Software Security Testing

Bhupender Kumar Saini(219100887)

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

Abstract. A lot of research has been done recently towards creating effective approaches to increase the security testing scope and effectiveness. Software security testing is a key aspect to ensure reliability, confidence, and trust in software applications. Secure software can contribute to the software quality and plays a vital role in reducing the risk of intentional/unintentional cyber attacks or failures that can severely affect the reputation of a software company and can lead to negative consequences. This paper discusses the importance of security testing, challenges in implementing testing techniques, and unique testing approaches in the software development life cycle. With the day by day advancement of technology, the frequency of cyber attacks are increasing and the nature of cyber attacks are becoming more complex. This paper presents the widely used testing techniques and also respective tools. Finally, a brief explanation about the important prerequisite for designing effective security test sequences and increasing testing scope in secure software testing: mindset.

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

1 Motivation

With the evolution of technology, there is an increase in cyber-attacks which results in data loss, threat to privacy, human life, etc. 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 a high-risk app are installed, more attacks groups are forming and on average 55 organization 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 data due to an internal or external security threat. Enterprises lose half a million

[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

US dollars on average due to security breaches. However, quantifying only based on monetary loss is not enough. Additionally, the top three most expensive types of security breaches are third party failure, fraud by employees, and cyber espionage. The top three IT security threats that lead to data loss are malware, phishing attacks, and accidental leaks by the staff. The consequences of security breaches are loss of access to business-critical information, damage to company reputation, and temporary loss of ability. One should keep in mind that the cost of a security breach is always comparably higher than the cost of protection.
As per Edgescan 2019 vulnerability report [3], 19% of all vulnerabilities were associated with (Layer 7) web applications, API's, etc. and 81% were network vulnerabilities. Security implementation has a much broader scope that has to be introduced in every part of the process that could be monitoring access of employees in the organization, emphasizing security awareness, or implementing practices designed by security experts in the organization. In this paper, our focus is only on one aspect: Software security. Software security defined as the process to identify the security features of software implementation are consistent with the design.
Most of the security breaches like exploiting known vulnerabilities and malware are preventable, if the process follows security guidelines, plan accordingly, and uses security techniques carefully. Introducing security activities into the SDLC, DevSecOps, patch management, continuous vulnerability management, and continuous asset profiling, can help in identifying weaknesses at an earlier phase. 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

Nowadays, computers and software are significantly expanding and creating dependencies in every aspect of life, which also enhances the possibility of unavoidable vulnerabilities which on later stage can be exploited by attackers as well as increasing the risk of occurring on production. Therefore, introducing an efficient way of security testing is one of the major concern for organization.
There are many definitions for the Software Security,

"Software Security is the software's ability to highly resist, tolerate, and recover from cases that strongly threaten the product." - Julia Allen [4]

"The primary purpose of engineering software that continues to function correctly and efficiently under several types of malicious attacks." - G. McGraw [34]

"Software security is simply about the process of building secure software by

[3] https://www.edgescan.com/wp-content/uploads/2019/02/edgescan-Vulnerability-Stats-Report-2019.pdf
[4] https://www.tandfonline.com/doi/full/10.1080/07366980701500734

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." -G. McGraw [34]

According to Tian-Yang [40], 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* that should not be misunderstood as an *absence of vulnerabilities* [14]. Anyway, it increases confidence in the product and reduce risk from attacks or unexpected behavior which can cost money and reputation if discovered at a later stage.
A very important and different perspective was presented by Arkin, Brad[1], there is no relation between software security defects and vulnerabilities to security functionality — rather, they originate from an attacker's unexpected but intentional misuse 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 negative test scenarios possibly driven by abuse cases and architectural risks to simulate the behavior of a system under attack. This arises the limitation, the scope of generative negative test scenarios solely depends on the security tester imagination, expertise, and knowledge. Insufficient planning and negligence in security testing can lead to unexpected consequences:

- Software product with vulnerabilities, like poor encryption, unaddressed or unprotected bugs 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.
- You lose customers trust hence losing customers and profits.
- Software quality, reliability and security are tightly coupled[21].

