simulating the vulnerable tools and technologies which are listed in the Common Vulnerability Exposure (CVE) details list [39].

3.5.2 Test Tool Selection

Tools have been selected to support the main phases of software development. However, there are not many automated tools that support for requirement gathering and design stages. Even though there are some models suggested in the academic context, it is difficult to implement it in the practical scenarios due to lack of infrastructure support.

However, tools have been supported in terms of Static Analysis Security Tools (SAST) and Dynamic Analysis Security Tools (DAST). The following tools have been selected for the study.

SAST

- FindBugs
- FindSecurityBugs with Findbugs
- SonarQube
- PMD
- DependencyChek

DAST

- OWASP ZAP
- Arachni

This study is focused mainly on open source tools due to two reasons. Firstly, all the commercial tools require a license to be used for this kind of a purpose and also there are legal issues related to publishing the results for some of the products. Secondly, the preliminary qualitative studies show that the difference of vulnerability scanning between commercial and open source counterparties to be insignificant and for this study the focus is on the effectiveness of vulnerability assessment for the composed system rather than comparing commercial and open source alternatives.

3.6 Experiment Setup

The experiment has been designed considering three main areas. Firstly, tools have been tested on the effectiveness in a quantitative approach. Secondly, the effectiveness of the tools in the initial prototyping level of the development with different components has been checked more qualitatively. Thirdly, a qualitative measure is used to check the product support for the development methodologies in its architecture and basic release delivery procedures.

3.6.1 Quantitative Analysis of Tools

Experiment I

OWASP benchmark has been used in a number of test cases and evaluated with the number of vulnerabilities found in the assessment based on the false positives, true positives, etc.

Experimental Environment

OS - Windows 10 Enterprise
Processor - Intel Core i7-5600U CPU @ 2.6GHz 2.59GHz
Memory - 16 GB

Data Set

OWASP Benchmark 1.2 has been used as the test benchmark and its 2740 test cases categorised below are used to test the tools.

Table 3.1: Number of vulnerabilities in each category

Category	**CWE #**	**Total**
Command Injection	78	251
Cross-Site Scripting	79	455
Insecure Cookie	614	67
LDAP Injection	90	59
Path Traversal	22	268
SQL Injection	89	504
Trust Boundary Violation	501	126
Weak Encryption Algorithm	327	246
Weak Hash Algorithm	328	236
Weak Random Number	330	493
XPath Injection	643	35
Totals		**2740**

The following tools with the particular version have been used as the tools in this experiment.

Table 3.2: Products and versions used in experiment I

Tool	**Version**
FBwFindSecBugs	1.4.6
FindBugs	3.0.1
PMD	5.2.3
SonarQube Java Plugin	3.14
OWASP ZAP	vD-2016-09-05

3.6.2 Qualitative Analysis of Tools for CBSD

Experiment II

Experiment II uses the customised test cases or the application to check the effectiveness of the vulnerability scanner is in terms of CBSD systems.

Experimental Environment

OS - Windows 10 Enterprise

Processor - Intel Core i7-5600U CPU @ 2.6GHz 2.59GHz

Memory -16 GB

Data Set

The following vulnerable development tools and technologies have been used as data and as the main test bench.

Table 3.3: Details of vulnerable components in development

Tool	**Version**	**No of test cases**	**CVE #**
GWT	2.5.1 RC	1	CVE-2013-4204
Jenkins	1.400.x before Some specific versions	8	CVE-2013-2034 CVE-2013-2033
Hibernate	5.2.3	1	CVE-2014-3558
Maven	3.14		CVE-2013-0253
Oracle Java 7	1.7 SE 7u111	7	Many CVE numbers
Spring MVC	before 3.2.4	5	Many CVE numbers
Eclipse IDE	before 3.6.2	2	Many CVE numbers

3.6.3 Support for Agile Development

Experiment III

Frequent releases of the tools, update policy and mechanism of the rule database have been evaluated to study the support of Rapid Application Development and Agile development methodologies.

OWASP benchmark has been used with many numbers of test cases and evaluated with the number of vulnerabilities found in terms of false positives, true positives with a set of versions for FindSecBugs and OWASP-ZAP's two contiguous releases. Furthermore, the features of relevant tools are evaluated in order to infer the supportiveness for the component-based development and new methodologies.

Experimental Environment

OS - Windows 10 Enterprise
Processor - Intel Core i7-5600U CPU @ 2.6GHz 2.59GHz
Memory -16 GB

Data Set

Table 3.4: Incremental product versions in the Excrement II

Tool	**Version**
FBwFindSecBugs	1.4.4
FBwFindSecBugs	1.4.5
FBwFindSecBugs	1.4.6
OWASP ZAP	vD-2015-08-24
OWASP ZAP	vD-2016-09-05

3.7 Measurement Methods

In Experiment I, quantitative data have been collected for all the test cases and the results were categorised as follows;

- Tool correctly identifies a real vulnerability (True Positive - TP)
- Tool fails to identify a real vulnerability (False Negative - FN)
- Tool correctly ignores a false alarm (True Negative - TN)
- Tool fails to ignore a false alarm (False Positive - FP)

From the above values, the following could be calculated:

- Sensitivity or True Positive Rate (TPR) = TP/(TP+FN)
- Fallout or False Positive Rate (FPR) = FP/ (FP+TN)
- Specificity or True Negative Rate = 1− FPR = TN/(TN +FP)
- Youden's index = Sensitivity+Specificity−1

If a graph is plotted as TPR against TNR, then the top right corner of the graph will have the tools that have high sensitivity and high specificity. However, this will not allow a good understanding of the accuracy of the tools since it does not consider the flaws, i.e., False Positives. For example, assume a tool will treat vulnerabilities as only True Positives and True Negatives. Then such a tool will calculate TPR and TNR as 1 because of the value of the denominator and the numerator in the formula becomes 1. Even if the number of false positives and false negatives will be a low value, then the denominator is very much closer to 1 and the final value will be again closer to 1 leading it to be incorrectly considered as a better tool. Because of that, it is difficult to figure out a tool which will conclusively identify the flaws and true vulnerabilities with high accuracy.

We need an informed decision rather than a random prediction. Therefore, the identification of false positives and false negatives should be considered equally in the final score in terms of accuracy. If TPR is plotted against FPR (1-specificity), where

the false positive rate represents the percentage rate of the tools which report false positives, then that will provide an accurate score for the tool.

Benchmark score is calculated by using Youden's index [40] which is calculated by deducting 1 from the sum of a test's sensitivity and specificity expressed not as a percentage but as a part of a whole number: (sensitivity + specificity) – 1. For a test with poor diagnostic accuracy, Youden's index equals 0 and in a perfect test, Youden's index equals 1. Benchmark Score is the length of the line from the point down to the diagonal "guessing" line

Finally, the score for a specific tool has been derived as follows:

Score =TPR-FPR

For experiment II, there are specific vulnerabilities with respect to the category of the development environment, runtime environment and build environment and development tools. From the study, it was decided whether the assessment tool correctly found out the vulnerability in the perspective of utility tools, development tools, and runtime configurations.

From experiment III, time-frequency of the tool's releases and update frequency of the rule database have been compared.

4. Results & Evaluation

This section will present the quantitative and qualitative results of the research study. The results are based on the data gathered and evaluated for the effectiveness of the tools corresponding to selected benchmarks.

4.1 Experiment I

In this experiment, prominent open source SAST and DAST tools were chosen to execute against OWASP Benchmark which contains 2740 test cases covering vulnerabilities of eleven categories. From the benchmark reports, we can infer the accuracy of the vulnerability assessment by the values mentioned in section 3.7 measuring methods.

