Software Security Testing

Bhupender Kumar Saini

Institute of Software Technology,
University of Koblenz and Landau, Koblenz, Germany
bksaini@uni-koblenz.de

Abstract. Software security testing is the key aspect to ensure reliability, confidence, and trust on software application. Secure software enhances the quality of the application and reduces the risk of intentional/unintentional cyber attacks or failures that can severely affect the reputation of a software company and can leads to negative consequences. This paper discuss about the importance of the security testing, challenges in implementing, and different testing approaches in software development life cycle. With the advancement of technology day by day, the frequency of cyber attacks are increasing and the nature of cyber attacks are becoming more complex. Many research has been done recently towards creating effective approaches in order to increase the security testing scope and effectiveness. In this paper, analyzed the widely used testing techniques and also respective manual/automation tools. Finally, brief discussion about the important prerequisite for designing effective security test cases and increasing testing scope in secure software testing i.e. mindset.

Keywords: Software Security,· Security Testing · IT security · Secure software Development · Security testing tools · Security testing techniques

1 Motivation

To realize the importance of need of security, one need to understand that with the advancement of the technology, the risk of cyber attacks, privacy compromise events, data loss, safety and security of the human being is also increasing every year. As per Symantec Internet security report 2019 [1], one in ten URLS are malicious, web attacks significantly increased by 56% , 33% increase in Mobile ransomware attacks and also one in 36 mobile devices has a high risk app installed , more attacks groups are forming and on average 55 organisation attacked by them.
As per Kaspersky Lab an antivirus company report [2] states that 90% of businesses admitted a security incident. Additionally, 46% of businesses lost sensitive

[1] https://www-west.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf

[2] https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf

data due to an internal or external security threat. Enterprises lose half a million US dollar on average from a security breach. However, quantifying only on the basis of monetary loss is not a good idea. Additionally, Top three most expensive types of security breaches are third party failure, fraud by employees and cyber espionage. Top three IT security threats that lead to data loss are Malware, Phishing attacks and accidental leaks by the staff. With respect to consequences of security breaches loss of access of business-critical information, damage of company reputation and temporary loss of ability. Only certainty that we have - the cost of a security breach is always higher than the cost of protection.
Security implementation has broader scope which has to be introduced in each and every part of the process that could be a monitoring access of employees in the organisation, emphasizing security awareness, or implementing practices designed by security experts in the organisation. In this paper, our focus only on one aspect i.e.Software security which can be defined as the process to identify whether the security features of software implementation are consistent with the design. But,
Why bother with software security?
The answer is quite simple: the software is touching all the aspect of human being and world is more connected now than it has ever been, and no doubt it will grow even more over time. This incredible level of dependency on software has created a huge threat environment and, hence, hugely escalated risk for all software users. Hence, now a days it is an important and growing concern for software organisation.
If we see the statistics presented by Edgescan 2019 vulnerability report [3], 19% of all vulnerabilities were associated with (Layer 7) web applications, API's, etc., and 81% were network vulnerabilities. Many breaches using hacking attacks by exploiting vulnerability and malware are preventable. Introducing activities such as security integration into the SDLC, DevSecOps, patch management, continuous vulnerability management and continuous asset profiling (i.e. visibility), can help us identify and mitigate such weaknesses before we deploy systems, or at least before they become a real problem.In this paper, the security testing methods, techniques of security testing, and mindset required for security testing is briefly explained.

2 Introduction to Software Security Testing

Now a days, computer and software are touching each and every aspect of life and growing significantly on large scale, which also increasing the concern towards the possibility of unavoidable vulnerabilities which on later stage can be exploited by attackers as well as increasing the risk of occurring during production. Therefore, introducing efficient way of security testing is one of the major concern for organisation.
There are many definition for the Software Security ,

[3] `https://www.edgescan.com/wp-content/uploads/2019/02/edgescan-Vulnerability-Stats-Report-2019.pdf`

“ Software Security is the software’s ability to highly resist, tolerate, and recover from cases that strongly threaten the product.”

“The primary purpose of engineering software that continues to function correctly and efficiently under several types of malicious attacks.”

“Software security is simply about the process of building secure software by designing software to be secure, emphasizing that the software is secure, and educating software architects, developers, and users about how to build and use secure things.”
According to Tian-Yang [38], Software security requirements mainly include data confidentiality, integrity, availability, authentication, authorization, access control, audit, privacy protection, and security management. The main focus of the software security testing is to reveal vulnerabilities in the software product or the application which can be operating systems, software, database system, and more. But, as a limitation software security testing can only reveal the presence of vulnerabilities which should not be misunderstood as absence of vulnerabilities. Anyway, it increases confidence on the product and reduce risk from attacks or unexpected behaviour which can cost money and reputation if discover at later stage.
A different perspective presented by Arkin, Brad[1] that there is no relation between software security defects and vulnerabilities to security functionality —rather, they originate from an attacker’s unexpected but intentional misuses of the application. Furthermore, if we characterize functional testing as testing for positives—verifying feature how it should perform— then security testing is in somewhat testing for negatives test scenarios possibly driven by abuse cases and architectural risks to simulate the behaviour of system under attack. Which arises the limitation, the scope of generative negative test scenarios is solely depend on the security tester imagination, expertise and knowledge.
Why software security is important ?
Insufficient planning and negligence towards security testing can lead to unexpected consequences:

- Software product with vulnerabilities, like poor encryption, unaddressed or unprotected will often impede general productivity and negatively impact applications currently in production..
- Security flaws and breaches can lead to fines and sanctions for lack of regulatory compliance.
- Not addressing these issues before releasing a product, will eventually require you to devote additional time and effort.
- If users feel they can’t rely on your products to keep them or their information safe, you will lose customers and profits. It can also lead to loss of consumer trust.
- Software quality, reliability and security are tightly coupled.[19]

