2020 International Conference on Software Security and Assurance (ICSSA) | 978-1-6654-3246-7/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICSSA51305.2020.00012

Continuous Security through Integration Testing in an Electronic Health Records System

Saptarshi Purkayastha*, Shreya Goyal*, Tyler Phillips†,
Huanmei Wu*, Brandon Haakenson†, Xukai Zou†
*Department of BioHealth Informatics, †Department of Computer Science
Indiana University-Purdue University Indianapolis
Indianapolis, Indiana 46202, USA.
{saptpurk, shregoya, phillity, hw9, bhaakens, xzou}@iupui.edu

***Abstract*—The estimated average cost of a healthcare data breach in 2019 was $6.45 million, which is the highest among all industries. Yet, security remains an afterthought in many digital health applications. Formal methods for testing for bugs are commonplace in software development through the use of unit testing, integration testing, system testing, and acceptance testing. More so, in modern software engineering, continuous integration is a well-known concept to run automated tests soon after any code change, when the system builds and notifies the development team of the test results. In this paper, we describe the use of a popular Python unit testing framework to implement a formal method of security testing. Common Vulnerability Scoring System (CVSS) is used to calculate metrics that represent the state of security of a deployed system. We developed a series of Pytest Behavioral Driven Development (BDD) scripts to test the Authentication and Availability of a widely used Electronic Health Records System called OpenMRS. The advantage of using the BDD approach is that testing scripts, called Gherkin files, can be read, and understood by the developers as well as the non-developer stakeholders. The use of Gherkin serves two purposes: firstly, it serves as the project's documentation, and secondly, it automates the tests. The use of the CVSS score between 0 to 10 becomes an objective metric to compare every code change, thus achieving continuous security. We plan to expand BDD scripts to attacks like Denial of Service, Session Hijacking, SQL Injection, and other privilege escalation attacks.**

***Index Terms*—Behavior Driven Development, Common Vulnerability Scoring System, Continuous Security, Electronic Health Records, Integration testing, OpenMRS.**

I. Introduction

According to a study conducted for the reported data breaches in the United States between 2013-2017, it was seen that there were 128 breaches involving Electronic Health Records (EHR), which affected more than 4.8 million patient records [1]. Also, there were a total of 363 breaches classified as hacking incidents, which involved more than 130 million patient records, and 1149 breaches that were non-hacking incidents but impacted over 23 million records [1]. The healthcare data breach on an average was estimated to cost $6.45 million for the year 2019 [2]. From these, we can see the urgent need for robust security in health systems, right from software development to deployment. Also, there is a need to monitor the IT systems and ensure that they are updated and tested against various security attacks at regular intervals to identify the potential vulnerabilities that can take place [3]. For the United States, it was seen that in the year 2016, the estimated total cost for EHR hacking incidents was $3.6 billion, followed by unauthorized access or disclosure of personal health information at $466 million [4]. Thus, poor security practices are costing a lot of money and risking the identity and health of many.

Continuous Integration (CI), where code is built and unit tests are run after every code change [5], and *Continuous Delivery* (CD), where the software package is built with integration with other system components and deployed to a test server [6], are considered best practices in modern software development [7]. CI eliminates the discontinuities between the development and deployment phases of the software development cycle. CI is defined as a process or practice which is comprised of various inter-connected steps like compiling code, unit testing, and acceptance tests, checking compliance with the code standards, and building deployment packages [8]. *Continuous Security* (CS) builds on the ideas of CI and CD, but instead of testing features of individual components, it deploys the system in a real-like environment and simulates attacks of different types and validates the system [9]. *Continuous Security* allows better integration of the security testing activities with the coding and makes security a software development practice, rather than an afterthought. While CI and CD have emerged and benefited the software engineering practices with early detection of errors and minimizing the last-minute cost of fixing them, similar practices for security testing are uncommon, but highly desired. We see two main reasons for the lack of adoption of *Continuous Security* - metrics and software development practices [8]. CI/CD improvements are relatively easy to assess and measure because of concepts like test coverage, feature testing, or user acceptance test are easy to define in an automated process. Vulnerability metrics may range from network security [10] to software library, zero-day issues, or more common security-related bugs like buffer overflow or Cross-Site Request Forgery (CSRF). These have made CS harder to implement in practice than CI or CD [8], [11].