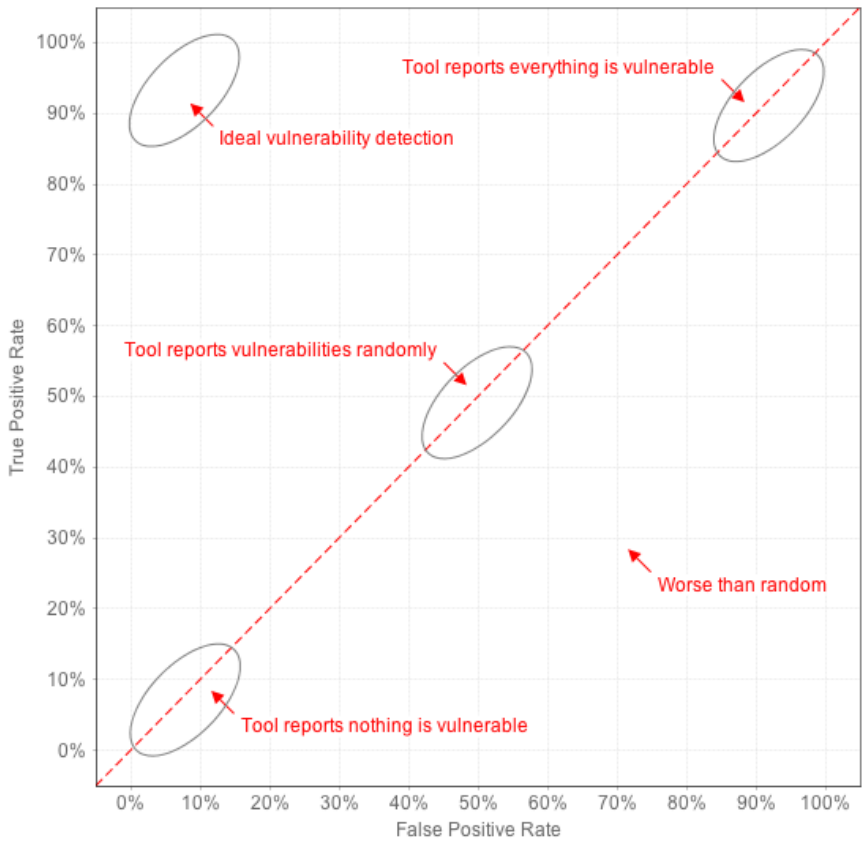

Figure 4.1: Interpreting the results [35]

In order to interpret the data from the experiment correctly, the above graph provides the guideline. The accuracy and effectiveness of the tools can be measured correctly based on the identified vulnerability ratios.

As shown in Figure 4.1, if the True Positives and False Positives are 0, it means that the tool reports nothing as vulnerable. On the other hand, if True Positives and False Positives are 100%, then it assumes that the tool is reporting everything as vulnerable. Thus, both these extreme ends can be treated as inefficient. And the line drawn between (0, 0) and (100, 100) can be treated as the random line showing the area where tool behaviour is random. Any performance point under this curve means the tool has reported more incorrect outcomes with less information and therefore, it was less effective than expected. If the score is above the curve and toward the top left corner, then the tool is in the ideal performance area. Also, the coverage and accuracy are also in the maximum ranges.

4.1.1 Assessment of SAST

In order to evaluate the effectiveness, the following information was gathered for all the tools. Table 4.1 consist of the information about FindSecBugs with FindBugs.

Table 4.1: FindSecBugs vulnerability assessment statistics

Category	**CWE #**	**TP**	**FN**	**TN**	**FP**	**Total**	**TPR**	**FPR**	**Score**
Command Injection	78	126	0	14	111	251	100.00%	88.80%	11.20%
Cross-Site Scripting	79	246	0	78	131	455	100.00%	62.68%	37.32%
Insecure Cookie	614	36	0	31	0	67	100.00%	0.00%	100.00%
LDAP Injection	90	27	0	5	27	59	100.00%	84.38%	15.63%
Path Traversal	22	128	5	18	117	268	96.24%	86.67%	9.57%
SQL Injection	89	272	0	22	210	504	100.00%	90.52%	9.48%
Trust Boundary Violation	501	83	0	8	35	126	100.00%	81.40%	18.60%
Weak Encryption Algorithm	327	130	0	63	53	246	100.00%	45.69%	54.31%
Weak Hash Algorithm	328	89	40	107	0	236	68.99%	0.00%	68.99%
Weak Random Number	330	218	0	275	0	493	100.00%	0.00%	100.00%
XPath Injection	643	15	0	1	19	35	100.00%	95.00%	5.00%
Totals		**1370**	**45**	**622**	**703**	**2740**			
Overall Results							**96.84%**	**57.74%**	**39.10%**

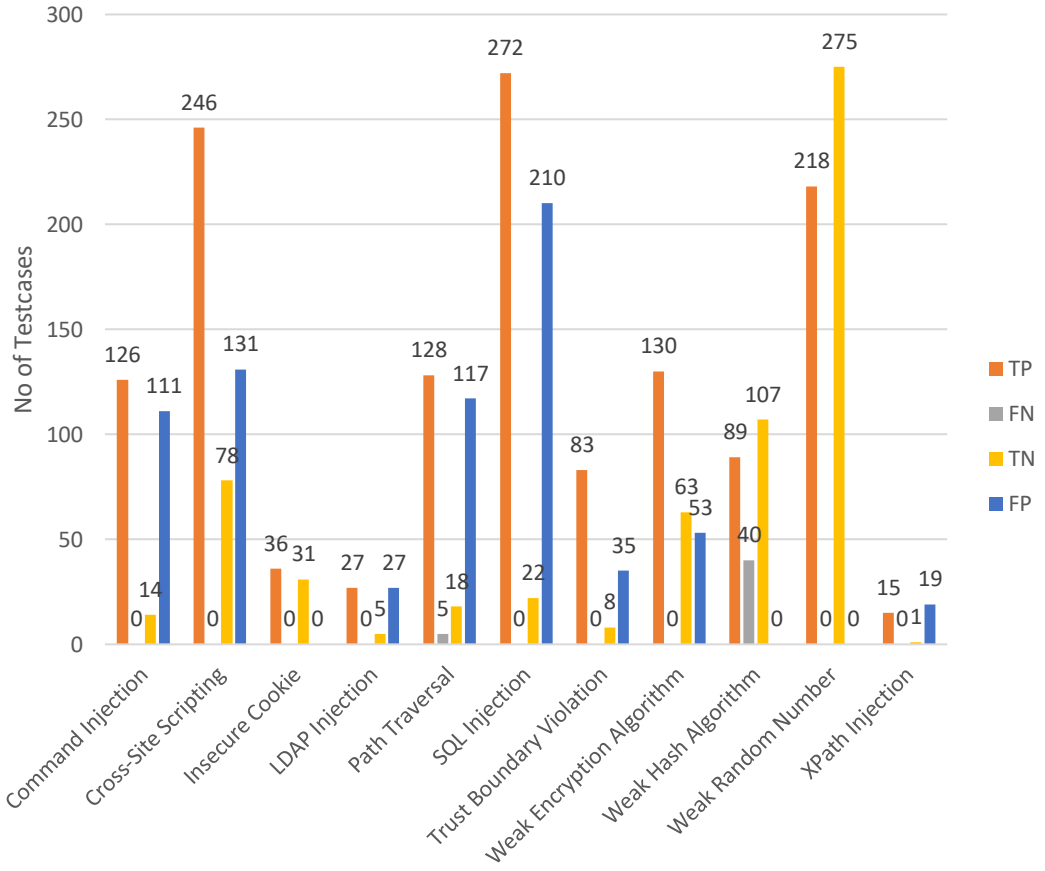

Figure 4.2: FindSecBugs vulnerability assessment statistics

FindSecbugs, a sub-plugin of FndBugs reported higher True Positive values for Cross-Site Scripting, SQL injection, and weak rank number. As shown in figure 4.2, for True Negative scenarios, the Weak Random Number vulnerability is reported correctly as not as vulnerabilities.