2.1 Software Security Testing Terminologies

To completely understand the intention of this paper, basic background information on security testing is required and definition of terms used in this paper are as follows:
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[16].

Bug is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways[5].
Vulnerability is a fault related to security properties. A vulnerability either means that the security requirement is completely missing or implemented in the wrong way[16].
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[6].
Threat is the possible cause of an undesirable incident that could harm or reduce the value of an asset. It can be hacker, malicious insider, compromised employee [16].
Exploit is a specific actions/software/inputs that uses the vulnerability in the system and causes the system to behave unintended[7].
Confidentiality is the assurance that information is not disclosed to unauthorized individuals, processes, or devices[15].
Availability guarantees timely, reliable access to data and information services for authorized users [15].
Integrity is provided when data is unchanged from its source and has not been accidentally or maliciously modified, altered, or destroyed[15].
Non-repudiation is the assurance that none of the partners taking part in a transaction can later deny having participated[15].
Authentication is a security measure designed to establish the validity of a transmission, message, or originator, or a means of verifying an individual's authorization to receive specific categories of information[15].
Authorization provides access privileges granted to a user, program, or process[15].

3 Scope And Challenges

To successfully develop secure software, responsible individuals should verify and validate the security requirements as well as should gather information about the 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[44]. Hence, in the event of security testing, it is important that the negative requirements and cases derived from risk analysis are incorporated. Additionally, one should follow an 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. The

[5] https://steelkiwi.com/blog/is-there-such-a-thing-as-bug-free-software/
[6] https://www.trendmicro.com/vinfo/us/security/definition/zero-day-vulnerability
[7] https://en.wikipedia.org/wiki/Exploit_(computer_security)

Common Weakness Enumeration (CWE) [8] 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 [9].Considering these details during security testing is highly recommended.

3.1 Challenges in Security Testing

American security today website [10], provided brief overview of challenges faced during Security testing are :

- **Speed of Agile Development:**
  The dynamic and fast-paced nature of agile development encourages the team to neglect issues to achieve project goals or meet the deadlines. Hence, there is a high chance that security testing guidelines will be bypassed or partially ignored.
- **Risks of Using Open Source Components:**
  Using open-source components with no/little cognizance about internal insight of the components may lead to vulnerabilities, unwanted complexity, and inconsistencies in the overall product. Avoid the 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:**
  Securely developed applications still can be at risk due to be vulnerabilities and weaknesses in programming languages. Every programming language is prone to its own vulnerabilities and limitation which can be utilized in attacking the application. For example, C programming language most common vulnerabilities are buffer overflow error, format string vulnerability, integer errors[11]. Increasing awareness regarding known issues of programming languages can significantly minimize the security risks. In addition, languages used in developing web or desktop application can create challenges in applying security policies[28].
- **Lack of AppSec Planning:**
  Appsec defined as the process of securing all the software a business uses. Lack of proper planning can lead to unmanageable security issues and unclear expectations of the 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 [31], 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.

[8] https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html
[9] https://www.sans.org/top25-software-errors
[10] https://americansecuritytoday.com/secure-software-development-challenges-and-considerations/
[11] https://medium.com/hackernoon/top-5-vulnerable-programming-languages-eab3144d6db7

