Software Vulnerability Analysis and its Prevention in IT Industry

Sushmitha Kasimsetty Ramesh
CS530: Fault Tolerant Computing
Prof Yashwant K Malaiya
Colorado State of University
Fort Collins, CO, USA
sush1997@colostate.edu

‘

***Abstract*—One of the most critical issues faced by IT organizations is secure software development. The core of all computer security issues is software vulnerabilities. Exploitable software vulnerabilities have received a lot of attention in recent years because of their potential for serious consequences in terms of computer security and information security. Many reasons can lead to software vulnerabilities. It takes a lot of effort to develop secure software. One of the most significant issues is the lack of secure practices in the software development life-cycle. To improve software security, tools, and procedures for measuring software security should be available. This paper discusses two methods of discovering software vulnerabilities. It also provides a technique for reducing software vulnerability and improving its security is proposed. This methodology is used in the software development life cycle processes. In addition, multiple measures are provided to analyze the software's level of security. This methodology proposes measures to evaluate security in each step in the development process inside the iteration rather than measuring security performance after each development iteration in the life cycle.**

***Index Terms*—Software vulnerabilities, IT Industry, Software development life-cycle;**

I. Introduction

[4]One of the most crucial concerns for IT firms is secure software development.A growing number of cyberattacks are based on software flaws, resulting in the loss of sensitive data and damage to a company's brand. Despite the numerous studies presented to help vulnerability detection, vulnerabilities continue to represent a danger to the secure operation of IT infrastructure. These flaws might have presented a risk to the safe use of digital products and devices around the world. Several studies have been conducted to establish effective approaches for developing and evaluating secure software systems. As a result, many components of software security are enforced in various phases.Many vulnerability detection approaches have been proposed to combat assaults induced by vulnerability exploitation, with the goal of identifying flaws prior to software distribution. Building powerful cryptosystems, implementing authentication protocols, and devising effective trust models and security rules are just a few of the difficult difficulties that come up while developing safe software systems. Despite these obstacles, most security attacks make use of either human errors, such as bad password selection and configuration, or software implementation weaknesses. As a result, software should be created in accordance with a desired level of security. To reach the highest professional and institutional quality and efficiency, software systems and applications require regular measures of the implementation operations, as well as the use of IT governance ideas based on transparency and accountability. This paper aids in the search for two deep learning-based vulnerability detection approaches. A technique for evaluating security measures during the software development phase in order to improve security. The foundation of this methodology is the quantification of some aspects of the software development life cycle (SDLC). [**?**].

II. Motivation

Recently, I came across an incident where the people in the bank were not able to access the information from the application due to server issues. These made them not complete their transactions and waited for a long period of time. Software vulnerabilities are the main reason for issues related to computer security. So, to mitigate the risk caused by vulnerabilities, certain models should be analyzed using the data so that they can be implemented on other software systems. These software security problems are one of the most notable issues in IT organizations. In IT organizations, numerous people will be working and there are more chances of leakage of information and other problems. Specifically, after the outbreak of covid, most companies are offering remote work, and the probability of issues related to security will be more. The risks for the companies will increase exponentially. If I consider an IT company as an example, considering one instance, in client/server systems, where data is scattered across several servers and locations, it's impracticable to design a centralized security system that can handle all the client and server security threats. The likelihood of such distributed systems and services being vulnerable to viruses and misuse in general, as well as other scenarios like Denial-of-Service assaults. Hackers and cyber criminals are always looking for new ways to exploit software flaws. Developers and stakeholders have greater opportunities to troubleshoot possible security issues and resolve them early on as an integrated part of the software development process if security

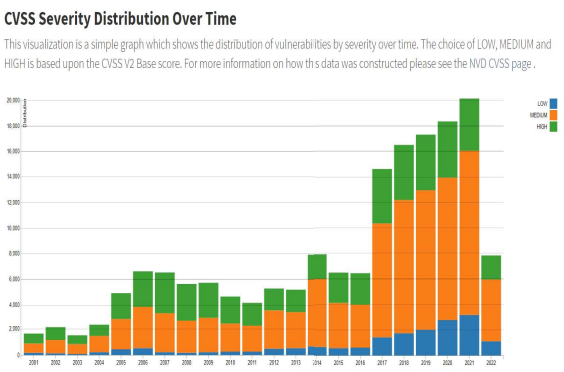

Fig. 1. CVSS distribution over a period of time

is prioritized throughout the SDLC. These issues motivated me to look for the problems, evaluate the risks occurring in them, and propose some ideas to prevent them.