However, False positive is also higher in Cross-Site Scripting, SQL injection, Path Traversal and Command Injection. Identification of false negatives is also insufficient. Except for the Weak Random Number and Insecure cookies, FindSecBug performance is low and shows random behaviour.

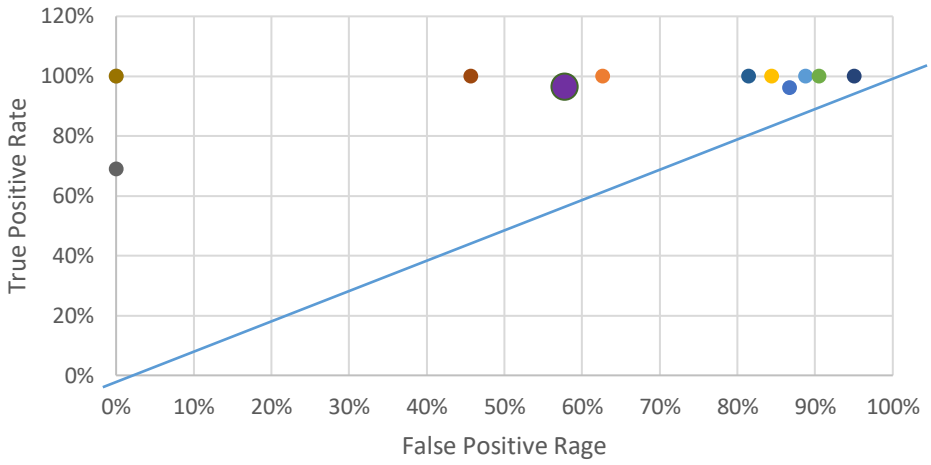

Figure 4.3: FindSecBugs effectiveness

In using the FindSecBugs with FindBugs individual assessment, it was efficient in identifying vulnerabilities of insecure cookies and a weak random number. Even though it achieved a 100% coverage, and accuracy in those vulnerabilities, effectiveness in XPath injection, SQL injection and PathTraversal are lower than 10%. On average, FindSecBugs scored 39%, which is remarkably low against the expected value due to the inaccuracy of most of the reported vulnerability. However, among the selected SAST tools, FindSecBug scored the highest marks.

Due to the absence of specific security rules in the Findbug, its performance is negligible and the overall average is -0.07, which is below even than the expected value. While this tool contains rules to find other bugs, the reported security vulnerabilities from that are even more inaccurate and detection efficiency is also in lower levels.

PMD is falling into the same category as FindBugs where there are no security rules to detect security flaws. Even though it obtained an overall score of 0%, it is better in the accuracy since it has not reported any false positives as well as not any True positives in case of Negative scenario. Therefore PMD analysis can be stated as ineffective.

Even though it is expected for SonarQube to outperform all of the other SASTs; its performance was less than the performance achieved by FindSecBug where

FindSecBug score reported as 33.34% on average. SonarQube has performed well against insecure cookies and weak random numbers in the same manner as FindSecBugs. However, it outperformed FindSecBugs in detecting weak encryption algorithms.

4.1.2 Assessment of DAST

OWASP Zed attack proxy is the most popular and actively maintained web project in the community. For this tool, the OWASP Benchmark project has been used to implement the test cases as a web application project where the OWASP ZAP analysed the provided vulnerable websites and created the results set related to those sites. Table 4.2 shows the categorised details of the vulnerability assessment of OWASP ZAP.

Table 4.2: OWASP ZAP vulnerability assessment statistics

Category	**CWE #**	**TP**	**FN**	**TN**	**FP**	**Total**	**TPR**	**FPR**	**Score**
Command Injection	78	41	85	125	0	251	32.54%	0.00%	32.54%
Cross-Site Scripting	79	71	175	209	0	455	28.86%	0.00%	28.86%
Insecure Cookie	614	36	0	31	0	67	100.00%	0.00%	100.00%
LDAP Injection	90	0	27	32	0	59	0.00%	0.00%	0.00%
Path Traversal	22	0	133	135	0	268	0.00%	0.00%	0.00%
SQL Injection	89	158	114	229	3	504	58.09%	1.29%	56.80%
Trust Boundary Violation	501	0	83	43	0	126	0.00%	0.00%	0.00%
Weak Encryption Algorithm	327	0	130	116	0	246	0.00%	0.00%	0.00%
Weak Hash Algorithm	328	0	129	107	0	236	0.00%	0.00%	0.00%
Weak Random Number	330	0	218	275	0	493	0.00%	0.00%	0.00%
XPath Injection	643	0	15	20	0	35	0.00%	0.00%	0.00%
Totals		**306**	**1109**	**1322**	**3**	**2740**			
Overall Results							**19.95%**	**0.12%**	**19.84%**