2.1 Software Security Testing Terminologies

- **Defect** is a variation or deviation from the original business requirements
- **Fault** is a condition that causes the software to fail to perform its required function
- **Bug** is the consequence/outcome of a coding fault.
- **Asset** is a data item, or a system component that has to be protected like computer, server, etc. In the security context, such an asset has one or multiple security properties assigned that have to hold for that asset.
- **Vulnerability** is a special type of fault. If the fault is related to security properties, it is called a vulnerability. A vulnerability is always related to one or more assets and their corresponding security properties. An exploitation of a vulnerability attacks an asset by violating the associated security property. Since vulnerabilities are always associated with the protection of an asset, the security relevant fault is usually correlated with a mechanism that protects the asset. A vulnerability either means that (1) the responsible security mechanism is completely missing, or (2) the security mechanism is in place but is implemented in a faulty way.
- **Zero day Vulnerability** is a vulnerability in a system or device that has been disclosed but is not yet patched. An exploit that attacks a zero-day vulnerability is called a zero-day exploit.[4]. The term "zero-day" refers to a newly discovered software vulnerability. Because the developer has just learned of the flaw, it also means an official patch or update to fix the issue hasn't been released
- **Threat** is the potential cause of an unwanted incident that harms or reduces the value of an asset. For instance, a threat may be a hacker, power outages, or malicious insiders. An attack is defined by the steps a malicious or inadvertently incorrectly behaving entity performs to the end of turning a threat into an actual corruption of an asset's properties. This is usually done by exploiting a vulnerability.
- **Exploit** is a concrete malicious input that makes use of the vulnerability in the system under test (SUT) and violates the property of an asset. Vulnerabilities can often be exploited in different ways. One concrete exploit selects a specific asset and a specific property, and makes use of the vulnerability to violate the property for the selected asset.
- **Risk assessment** is the identification of hazards that could negatively impact an organization's ability to conduct business. These assessments help identify these inherent business risks and provide measures, processes and controls to reduce the impact of these risks to business operations.
- **Hacking** is an attempt to exploit a computer system or a private network inside a computer. Simply put, it is the unauthorised access to or control over computer network security systems for some illicit purpose.

[4] `https://www.trendmicro.com/vinfo/us/security/definition/zero-day-vulnerability`

- **Penetration testing** Penetration testing, also called pen testing or ethical hacking, is the practice of testing a computer system, network or web application to find security vulnerabilities that an attacker could exploit. Penetration testing can be automated with software applications or performed manually.
- **Confidentiality** is the assurance that information is not disclosed to unauthorized individuals, processes, or devices.
- **Availability** guarantees timely, reliable access to data and information services for authorized users.
- **Integrity** is provided when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed.
- **Non-repudiation** is the assurance that none of the partners taking part in a transaction can later deny of having participated.
- **Authentication** is a security measure designed to establish the validity of a trans- mission, message, or originator, or a means of verifying an individual's authorization to receive specific categories of information.
- **Authorization** provides access privileges granted to a user, program, or process.

3 Scope And Challenges

To successfully develop a secure software, responsible individual should verify and validate the security requirements as well as should gather information about the already known issues. Security requirements are taken as a foundation to derive and execute tests against a system under test. Yet, these positive requirements by far do not cover all the relevant security aspects[42].Hence, especially in the event of security testing, negative requirements, derived from risk analysis, are vital to be incorporated. Additionally, one should follow online community like Open Web Application Security Project(OWASP) that caters with articles, methodologies, documentation, tools and technologies in the field of the web application security to gain much required information for security testing. Additionally, it also provide top 10 known vulnerability which are as follows[5]:

1. Injection.
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting XSS
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging Monitoring

[5] `https://owasp.org/www-project-top-ten/`

The Common Weakness Enumeration (CWE) [6] provides a list of Most Dangerous Software Errors.. The SANS Top-25 list shows the most widespread and critical errors that are applicable to all types of applications [7].Considering these details during security testing is highly recommended.

3.1 Challenges in Security Testing

American security today website [8], provided brief overview of challenges faced during Security testing are :

Speed of Agile Development:
Agile is majorly adopted methodologies,but also posses challenges because of its fast paced working strategy to reduce delivery time. It encourages developers to ignore issues in order to meet deadlines. There may be chances that security testing procedures will be bypassed or partially ignored. However, employing security testing process from early phase of development can prevent vulnerabilities in first place.Static Application Security Testing (SAST), Dynamic Application Security Testing (DAST), manual testing, threat modeling and using an application vulnerability management system are all practices can be introduced to overcome these challenges.

Risks of Using Open Source Components:
To save time, developer uses open source component with no/little cognisance about internal insight or dependencies of the component.These open source component are used in a black box fashion. This results in vulnerabilities, complexity, and inconsistencies in the overall product. Avoiding use of open source components until it not possible to write the code. Using application vulnerability tools that perform Software Composition Analysis (SCA) can help locating and tracking vulnerable components.

Vulnerabilities in Code:
Every programming language are also prone to vulnerabilties and has its own vulnerabilities which can affect the security of the application. For example, injection attacks with JavaScript or SQL or Cross-Site Scripting (XSS) with PHP and Ruby. Injection attack allows an attacker to read or modify databases by submitting code through unfiltered user input mechanisms. XSS allows attackers to execute scripts in user browsers, redirecting users to malicious sites or stealing information through cookies. Increasing awareness regarding known issues of programming languages can significantly minimize the security risks. Also, the web development languages which we use may lack in enforcing the security policy which may even violate the integrity and confidentiality of the web application [26].

Lack of AppSec Planning:
Appsec defined as the process of securing all the software a business uses. Lack

[6] https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html
[7] https://www.sans.org/top25-software-errors
[8] https://americansecuritytoday.com/secure-software-development-challenges-and-considerations/

of proper planning can lead to unmanageable security issues and unclear expectations of requirement for production ready products which can also lead the team towards ineffective methods.