The figure is taken from the mentioned URL. https://nvd.nist.gov/general/visualizations/vulnerability-visualizations/cvss-severity-distribution-over-time

III. Background

An IT company should be secured with the utmost level of precautions. When a company is started, there are several practices that must be considered which tend to possess risks. The two famous methodologies called waterfall and agile are followed by most companies. It's a way of managing projects with several programmers and systems that are very regimented and risk averse. But everything comes with a cost, clients may not know exactly what they want until they see functional software, so they change their minds, resulting in the redesign, rebuilding, and retesting, as well as higher expenses. [5]Some of the risks are, locking down version control repositories, scanning code for vulnerabilities, specifying minimal privileges to allow deployments, encrypting connections, and doing penetration testing used to be the norm when it came to securing in-house software. Locking down the network and infrastructure was an entirely different security domain that required different tools and procedures maintained by IT operations. It's simple to state that the company prioritizes security, and many companies do adhere to best security practices in agile and DevOps. However, because information security teams are frequently understaffed in comparison to development teams, it's simple to see how other businesses and technical debt objectives dominate agile team backlogs and why security standards are not implemented universally across the enterprise. Organizations rely on software development teams for their creativity, innovation, and technical chops to handle pressing business difficulties. However, requirements can occasionally lead development teams down the path of tackling challenging technological difficulties and implementing solutions that they could potentially accept from third-party sources.

IV. Research questions

Software vulnerability:An attacker could take advantage of a security defect, malfunction, or weakness detected in software code.

Most common software vulnerabilities:

• Missing data encryption:

Before being stored or transmitted, the software does not encrypt sensitive or critical information. Additional Information Without effective data encryption, the guarantees of confidentiality, integrity, and accountability that encryption provides are lost.

• SQL Injection:

SQL injection is a sort of web security threat which enables the client to alter database queries in a web application. It gives an attacker access to data they wouldn't otherwise have access to. These include information from other individuals as well as any other information the app has access to. An attacker can frequently alter or destroy this data, permanently altering the application's content or functionality.

A SQL injection attempt can be used to compromise the supporting webserver or other rear architecture, or in some situations, to perform a denial-of-service attack.

• OS command injection:

OS command injection (also known as shell injection) is a web security flaw that allows an attacker to run arbitrary operating system (OS) commands on the server hosting an application, effectively compromising the program and all of its data. Using trust connections to pivot the attack to other systems inside the company, an attacker can frequently use an OS command injection vulnerability to compromise other portions of the hosting infrastructure.

• Missing authentication for critical function:

This flaw depicts a situation in which software fails to validate a user's identity before granting access to privileged application capabilities.

This vulnerability is frequently introduced during the application development process' architecture and design phase.

A critical vulnerability in McAfee Advanced Threat Defense's web interface (CVE-2017–4052) is a real-world example of such a problem.

The flaw allows an unauthenticated remote attacker to change configuration settings or obtain administrative access to the affected application by sending a carefully crafted HTTP request.

• Missing authorization:

Because of this flaw, the software is unable to identify whether the actor's authorization to access data or perform activities was correct and accurate or not. Due to a lack of access control checks, any user can access all resources and perform whatever action they wish. It can expose sensitive information, create a denial of service, and execute arbitrary code. As a result of this flaw, attackers can read and manipulate sensitive data as well as gain access to privileged capabilities. During the Architecture Design, Implementation, and Operation stages, the vulnerability is introduced.

V. Proposed Approach

There are several methods for vulnerability detection with deep learning.
First method:
Penetration testing:
[3]Software Penetration testing examines a system's security by simulating hostile user attacks and determining if the attacks are successful. Because penetration testing is not based on falsifiable hypotheses, it is more akin to an art than a science. Penetration testing performed by software vendors' penetration testers is usually considered the final test before the product is released. Penetration testing is classified into three categories based on the amount of information offered by suppliers:
1) Black box penetration testing: It makes no assumptions about the infrastructure being tested. Before they can begin their investigation, the testers must first determine the location and scope of the systems. Black-box penetration testing simulates an assault from an outsider with no prior knowledge of the system.
2) White-box penetration testing. It gives testers a thorough understanding of the infrastructure they'll be testing, including network diagrams, source code, and IP addressing information. White-box penetration testing replicates what may happen in the event of an "inside job" or a "leak" of sensitive data, in which the attacker has access to source code, network layouts, and possibly even certain passwords.
3) Gray-box penetration testing: Black-box and White-box penetration testing, also known as Gray-box penetration testing, have notable differences. Which one to choose for software suppliers while conducting penetration testing is determined by the aims of the vendors.