Behavior Driven Development (BDD) allows writing tests, similar to in spoken English, which is easily understandable by the non-technical stakeholders [12]. The main goal of BDD

978-1-6654-3246-7/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSSA51305.2020.00012

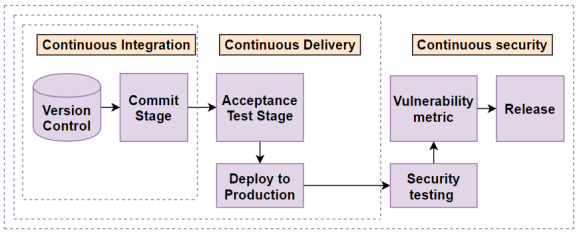

Fig. 1. Conceptual relationship between CI, CD and CS

is to get executable specifications of a system. BDD starts with the textual description of the conditions using specific keywords like *Given*, *When* and *Then* to tag the type of sentence and indicate how the sentence should be treated in the testing phase [13]. OpenMRS is a popular open-source Electronic Health Record (EHR) system that is used in over 40 countries around the world, including for clinical practice, research and teaching [14]. By using BDD in the process of software development for OpenMRS, we were able to simulate the attacks and validate the security of the system whenever code is merged into the master repository. OpenMRS already has a CI/CD process in its software development practice. By adding Continuous Security in the process, we demonstrate that vulnerabilities in the system can be identified and fixed during software development itself.

II. Related Work

Staff and Ernst introduced the concept of 'continuous' testing and suggested that continuous testing is an effective tool for reducing the overall development time by almost 15%, indicating that continuous testing can be incorporated to reduce the waiting time [15]. A literature review by Shahin et al. on the approaches and tools for the implementation of continuous practice (CI/CD) identified around 30 approaches and tools that facilitate the reduction of build and test time in continuous integration, increase visibility and awareness on the test, automates the continuous testing and improves the reliability of deployment methods [16]. While the tools and apps are addressing and solving a wide range of problems, there exist some of the challenges for adopting continuous practices. These challenges include lack of awareness, high cost associated with it, lack of skilled labor and expertise, lack of suitable tools, resistance to change, and lack of proper test strategy [16].

The use of the Common Vulnerability Scoring System (CVSS) is emerging and being used by many organizations for risk assessment. Mell et al. reviewed that several organizations such as Cisco, US National Institute of Standards and Technology (through the US National Vulnerability Database (NVD)), Qualys, Oracle, and Tenable Network Security are using CVSS [17]. The benefits of using the CVSS framework are standardized risk scores, contextual scoring, and its an open-source framework [17]. Another study evaluates the security of the medical devices by describing the test scenarios and calculating the corresponding CVSS scores. The study highlights that the healthcare industry have a low adoption percentage (around 61%) for the security frameworks. The study also found that the NIST framework was among the top four frameworks adopted by the organization. The challenges mentioned in the previous studies The NIST CVSS framework is a low-cost, easy to use system for identifying the security vulnerabilities and enhance the risk management processes [18]. The another added advantage of this framework includes easy to understand and interpret results, thus requiring no additional training or tools. The scores are interpreted as: Low: 0 to 3.9; Medium: 4 to 6.9; and High: 7 to 10. Higher score indicates higher vulnerability to attacks.

