Spring 5-1-2019

Compliance of Open Source EHR Applications with HIPAA and ONC Security and Privacy Requirements

Maryam Farhadi

Hisham Haddad

Hossain Shahriar

Compliance of Open Source EHR Applications with HIPAA and ONC Security and Privacy Requirements

Master's Thesis

by

Maryam Farhadi
MSCS Student
Department of Computer Science
Kennesaw State University, USA

Submitted in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

April 2019

DEDICATION

This thesis is dedicated to my family,

I love you and thank you for always being there for me.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisors Dr. Hisham Haddad and Dr. Hossain Shahriar for their support and encouragement through this entire process. I am very thankful for this experience.

I would also like to thank Dr. He, my thesis committee member, for her insightful comments and valuable suggestions.

ABSTRACT

Electronic Health Record (EHR) applications are digital versions of paper-based patient's health information. They are increasingly adopted to improved quality in healthcare, such as convenient access to histories of patient medication and clinic visits, easier follow up of patient treatment plans, and precise medical decision-making process. EHR applications are guided by measures of the Health Insurance Portability and Accountability Act (HIPAA) to ensure confidentiality, integrity, and availability. Furthermore, Office of the National Coordinator (ONC) for Health Information Technology certification criteria, which focuses on usability aspects of EHRs, have overlapping security and privacy requirements. A compliance checking approach attempts to identify whether or not an adopted EHR application meets the security and privacy criteria. Among others, static code analysis is a common approach to discover vulnerabilities in software applications. However, there is no study in the literature to know whether traditional static code analysis-based vulnerability discovered can assist in compliance checking of regulatory requirements of HIPAA and ONC.

The aim of this thesis is to (i) Identify key security requirements of HIPAA and ONC framework; (ii) Evaluate a number of open-source EHR applications for security vulnerabilities using open source scanner tools; (iii) Compare and contrast security vulnerabilities identified by scanner tools with HIPAA and ONC security requirements; and (iv) Propose an approach to secure personal health identifier information within EHR applications.

Table of Contents

LIST OF TABLES 6

CHAPTER 1: MOTIVATION AND PROBLEM STATEMENT 7

1.1 BACKGROUND 7
1.2 MOTIVATION AND PROBLEM STATEMENT 8
1.3 GOALS, RESEARCH METHODOLOGY AND CONTRIBUTIONS 9

CHAPTER 2: LITERATURE REVIEW 11

2.1 OVERVIEW 11
2.2 IMPLEMENTATION VULNERABILITIES 11
2.3 ACCESS CONTROL IN EHR 15
2.4 PRIVACY AND MONITORING OF EHR ACTIVITIES 16
3.1 OVERVIEW 19

CHAPTER 3: OPENEMR AND OPENCLINIC COMPLIANCE WITH HIPAA 19

3.2 COMMON WEB SECURITY VULNERABILITIES 19
3.3 HIPAA SECURITY REQUIREMENTS 20
3.4 EVALUATION OF OPENEMR AND OPENCLINIC 23
3.5 RESULTS 24

CHAPTER 4: COMPLIANCE CHECKING FOR ONC SECURITY AND PRIVACY REQUIREMENTS 30

4.1 OVERVIEW 30
4.2 COMPLIANCE WITH ONC CERTIFICATION 30
4.3 IMPROVING COMPLIANCE 32

CHAPTER 5: CONCLUSION 35

APPENDIX A: RIPS RESULTS FOR OPENEMR 36

APPENDIX B: PHP VULNHUNTER RESULTS FOR OPENEMR 50

APPENDIX C: RIPS RESULTS FOR OPENCLINIC 54

APPENDIX D: PHP VULNHUNTER RESULTS FOR OPENCLINIC 55

APPENDIX E: HIPAA SECURITY CHECKLIST 58

REFERENCES 62

List of Tables

Table 1: List of vulnerabilities missing in CCHI criteria... 13

Table 2: True and false positive vulnerabilities detected by static analysis and automated penetration testing 14

Table 3: HIPAA technical safeguard checklist 22

Table 4: Detected vulnerabilities found by RIPS and vulnhunter in OpenEMR and OpenClinic 24