[2]In general, penetration testing involves two processes: vendors arranging the test and testers carrying it out. Risk analysis and threat modeling should be the foundation of the test plan. Suppliers can use STRIDE, one of the best and most extensively used methodologies for threat modeling, and the results can be displayed as Threat Trees [32]. Suppliers then create a test strategy based on the dangers identified in the previous steps, which may include things like penetration testing type (Black-box, White-box, or Gray-box), logistical arrangements, timeline, tools, and requirements for the problem report, among other things. PTES (Penetration Testing Execution Standard) is recommended for testers who execute tests. PTES is made up of seven steps: a. Interactions prior to engagement; b. Intelligence gathering; c. Threat modeling; d. Vulnerability analysis; e. Exploitation; f. Post-exploitation; g. Reporting It's worth noting that the submitted report shouldn't merely state whether the target software has any vulnerabilities. More detailed information, such as reproduction methods, severity, exploit scenarios, and exploit code samples, should be included in the report. Suppliers might use these reports to assess the target's security risk and devise a new publication strategy [34]. There are a few aspects of penetration testing that are easily misinterpreted. To begin with, some people may conflate penetration testing with black-box testing, which leads to the problem of "security via obscurity," in which the defense is based on the attackers' lack of knowledge. This is the explanation for a lot of successful attacks. Penetration testers should presume that attackers have access to one or more versions of the application's source code, according to current best practices (e.g., OWASP OSSTMM). Second, after performing penetration testing, some people believe the target software is impervious to any attack. Rather, having no vulnerabilities revealed or resolving all vulnerabilities does not mean that the target has no more vulnerabilities, because the problem of whether there are further vulnerabilities or not is indeterminable. Finally, some may consider penetration testing to be unethical. In reality, penetration testing is merely a technique for resolving security issues, and its value depends on the practitioner's and client's intentions.

Second method:
Binary code vulnerability detector:

[7]Cyber Vulnerability Intelligence for Internet of Things Binary The application domain of neural model-based vulnerability detection is expanded from source code to binary code in this paper. Commercial software and firmware for Internet of Things (IoT) devices are typically delivered as binary code. This technique makes it possible to detect vulnerabilities without relying on source code. Data gathering, data preparation, model development, and evaluation/test are typically used in deep learning-based vulnerability detection approaches. The goal of data collecting is to collect labeled vulnerable and non-vulnerable data in order to train neural models. Data preparation seeks to turn raw data (in most cases, textual data) into vector representations that neural models can understand.
These two phases are linked to data, which is essential for developing a reliable detection system. Applying or customizing a deep neural network model to identify potentially vulnerable patterns in order to develop a vulnerability detector is known as model building. The developed detector is evaluated or tested in specific application scenarios. [6] This method presents a binary code vulnerability detector based on deep learning. Its goal is to widen the application field of vulnerability detection by addressing the lack of source code availability. Binary segments are sent to a bidirectional LSTM neural network with attention for data training (Att-BiLSTM). There are three steps to data processing. First, the binary segments were extracted from the original binary code using the IDA Pro program.The second stage is to extract functions from binary segments and label them as "vulnerable" or "not susceptible." Before feeding it to the Att-embedding BiLSTM's layer, the third step converts the binary segment to a binary feature. Furthermore, detection is granular down to the function level.

Many experiments are carried out on an open-source project dataset to evaluate the suggested method's performance. On binary code, the suggested method beats source code-based vulnerability detection tools, according to the findings of the experiments. However, there are drawbacks to this technology that could be addressed in the future. The detection accuracy is very low, with each dataset's detection accuracy falling below 80% . It also ignores the case of function inlining, in which the binary code's structure frequently changes as a result of function inlining. Deep learning approaches were used to discover binary code vulnerabilities in this work. It has sparked more research into using deep learning approaches to detect binary code vulnerabilities. It clears the way for deep learning-based vulnerability detection apps to migrate from source code to binary code.