Another study compared different testing frameworks and found that the Pytest framework was the most efficient and suitable one. The reason being its straightforward test creation, ease of writing the test applications and integration with another platforms. Additionally, because Pytest is a Python package we can take advantage of other open source Python packages available with desired functionality [19]

III. Methodology

We implemented our *Continuous Security* software development practice using Pytest [20] BDD, as a way to enhance collaboration between developers and EHR implementers. The BDD approach has allowed the product planners and software designers to specify the requirements in simple English, which is matched to unit tests that perform security testing after deployment [12]. The enhanced collaboration also helped in troubleshooting and writing concrete automation tests. The unit tests eventually produce scores using the Common Vulnerability Scoring System framework, such that each identified severity and associated risk can be measured. The CVSS specification defines a framework that the assessor can use to transform information about the system vulnerability into a CVSS score [21].

The CVSS framework provides several dimensions over which a vulnerability score is calculated. These dimensions are classified into three groups, or metrics: Base Metric, Temporal Metric, and Environmental Metric. The Base Metric Group is, by far, the most used in practice [22], [23]. The Base Score calculation is organized in two conceptually different groups - *Exploitability metrics* and *Impact metrics*. Exploitability metric describes the means by which an attacker can deliver a successful attack, whereas Impact metrics provide an assessment of the consequences of a successful attack on the impacted system [21]. Exploitability metrics under CVSS v3 are measured over four parameters: Attack Vector (AV), Attack Complexity (AC), Privileges Required (PR), and User Interaction (UI). Impact metrics in CVSS v3 are measured over the CIA triad Confidentiality, Integrity, and Availability [21].

We performed vulnerability scans and penetration tests to discover weaknesses in our OpenMRS deployment. Penetration tests consisting of a series of automated testing scenarios were then defined in a Gherkin feature file. A Gherkin file is Pytest-BDD standard description format, where

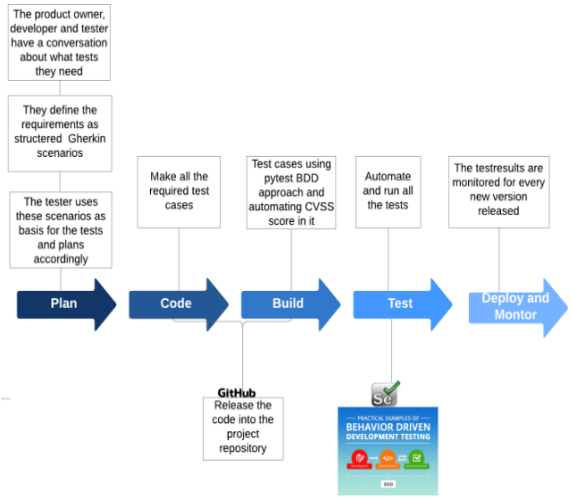

Fig. 2. Pytest BDD approach

we describe the test scenario, and then based on the results of our Pytest script, a CVSS score is generated. Our paper focuses on the six elements of information security defined by the Parkerian Hexad along with the three major tenants of access control. The security attributes of Parkerian Hexad consists of Confidentiality, Integrity, Availability, Authenticity, Utility, and Possession [24].

After performing the literature search, we came up with some of the most common security attacks for an EHR and

A. Test-case for Authenticity

Test Scenarios for Authenticity attack include SQL injection attacks, where the attacker can bypass the identity of the user and get access to the resources by sending the smart inputs. The smart inputs, called tautology, bypass the authentication step. We verify for these vulnerabilities, and check if the system allows us to login as the user we claimed to be or not. If we are able to log in, it indicates that the system is highly vulnerable to SQL injection attacks, and a lower CVSS score is given, as per Table I.

TABLE I. Parameters for Test Case 1 (Authenticity)

Parameter Description	Parameter Value	Description
Attack Vector	Netw(N),0.85	remotely exploitable
Attack Complexity	Low(L),0.77	no special conditions required
Privilege Required	None(N),0.85	privilege required is none
User Interaction	None(N),0.85	No user interaction required
Scope	Unchanged(U)	Impacted and vulnerable component same
Confidentiality	Admin 0.56, Other 0.22	Admin access(H), Other user access like Physician and Patient(L)
Integrity	Admin 0.56, Physician 0.22, Patient 0	Admin access(H), Other user access like Physician(L), Patient(N)
Availability	None(N),0	No loss of Availability

B. Test Case for Session Management

Authentication & session management is a critical part of web application security. Flaws in any of these areas can cause failure to protect the user credentials and session token used within one life cycle [25]. EHR systems for patient privacy and security, including OpenMRS, are usually configured to logout after every 10 mins of inactivity. Ending user sessions is an important tenant for validating the system against authentication attacks. If the session is disconnected or logged out, the user should not be allowed access to the account on hitting the back button. Test Scenarios are developed to ensure that the current session terminates when the user is logged out. The impact parameters for this use-case include Confidentiality and Integrity and shown in Table II.