3.2 Requirements for Security Testing

In a paper presented by Firesmith, Donald [18] about security requirement has explicated 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 application users 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 authorized 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 user, component, or application should not deny having participated in interactions afterward.
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 the company and its assets are protected against terrorism attacks, damage, theft, sabotage, etc.
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 the software development life cycle [37]. In other words, the cost of fixing issues will be higher in the later stages. Hence, security testing techniques should be applied as early as possible in a secure

software development life-cycle. A secure software development life cycle takes security aspects into account in each phase of software development [16]. Bachmann et. al.[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 modeling security testing methods are one of the most crucial methods which help in selecting tools and techniques for testing in later stage :
  - Architecture Security Reviews is the manual review of the product architecture which ensures fulfillment of the security requirement. Detecting architectural flaws at the early stage result in saving cost and effort as a benefit.
  - Threat modeling is a structured manual analysis of an application-specific business case or user scenarios[3]. This analysis is guided by a set of precompiled security threats. With the identification of threats, their impact and potential countermeasures specific to the development of the software product can be introduced. These methods help in identifying the attack surface and the most critical components. This provides what to focus on during security testing activities.
- **During Software development**
  In the 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 that help in detecting insecure programming, outdated libraries, and configurations which is one of the challenges discussed in the 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 are still secure and accidental violations related to authorization or authentication has not occurred. In addition to that, passive security testing techniques like intrusion detection system or monitoring system can be utilized to observe the behavior of the software and, thus, highly recommended practice. Additionally, during this stage, rigorous security testing of updates and patches is 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 a 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 run a regression test to verify that changes have not broken existing functionality[13]. In agile software development, the primary focus is on feature implementation and delivering 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 [12] discuss the Agile Testing quadrants, which are widely acknowledged in practice. Each quadrants in figure 1 reflects different reasons to test.

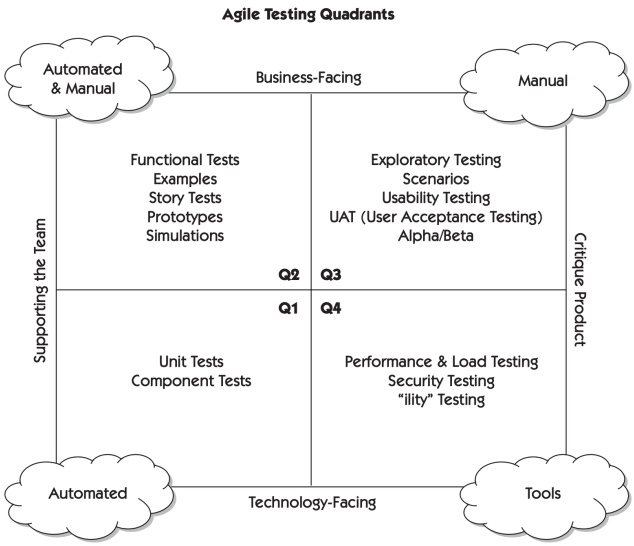

Fig. 1. Four Quadrants of Agile Testing(Adopted from [12])

Software companies somewhat focus on the right-hand side i.e Q3 and Q4, and doesn't play enough in supporting the left side i.e. Q1 and Q2 [13]. 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 constraints in agile testing, Automation is an important savior for agile testing [13]. 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, the main scope is testing non-functional aspect and heavily dependent on testing tools which require specialized skill sets and expertise in the product domain. But, exploratory testing by security experts 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 [13].
There are other known software security methodology 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.[11] 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 organization are the Security Development Lifecycle (SDL) [26] from Microsoft and the Open Software Assurance Maturity Model (Open-SAMM)from OWASP [12].

5 Techniques of Security testing

Implementing security activities 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[24]:

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

In penetration testing, tester attempts to circumvent the security features of a system based on their understanding of the system design and implementation [24]. This technique is used to find security problems that are likely to originate in the software's architecture and design as this type of vulnerability overlooked by other testing techniques[21]. Penetration testing mostly performed at the later stage of the software testing life cycle once the software product is developed. There are automated tools like Nmap, Nessus, Metasploit, OpenVAs, and Manual tools like OWASP Zed Attack Proxy (ZAP), w3af audit framework, Wireshark available to perform such activities. Selecting a penetration testing tool depends upon the type and feature of the software product. But using tools is not enough to get secure software, instead, pen tester has to acquire the mindset of the attackers and try to find out negative test scenarios as much as possible. The main difference between traditional software testing and security testing is software tester try to design positive test scenarios based on the functional requirement specification which can be accessed easily from product artifacts. On the other hand, designing negative test scenarios completely depends on the tester expertise on the product domain and creativity. Tester with the help of

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

his imagination try to abuse the same functional requirement for unexpected purpose. Penetration Testing Execution Standard (PTES) provided by OSWAP defines penetration testing as 7 phases [13].