Lack of effective approaches to detect when data is encrypted: According to Kirubakaran [28], Software application using encrypted data communication like HTTPS makes traffic analysis impossible. Monitoring only sensitive data can be useful as it is the main target of the attacks.

3.2 Requirements for Security Testing

In paper presented by Firesmith, Donald [15] about security requirement has explicate the requirements worth remembering in the era of virus alerts, malicious crackers and the risk of cyber terrorism based on objectives of security requirements are :

1. **Identification Requirements**
   Ensuring users and client application identities identified and verified.
2. **Authentication Requirements**
   Ensure that externals are actually who or what they claim to be and thereby to avoid compromising security to an impostor.
3. **Authorization Requirements**
   Ensuring that users and client applications can only access data and services they are authorised for.
4. **Immunity Requirements**
   Ensuring that prevention from the malicious attacks do infect the application.
5. **Integrity Requirements**
   Ensuring prevention from intentional corruption of data.
6. **Intrusion Detection Requirements**
   Ensure an application or component shall detect and record attempted access or modification by unauthorized individuals.
7. **Non-repudiation Requirements**
   Ensuring a business, application, or component shall prevent a party to one of its interactions (e.g., message, transaction) from denying having participated in all or part of the interaction.
8. **Privacy Requirements**
   Ensuring confidentiality and data are kept private.
9. **Security Auditing Requirements**
   Timely basis independent audit of the security mechanism and status.
10. **Survivability Requirements**
    Ensuring applications survive attack, possibly resilient towards attack.
11. **Physical Protection Requirements**
    Ensuring that centers and their components and personnel are protected against destruction, damage, theft, or surreptitious replacement (e.g., due to vandalism,sabotage, or terrorism).
12. **System Maintenance Security Requirements**
    Ensuring system maintenance does not unintentionally disrupt the security mechanisms of the application,or component.

4 Security Testing in the Secure Software Development Life cycle

It is well known accepted fact that the cost of fixing bugs and security vulnerability increases as we move right in software development life cycle. In other words, cost of fixing issues will be higher in later stages. Hence, security testing techniques should be applied as early as possible in a secure software development life-cycle.A secure software development lifecycle takes security aspects into account in each phase of software development [13]. Bachmann, Ruediger and Brucker, Achim D[3] described how security testing can be performed during Secure SDLC:

During Planning and Design Phase

In this phase, using static approaches like security review of the architecture and threat modelling security testing methods are one of the most crucial methods which help in selecting tools and technique for testing in later stage :

- Architecture Security Reviews is the manual review of the product architecture which ensure fulfillment of the security requirement. Detecting architectural flaws at the early stage result in saving cost and effort as a benefit.
- Threat modelling is a structured manual analysis of an application specific business case or usage scenario. This analysis is guided by a set of precompiled security threats. With identification of threats, their impact and potential countermeasures specific to the development of the software product can be introduced. These methods helps in identifying the attack surface and the most critical components. This provide what to focus during security testing activities.

During Software development

In development stages, the following techniques are applicable:

- With the help of Static Source Code Analysis (SAST) and Manual Code Review of the application source code for finding vulnerabilities which help in detecting insecure programming, outdated libraries, and configurations which is one of the challenges discussed in earlier section.
- In Static Binary Code Analysis and Manual Binary Review, analysis of the compiled application (binary) for finding vulnerabilities without actually executing the application.

During Executable in test environment

Various techniques like penetration testing(Manual or Automated),Vulnerability scanners test, fuzz testing and many more which is explained in section 5 should be performed. However, these dynamic techniques usually achieve lower coverage than static approaches and mainly focused on detecting vulnerabilities related to data flows across the system and already known vulnerabilities.

During Maintenance and operation

Ensuring that software configurations is still secure and accidental violations related to authorization or authentication has not occured. In addition to that,

passive security testing techniques like intrusion detection system or monitoring system can be utilized to observe the behaviour of the software and, thus, highly recommended practice. Additionally, during this stage rigorous security testing of updates and patches are performed and ensuring that new vulnerabilities should not arise as a side affects.

4.1 Four Quadrants of Agile Testing

Recently, Agile testing approach has gained lot of attention and being adopted in the software Industry. Agile testing involves immediate and continuous testing of all changes and updating test cases to be able to run a regression test at any time to verify that changes have not broken existing functionality[12]. In agile software development, main focus is on the feature implementation and delivering of value to the customer. As a drawback, non-functional aspects like security of system often neglected because of time pressure, cost and awareness. Crispin and Gregory [11] discuss the Agile Testing quadrants which is widely accepted in practice. Each quadrants in figure 1 reflects different reasons to test.

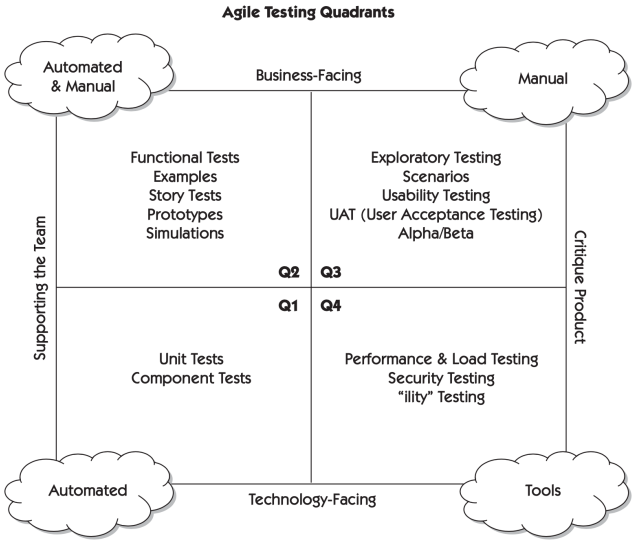

Fig. 1. Four Quadrants of Agile Testing(Adopted from [11])