TABLE II. Parameters for Test Case 2 (Session Management)

Parameter Description	Parameter Value	Description
Attack Vector	Network(L),0.55	physical access to vulnerable component required
Attack Complexity	Low(L),0.77	no special conditions required
Privilege Required	None(N),0.85	privilege required is none
User Interaction	None(N),0.85	No user interaction required
Scope	Unchanged(U)	Impacted and vulnerable component same
Confidentiality	Admin 0.56, Other 0.22	Admin access(H), Other user access(L)
Integrity	Admin 0.56, Physician 0.22, Patient 0	Admin access(H), Other user access(L), Patient(N)
Availability	None(N),0	No loss of Availability

C. Test Case for Brute force attack

Test Scenarios were developed to test the OpenMRS platform against Brute Force Attack. Known usernames and randomly generated strings for passwords were used to exploit the weakness of the system. The system should ideally suspend the account after some limited number of incorrect password attempts. For our test case, the random guess for username and password attempts were made for ten times and examined for account suspension. Confidentiality and Integrity are the two important impact parameters for this case, and CVSS parameter values are shown in Table III.

TABLE III. Parameters for Test Case 3 (Brute force authentication)

Parameter Description	Parameter Value	Description
Attack Vector	Network(N),0.85	remotely exploitable
Attack Complexity	High(H),0.44	Username should be known for exploitation
Privilege Required	None(N),0.85	privilege required is none
User Interaction	None(N),0.85	No user interaction required
Scope	Unchanged(U)	Impacted and vulnerable component same
Confidentiality	Admin 0.56, Other 0.22	Admin access(H), Other user access(L)
Integrity	Admin 0.56, Physician 0.22, Patient 0	Admin access(H), Other access(L), Patient(N)
Availability	None(N),0.0	No loss of Availability

The CVSS score for Confidentiality and Integrity will depend on the privilege accessed by the attacker after logging into the system. If the user gains admin privileges, then total loss of Confidentiality and Integrity will occur. If the user gains physician or nurse privilege, then a partial loss of Confidentiality and Integrity will occur. With patient privilege, the attacker will have limited access to resources. Such a

limited loss of Confidentiality will be no loss of system integrity as they cannot modify the records. The Execution time for the attack is also calculated.

D. Test-case for Authorization

Authorization attack includes privilege escalation and SQL injection attacks. Test Scenarios were developed to determine the venerability of the system against such attacks. If successful, this type of attack can result in gaining privileges as high as the administrative level.

TABLE IV. Parameters for Test Case 4 (Authorization)

Parameter Description	**Parameter Value**	**Description**
Attack Vector	Local(L),0.55	physical access to vulnerable component required
Attack Complexity	Low(L),0.77	no special conditions required
Privilege Required	Low(L),0.62	low privilege user tries to gain access to higher privileges
User Interaction	None(N),0.85	No user interaction required
Scope	Unchanged(U)	Impacted and vulnerable component same
Confidentiality	Admin 0.56, Other 0.22	Admin access(H), Other access Physician(L)
Integrity	Admin 0.56, Other 0.22	Admin access(H), Other user access like Physician(L)
Availability	None(N),0	No loss of Availability

Privilege Escalation occurs in two ways: Vertical Privilege escalation where the user with lower privilege receives the access of users with high-level privileges and Horizontal Privilege Escalation where a normal user receives the privileges of other normal users [26]. Test scenarios for privilege escalation attacks involve storing and retrieving data from the database for different user roles and privileges. The user privileges are checked before granting access to create, receive, or delete any part of the information from the database. CVSS score parameters are shown in Table IV.