Third method:
VulDeePecker: A Deep Learning Based System for Vulnerability Detection:
[7]The "code gadgets" notion, which consists of numerous lines of code, is used to represent data or control dependencies. The code gadget is a more fine-grained item for exposing variable weaknesses than earlier research that employ abstract syntax trees (ASTs) for understanding potentially vulnerable patterns. This configuration allows a neural model to acquire reliable and exact information about specific vulnerability categories, such as buffer and resource management errors. This is the first research to use fine-grained program representation for learning high-level representations in a neural network.
It's the first time a bidirectional Long Short Term Memory (BiLSTM) model has been used to detect vulnerabilities. Long-range dependencies are extracted and learned by the BiLSTM from code sequences. It gets its training data from a code gadget that represents the program supplied to the BiLSTM. The code gadgets are processed in three phases. The first step is to extract the program slices that correspond to library/API function calls. The next step is to make and label the code gadgets. The code gadgets are then converted into vectors in the third stage. The long-range dependencies are then captured from the code gad receives. The detection granularity in this paper is at the slice level. A collection of experiments on open-source projects and the SARD dataset are used to evaluate the performance of VulDeePecker. The findings of the experiments reveal that VulDeePecker can manage many vulnerabilities, and that human experience can assist increase VulDeePecker's effectiveness. VulDeePecker is also more effective than static analysis techniques that rely on experts' set rules to find vulnerabilities. However, there are certain limitations to this system that could be improved. VulDeePecker's initial limitation is that it can only deal with C/C++ applications. VulDeePecker can only deal with vulnerabilities connected to library/API function calls, which is the second limitation. The third constraint is the short size of the performance review database, which only covers two aspects of vulnerabilities.. The BiLSTM model used in VulDeePecker has prompted other researchers to follow suit. The usage of BiLSTM allows researchers to examine code dependencies over large distances. Instead of recording data dependent relations, VulDeePecker recommends using fine-grained code gadgets to represent a program. This paper also sparked further research in, where code gad gets used to represent programs in order to capture data and govern dependency relationships. Furthermore, this publication and a few extended works are produced by the same research team, which is constantly improving and innovating.

Fourth method:
Fuzzing:
[3] In 1990, Miller et al proposed the concept of Fuzzing and built the first Fuzzing program, "fuzz," which generates streams of random characters, in response to Modem applications' tendency to fail owing to random input caused by line noise over "fuzzy" telephone connections. It crashed more than 24 percent of the ninety distinct utility programs tested on seven different UNIX versions. In academics, there are numerous meanings of Fuzzing. Fuzzing is a highly automated testing technique that covers multiple border cases using incorrect data (from files, network protocols, API requests, and other destinations) as application input to better assure the absence of exploitable vulnerabilities, according to Miller et al. and Oehlert. Fuzzing is considered brute force testing by M. Sutton et al. Fuzzing is a type of black-box testing, according to A. Lanzi et al. This paper will define Fuzzing from the standpoint of its procedure: It generates semi-valid or random data first, then delivers it to the target application; finally, it monitors the program as it consumes the data to check if it fails. Semi-valid data is information that is right enough to pass input tests but still incorrect enough to cause issues. The key to Fuzzing is data production and target monitoring. Data was initially generated in a completely random manner, which resulted in the majority of the data being completely invalid. Researchers later presented two basic data generation methods: data-generation methodology and data-mutation technique. To generate data, most data-generation techniques rely on specifications such as file format requirements and network protocol specifications. This technique necessitates extensive user knowledge of the examined file formats and protocols, as well as significant human engagement in the process. Users just need a basic understanding of file formats and protocols to use the data-mutation technique, which creates data by changing specific fields of valid inputs. Data-mutation is preferable to data-generation when specifications are extremely complex and sample data is readily available. Data-mutation, on the other hand, is insufficient for extremely sensitive circumstances since it is significantly dependent on initial values, which means that differing initial values might result in drastically different code coverage rates and effects. Autodafe and APIKE Proxy are technologies that use a data-mutation technique to produce data. Peach is a program that combines both methods. When it comes to target monitoring, most tools rely on third-party debuggers like OllyDbg and

GDB, while some, like Sullery and Autodafe, employ their own specialized debuggers. In addition to target application debugging. Other statistics to keep an eye on include memory consumption, CPU utilization, and network activity. Many people associate fuzzing with black-box testing, however this is not the case. White-box Fuzzing, which combines Fuzzing and dynamic test creation, was proposed by P. Godefroid et al. Symbol execution observes P's processes on I0 and a path constraint that is in the form of logical formulae; second, it negates some marks of the route constraint, solves a new constraint via a constraint solver, and creates a new input I1 whose execution path is distinct from I0's. Finally, it processes I1 in the same way as I0 was processed, then continues the previous three procedures until time runs out or no new input is generated. White-box fuzzing's purpose is to run as many control routes as possible and find vulnerabilities as quickly as feasible using various search algorithms. P. Godefroid et al. incorporated grammar specifying legal input into white-box Fuzzing and replaced bytes in program inputs with tokens produced from a lexical analyzer. This solution solves early path explosion problems and situations where code beyond the initial stages cannot be reached in applications with highly structured inputs. P. Godefrorid et al. discovered through experiments that white-box Fuzzing based on grammar is superior to white-box Fuzzing and black-box Fuzzing; additionally, white-box Fuzzing based on grammar is superior to black-box Fuzzing. Grammar-based sleuthing People will run into numerous issues when using Fuzzing to detect vulnerabilities. Input generation is hampered by file format and protocol validation. Some programs match key phrases with hash values, which makes solving constraints more difficult. In addition, inputs may be processed using signature, encryption, or compression methods. Ffuzzer should be able to re-sign, re-encrypt, and re-compress files in this regard.