Software companies somewhat focus on right hand side i.e Q3 and Q4, criticizing the product, but doesn't play enough in supporting left side i.e. Q1 and Q2. In agile testing, due to continuous interaction with developers and customers extend the contribution of the testers from only identifying the vulnerability to prevention. Due to time constraint in agile testing, Automation is an important

saviour for agile testing [12]. Introducing test automation in Q1 is usually easiest to implement, and has a big impact on the process effectiveness. Tests in Q3 are usually performed manually. In Q4, main scope is testing non-functional aspect and heavily dependent on testing tools which require specialized skill sets and expertise in product domain. But, exploratory testing by security expert is highly recommended as it increases the test coverage and can reveal vulnerability missed by automation tools. Authorization is often the only aspect of security testing that the agile teams consider as part of business functionality [12].
There are other known software security methodology is also available, one most widely known software security methodology is Microsoft's framework, which is integrated into the Microsoft Agile Security Development Lifecycle. Other approaches like Baca et al.[2] demonstrated how security features can be integrated into an agile software development method process at Ericsson AB. The approach focuses on risk management.
Chólis et al.[10] describe a case study of a software security testing process based on the Microsoft Software Development Life cycle for Agile. Major security development processes followed by the organisation are the Security Development Lifecycle (SDL) [24] from Microsoft and the Open Software Assurance Maturity Model (Open-SAMM)from OWASP [9].

5 Techniques of Security testing

Implementing security activities at throughout the software development life cycle requires different methodologies and techniques to successfully develop secure software in the end.
Security testing basically follows two types of approaches[22]:

1. Testing software to validate its functionality and mechanism checks.
2. Performing risk based approach according to attackers mindset.

Different types of testing techniques and risk assessment is described in this section and but, not limited to only these techniques.

5.1 Penetration testing

Penetration testing is a type of black- box security testing in which tester attempt to circumvent the security features of a system based on their understanding of the system design and implementation [22].Penetration testing is used to find security problems that are likely to originate in the software's architecture and design as it is this type of vulnerability that tends to be overlooked by other testing techniques[19]. Penetration testing mostly performed at the later stage of software testing life cycle when software product is developed which is one of the limitation as discussed before. There are many tools available like Nmap, Nessus etc to perform such activities. Selecting penetration testing tool depends

[9] `https://owasp.org/2020/02/11/SAMM-v2.html`

upon the type and feature of the software product. But using tools is not enough to get highly secure software, instead pen tester have to acquire mindset of the attackers and try to find out as much as possible negative test scenarios. The main difference between traditional software testing and security testing is software tester try to design positive test scenarios where they try to pass the test cases based on the functional requirement where software product is acting as required which is easier as those details can be acquired from product artifacts. On the other hand, designing negative test scenarios completely depends on the expertise of the tester on the product and creativity in designing negative test scenarios Tester with the help of his imagination try to abuse same functional requirement for unexpected purpose.
Penetration Testing Execution Standard (PTES) provided by OSWAP defines penetration testing as 7 phases [10].

- Pre-engagement Interactions
- Intelligence Gathering
- Threat Modeling
- Vulnerability Analysis
- Exploitation
- Post Exploitation
- Reporting

Types of penetration testing include black-box, white box and grey box. In black-box penetration testing, the testers are given no knowledge of the application. White-box penetration is the opposite of black-box in that complete information about the application may be given to the testers. Grey-box penetration testing, the most commonly used, is where the tester is given the same privileges as a normal user to simulate a malicious insider. Penetration testing should focus on those aspects of system behavior, interaction, and vulnerability that cannot be observed through other tests [19]. Penetration testers should subject the system to sophisticated multi-pattern attacks designed to trigger complex series of behaviors across system components, including non-contiguous components, these are the types of behaviors that cannot be forced and observed by any other testing technique.

5.2 Model based security testing

The basic idea of model-based testing (MBT) is that instead of creating test cases manually, selected algorithms are generating them automatically from a (set of) model(s) of the system under test or of its environment[36]. Model based testing replaces manual test designs by automated test generation unlike test automation. Mark R. Blackburn, Robert Busser, and Aaron Nauman [5] presented paper on how model based security testing is different from other three generation of the test automation considering model based test automation as fourth generation. In other paper Mark Blackburn and Ramaswamy Chandramouli [6],

[10] `http://www.penteststandard.org/index.php/Main_Page`

they proposed an model-based approach to automate security functional testing that involves developing models of security function specifications (SFS) as the basis for automatic test vector and test driver generation.
Jan Jürjens [27] presented work towards systematic specification-based testing of security-critical systems based on UMLsec models. He also showed how to systematically generate test sequences fom security properties based on the model that can be used to test the implementation for vulnerabilities.
Wimmel, Guido and Jürjens, Jan [40] proposed a specification based testing for security critical systems, a specification based testing is where test sequences generated from an abstract system specification in order to gain confidence in the correctness of the implementation. And, they presented an approach where test sequences for transaction systems from a formal security model supported by the CASE tool Auto-Focus and those test sequences are determined with respect to the system's required security properties, using mutations of the system specification and attack scenarios.
Tejeddine, MouelhiFranck, FleureyBenoit, BaudryYves, and Le Traon [33], proposed a model-driven approach for specifying, deploying and testing security policies in Java applications. First, a security policy is specified independently of the underlying access control language (OrBAC, RBAC) based on a generic security meta-model which can be used for early consistency checks in the security policy. This model is then automatically transformed into security policy for the XACML platform and integrated in the application using aspect-oriented programming. To qualify test cases that validate the security policy in the application and they inject faults into the policy. The fault model and the fault injection process are defined at the meta-model level, making the qualification process language-independent.
Furthermore, relevant publications in the field of MBST are [27, 5, 6, 31, 35, 32, 40].