Table 5: Code examples of vulnerable files in OpenEMR and OpenClinic 25

Table 6: EHR compliance with HIPAA technical requirements 28

Table 7: Privacy and security-related of ONC certification criteria 31

Table 8: EHR compliance with ONC certification criteria.. 32

Table 9: Code examples of vulnerable EHR files and the solution code............................... 33

CHAPTER 1

Motivation and Problem Statement

1.1 Background

Digital version of electronic health data improved the quality of healthcare due to easier follow-ups, lowering cost of patient care, enabling data track over time, and making more precise medical decisions. Three types of health records are defined: (i) Electronic Medical Records (EMRs) refer to digital version of paper-based clinical data. The clinical data, gathered by clinicians, include information that enables the clinicians to make better medical decisions; (ii) Electronic Health Records (EHRs) provide a more comprehensive view of the patient's overall well-being. It contains information collected by all clinicians engaged in the patient's healthcare. Therefore, information in EHRs can be shared among all involved providers; and (iii) Personal Health Records (PHRs) are EHRs that are controlled and accessed by the patients [1, 2].

The prevalence of healthcare data security breach can be observed both inside and outside USA. According to 2016 Data Breach Investigations Report (DBIR), there were 115 cases of data breach in North America during 2015. It included 32% privilege misuse, 22% miscellaneous errors, 19% stolen assets, 7% point of sale, 3% cyber-espionage, 3% crimeware, 3% web applications, and 11% other incidents. Healthcare is among the top industries vulnerable to physical theft and loss, miscellaneous errors, insider and privilege misuse, and others. Physical theft and loss is any occurrence where information or a device containing information is missing. Miscellaneous errors occur when accidental actions weaken a security attribute. Insider threats and privilege misuses refer to all unapproved or malicious use [8].

According to Verizon survey report, some of the reported healthcare data breaches in 2015 were as follows: In February, Anthem, a Blue Cross health insurance member-company, reported a data breach where 80 million patients were affected. In March, Premera, another Blue Cross member, reported a data breach affecting 11 million patients. In both cases, ThreatConnect [9] announced that Chinese threat Actor "Deep Panda" was probably the attacker. Partners HealthCare, CareFirst Blue Cross and Blue Shield, MetroHealth and Bellevue Hospital reported breaches in April of

2015. In June of the same year, US Office of Personnel Management (OPM) reported mega-breaches for health insurance. The US Department of Health and Human Services reported a breach in August 2015 [8].

In 2017, Emory Healthcare's appointment system was hacked compromising almost 80,000 patients Personal Health Information (PHI) data such as names, birthdates, internal medical record and appointment information. The appointment information was unencrypted, which opened the door for hackers to obtain plain text information. According to a report [7], this incident is the largest breach in 2017 in the US.

In order to maintain better healthcare, individuals must ensure that their personal health information is private and secure. Otherwise, if patients do not feel that their information remains confidential, they may not want to disclose their health information to healthcare providers, which could endanger the patient's life. Moreover, when a security breach occurs, it may lead to financial harm for healthcare providers or the patient alike [10].

1.2 Motivation and Problem Statement

As healthcare application becomes more and more evidence-based, storing health data is becoming more important. Weak health data protection may lead to identity theft, obtaining medical care at the expense of others, ordering expensive drugs for resale, and fraudulent insurance claims [3]. Moreover, healthcare data hacks may threaten patient's health due to the change of patient's medical history. For example, if health records do not contain a correct listing of allergies, the patient could suffer serious consequences or death due to wrong prescription [4].

Compare to banks and financial institutions, patients' data has less protection. Banks are mostly equipped with two-factor authentication while healthcare applications are not. Two-factor authentication is an extra protection which includes not only username and password, but also some unique information that only the user has, such as a physical token. Furthermore, unlike bank accounts that can be locked and changed for protection, it is completely impossible to get back the compromised and disclosed health data [5, 6].

In the healthcare sector, the HIPAA Meaningful Usage act requires that any data security breaches affecting 500 or more patients be reported to public through US Health and Human Service Office for Civil Rights' Breach Portal and the affected healthcare provider must take appropriate steps within a certain time limit, otherwise, faces further penalties. Thus, PHI leakage not only brings reputation problem for healthcare providers, but also affects patient’s privacy and well-being.