1. Pre-engagement Interactions
2. Intelligence Gathering
3. Threat Modeling
4. Vulnerability Analysis
5. Exploitation
6. Post Exploitation
7. 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. Unlike black-box, in white-box penetration complete information about the application is available 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 aspects of system behavior, interaction, and vulnerability that cannot be observed through other tests [21]. 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 security testing (MBST) 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[38]. Model-based testing replaces manual test designs by automated test generation, unlike test automation.
Mark R. Blackburn et al. [5] presented a paper on how model-based security testing is different from the other three generations of the test automation considering model-based test automation as the fourth generation. In other paper, Mark Blackburn and Ramaswamy Chandramouli [6] proposed a 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 [29] presented work towards systematic specification-based testing of security-critical systems based on UMLsec. UMLsec is an extension to the Unified Modelling Language for integrating security-related information in UML specifications [30]. He also showed how to systematically generate test sequences from security properties based on the model that can be used to test the implementation for vulnerabilities.

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

Wimmel and Jürjens[42] proposed a specification-based testing for security-critical systems, a specification based testing is where test sequences generated from an abstract system specification 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 for the system's required security properties, using mutations of the system specification and attack scenarios.
Tejeddine et al. [35], 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 that can be used for early consistency checks in the security policy. This model is then automatically transformed into a security policy for the XACML platform and integrated into 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.
In another case, Matthias Büchler et al. [9] proposed a different approach of utilizing model checkers and penetration testing for security testing to bridge the gap between both. In the 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 a result, that prototype was successful in reproducing Role-Based Access Control (RBAC) and Cross-Site Scripting (XSS) attacks.

5.3 Risk-Based Security Testing

According to I. Schieferdecker et al. [38], Risk-based testing introduces two different goals in mind. One, optimizing the overall test process by risk analysis that provides guidance towards test identification and requirement engineering that results in systematic information with a 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 the notion of risk values, which estimates the likelihood and consequences of certain threat scenarios. These risk values enable the tester to weigh threat scenarios as well as prioritize them and thus, helps in focusing on more relevant vulnerabilities. Risk analysis and risk assessment, similar to other development activities are performed in early project phases, are mainly based on assumptions on the system itself. But, testing with the system enables us to gain empirical evidence on the presence of vulnerabilities, applicability, quality.
In particular, risk-based testing can help in [38]

- 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 a 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 compledtion of risk analysis result by integrating vulnerabilities, potential threats, threat scenarios and unwanted incidents.
Zech, Philipp [44], proposed a model-driven methodology for the security testing of cloud environments, ingesting misuse cases, defined by negative requirements derived from risk analysis.
Ming Ni [36] paper focused on speed enhancements techniques that offer clear and meaningful ways to enhance human understanding and comprehension of security levels.
Julien Botella [7] presented an 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 a feature to drive the test generation process.
Yan Li [32], proposed a conceptual framework which is based on the two possible combinations of model-based security testing(MBST) 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 [23], worked on developing a graphical approach to threat and risk modeling that supports the security analysis process and approach is contributing to 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 modeling approach that emphasizes the modeling of threat scenarios and provides formalism to annotate the threat scenarios with probability values and formalism to reason with these annotations.