VI. Software Development Life Cycle

In fig2, SDLC is given [1].The Software Development Life Cycle is the process of creating software applications using standard business procedures. Planning, Requirements, Design, Build, Document, Test, Deploy, and Maintain are the traditional six to eight processes. Some project managers will mix, split, or omit processes depending on the scope of the project. These key components should be included in any software development initiatives.SDLC assists in achieving these objectives by finding inefficiencies and increasing costs and correcting them so that everything runs smoothly.

The Software Development Life Cycle essentially lists all of the steps involved in creating a software application. This helps to eliminate waste and improve the development process' efficiency. Monitoring ensures that the project stays on schedule and is a viable investment for the business.

Many businesses will break down these processes into smaller chunks. Technology research, marketing research, and a cost-benefit analysis are all parts of planning. Other steps

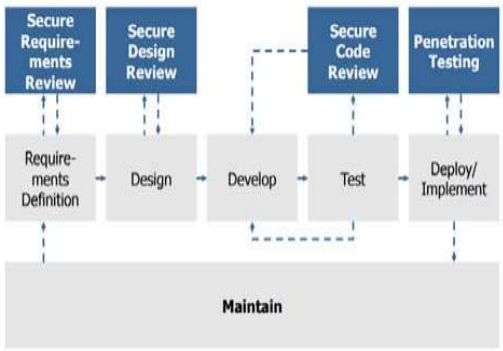

Fig. 2. Software development life cycle

may interact with one another. Because developers must remedy faults that occur during testing, the Testing phase can run concurrently with the Development phase.

The SDLC is a method for evaluating and improving the development process. It enables for a fine-grained study of each process phase. As a result, businesses may maximize efficiency at each level. As processing power grows, the demand for software and developers grows. Companies must cut expenses, deploy software more quickly, and meet or exceed consumer expectations.The SDLC is the planning, creation, testing, and deployment process. As it cascades from feasibility research to systems analysis, design, implementation, testing, and finally installation and maintenance, its life cycle is commonly referred to as a waterfall model. While there are numerous benefits to using this framework for a design project, the following are the most prevalent. Optimizing software security throughout the SDLC has several advantages: • Software performance has improved. • Business risks are reduced. • Reduced costs for detecting and fixing software flaws • Consistent adherence to security laws and regulations, resulting in a reduction in fines and penalties. • Increased consumer loyalty and trust • Internal organizational security is improved.

VII. My work

Even if you follow software security best practices to a tee, you'll always be vulnerable to a breach if you don't have a solid incident response plan in place. Even if attackers enter your networks, if you prepare, you can prevent them from completing their purpose. Have a comprehensive incident response (IR) plan in place to detect and mitigate the effects of an attack.

VIII. Methodology to enhance security during development process

Fig 3, [1]A Secure SDLC is created by adding security-related activities to a present development process.

Measurement of both the product and development processes has long been regarded as a necessary component of effective software development. Realistic project planning,