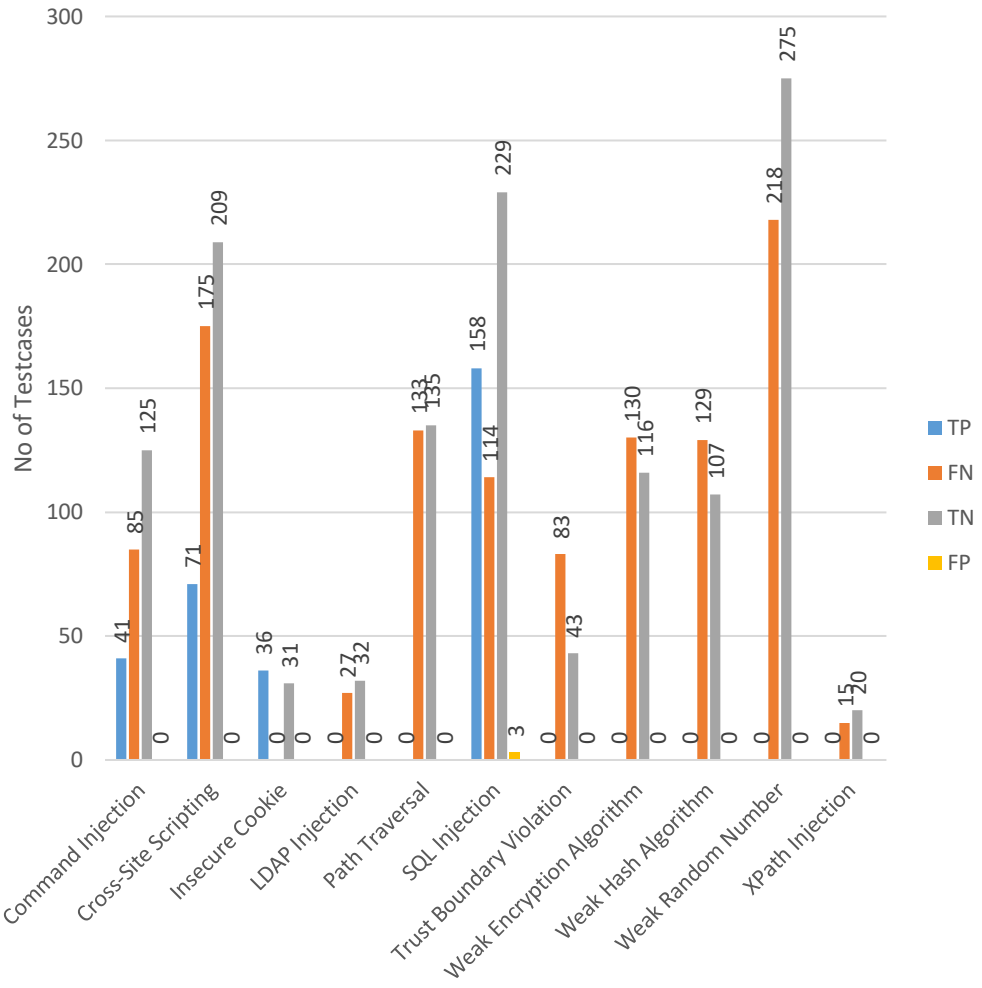

Figure 4.4: OWASP ZAP vulnerability assessment

Figure 4.4 shows the vulnerabilities assessment response of OWASP ZAP, which is based on the DAST methodology. According to the number of True Negatives, it reported most of the not vulnerable scenarios correctly. For example, weak random number, SQL Injection, and Cross-Site Scripting are reported with the highest values. Also in the same categories, it reported higher values relative to the other categories, but compared to the other tools, a number of issues reported are quantitatively lesser. A significant characteristic of the OWASP ZAP is that it does not report many False Positives. Only 3 False Positives have been reported in the SQL Injection category.

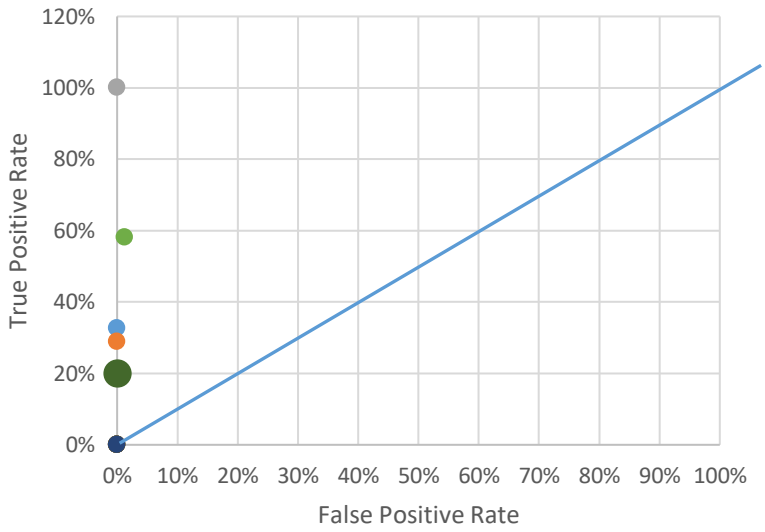

Figure 4.5: OWASP ZAP vulnerability assessment effectiveness

Figure 4.5 shows the effectiveness OWASP Zap precisely. All the values in the categories are scattered above the random line and are located left-sided, which means that its rate of False Positives is low and True Positive Rate is average. That is due to the higher number of True Negatives and True Positives and absence of False Negatives. Hence, the accuracy of this tool is very high. However, the number of True Positives reduce the coverage to the tool in low ranges.

Arachni is selected due to its popularity in the community, and the selected version executed in the specified environment with the OWASP Benchmark 1.2 for 6 hours which is the best possible scenario. However, it could not complete the complete analysis. It has a resource problem with the Request concurrency and crashes the system. Due to the incompleteness of the analysis, the Arachni is not compared with the other systems.

4.1.3 Overall Assessment

Table 4.3 Overall vulnerability assessment statistics

Tool	TP	FN	TN	FP	TPR	FPR	Score
FBwFindSecBugs v1.4.6	1370	45	622	703	96.84%	57.74%	39.10%
FindBugs v3.0.1	150	1265	1194	131	5.12%	5.19%	-0.07%
PMD v5.2.3	0	1415	1325	0	0.00%	0.00%	0.00%
SonarQube Java Plugin v3.14	607	808	1184	141	50.36%	17.02%	33.34%
OWASP ZAP vD-2016-09-05	306	1109	1322	3	19.95%	0.12%	19.84%

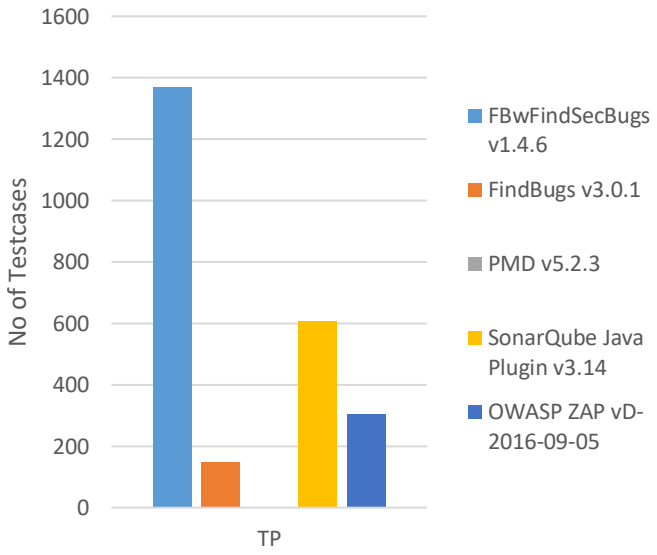

Figure 4.6: True Positive count of the products

Figure 4.6 represents the Comparison of the True Positives reported by the selected tools. From the True Positive value, we can identify how much issues can be identified by the tool. FindBugsSec is the premier tool in the set from this aspect where it reported 1370 issues. This performance of FindBugsSec is more than twice as better as the second-best SonarQube. PMD was the lowest performing tool where it reported a

value of 0, which means that no true vulnerability was found. This was due to it not having any related security rules implemented.

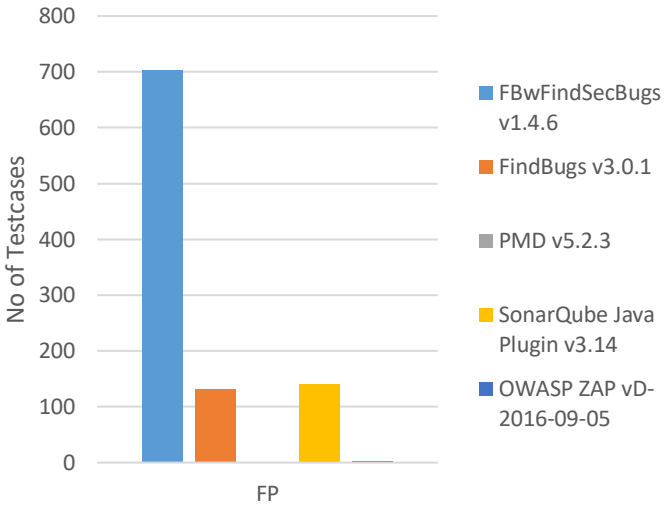

Figure 4.7: False Positive count of the products

Figure 4.7 represents the False Positive count of the product where it reported as vulnerabilities for non-vulnerabilities. Again, the FindSecBugs plugin performed as the highest scorer despite expecting a lesser value for this category. While SonarQube and FindBug reported moderate values for the test, PMD scored the lowest value due to lack of security rules. OWASP ZAP is the most accurate, only reporting 3 False Positives.