Examples of privilege escalation include URL manipulation [27]. In this attack, we manipulate the URL query strings & capture important account information. Information from the HTTP GET request passed as URL parameters are captured and modified in our test scripts. If the server accepts it, it indicates that the system is highly vulnerable, and the calculated CVSS score will be high. The developed test scenarios assume that the attacker has the lowest level of privilege i.e., patient level, and he tries to gain access to physician or admin level.

E. Test-case for Denial of Service attack

A common Availability attack, particularly for network-hosted systems, is the Denial of Service attack [28]. In a Denial of Service (DOS) attack, the service becomes unavailable due to capacity overload. Test scenarios were developed to simulate DOS attack, and the OpenMRS system is expected to block IPs after rate-limits on its API have been reached. The test scenarios verify that these rate-limits are in place and calculate the CVSS scores based on these criteria, as shown in Table V.

TABLE V. Parameters for Test Case 5 (Denial of Service)

Parameter Description	Parameter Value	Description
Attack Vector	Network(N),0.85	remotely exploitable
Attack Complexity	Low(L),0.77	no special conditions required
Privilege Required	None(N),0.85	privilege required is none
User Interaction	None(N),0.85	No user interaction required
Scope	Unchanged(U)	Impacted and vulnerable component same
Confidentiality	None(N),0.0	No loss of confidentiality
Integrity	None(N),0.0	No loss of integrity
Availability	High(H),0.56	High loss of Availability

The Gherkin files, as shown below, describe the security parameters on a deployed *demo OpenMRS* for multiple scenarios. For each test scenario, corresponding python functions for Given, When and Then are written. They are then integrated with the other codes in the project's main repository to check for security attacks at various time intervals.

Listing 1: Gherkin file for Brute Force Authentication

```
Feature: TestCase3
  As a user or attacker,
  I want to perform various attacks on the system,
  So that I can find out how vulnerable or secure our
  system is against those attacks.

  Scenario: Brute force attack with admin username
  and 10 incorrect attempts
    Given OpenMRS home page is displayed
    When attacker tries to login with admin and
    invalid "password"
    Then check after 10 incorrect attempts, the
    system suspends the account, blocks the account
    even if correct credentials are provided
```

Listing 2: Pytest code related to Listing 1 Gherkin

```
# Brute force attack TestCase 3 with known username
# Constants
OPENMRS_HOME = 'https://demo.openmrs.org/openmrs'
HOME_PAGE = OPENMRS_HOME + '/home.page'

@pytest.fixture
def browser():
  driver = Firefox()
  driver.implicitly_wait(10)
  yield driver
  driver.quit()

# Scenarios
@pytest_bdd.scenario('TestCase3.feature',
'Brute force attack with known username',
features_base_dir='', strict_gherkin=False)
def test_incorrect_password():
  pass

# Given Steps
@pytest_bdd.given('OpenMRS home page is displayed')
def demo_home(browser):
  browser.get(OPENMRS_HOME)

# When Steps
@pytest_bdd.when('attacker tries to login with admin
and invalid "password"')

def random(stringLength=10):
  letters = string.ascii_lowercase
  text = ""
  for i in range(stringLength):
    text = text.join(random.choice(letters))
  return text

def CVSS_Score(AV, AC, PR, UI, A, C, I):
  ISS = 1-((1-C) * (1-I) * (1-A))
```

```
  Impact = 6.42 * ISS
  Exploitability = 8.22*AV*AC*PR*UI

  if Impact<= 0:
    Base_score = 0
  else:
    Base_score = min((Impact + Exploitability), 10)
    Base_score1 = round(Base_score, 2)
    Base_score2 = round(Base_score1, 1)
  print(Base_score2)
  return Base_score2

def login_username(browser):
  pas=[random(10),random(10),random(10),random(10),
  random(10),random(10),random(10),random(10),
  random(10),random(10),'Admin123']

  s_input = browser.find_element_by_id('username')
  s_input.send_keys('admin')
  s_input = browser.find_element_by_id('password')
  s_input.send_keys((pas[i]) + Keys.TAB)
  login_button = browser.find_element_by_id('loginButton')
  login_button.click()
  if browser.current_url == HOME_PAGE:
    CVSS_Score(AV=0.85,AC=0.44,PR=0.85,UI=0.85, A=0, C=0.22,
    I=0.22)
  else:
    CVSS_Score(AV=0.85,AC=0.44,PR=0.85,UI=0.85,A=0, C=0, I=0)

# Then Steps
@pytest_bdd.then('check after 10 incorrect attempts, the
system suspends the account, blocks the account even if
correct credentials are provided"')
def login_results(browser):
   assert browser.current_url != OPENMRS_HOME +
            '/referenceapplication/home.page'
```