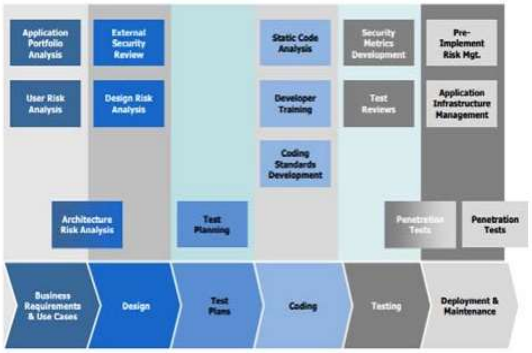

Fig. 3. Security modeling in Software development life cycle

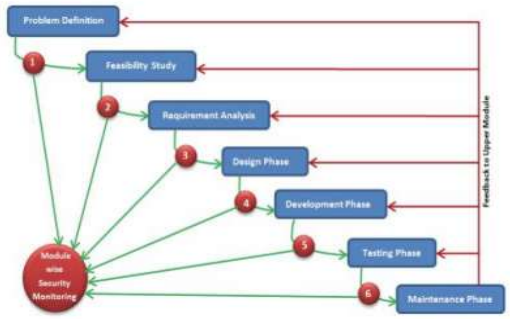

Fig. 4. Security in each phase

timely monitoring of project progress and status, identification of project hazards, and successful process improvement are all enabled by good measuring procedures and data. During project execution, appropriate metrics and indicators of software artifacts such as requirements, designs, and source code can be examined to diagnose problems and find solutions, reducing defects, rework (effort, resources, etc.) and life cycle time. These approaches assist businesses to produce higher-quality products that are more mature. The following SLDC areas are addressed in relation to the definition and application of measures for secure development [1]:

- Requirements Engineering
- Architectural Risk Analysis
- Assembly, Integration, and Evolution
- Code Analysis
- Risk-Based and Functional Security Testing
- Software Development Life-Cycle (SDLC)
- Coding Rules
- Training Awareness
- Project Management

This methodology focuses on measuring entire SDLC phases, as mentioned in the subsections below.

A. Requirements Engineering: It is designed to model the system's intended functions and environment, as well as to extract, detail, prioritize, and organize the set (package) of requirements, including text, attributes, and traceability for all requirements that describe the software of the target release. During requirement gathering, the analyst should follow his check list as we mention in our process, and he should peer review his document and get approval for the do.

B. System Architecture: System Architecture defines the architecturally significant components, modules, interfaces, and data for a software system in order to meet specified requirements based on Method Task Establish and Maintain Software Architecture and should make sure that the software architecture is up to date and peer review has to be done.

C.Development: Development should follow all code standards to avoid SQL injection, XSS, DOS, and other vulnerabilities, conduct coding reviews, and build and validate the build for either a subsystem or the complete system, depending on the method task (Integrate Software Elements).

D. Execution and testing: Execution and testing should be designed to run the relevant collections of tests needed to validate the overall system's behavior quality and uncover faults in the system's functional and non-functional requirements.

E. Quality Management: Quality management ensures that the project's chosen product meets the project's specified requirements, as well as providing staff and management with objective insight into processes and associated work products in comparison to applicable process descriptions, standards, and procedures.
[6]

IX. Results in terms of vulnerabilities

Two scenarios have been implemented to assess the efficacy of the suggested methodology:

The vulnerability is scanned in the first scenario before the proposed methodology is implemented. In this scenario, there are a considerable number of vulnerabilities (= 46), which has a significant impact on the severity of risk (= critical (5)) and overall product quality.

The vulnerability is scanned after the proposed methodology is implemented in the second scenario. Following the implementation of the technique, the acquired results showed that there were few vulnerabilities (=6) with a minimal influence on risk severity (= low (1)) and overall product quality. The reference is fig 5 and fig 6 [1]

X. CONCLUSION

According to the statistics of last year, the NVD database contains 21,957 vulnerabilities. This figure is greater than in previous years (18,362 in 2020, 17,382 in 2019, and 17,252 in 2018). As the software vulnerabilities increases, risks increase proportionally. So, to mitigate it,

- A method for the analysis of detection of the software vulnerability called penetration testing and binary code vulnerability, Vuldeepecker, Fuzzing is provided.
- Also, a methodology for secure software development life

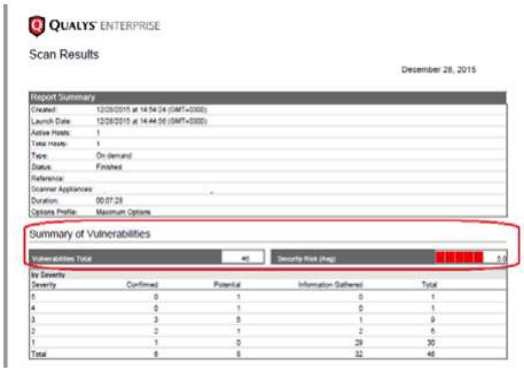

Fig. 5. Vulnerability before implementing proposed SDLC

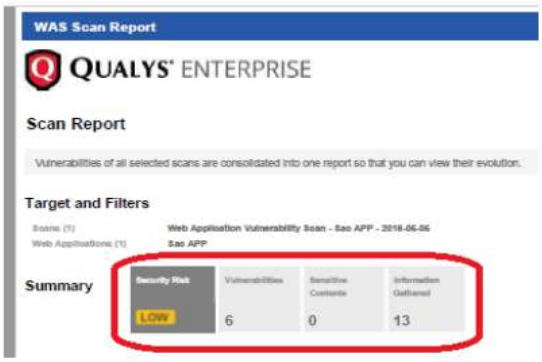

Fig. 6. Vulnerability after implementing proposed SDLC

cycle has been proposed.
Hence, the vulnerabilities can be reduced. By integrating the security at each step in the development cycle, most of the risks can be reduced. Vulnerabilities' criticality is also greatly decreased. All of this results in significant cost and time savings, which are especially noticeable in the latter stages of development. Software security and quality, as well as development productivity, are improved.