5.4 Fuzz testing

Because of its effectiveness in discovering the vulnerability, fuzz testing has gained a lot of attention. In Fuzz testing, injecting random data into a 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[38]. Fuzzy testing is illogical, just creates clutter data and require

very less computational cost and time. Fuzz testing is mostly implemented by a program or script that submits a combination of inputs to the software to disclose how the 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[21].
Patrice Godefroid [22] 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 et al. [41] proposed a lightweight, 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 fuzzy 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 behavior [14] 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 the 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 a powerful tool that allows us to detect vulnerabilities in source code. According to the security community, there is no alternate for actually looking at the code for detecting vulnerabilities and many serious security weaknesses cannot 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

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

doors, trojans, and other forms of malicious code can be exposed by source code reviews. Code-based and Binary code analysis share similarities, hence not discussing binary code analysis. Code reviews can be performed manually or automated and are 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 [43].

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 the security mechanism of the software. It is mainly used to test program-based attacks, such as stack buffer overflow.

V. Benjamin Livshits and Monica S. Lam [33] 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 find all vulnerabilities matching a specification in the statically analyzed code.

In Seung-Hyun Seoa and Aditi, Gupta [39] 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 [15] and RATS [16], 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.

Property-based testing is a special case of static analysis. 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 the evaluation of test executions for correctness and completeness. Paper [17] describes a method that transforms the security property of software into specification described by TASPEC language. It would extract the code about specific property by program slicing technology, and discover a violation of the code against security property specification. Property-based testing focuses on some specific security properties, which can meet the requirement of classification and priority. 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[21].

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

[16] `www.securesoftware.com`

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[21]. Testing space scanning deals with network port, string, procedure data, network data, and other elements scanning, for example, network port scanning can reveal issues related to vulnerable ports. Vulnerability scanning tools scan application inputs and outputs to look for known vulnerability signature in application-level software.
Common scanning tools include: Netsparker, Acunetix, and Retina CS Community with automation features and Security Profile Inspector (SPI), Internet Security Scanner (ISS), Security Analysis Tool for Auditing Networks (SATAN), Tiger, Sscan, Nmap, COPS, and Tripwire.
Jose Fonseca et al. [19] also proposed a method where they inject common types of software fault and then evaluate and benchmark automatic web vulnerability scanners and the results show a 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 et al. [27] proposed a customized vulnerability scanning system using a secure mobile agent that is resistant to attack and also can 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 tools. 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 Click-jacking vulnerabilities brought by HTML5, but also provide efficient Web application security scanning and evaluation services for the websites.
Jun Gao et al. [20] researched on the evolution of app vulnerabilities, they first developed a lineage of an android app which represent a sequence of historical release of that app and based on those lineages they observed the evolution vulnerabilities scanned by well-known scanner.

5.7 Tools for Security testing

Tools for Manual/Automation security testing techniques discussed in this paper shown in table 1. Based on project requirements, technologies offered, support, attack surface, and cost tools can be selected.
For instance, if someone wants to develop a secure web application, consider using the OWASP risk assessment framework and verifying code using static analysis tools like Synopsys Static Analysis (formerly Synopsys Coverity). Synopsys static analysis tool offer plugin for IDE and helps users view the impact of rule changes by displaying a comparison of results before and after the change without requiring a new scan. The same tool gained the position of leader in Gartner's Magic Quadrant for its high-assurance, fast, and accurate analysis.

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	Synopsis ; CAST ; 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

For generating test cases automatically, techniques introduced by Jan Jürjens [29] should be used. Using UWE(UML for web engineering) extended with security properties for a web application like UMLSec and systematically generating test sequences.
For penetration testing, usage of Nmap, ZAP, and Wireshark is highly recommended with the Penetration Testing Execution Standard (PTES) because of their analysis and vulnerability revealing capability. As a vulnerability scanner, Nessus Professional is the widely used and industry standard for vulnerability assessment. It helps professionals quickly identify and rectify vulnerabilities including software flaws, missing patches, malware, and misconfiguration on various operating systems, devices, and applications. To reveal unknown vulnerabilities in application fuzz testing techniques must be included, either manual or automated because it has the capability of revealing the most serious security faults or defects. Tools like WebScrab should be introduced as it has a framework developed on Java and designed for analyzing apps that are communicating via HTTPS and HTTP protocols.