IV. Result

We were able to implement the practice of Continuous Security with the help of the Pytest BDD framework. The unit testing examples shown can be used by other projects and software systems to implement automated test scenarios to implement Continuous Security in their projects. The higher the CVSS score, the poorer the security of the system against those attacks. Every time a security enhancement is made in the system, the test must be rerun, and a new CVSS score should be calculated. If the CVSS score has decreased, it implies that the security enhancements have worked and have made the system less vulnerable to such security attacks. We have released two versions of the software system through this process, which has resulted in lowering the CVSS score for every release. However, this is a continuous process, and with every code change, developers are notified if their code change is introducing any new vulnerabilities or not. Our process has also shown that developers will release new versions of the software system, not just on feature completeness, but rather also achieving statistically significant gain in the CVSS score between releases. For each Test Scenario discussed in the methodology section, the corresponding pytest test script is created and the CVSS score is computed for all the Test Scenarios. The effect of the parameters for different test scenarios is analyzed. Table VI below shows the CVSS score for all the Test Scenarios discussed earlier. To reflect on the results of Table VI, the Attack Vector (AV) and Attack Complexity (AC) have largely determined the weakest link of the system, as shown by the larger score in most of the test cases.

TABLE VI. CVSS Scores on Test Cases for demo OpenMRS v2.10.0

CVSS Score for Test Case 1								
User	AV	AC	PR	UI	C	I	A	Score
Admin	0.85	0.77	0.85	0.85	0.56	0.56	0.0	9.06
Doctor	0.85	0.77	0.85	0.85	0.22	0.22	0.0	6.40
Patient	0.85	0.77	0.85	0.85	0.22	0.0	0.0	5.30
CVSS Score for Test Case 2								
User	AV	AC	PR	UI	C	I	A	Score
Admin	0.55	0.77	0.85	0.85	0.56	0.56	0.0	7.69
Doctor	0.55	0.77	0.85	0.85	0.22	0.22	0.0	5.03
Patient	0.55	0.77	0.85	0.85	0.22	0.0	0.0	3.93
CVSS Score for Test Case 3								
User	AV	AC	PR	UI	C	I	A	Score
Admin	0.85	0.44	0.85	0.85	0.56	0.56	0.0	7.40
Doctor	0.85	0.44	0.85	0.85	0.22	0.22	0.0	4.74
Patient	0.85	0.44	0.85	0.85	0.22	0.0	0.0	3.63
CVSS Score for Test Case 4								
User	AV	AC	PR	UI	C	I	A	Score
Admin	0.55	0.77	0.62	0.85	0.56	0.56	0.0	7.01
Doctor	0.55	0.77	0.62	0.85	0.22	0.22	0.0	4.35
CVSS Score for Test Case 5								
User	AV	AC	PR	UI	C	I	A	Score
Admin	0.85	0.77	0.85	0.85	0.0	0.0	0.56	7.48

It can be observed that the CVSS is highest for the first test scenario, indicating that the system is highly vulnerable to SQL injection attack. Some core modules also result in increased scores for Privilege Required (PR) and User Interaction (UI) parameters of the CVSS framework, according to our test cases. While these high scores are harmless in most cases in the current release, there might be future regression bugs that will be caught through our Continuous Security process. Confidentiality (C), Integrity (I) and Authenticity (A) of the system is robust and highlights that the EHR system is secure for patient privacy and security.