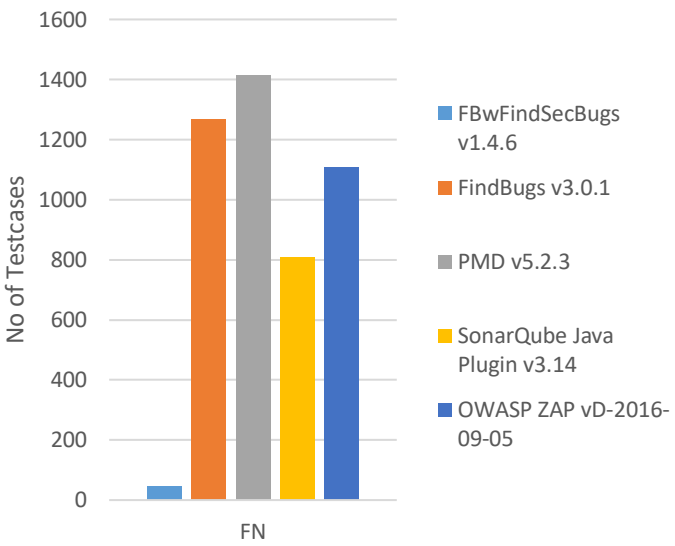

Figure 4.8 False Negative count of the products

Figure 4.8 represents the number of occurrences where the tool was unable to report an issue for a vulnerable where it should have done so. In the test, PMD was the highest scorer where it reported none of the issues as vulnerable due to the absence of security rules and Findbugs was the second due to the same reason. However, despite the existence of focused security rules, OWASP-ZAP reported a large number of false negatives which reduced its coverage significantly. In this category as well, FindSecBug reported the lowest score as 45 which contributed to increase its coverage.

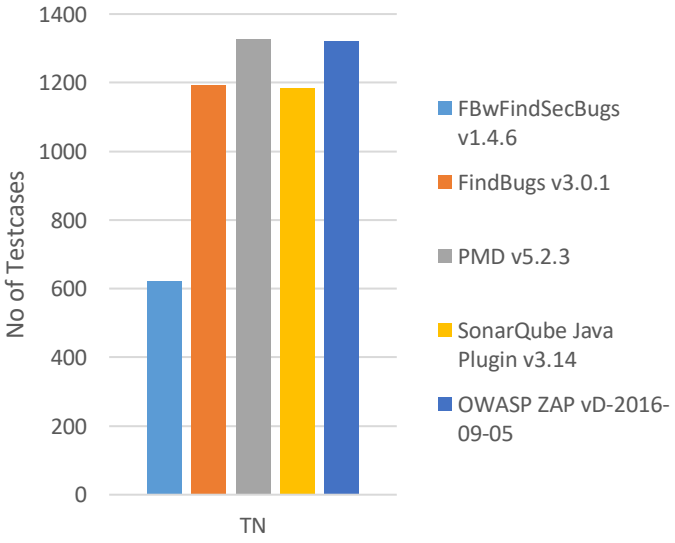

Figure 4. 9: Product-wise True Negative count

Figure 4.9 shows the scenarios where it has not reported vulnerability for non-vulnerability issues correctly. In this category, PMD reported the highest value again which is due to the absence of the security rules. OWASP Zap is the real leader in this category which helped it to increase its accuracy. FindBugs and SonarQube reported a fair number of issues as True Negatives. FindSecuBug reported the lowest number of True Negatives which lead to lower its final score in the accuracy measure.

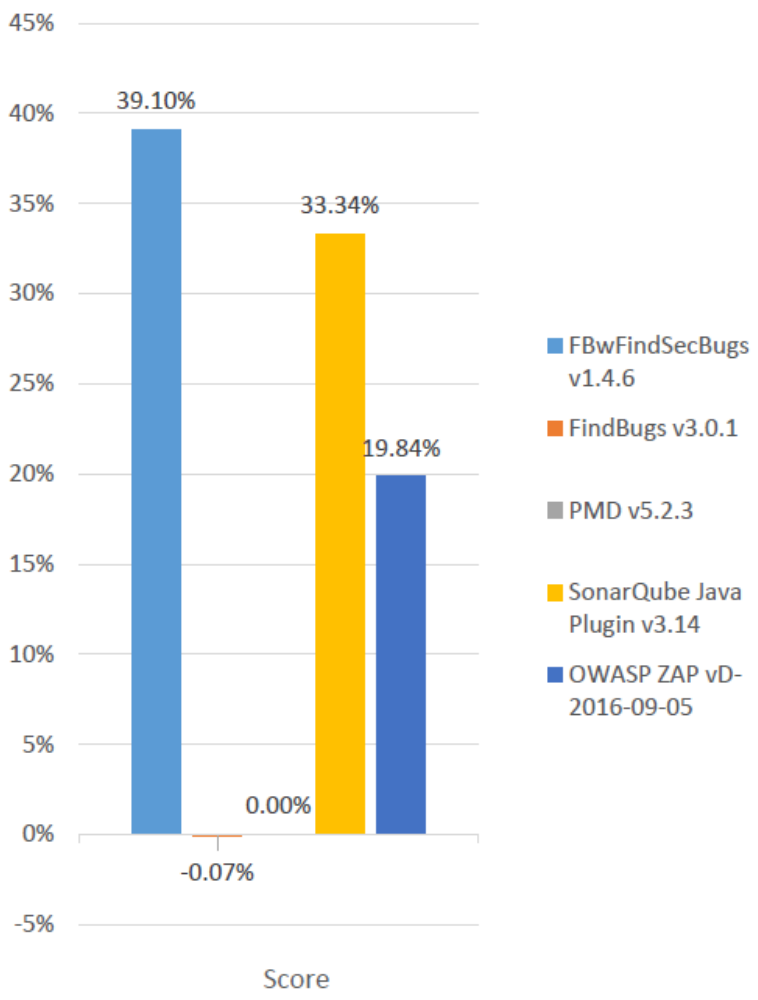

Figure 4.10: Overall score gained by the tools

Overall score has been calculated by reducing the True Negative Rate from True Positive Rate in order to take the correct measurement. FindBug has the lowest score as a minus value due to the number of False Positives reported. PMD has a far better value numerically due to the fact that it does not report any false positive, but it does not contain any security rules. OWASP-ZAP has the next lowest due to lack of detection or the coverage. SonarQube has the second-best score due to increase in coverage than the rest of the tools. FindSecbug is the highest scorer merely because it has the highest coverage despite the loss of its accuracy.

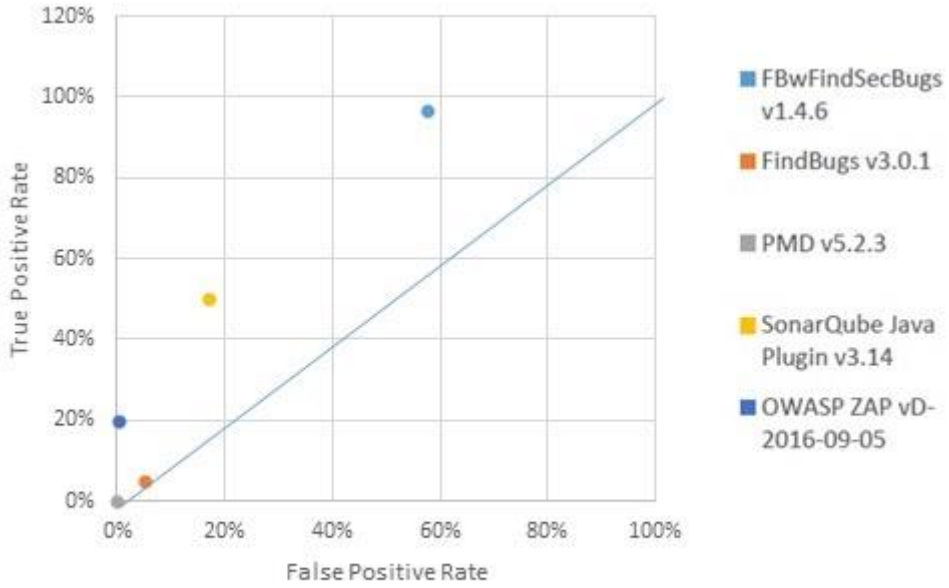

Figure 4.11: Overall effectiveness of products

Figure 4.11 shows the overall effectiveness of the tools. If a tool is plotted near the random line, then it is described as a tool that characterises a random behaviour and therefore deemed to be ineffective. As clearly shown by the test results, PMD and FindBug fall into this category due to lack of security rules. Rest of the tools are represented in the graph above the random line meaning they have some level of effectiveness. Even though FindSecBug reported the highest True Positive rate, it is plotted in the middle of the graph due to the highest number of false positives it has. SonarQube has been plotted more towards the left of the graph but with moderate

coverage. Although OWASP Zap is located as a leftmost item in the graph behind SonarQube, it is also not in the optimal range due to lack of coverage. According to the graph, there is no tool plotted on the top-left of the graph where the ideal tool should be located.