CONFERENCE

[1] Conference: 2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO) Amity University, Noida, India. Sep 3-4, 2021
URL:https://www.amity.edu/aiit/icrito2021/
Description:The goal of the conference is to The establishment of knowledge bases and simple access to organized databases on systems, processes, and technology based on a quantitative study that can provide a competitive advantage in the marketplace through high-quality processes, products, and services..

[2] Conference:2019 IEEE International Conference on Smart Cloud (SmartCloud), Dec. 10 2019 to Dec. 12 2019,Tokyo, Japan,
URL:https://www.computer.org/csdl/proceedings/smartcloud/2019/1jPb86kKQj6
Description:This conference will bring together academics and industry professionals to explore the foundations, methodologies, and tools for automating the analysis, design, implementation, testing, and maintenance of complex software systems.

[3] Conference:2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)
URL:2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)
Description:The purpose of this conference is to bring together researchers, practitioners, and educators to present and discuss the most recent developments, trends, experiences, and concerns in the field of software engineering.

[4] Conference:2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering ,Aug 5 2019 to Aug. 8 2019Rotorua, New Zealand
URL:https://www.computer.org/csdl/proceedings/trustcombigdatase/2019/1
Description: The conference intends to bring together scholars and practitioners from around the world who are working on trust, security, privacy, reliability, and dependability in computing and communications

[5] Conference:2019 IEEE International Conference on Systems, Man and Cybernetics (SMC) Bari, Italy. October 6-9, 2019
URL:http://smc2019.org/index.html
Description:This is a one-day conference. It serves as an international venue for scholars and practitioners to share cutting-edge research and development, describe the state-of-the-art, and exchange ideas and advances in all areas of systems science and engineering, human machine systems, and cybernetics.

[6] Conference:2021 IEEE International Conference on Big Data (Big Data),Dec. 15, 2021, to Dec. 18 2021 Orlando, FL, USA
URL:https://www.computer.org/csdl/proceedings/bigdata/2021/1A8gmCnij
Description: Its goal is to bring together researchers and industry practitioners in a synergistic way.

[7] Conference:2020 27th Asia-Pacific Software Engineering Conference (APSEC)Dec. 1 2020 to Dec. 4 2020, Singapore,
URL:https://www.computer.org/csdl/proceedings/apsec/2020/1rCgBwZRc5O
Description :The main research track of this conference will highlight the most current and significant advances in the field of software engineering and all of its sub-disciplines.

[8] Conference:2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion

(QRS-C), July 22-26,2019. Sofia, Bulgaria
URL:https://qrs19.techconf.org/
Description :This conference gives an opportunity for engineers and scientists from both business and academia to present their current work, share their research findings and experiences, and discuss the most effective and efficient approaches for developing dependable, secure, and trustworthy systems.

[9] Conference: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)
URL: https://icsme2020.github.io/
Description:This conference brings together academics, industry professionals, and government officials to present, debate, and discuss the most recent ideas, experiences, and issues in software maintenance and evolution.

JOURNALS

[1] Journal: Journal: IEEE Transactions on Dependable and Secure Computing (Volume: 17, Issue: 4, July-Aug. 1 2020)
URL: https://www.computer.org/csdl/journal/tq
Description: All areas of reliability and security are covered in this peer-reviewed scientific journal.

[2] Journal: Journal: IEEE Transactions on Fuzzy Systems (Volume: 28, Issue: 7, July 2020)
URL: https://cis.ieee.org/publications/t-fuzzy-systems
Description: The theory, design, and applications of fuzzy systems, ranging from hardware to software, are the focus of this publication.

[3] Journal: Journal:IEEE Transactions on Cloud Computing (Volume: 9, Issue: 2, April-June 1 2021)
URL:https://dblp.org/db/journals/tcc/tcc9.html
Description: The diverse topic of cloud computing and software security efficiency is the focus of this publication.