V. Discussion

Our research addresses the problems mentioned in some of the above studies [16] for the lack of adoption of continuous security frameworks. The use of Pytest BDD framework does not require anything complex. It only requires the working knowledge in python. All you need is a working desktop that has a command line interface, python package manager and an IDE for development. The other testing tools require the developer or tester to use a debugger or check the logs and detect where a certain value is coming from. Pytest helps to develop and write test cases in a way that gives you the ability to store all values inside the test cases and finally inform the user which value failed, and which value is asserted. Moreover, the use of BDD allows the test cases to be written in simple readable language. Readability in the test codes improves the software quality, communication and collaboration between product owners, test engineers, developers, and customers. Also, the use of CVSS vulnerability assessment was done for assessing the security of an EHR for achieving continuous security with the advantage of identifying and ensuring the security with every code commit. The CVSS system is easy to use, low cost and

yields easy to interpret results. Hence, the overall framework proved to be a robust approach to implement *Continuous Security* in the EHR software development.This study will allow the developers and testers to notice small discrepancies in their system, with the help of which they can catch problems early on and use effective solutions to avoid serious failures.

However, as far as the best of our search on GitHub (for code), Google searches (for open-source projects), and Scopus (for published literature), we have not seen other descriptions of hands-on implementations of *Continuous Security*. Maybe, many organizations and closed-source software and software vendors have deployed such test scripts, but these have not been popularized. Beyond digital health projects, we see an opportunity to implement *Continuous Security* whenever a commit is made in the project codebase, and the testing will calculate the CVSS score. This type of software assurance process should be the norm, and we hope more open-source projects adopt this approach.

VI. Conclusion & Future Work

With the advancements in information technology, the risks of security attacks have escalated, causing substantial monetary losses. There is a need for continuous and integration security testing for EHRs. The CVSS Score provides a meaningful insight with the identification of the severity of risks and vulnerabilities and implementing appropriate security measures against the risks. The use of BDD allows individuals with limited programming experience to participate actively in achieving the testing requirements and monitor the security aspects of the software.

The design of the study was limited to only one open-source EHR system, OpenMRS. This is a threat to external validity. Not all the vulnerabilities associated with the EHR were identified by our study, which could be extended further for finding out various other vulnerabilities associated with the system and calculating the corresponding CVSS scores.

VII. Acknowledgement

The U.S. National Science Foundation supported this work under grant OAC-1839746. This work was made possible through a research allocation on the JetStream [29] public cloud infrastructure and XSEDE resources.

References

[1] J. G. Ronquillo, J. Erik Winterholler, K. Cwikla, R. Szymanski, and C. Levy, "Health it, hacking, and cybersecurity: national trends in data breaches of protected health information," *JAMIA Open*, vol. 1, no. 1, pp. 15–19, 2018.

[2] P. C. Reich, "Healthcare: A critical information infrastructure," 2019.

[3] R. Clarke and T. Youngstein, "Cyberattack on britain's national health service—a wake-up call for modern medicine," *N Engl J Med*, vol. 377, no. 5, pp. 409–11, 2017.

[4] A. Madhavi and S. Lincke, "Security risk assessment in electronic health record system," in *2018 IEEE Technology and Engineering Management Conference (TEMSCON)*. IEEE, 2018, pp. 1–4.

[5] M. Fowler and M. Foemmel, "Continuous integration," 2006.

[6] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[7] D. Ståhl and J. Bosch, "Modeling continuous integration practice differences in industry software development," *Journal of Systems and Software*, vol. 87, pp. 48–59, 2014.

[8] B. Fitzgerald and K.-J. Stol, "Continuous software engineering and beyond: trends and challenges," in *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, 2014, pp. 1–9.