4.2 Experiment II

This test was specifically designed to check some aspects of the component-based software development where it will check the vulnerability identification early in the SDLC especially with vulnerable development utility tools.

Table 4.4: Experiment II results

Tool	**FindBugs**	**FindSecBug**	**PMD**	**SonarQube**	**DependencyCheck**	**OWASP ZAP**
GWT	-	-	-	-	✓	-
Jenkins	-	-	-	-	-	✓
Hibernate	-	-	-	-	✓	-
Maven	-	-	-	-	-	-
Oracle Java 7	-	-	-	-	-	-
Spring MVC	-	-	-	-	✓	-
Eclipse IDE	-	-	-	-	-	-

In the experiment, it shows that none of the tools has been able to address the vulnerabilities except the DependencyCheck. It is due to those specific tools being designed to test the direct code and the execution paths of that code in the runtime. Therefore, those tools are incapable of identifying dependent third-party components when the source code is not available and the vulnerable code is not used in the application execution. DependencyCheck is designed to analyse the security of the third-party dependencies. Therefore, it was able to figure out those vulnerabilities. However, it was unable to figure out the vulnerabilities in the development of infrastructure tools.

4.3 Experiment III

Impact of release frequency of the security product needs to be analysed against modern development practices to study the effectiveness. Hence, incremental releases of the following main tools have been analysed to deduce the fact that the updated tool version often contains improvements in vulnerability detection. Table 4.5 shows the statistics of OWASP Benchmark.

Table 4.5: Vulnerability assessment in incremental versions

Tool version	Release Date	TP	FN	TN	FP	TPR	FPR	Score
FBwFindSecBugs v1.4.0	31-Mar-15	716	699	899	426	47.64%	35.99%	11.65%
FBwFindSecBugs v1.4.3	17-Sep-15	1026	389	791	534	77.60%	45.21%	32.39%
FBwFindSecBugs v1.4.4	20-Nov-15	1044	371	788	537	78.77%	44.64%	34.13%
FBwFindSecBugs v1.4.5	5-Jan-16	1355	60	622	703	95.20%	57.74%	37.46%
FBwFindSecBugs v1.4.6	3-Jun-16	1370	45	622	703	96.84%	57.74%	39.10%
OWASP ZAP vD-2015-08-24	24-Aug-15	245	1170	1324	1	18.03%	0.04%	17.99%
OWASP ZAP vD-2016-09-05	5-Sep-16	306	1109	1322	3	19.95%	0.12%	19.84%

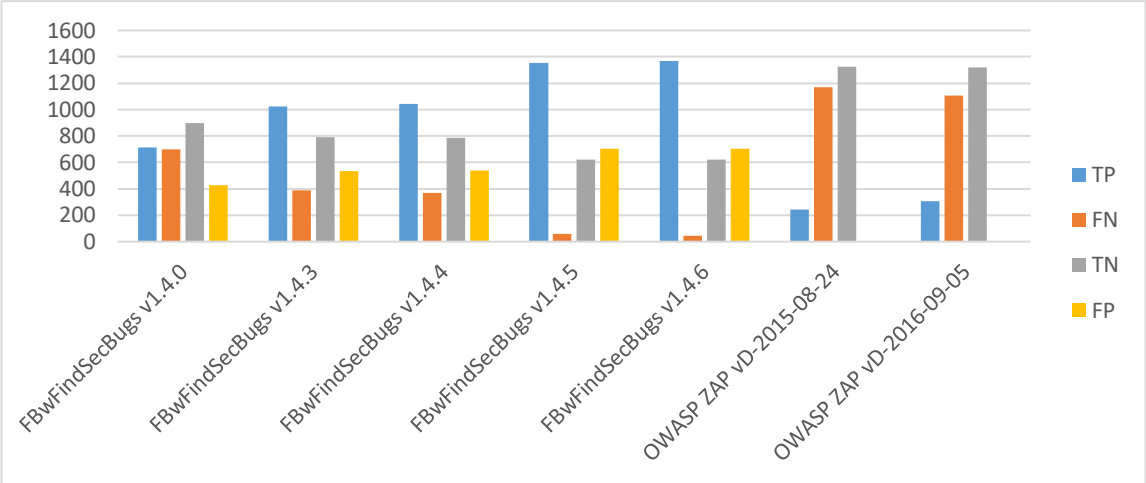

Figure 4.12: Number of issues in incremental versions

In the experiment, FindSecBug is used as a SAST tool and OWASP ZAP tool has been used as a DAST tool. Compared to release version 1.4.0 of FindSecBug, version 1.4.3 has significant improvement in the number of the detected issues. However, the average gap between the two releases is two months, and a rough average of detected false positive is 15. The False Positives are also reducing significantly in the incremental versions. However, even though OWASP ZAP has frequent updates, improvements are slightly less. Comparing the results between 2015 and 2016 releases, the number of True Positives increased only by 60, and False Positive count has increased only seldom. Since OWASP Zap uses DAST, it has been able to score a high accuracy.

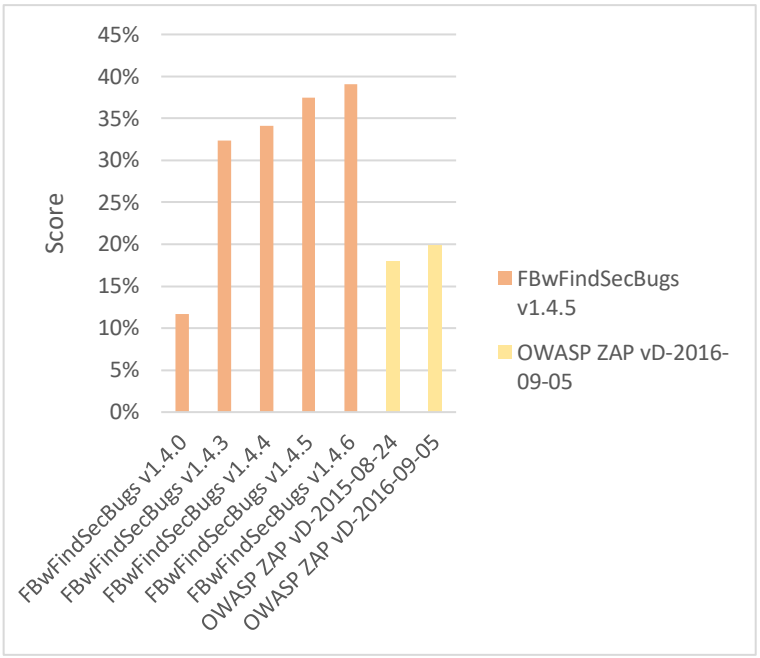

Figure 4.13 overall progression in incremental versions

Figure 4.13 shows the overall progression of the OWASP ZAP and FindSecBug plugin. In the graph, it shows that FindSecBug has significant improvements in its overall score in its incremental releases. However, despite annually released versions of OWASP-ZAP, it has only about 1.84% improvement in overall score.