5.3 Risk-Based Security Testing

According to I. Schieferdecker, J. Grossmann, and M. Schneider [36], Risk-based testing introduces two different goals in mind.One, optimizing overall test process by risk analysis that provide guidance towards test identification and requirement engineering that result in systematic information with focus on threats and vulnerabilities. Second, simulation of attack to find deviations of the software under test that leads to vulnerabilities for example performing a denial-of-service attack.
Additionally, risk assessment introduces notion of risk values, which estimates the likelyhood and consequences of certain threat scenarios. These risk values enable tester to weigh threat scenarios as well as prioritize them and thus, helps in focusing on vulnerabilities that are more relevant. Risk analysis and risk assessment, similar to other development activities is performed in early project phases, are mainly based on assumptions on the system itself. But, testing with the system enables to gain the empirical evidence on the presence of vulnerabil-

ities, applicability, quality.
In particular, risk-based testing can help in [36]

- providing evidence on the functional correctness of countermeasures,
- providing evidence on the absence of known vulnerabilities, and
- discovering unknown vulnerabilities,
- optimizing risk analysis by identifying new risk factors and reassessing the risk values.

Risk based approaches can be utilized in another way like risk based test selection and risk control. Risk-based test selection is used to find an optimal set of test cases along certain selection strategy. Risk control deals with the revision of risk assessment results by correcting assumptions on probabilities, consequences or the maturity of treatments scenarios or deals with the completion of risk analysis result by integrating vulnerabilities, potentially threats, threat scenarios and unwanted incidents.
Zech, Philipp [42], proposed model-driven methodology for the security testing of cloud environments, ingesting misuse cases, defined by negative requirements derived from risk analysis.
Ming Ni [34] paper focussed on speed enhancements techniques that offer clear and meaningful ways to enhance human understanding and comprehension of security levels.
Julien Botella [7] presented approach based on a mixed modeling, the model used for automatically generating test cases by capturing some behavioral aspects of the Web applications, which also includes vulnerability test purposes as feature in order to drive the test generation process.
Yan Li [29], proposed a conceptual framework which is based on the two possible combinations of model-based security testing (MST) and model-based security risk analysis (MSR) called risk-driven model-based security testing (RMST) and test-driven model-based security risk analysis (TMSR).
Ida Hogganvik [21],worked on developing a graphical approach to threat and risk modeling that supports the security analysis process and approach is contributing in solving three issues related to security analysis:

1. how to facilitate communication in a group consisting of people with different backgrounds and competences?
2. how to estimate the likelihood and consequences of the risks?
3. how to document the security analysis in a comprehensible manner?

His approach incorporated the CORAS security risk modeling language. It is a graph based modelling approach that emphasizes the modelling of threat scenarios and provides formalism to annotate the threat scenarios with probability values and formalism to reason with these annotation.

5.4 Fuzz testing

Because of its effectiveness in discovering vulnerability, it has gain a lot of attention. In Fuzz testing, injecting random data into program to test whether

it can run normally under unexpected inputs. Fuzzy testing would find flaws of tested software, which are difficult for the other logical testing method[36]. Fuzzy testing is illogical, just creates clutter data and require very less computational cost and time. Fuzz testing is mostly implemented by a program of script that submits a combination of inputs to the software to disclose how software performs. Fuzzing might be characterized as a blind fishing mission that hopes to uncover completely unsuspected problems in the software. For example, suppose the tester intercepts the data that an application reads from a file and replaces that data with random bytes. If the application crashes as a result, it may indicate that the application does not perform demanded checks on the data from that file but instead assumes that the file is in the right format. As fuzzing is essentially functional testing, it can be conducted in various steps during the overall development and testing process[19].
Patrice Godefroid [20] proposed a method to records an actual run of the program under test on a well-formed input, symbolically evaluates the recorded trace, and gathers constraints on inputs capturing how the program uses these.The collected constraints are then negated one by one and solved with a constraint solver, producing new inputs that exercise different control paths in the program. They have implemented this algorithm in SAGE (Scalable, Automated, Guided Execution), a tool employing x86 instruction-level tracing and emulation for white-box fuzzing of arbitrary file-reading Windows applications. As a result, they have revealed MS07-017 ANI vulnerability, which was missed by extensive black-box fuzzing and static analysis tools.
Petar Tsankov, Mohammad Torabi Dashti, and David Basin [39] proposed a light weight, yet effective, technique for fuzz testing security protocols. They used a concrete implementation of the protocol to generate valid inputs and mutate the inputs using a set of fuzz operators. A dynamic memory analysis tool monitors the execution as an oracle to detect the vulnerabilities exposed by fuzz-testing. For encrypted messages, they provide the fuzzer with the necessary keys and cryptography algorithms in order to properly mutate encrypted messages.
The great advantage of fuzz testing is that the test design is extremely simple, and free of preconceptions about system behaviour [11] and Fuzzers work best for discovering vulnerabilities that can be exploited by SQL injection, buffer overflow, denial of service (DOS), and cross-site scripting. On the other hand, fuzzer has limitations it tries to find simple bugs and when we do black-box testing, which increases difficulty to evaluate the threatening/impact of the found vulnerability (no debugging possibilities).

5.5 Code-Based Testing and Static Analysis

In Static analysis, we analyze the code of the application without actually executing it. It is powerful tool which allows to detect vulnerabilities in source code. According to security community, there is no alternate for actually looking at the code for detecting vulnerabilities and many serious security weaknesses cannot

[11] `http://en.wikipedia.org/wiki/Fuzz_testing`

be detected with any other procedure of analysis or testing. The issues related to concurrency problems, time bombs, logic bombs, flawed business logic, access control problems, and cryptography weaknesses as well as back doors, Trojans and other forms of malicious code can be exposed by source code reviews. Code-based and Binary code based analysis share similarity, hence not discussing binary code analysis. Code reviews can be performed manually or automated and is often called static code analysis (SCA) or Static Application Security Testing (SAST). The main principle of static analysis are deducing, data flow analysis and constraint analysis [41].