[4] Journal: Journal:IEEE Internet of Things Journal (Volume: 8, Issue: 13, July1, 1 2021)
URL:https://ieee-iotj.org/
Description: Description: It includes studies on the Internet of Things.

[5] Journal: Journal:[5] Journal: IEEE Access (Volume: 9), 26 February 2021
URL:https://dblp.org/db/journals/access/access9.html
Description: Description: It aims at research of Software Vulnerability Security Bug Report Usage.

[6] Journal: IEEE Transactions on Software Engineering (Volume: 47, Issue: 12, Dec. 1 2021)
URL: https://dblp.org/db/journals/tse/tse47.html
Description: It aims at research of Security Vulnerabilities in Machine Learning, topics related to IOT .

[7] Journal: : IEEE Transactions on Network and Service Management (Volume: 18, Issue: 3, Sept. 2021)
URL: https://dblp.org/db/journals/tnsm/tnsm18.html
Description: It aims at discussing the security of the system and software data leakage challenges .

[8] Journal:IEEE Access (Volume: 7)
URL:https://dblp.org/db/journals/access/access7.html
Description:This journal Strategies in a Software Testing Learning Environment and helps in giving information about software engineering.

[9] Journal:2IEEE Transactions on Software Engineering (Volume: 47, Issue: 5, May 1 2021)
URL:https://www.computer.org/csdl/journal/ts
Description:This journal is looking for well-defined theoretical conclusions and empirical investigations that could have an impact on software development, analysis, or management.

[10] Journal: IEEE Transactions on Parallel and Distributed Systems (Volume: 31, Issue: 2, Feb. 1 2020)
URL:https://www.computer.org/csdl/journal/td
Description:This journal publishes research on parallel and distributed computing that is both basic and translational analysis.

[11] Journal:IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 48, NO. 2, FEBRUARY 2022
URL: https://www.computer.org/csdl/journal/ts/2022/01
Description:This journal contains topics related to software engineering

XI. TOP ORGANIZATIONS RELATED TO SOFTWARE VULNERABILITIES

The organizations for vulnerability Management Solutions Companies

[1] F Secure Radar

Flexera

GFI Software

Layer Seven Security

Qualys

Acunetix

Beyond Security

Netsparker

NEWS

[1]https://techxplore.com/news/2022-04-technique-fasternonvolatilememory-tech.html

[2] https://www.nbcnews.com/tech/security/us-warns-new-softwareflaw-leaves-millions-computers-vulnerable-rcna8693

[3]https://portswigger.net/daily-swig/git-security-vulnerabilitiesprompt-updates

[4] https://thehackernews.com/2022/03/the-continuing-threat-ofunpatched.html

[5] https://www.softwaretestingnews.co.uk/the-limits-of-regression-test-automation

[6] https://www.browserstack.com/guide/software-testing-challenges

[7] https://www.theguardian.com/technology/2021/dec/10/softwareflaw-most-critical-vulnerability-log-4-shell

[8] https://www.darkreading.com/vulnerabilities-threats

[9] https://www.channele2e.com/business/enterprise/5-key-challenges-of-software-testing

[10] https://u-tor.com/topic/testing-trends-for-2021

REFERENCES

[1] A. S. Farhan and G. M. Mostafa. A methodology for enhancing software security during development processes. In *2018 21st Saudi Computer Society National Computer Conference (NCC)*, pages 1–6. IEEE, 2018.

[2] G. Lin, J. Zhang, W. Luo, L. Pan, O. De Vel, P. Montague, and Y. Xiang. Software vulnerability discovery via learning multi-domain knowledge bases. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2469–2485, 2019.

[3] B. Liu, L. Shi, Z. Cai, and M. Li. Software vulnerability discovery techniques: A survey. In *2012 fourth international conference on multimedia information networking and security*, pages 152–156. IEEE, 2012.

[4] J. C. S. Núñez, A. C. Lindo, and P. G. Rodríguez. A preventive secure software development model for a software factory: a case study. *IEEE Access*, 8:77653–77665, 2020.

[5] J. D. Pereira. Techniques and tools for advanced software vulnerability detection. In *2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 123–126. IEEE, 2020.

[6] N. Visalli, L. Deng, A. Al-Suwaida, Z. Brown, M. Joshi, and B. Wei. Towards automated security vulnerability and software defect localization. In *2019 IEEE 17th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 90–93. IEEE, 2019.

[7] P. Zeng, G. Lin, L. Pan, Y. Tai, and J. Zhang. Software vulnerability analysis and discovery using deep learning techniques: A survey. *IEEE Access*, 8:197158–197172, 2020.