In order to test the support for modern development methodologies and concepts, features of the same set of security tools have been analysed and Table 4.6 list the facts gathered.

Table 4.6: Feature analysis of the security tools

Feature	**FindBugs**	**FindSecBug**	**PMD**	**SonarQube**	**DependencyCheck**	**OWASP ZAP**
Release frequency	6 months	2-3 months	monthly	1-3 months	2-3 months	weekly
Security Rules/Database updates frequency	NA	NA	NA	NA	weekly	weekly
Tool support for Agile Development	✓	✓	✓	✓	✓	✓

According to the analysis of the set of features, almost all of the tools have support for the Agile development concept such as Continuous Integration and Continuous Delivery (CICD). However, except for OWASP ZAP, all the other tools have a release cycle of more than one month long. It is interesting to note that DepenencyChek has a mechanism to update the security repository bi-weekly. Therefore, some of the tools need to think about updating mechanism of their security policies or the repositories in a more regular manner.

4.4 Limitations & Improvements to the Experiment

Experiment II and III has been done from a qualitative perspective to check the supportiveness for the specific features and effectiveness of the product. However, as a future improvement to the tests carried out in this research, developing a set of more formal test cases would be beneficial to gather improved qualitative measurements.

Due to the lack of product support for gathering security vulnerabilities in the requirement phase, this research study did not cover that phase but focused more on the development and testing phases. It is another area that could have greater emphasis in future research studies with appropriate tool development.

Since this research had a developer-oriented perspective, and also many system level binary analysis work has been done in the area beforehand, our tests did not place much emphasis on binary analysis. Even though it was not tested in the experiments, it would be better to recheck the use of binary analysis since empirical evidence has shown that the claims made by vendors are not met in reality.

The experiments conducted in this research did not cover the deployment phase as it is more similar to post-production testing. However, nowadays this deployment happened more frequently, and therefore vulnerability assessment of binary-level is possible in every iteration. Hence, the tool support for security testing in such dynamic deployment environments is another area that this experiment and study can be improved. Furthermore, this vulnerability assessment did not monitor resource utilisation. Hence, as an improvement, such an experiment can be performed to get a measurement of resource usage and resource efficiency.

Since the primary goal of the study was to evaluate the effectiveness of vulnerability assessment done by widely used tools, open source products have been used. However, to infer a better understanding about the performance of proprietary tools, the study can be extended to evaluate commercial tools as well.

In the quantitative methodology, the accuracy of the measurement is entirely based on the benchmark used and the test cases included in it. Despite the hardness of writing proper automated tests cases to test security vulnerabilities, it is useful if it is possible to extend the test cases to cover more areas. As a future improvement in the experiment, a merged test benchmark can be created from the existing test tools where OWASP Benchmark provides a much-needed platform.

5. Conclusion

5.1 Findings

From the preliminary review of the literature and other reviewed articles and publications, it was found that providing tool support to cover the security aspects in the requirement gathering phase is difficult from a process automation perspective. Even though several good suggestions on this area could be found in the academic research literature, it was not possible to locate a real-world tool implementation to test the effectiveness in that area. However, in place of tools, there are best practices introduced by ISO/IEC 15408 to evaluate security requirements.

In the development and test stages, SAST and DAST are the main methodologies used to analyse the security vulnerabilities of the composed systems. Even though many vendors in the industry claim highly of the efficiency of their relevant tools, by the experiments conducted in this research, it was found that this is not the real picture in terms of the security vulnerability assessment. According to the overall score for products graphed in figure 4.10, maximum score found for SAST is around 40%, and for DAST it is around 20%.

As figure 4.2 and figure 4.3 shows for FindSecBugs effectiveness and figure 4.4 and figure 4.5 shows for OWASP Zap effectiveness, the existing evaluated products are most useful in finding out a particular category of specific vulnerabilities. However, those tools did not seem capable of covering all the categories with expected accuracy.

In terms of component-based development, existing products are more effective and focused on certain types of applications such as web applications. Current product support for component-based development is in written code and the runtime application only. However, as results of experiment II in Table 4.4 demonstrates, the vulnerabilities in interactions and the tools used in the software development itself and third-party libraries have not been identified correctly by most of the current tools.

According to the feature analysis of security tools in table 4.6, most of the open source tools support Agile methodologies and frameworks and also provides interaction for continuous integration and delivery. However, the security vulnerability databases and methodologies used in those open source tools were not supportive enough to get quick feedback.

5.2 Conclusion

Since software has become one of the most critical parts of all the industries and also the increasing trend of cybercrimes targeting vulnerabilities in such software systems, securing of software is now an essential requirement. Even though, there exist facilities to assess the security of the software in post-production environments, the support for pre-production is minimal. On the other hand, today, most of the systems are being developed by composing a diverse range of components where holistic security is unknown or untested. Furthermore, the development methodologies have been changed over time where more iterative, Agile methodologies have been adopted. Therefore, this study focused on analysing the vulnerability assessment in composed systems and more specifically from a software development perspective as against a software usage perspective.

In the experimental design, a quantitative approach is used to measure the effectiveness of the selected products. On the other hand, qualitative features of the products have been analysed against their support for the component-based development and Agile development methodologies.

In the quantitative analysis, it was found that despite the vendor's claims about the effectiveness of their products, it is far less than it expected. According to the overall score gained by the products shown in figure 4.10, the effectiveness of SAST is about 40%, and DAST is around 20%. As depicted in figure 4.11 of overall effectiveness, DAST tools accuracy is high, but coverage is very minimal in most areas of the analysis. Even though SAST has more coverage in terms of the number of vulnerabilities, its accuracy is dubious due to the False Positives and True Negatives.

In the analysis of product features qualitatively, it is found that most of the products support for Agile methodologies, continuous integration, and continuous delivery. According to table 4.6, the open source products that have been evaluated are supported with frequent releases, but still, it needs to have shorter release cycles in order to be more effective. OWASP ZAP and DependencyCheck is the only tools from the considered tools that separated its security policy mechanism from the core product, which can be considered as a valid action from an architecture perspective.

According to the findings, SAST focuses mostly on the development stage of the SDLC and DAST is focused on the testing stage. But it is required for these tools to improve accuracy and coverage in order to make sure they are effective in vulnerability testing of the software systems. Even though IAST is a currently promising area in the field, it is still a questionable methodology in usability in an Agile development environment due to the amount of time and resources it takes to complete the analysis. The existing tools that support requirement gathering and deployment stage in modern development need to be improved by reducing feedback time and increasing the quality of the feedback. Each type of product has its strengths and weaknesses. However in the perspective of today's software development methodologies, having one tool which covers all the phases in SDLC with higher accuracy, and coverage is the ultimate expectation for the future in this arena.

In conclusion, despite the fact of valuable service provided by today's products, there will be more work that needs to be done in this area to have an effective software vulnerability measurement in the modern component-based software development.

5.3 Future Improvements

Dynamic reconfigurability and deployability of tools would be a significant improvement considering the deployment stages. Therefore, support for virtual machines and containers is another stream that can be improved.

Currently, most of the benchmarks are focused on web applications. Extending support for other application types such as application servers, and mobile systems is

also a trending area. Optimising test cases to run on the application containers is another area of improvements that can be done in the benchmark tools and test cases perspective. Center for Internet Security (CIS) benchmark provides a wide range of test cases for application servers, container-based environments and many new technologies [41]. However, those CIS benchmarks could be automated to increase popularity and usability.

From a vulnerability assessment product perspective, those can be improved by detailed analysis of the assessment report against a benchmark. Also, those need to increase more coverage and accuracy and needs to be optimised by specific strategies and categories. However, having a merged or combined tool of those security rules and strategies is more beneficial to analyse the vulnerability of component-based software development.