Ben Breech and Lori Pollock suggested a dynamic compiler based security testing framework [8], which could insert attack code into the running program and test security mechanism of software. It is mainly used to test program-based attacks, such as stack buffer overflow.

V. Benjamin Livshits and Monica S. Lam [30] proposed a static analysis technique for detecting many recently discovered application vulnerabilities such as SQL injections, cross-site scripting, and HTTP splitting attacks.In their Java based system, user-provided specifications of vulnerabilities are automatically translated into static analyzers and finds all vulnerabilities matching a specification in the statically analyzed code.

In Seung-Hyun Seoa and Aditi,Gupta [37] work, they have proposed a static analyzer tool called DroidAnalyzer which identifies potential vulnerabilities of Android apps and the presence of root exploit .Using this tool, they analyzed various mobile malware samples and targeting apps such as banking, flight tracking and booking, home and office monitoring apps to examine potential vulnerabilities.

Basic lexical analysis is the approach taken by early static analysis tools, including ITS4, FlawFinder [12] . and RATS [13], all of which preprocess and tokenize source files (the same first steps a compiler would take) and then match the resulting token stream against a library of vulnerable constructs.

The advantages of code review are completeness, catching implementation bugs early, effectiveness, and Accuracy. Disadvantages are not suitable for large code bases, requires highly skilled reviewers, labor intensive, and infeasible to detect run-time errors[19].

5.6 Vulnerability Scanning

Application vulnerability scanners are a very important software security testing technique to find software security risks, includes testing space scanning and known defects scanning[19]. Testing space scanning deals with network port, string, procedure data, network data and other elements scanning, for example, network port scanning can reveal issue related to vulnerable ports.. Vulnerability scanning tools scan application inputs and outputs to look for known vulnerability signature in application level software.

[12] `www.dwheeler.com/flawfinder/`

[13] `www.securesoftware.com`

Common scanning tools include: Security Profile Inspector (SPI), Internet Security Scanner (ISS), Security Analysis Tool for Auditing Networks (SATAN), Tiger, Sscan, Nmap, COPS, and Tripwire.
Jose Fonseca, Marco Vieir, and Henrique Madeira [16] also proposed a method where they inject common types of software fault and then evaluate and benchmark automatic web vulnerability scanners and the results shows vulnerability scanner in general has low coverage and the percentage of false positives is very high. So, vulnerability scanner cannot be blindly trusted.
Jeffrey W. Humphries, Curtis A. Carver and Jr., Member [25] proposed customized vulnerability scanning system using secure mobile agent which is resistant to attack and also has the ability to quickly look for newly published vulnerabilities. A mobile agent is simply a program that represents a user in a computer network.
Now, websites are moving towards HTML5 but most of web application based scanner cannot detect security vulnerabilities related to HTML5 hence HTML5 based issues become blind spots for scanner tool. Qianqian and Liu Xiangjun [4] customized W3af(Web Application Attack and Audit Framework) and designed a web application security scanner. This web application security scanner can detect Clickjacking vulnerabilities brought by HTML5, but also provide efficient Web application security scanning and evaluation services for the websites.
Jun Gao, Li Li, and Pingfan Kong [18] researched on evolution of app vulnerabilities, they first developed a lineage of android app which represent sequence of historical release of that app and based on those lineage they observed the evolution vulnerabilities scanned by well known scanner.

5.7 Property-based testing

Many errors in software are caused by generalized flaws in the source code. Property-based testing assures that a given program is free of specified generic flaws. Property-based testing leverages property specifications and a data-flow analysis of the program to guide evaluation of test executions for correctness and completeness. Paper [14] describes a method that transforms security property of software into specification described by TASPEC language. It would extract the code in relation to specific property by program slicing technology, and discover violation of the code against security property specification. Property-based testing focuses on some specific security properties, which can meet requirement of classification and priority.

5.8 Semi-Automatic Security Testing

Recently, model-checkers dedicated to security analysis have proved their ability to identify complex attacks on web-based security protocols. Model checkers are formal verification tools, capable of providing counterexamples to violated properties. Normally, these counterexamples are meant to guide an analyst when searching for the root cause of a property violation[17].

Matthias Büchler, Johan Oudinet and Alexander Pretschner proposed a creative approach of utilizing model checkers and penetration testing for security testing to bridge the gap between both. In proposed idea, first they mutate the model to introduce specific vulnerabilities present in web applications, this model check outputs attack traces that exploit those vulnerabilities. Next, that attack traces converted into test cases by using 2- step mapping. Finally, that tests are performed on real software automatically. This prototype implemented and evaluated on Web Goat provided by OWSAP. As result, that prototype was successful in reproducing Role-Based Access Control (RBAC) and Cross-Site Scripting (XSS) attacks.

5.9 Tools for Security testing

There are so many tools for Manual/Automation security testing techniques discussed in this paper shown in table 1. Based on project requirement, technologies offered, support, attack surface and cost tools can be selected.