[9] K. A. Torkura, M. I. Sukmana, and C. Meinel, "Integrating continuous security assessments in microservices and cloud native applications," in *Proceedings of the 10th International Conference on Utility and Cloud Computing*, 2017, pp. 171–180.

[10] R. P. Lippmann, J. Riordan, T. Yu, and K. Watson, "Continuous security metrics for prevalent network threats: introduction and first four metrics," Massachusetts Inst of Tech Lexington Lincoln Lab, Tech. Rep., 2012.

[11] M. Aslam, C. Gehrmann, and M. Björkman, "Continuous security evaluation and auditing of remote platforms by combining trusted computing and security automation techniques," in *Proceedings of the 6th International Conference on Security of Information and Networks*, 2013, pp. 136–143.

[12] C. Hanson, "The network certification description," Ph.D. dissertation, University of Colorado at Colorado Springs, 2017.

[13] C. Solis and X. Wang, "A study of the characteristics of behaviour driven development," in *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE, 2011, pp. 383–387.

[14] S. Purkayastha, J. W. Gichoya, and A. S. Addepally, "Implementation of a single sign-on system between practice, research and learning systems," *Applied clinical informatics*, vol. 26, no. 01, pp. 306–312, 2017.

[15] D. Saff and M. D. Ernst, "Reducing wasted development time via continuous testing," in *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* IEEE, 2003, pp. 281–292.

[16] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[17] P. Mell, K. Scarfone, and S. Romanosky, "Common vulnerability scoring system," *IEEE Security & Privacy*, vol. 4, no. 6, pp. 85–89, 2006.

[18] I. Stine, M. Rice, S. Dunlap, and J. Pecarina, "A cyber risk scoring system for medical devices," *International Journal of Critical Infrastructure Protection*, vol. 19, pp. 32–46, 2017.

[19] J. Voss, J. A. Garcia, W. C. Proctor, and R. T. Evans, "Automated system health and performance benchmarking platform: high performance computing test harness with jenkins," in *Proceedings of the HPC Systems Professionals Workshop*, 2017, pp. 1–8.

[20] D. Sale, *Testing Python: Applying Unit Testing, TDD, BDD and Acceptance Testing*. John Wiley & Sons, 2014.

[21] L. Allodi, S. Banescu, H. Femmer, and K. Beckers, "Identifying relevant information cues for vulnerability assessment using cvss," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, 2018, pp. 119–126.

[22] S. H. Houmb, V. N. Franqueira, and E. A. Engum, "Quantifying security risk level from cvss estimates of frequency and impact," *Journal of Systems and Software*, vol. 83, no. 9, pp. 1622–1634, 2010.

[23] N. Mendes, H. Madeira, and J. Duraes, "Security benchmarks for web serving systems," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*. IEEE, 2014, pp. 1–12.

[24] G. Pender-Bey, "The parkerian hexad," *Information Security Program at Lewis University*, 2019.

[25] K. Patel, "A survey on vulnerability assessment & penetration testing for secure communication," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*. IEEE, 2019, pp. 320–325.

[26] S. Nagpure and S. Kurkure, "Vulnerability assessment and penetration testing of web application," in *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA)*. IEEE, 2017, pp. 1–6.

[27] P. Sharma and B. Nagpal, "A study on url manipulation attack methods and their countermeasures," 2015.

[28] G. Carl, G. Kesidis, R. R. Brooks, and S. Rai, "Denial-of-service attack-detection techniques," *IEEE Internet computing*, vol. 10, no. 1, pp. 82–89, 2006.

[29] C. A. Stewart, T. M. Cockerill, I. Foster, D. Hancock, N. Merchant, E. Skidmore, D. Stanzione, J. Taylor, S. Tuecke, G. Turner *et al.*, "Jetstream: a self-provisioned, scalable science and engineering cloud environment," in *Proceedings of the 2015 XSEDE Conference: Scientific Advancements Enabled by Enhanced Cyberinfrastructure*, 2015, pp. 1–8.