References

[1] Chess, Brian, and Gary McGraw. "Static analysis for security." IEEE security & privacy 2, no. 6 (2004): 76-79.

[2] Khan, U. A., I. A. Al-Bidewi, and K. Gupta. "Challenges in Component-Based Software Engineering as the Technology of the Modern Era." International Journal of Internet Computing (IJIC) 1 (2011).

[3] Hopkins, Jon. "Component primer." Communications of the ACM 43, no. 10 (2000): 27-27.

[4] Szyperski, Clemens, Dominik Gruntz, and Stephan Murer. Component software: beyond object-oriented programming. Pearson Education, 2002.

[5] D'souza, Desmond F., and Alan Cameron Wills. Objects, components, and frameworks with UML: the catalysis approach. Vol. 1. Reading: Addison-Wesley, 1998.

[6] Meijler, Theo Dirk, and Oscar Nierstrasz. "Beyond objects: components." Cooperative information systems: current trends and directions (1997): 49-78.

[7] Han, Jun. "A comprehensive interface definition framework for software components." In Proceedings 1998 Asia Pacific Software Engineering Conference (Cat. No. 98EX240), pp. 110-117. IEEE, 1998.

[8] Egyed, Alexander, and Robert Balzer. "Unfriendly COTS integration-instrumentation and interfaces for improved plugability." In Proceedings 16th Annual International Conference on Automated Software Engineering (ASE 2001), pp. 223-231. IEEE, 2001.

[9] Boehm, Barry, and Chris Abts. "COTS Integration: Plug and pray?." Computer 32, no. 1 (1999): 135-138.

[10] Bachmann, Felix, Len Bass, Charles Buhman, Santiago Comella-Dorda, Fred Long, John Robert, Robert Seacord, and Kurt Wallnau. Volume II: Technical concepts of component-based software engineering. Technical Report CMU/SEI-2000-TR-008, Carnegie Mellon Software Engineering Institute, 2000.

[11] P. Eskelin, “Component interaction patterns,” in 6th Annual Conf. on the Pattern Languages of Programs (PLoP). Urbana, IL, USA, 1999.

[12] Fowler, Martin. "Inversion of control containers and the dependency injection pattern." (2004).

[13] Yang, Hong Yul, Ewan Tempero, and Hayden Melton. "An empirical study into use of dependency injection in java." In 19th Australian Conference on Software Engineering (aswec 2008), pp. 239-247. IEEE, 2008.

[14] Vanbrabant, Robbie. Google Guice: agile lightweight dependency injection framework. APress, 2008.

[15] Prasanna, Dhanji R. Dependency injection. Manning Publications Co., 2009.

[16] Krueger, Charles W. "Software reuse." ACM Computing Surveys (CSUR) 24, no. 2 (1992): 131-183.

[17] Gao, Jerry, H-SJ Tsao, and Ye Wu. Testing and quality assurance for component-based software. Artech House, 2003.

[18] Alnusair, Awny, and Tian Zhao. "Component search and reuse: An ontology-based approach." In 2010 IEEE International Conference on Information Reuse & Integration, pp. 258-261. IEEE, 2010.

[19] Chacon, Scott, and Ben Straub. Pro git. Apress, 2014.

[20] Weiss, Dawid. "Quantitative analysis of open source projects on SourceForge." In Proceedings of the First International Conference on Open Source Systems, Genova, pp. 140-147. 2005.

[21] Sonatype Inc, “Concepts and Benefits of Repo Management.” [Online]. Available: https://www.sonatype.com/concepts-benefits-repo-management. [Accessed: 30-Jun-2018].

[22] JFrog Ltd ,“JFrog Enterprise+: An End-to-End Platform for Global DevOps” [Online]. Available: https://jfrog.com/wp-content/uploads/2018/05/White-Paper-Enterprise-Plus-An-End-To-End-Platform-For-Global-DevOps.pdf. [Accessed: 30-Jun-2018].

[23] Khan, Khaled, Jun Han, and Yuliang Zheng. "Security properties of software components." In International Workshop on Information Security, pp. 52-56. Springer, Berlin, Heidelberg, 1999.

[24] Khan, K., Jun Han, and Yuliang Zheng. "A scenario-based security characterisation of software components." In Proceedings of the 3rd Australasian Workshop on Software and System Architectures, pp. 55-63. 2000.

[25] Khan, Khaled M., Jun Han, and Yuliang Zheng. "Characterising user data protection of software components." In Proceedings 2000 Australian Software Engineering Conference, pp. 3-11. IEEE, 2000.

[26] Khan, Khaled M., and Jun Han. "A process framework for characterising security properties of component-based software systems." In 2004 Australian Software Engineering Conference. Proceedings., pp. 358-367. IEEE, 2004.

[27] Khan, Khaled M., and Jun Han. "Assessing security properties of software components: A software engineer's perspective." In Australian Software Engineering Conference (ASWEC'06), pp. 10-pp. IEEE, 2006.

[28] Busch, Axel, Misha Strittmatter, and Anne Koziolek. "Assessing security to compare architecture alternatives of component-based systems." In 2015 IEEE International Conference on Software Quality, Reliability and Security, pp. 99-108. IEEE, 2015.

[29] Nazir, Shah, Sara Shahzad, Muhammad Nazir, and Hanif ur Rehman. "Evaluating security of software components using analytic network process." In 2013 11th International Conference on Frontiers of Information Technology, pp. 183-188. IEEE, 2013.

[30] Swiderski, Frank, and Window Snyder. Threat modelling. Microsoft Press, 2004.

[31] “ISO/IEC 15408-1:2009(en), Information technology — Security techniques — Evaluation criteria for IT security — Part 1: Introduction and general model.” [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:15408:-1:ed-3:v2:en. [Accessed: 30-Jun-2018].

[23] “ISO/IEC 15408-2:2008(en), Information technology — Security techniques — Evaluation criteria for IT security — Part 2: Security functional components.” [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:15408:-2:ed-3:v2:en. [Accessed: 30-Jun-2018].

[33] “ISO/IEC 15408-3:2008(en), Information technology — Security techniques — Evaluation criteria for IT security — Part 3: Security assurance components.” [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:15408:-3:ed-3:v2:en. [Accessed: 30-Jun-2018].

[34] Olivier Gaudin, “SonarSource Blog: What makes Checkstyle, PMD, Findbugs and Macker complementary ?” [Online]. Available: https://blog.sonarsource.com/what-makes-checkstyle-pmd-findbugs-and-macker-complementary. [Accessed: 30-Jun-2018].

[35] N. Antunes and M. Vieira, “Benchmarking Vulnerability Detection Tools for Web Services,” in 2010 IEEE International Conference on Web Services, 2010, pp. 203–210.

[36] Chen, Shay. "The web application vulnerability scanners benchmark." Denim Group (2014).

[37] National Institute of Standards and Technology (NIST),"Juliet Test Suite v1.2 for Java" [Online]. Available: https://samate.nist.gov/SARD/resources/ Juliet_Test_Suite_v1.2_for_Java_-_User_Guide.pdf [Accessed: 30-Jun-2018].

[38] “Benchmark - OWASP.” [Online]. Available: https://www.owasp.org/index.php/Benchmark. [Accessed: 30-Jun-2018].

[39] CVE Details List. [Online]. Available: https://www.cvedetails.com/vulnerability-list/ [Accessed: 30-Jun-2018].

[40] Youden, William J. "Index for rating diagnostic tests." Cancer3, no. 1 (1950): 32-35.

[41] Center for Internet Security, “CIS Benchmarks.” [Online]. Available: https://www.cisecurity.org/cis-benchmarks/. [Accessed: 30-Jun-2018].