Table 1. Security testing tools for different techniques

Technique	Tools
Penetration testing	Netsparker ; Acunetix ; Indusface ; ImmuniWeb ; Owasp ; WireShark ; w3af ; Metaspoilt ; kali ; Aircrack ; ZAP ; Sqlmap ; Sqlninja ; BeEF ; Ettercap ; IBM Security AppScan ; Websecurify ; Wapiti ; Kismet ;OpenSSL; snort ; THC Hydra ; John the Ripper ; CloudFlare ; Nmap ; Fiddler ; OSINT ; Ratproxy
Vulnerability scanning	Abbey Scan ; Acunetix WVS ; AppScan on Cloud ; AppScan ; App Scanner ; AppSpider ; AppTrana Website Security Scan ; Arachni ; AVDS ; BlueClosure BC Detect ; Burp Suite ; edgescan ; Grabber ; Gravityscan ; Intruder ; Nessus ; Retina ; Wapiti ; Websecurify Suite ; Security Profile Inspector (SPI) ; Internet Security Scanner (ISS); Security Analysis Tool for Auditing Networks (SATAN) ; Tiger ; Sscan ; Nmap ; COPS ; Tripwire
Fuzz testing	JBroFuzz ; WSFuzzer ; american fuzzy lop ; Radamsa - a flock of fuzzers ; Microsoft SDL MiniFuzz File Fuzzer ; Microsoft SDL Regex Fuzzer ; ABNF Fuzzer ; Codenomicon's product suite ; Spirent Avalanche NEXT ; Beyond Security's beSTORM product ; Sulley Fuzzing Framework
Code and Static analysis	Bandit ; Brakeman ; Codesake Dawn ; Deep Dive ; FindBugs ; FindSecBugs ; Flawfinder ; GolangCI-Lint ; Google CodeSearchDiggity ; Graudit ; HCL AppScan CodeSweep ; LGTM ; .NET Security Guard ; phpcs-security-audit ; PMD ; PreFast ; Progpilot ; Puma Scan ; ShiftLeft Scan ; SonarQube ;VisualCodeGrepper (VCG) ; SourceGuard
Risk Based security testing	OWASP Risk Assessment Framework ; RAF SAST Tool
Model checker	BLAST ; CADP ; QComp ; PRISM-TUMheuristics ; DFTRES ; Storm ; Java Pathfinder ; SPIN ; TAPAs ; ROMEO ; TLA+ Model Checker (TLC) ; NuSMV ; mCRL2 ; MRMC

6 Selection Criteria for Security Testing Approach

For selecting suitable security testing approach as well as tool for a particular software application many aspect need to consider. Felderer, Michael and

Büchler [13] has briefly explained what one should consider as selection criteria for security testing approach and tools:

- **Attack surface:** As discussed in security testing techniques, different security testing methods helps in revealing different vulnerabilities. And, some testing approaches are complementary to each other. So, it is recommended to use various testing techniques in order increase test coverage and in order to find different types of security flaws in software application.
- **Application type:** This is an obvious aspect that should always be considered as different security testing approach behave differently depends on application type.
- **Quality of results and usability:** Before selecting any security testing approach, team should research about the effectiveness like false positive rate and usability aspect with respect to the project. Then, choose the best available approach for their application.
- **Supported technologies:** One should also consider the technologies used in their project (programming languages, interfaces etc.) are compatible with the approach.
- **Performance and resource utilization:** Making decision on different one should consider the available computing power and manpower in the project as different tools and methods require different computing power and different manual efforts.
- **Costs for licenses, maintenance and support:** Tools in market are either available free or cost money. Team should consider the support regarding the tool in case or other specific features like bug tracking is provided by the tools team.

6.1 Mindset for Security testing

Most of the developers and tester believes that implementing/testing security features is complete security testing.On the other hand, attacker's think differently and always looking for loopholes, mainly introduced because of developers incorrect belief or negligence.And, that mistake sometimes lead to zero day vulnerabilities. If mindset not changed then implementing any methodologies or technologies will not be effective. So, software industries should motivate security related mindset in project and also work towards providing training related to that.
Hooshangi, Sara and Weiss, Richard and Cappos, and Justin [23] research on how security mindset makes better tester, revealed that students who wrote good defense monitors also wrote good attack tests. Furthermore, student's knowledge of security can influence the quality of the programs and systems they develop [9].
Books like How to Break Software Security and Exploiting Software help educate testing professionals on how to think like attackers. Some key aspect required for security testing are :

1. Extensive knowledge of the domain.

2. Imagination helps in getting the idea of how certain thinks can work and in effective test case generation.
3. A hacker or attacker mindset results in developing impacting test cases and cover all the creative ways an unauthorized attacker could exploit the application.

7 Conclusion

This paper discusses about the definition, scope, challenges, requirements and techniques of software security testing. Main focus of this paper is security testing techniques used in different phases of software development life cycle with capability and limitation. As techniques, brief overview of penetration testing, vulnerability scanning, risk based analysis, property based and static analysis is collated with the different researches published on these techniques.
Introducing security practices from the initial stage of the project, designing security test cases, selecting effective tools and techniques, and most importantly acquiring security mindset can motivate to build secured application which can increase the reliability and confidence on the product.

References

[1] Brad Arkin, Scott Stender, and Gary McGraw. "Software penetration testing". In: *IEEE Security & Privacy* 3.1 (2005), pp. 84–87.
[2] Dejan Baca et al. "A novel security-enhanced agile software development process applied in an industrial setting". In: *2015 10th International Conference on Availability, Reliability and Security*. IEEE. 2015, pp. 11–19.
[3] Ruediger Bachmann and Achim D Brucker. "Developing secure software: A holistic approach to security testing". In: *Datenschutz und Datensicherheit (DuD)* 38 (2014), pp. 257–261.
[4] Jason Bau et al. "State of the art: Automated black-box web application vulnerability testing". In: *2010 IEEE Symposium on Security and Privacy*. IEEE. 2010, pp. 332–345.
[5] Mark Blackburn, Robert Busser, and Aaron Nauman. "Why model-based test automation is different and what you should know to get started". In: *International conference on practical software quality and testing*. 2004, pp. 212–232.
[6] Mark Blackburn et al. "Model-based approach to security test automation". In: *Proceeding of Quality Week 2001*. 2001.
[7] Julien Botella et al. "Risk-based vulnerability testing using security test patterns". In: *International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*. Springer. 2014, pp. 337–352.
[8] Ben Breech and Lori Pollock. "A framework for testing security mechanisms for program-based attacks". In: *ACM SIGSOFT Software Engineering Notes* 30.4 (2005), pp. 1–7.

[9] Ingrid A Buckley, Janusz Zalewski, and Peter J Clarke. "Introducing a Cybersecurity Mindset into Software Engineering Undergraduate Courses". In: *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS* 9.6 (2018), pp. 448–452.