6 Selection Criteria for Security Testing Approach

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

Büchler [16] has briefly explained what one should consider as selection criteria for security testing approach and tools:

- **Attack surface:** The Attack Surface is collection of all of the different points where an attacker could get into a system and where they could get out [17]. Approaches discussed in this paper reveals different types of vulnerability or points to get into a system. So, it is recommended to use various testing approaches to increase test coverage and 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, the team should research about the effectiveness like false positive rate and usability aspect concerning 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.) that are compatible with the approach.
- **Performance and resource utilization:** One should also consider the available computing power and manpower in the project as different tools and methods have different computational requirements.
- **Costs for licenses, maintenance, and support:** Tools in the market are either available free or cost money. The team should consider the support regarding the tool 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, attackers 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 the project and also work towards providing training related to that.
Hooshangi, Sara et al. [25] research on how security mindset makes better tester, revealed that students who wrote good defense monitors also wrote good attack tests. Furthermore, the student's knowledge of security can influence the quality of the programs and systems they develop [10].
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 :

[17] `https://cheatsheetseries.owasp.org/cheatsheets/Attack_Surface_Analysis_Cheat_Sheet.html`

1. Extensive knowledge of the domain.
2. Imagination helps in getting the idea of how certain things can work and ineffective 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 the definition, scope, challenges, requirements, and techniques of software security testing. The main focus of this paper is security testing techniques used in different phases of the software development life cycle with capability and limitation. As techniques, a brief overview of penetration testing, vulnerability scanning, risk-based analysis, 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 a security mindset can motivate to build secured applications which can increase the reliability and confidence in the product.
Information gathering is a crucial part of security testing. The team should collect known vulnerabilities and consider into test sequences. Exhaustive security testing is not possible, still, organizations have to concentrate equally or more on the security aspect of the applications from the beginning of life cycle and should adopt to systematic approaches/frameworks of security testing. To enhance the security test coverage in a fast-paced development environment, making use of various tools Wireshark, Nmap, and Webscrab, etc. will be very helpful in revealing known or unknown vulnerabilities and the saved time can be invested in generating some more abuse cases. As every tool is created to find different vulnerabilities as well as developed in different environments and languages which makes automation hard to realize. Research on the complete package of tools of all approaches is still open and which also focused on automation of security testing. As technology evolving towards ubiquitous computing and semantic web (mainly web-based application) which will be comparatively more complex and the nature of cyber attacks will also going to be different and complex. Hence, research on the security aspect of upcoming technologies demand more focus.

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] Matthias Büchler, Johan Oudinet, and Alexander Pretschner. "Semi-automatic security testing of web applications from a secure model". In: *2012 IEEE Sixth International Conference on Software Security and Reliability*. IEEE. 2012, pp. 253–262.

[10] 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.

[11] 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.

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

[13] 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.

[14] Srinivasan Desikan and Gopalaswamy Ramesh. *Software testing: principles and practice*. Pearson Education India, 2006.

[15] Michael Felderer et al. "A classification for model-based security testing". In: *Advances in System Testing and Validation Lifecycle (VALID 2011)* (2011), pp. 109–114.

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

[17] 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.

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

[19] 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.

[20] 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.

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

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

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

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

[25] 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.

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

[27] 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.

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

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

[30] Jan Jürjens. "UMLsec: Extending UML for secure systems development". In: *International Conference on The Unified Modeling Language*. Springer. 2002, pp. 412–425.

[31] 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.

[32] 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.

[33] 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.

[34] G. McGraw. "Software security". In: *IEEE Security Privacy* 2.2 (2004), pp. 80–83.

[35] 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.

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

[37] Ron Patton. *Software testing*. Pearson Education India, 2006.

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

[39] 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.

[40] 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.

[41] 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.

[42] 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.

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

[44] 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.