[10] Jesús Chóliz, Julián Vilas, and José Moreira. "Independent security testing on agile software development: a case study in a software company". In: *2015 10th International Conference on Availability, Reliability and Security*. IEEE. 2015, pp. 522–531.

[11] Lisa Crispin and Janet Gregory. *Agile testing: A practical guide for testers and agile teams*. Pearson Education, 2009.

[12] Daniela Soares Cruzes et al. "How is security testing done in agile teams? a cross-case analysis of four software teams". In: *International Conference on Agile Software Development*. Springer, Cham. 2017, pp. 201–216.

[13] Michael Felderer et al. "Security testing: A survey". In: *Advances in Computers*. Vol. 101. Elsevier, 2016, pp. 1–51.

[14] George Fink and Matt Bishop. "Property-based testing: a new approach to testing for assurance". In: *ACM SIGSOFT Software Engineering Notes* 22.4 (1997), pp. 74–80.

[15] Donald Firesmith et al. "Engineering security requirements." In: *Journal of object technology* 2.1 (2003), pp. 53–68.

[16] Jose Fonseca, Marco Vieira, and Henrique Madeira. "Testing and comparing web vulnerability scanning tools for SQL injection and XSS attacks". In: *13th Pacific Rim international symposium on dependable computing (PRDC 2007)*. IEEE. 2007, pp. 365–372.

[17] Gordon Fraser, Franz Wotawa, and Paul E Ammann. "Testing with model checkers: a survey". In: *Software Testing, Verification and Reliability* 19.3 (2009), pp. 215–261.

[18] Jun Gao et al. "Poster: On Vulnerability Evolution in Android Apps". In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. IEEE. 2018, pp. 276–277.

[19] ASAM Al-Ghamdi. "A survey on software security testing techniques". In: *Int J Comput Sci Telecommun* 4 (2013), pp. 14–18.

[20] Patrice Godefroid, Michael Y Levin, David A Molnar, et al. "Automated Whitebox Fuzz Testing." In: *NDSS*. Vol. 8. 2008, pp. 151–166.

[21] Ida Hogganvik. "A graphical approach to security risk analysis". In: (2007).

[22] Itti Hooda and Rajender Singh Chhillar. "Software test process, testing types and techniques". In: *International Journal of Computer Applications* 111.13 (2015).

[23] Sara Hooshangi, Richard Weiss, and Justin Cappos. "Can the security mindset make students better testers?" In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*. 2015, pp. 404–409.

[24] Michael Howard and Steve Lipner. *The security development lifecycle*. Vol. 8. Microsoft Press Redmond, 2006.

[25] Jeffrey W Humphries, Curtis A Carver Jr, and Udo W Pooch. "Secure mobile agents for network vulnerability scanning". In: *Proceedings of the*

2000 IEEE Workshop on Information Assurance and Security. 2000, pp. 6–7.

[26] Arunima Jaiswal, Gaurav Raj, and Dheerendra Singh. "Security Testing of Web Applications: Issues and Challenges". In: *International journal of computer applications* 88.3 (2014).

[27] Jan Jürjens. "Model-based security testing using umlsec: A case study". In: *Electronic Notes in Theoretical Computer Science* 220.1 (2008), pp. 93–104.

[28] B Kirubakaran and V Karthikeyani. "Mobile application testing—Challenges and solution approach through automation". In: *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*. IEEE. 2013, pp. 79–84.

[29] Yan Li. "Conceptual framework for security testing, security risk analysis and their combinations". In: *9th Workshop on Systems Testing and Validation (STV'12)*. 2012, pp. 17–21.

[30] V Benjamin Livshits and Monica S Lam. "Finding Security Vulnerabilities in Java Applications with Static Analysis." In: *USENIX Security Symposium*. Vol. 14. 2005, pp. 18–18.

[31] Malte Lochau et al. "Model-Based Testing". In: *Formal Methods for Executable Software Models - 14^{th} International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM*. Written at TU Darmstadt. Springer, 2014, pp. 310–342. DOI: 10.1007/978-3-319-07317-0_8.

[32] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. *Model-driven risk analysis: the CORAS approach*. Springer Science & Business Media, 2010.

[33] Tejeddine Mouelhi et al. "A model-based framework for security policy specification, deployment and testing". In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2008, pp. 537–552.

[34] Ming Ni et al. "Software implementation of online risk-based security assessment". In: *IEEE transactions on power systems* 18.3 (2003), pp. 1165–1172.

[35] Sven Peldszus. "Model-driven Development of Evolving Secure Software Systems". In: *Proceedings of the 7th Collaborative Workshop on Evolution and Maintenance of Long-Living Software Systems*. To Appear. Feb. 2020.

[36] Ina Schieferdecker, Juergen Grossmann, and Martin Schneider. "Model-based security testing". In: *arXiv preprint arXiv:1202.6118* (2012).

[37] Seung-Hyun Seo et al. "Detecting mobile malware threats to homeland security through static analysis". In: *Journal of Network and Computer Applications* 38 (2014), pp. 43–53.

[38] Gu Tian-yang, Shi Yin-Sheng, and Fang You-yuan. "Research on software security testing". In: *World Academy of science, engineering and Technology* 70 (2010), pp. 647–651.

[39] Petar Tsankov, Mohammad Torabi Dashti, and David Basin. "SECFUZZ: Fuzz-testing security protocols". In: *2012 7th International Workshop on Automation of Software Test (AST)*. IEEE. 2012, pp. 1–7.

[40] Guido Wimmel and Jan Jürjens. "Specification-based test generation for security-critical systems using mutations". In: *International Conference on Formal Engineering Methods*. Springer. 2002, pp. 471–482.

[41] Xia Yiming. "Security Vulnerability Detection Study Based on Static Analysis". In: *Computer Science* 33.10 (2006), pp. 279–283.

[42] Philipp Zech. "Risk-based security testing in cloud computing environments". In: *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE. 2011, pp. 411–414.