Many security vulnerabilities remain undetected for long time [15]. Fixing security vulnerabilities becomes more expensive when they are discovered later. Moreover, many security-related mistakes are repeated all the time. Static analysis is a method that directly examines the code of a program without executing the program. It can detect common vulnerabilities before releasing the program. Since manual static analysis takes a long time to be performed, static analysis tools are used to speed up the process of evaluating programs. Static analysis tools examine the text of a program statically, without attempting to execute it. Based on our research, there is no work in the literature that address whether vulnerability discovered using traditional static code analysis can assist in compliance checking of regulatory requirements of HIPAA and ONC.

1.3 Goals, Research Methodology and Contributions

Given that large scale PHI data security breach occurs in real world, it is imperative to look at the implemented EHR and check them if they complied with HIPAA safeguards, particularly technical safeguards. However, there is no single tool available in the market that can test an EHR application. This thesis brings two prong approaches to address the problem. First, examine EHR applications with available code analysis tools. In particular, static analysis tools that are intended to reveal implementation vulnerabilities related to traditional computer security breaches including data leakage, alteration or deletion. Second, analyze EHR applications at the source-code level to secure and harden the code to comply with HIPAA security requirements.

The specific goals of this thesis are to (i) Identify key security requirements of HIPAA framework; (ii) Evaluate two open source EHR applications for security vulnerabilities using open source scanner tools; (iii) Compare and contrast security vulnerabilities identified by scanner tools with HIPAA and ONC security requirements; and (iv) Propose an approach to secure personal health identifier information within the applications.

The research methodology involves the following activities:

1. Conduct literature search on existing EHR security and privacy vulnerabilities and their prevention technique, on HIPAA Security Requirements, and EHR systems that have been tested for HIPAA requirements.
2. Study and select open source EHR applications and run them to generate dummy PHI data, followed by applying open source static code analysis tools to identify security vulnerabilities and solutions to address them.
3. Analyze vulnerabilities and fix them within course code.
4. Disseminate the work results in the form of publications in related venues.

Our specific contributions to this field of study include:

1. An in-depth literature survey on state-of-the art approaches for addressing EHR application security and privacy [50].
2. Analysis of two EHR applications (e.g., OpenEMR, OpenClinic) for security vulnerabilities with static analysis tools (RIPS, VulnHunter), and checking the results for HIPAA technical security policy violations [51].
3. Improving the compliance of EHR applications (e.g., OpenEMR, OpenClinic) using static analysis and secure coding [52, 53].

CHAPTER 2

Literature Review

2.1 Overview

This Chapter presents a literature review of related work on privacy and security vulnerabilities in EHRs. In particular, we look at literature works for compliance with HIPAA and ONC requirements using four different perspectives: security vulnerability in EHR, access control, privacy and monitoring, and the gaps between HIPAA requirements and breach notification. We also map some of the existing vulnerabilities to HIPAA and ONC security rules.

2.2 Implementation Vulnerabilities

Since EHR applications are traditional web applications implemented using various languages (e.g., PHP, JSP) and deployed with databases (e.g., MySQL, MSSQL, Oracle) in well-known servers (e.g., Apache), they may be vulnerable in their implementations. Attackers may exploit the vulnerabilities by providing malicious inputs and compromising the data processed and stored by EHR applications. A number of literature works have explored the magnitude of vulnerabilities present in popular and open source EHRs and checked whether EHR implementations are complying with HIPAA related acts.

Smith et al. [4] empirically evaluated the ability of the Certification Commission of Healthcare IT (CCHIT) to identify a range of vulnerability types. CCHIT focuses on required functional capabilities in EHR applications, such as ambulatory (with prefix AM), ambulatory interoperability, and security (SC).

In a prior study, the authors discussed more than 400 vulnerabilities they discovered using automated security testing tools in OpenEMR [1]. In their current work, they tried to observe the consequences of the vulnerabilities rather than finding all vulnerabilities of a particular type. The authors exploited a range of common vulnerabilities at the code-level and design-level in EHR applications. Code-level refers to implementation vulnerabilities and design-level refers to design flaws. Some of the consequences of these exploits were denial of service, users' login information exposure, and editing health records by any users.

A team of instructed attackers was created to target the two EHR applications: OpenEMR and ProprietaryMed. The attackers' focus was on misuse cases of the CCHIT criteria not the overall security of the applications. Misuse cases are defined as actions that are not allowed in the system and can help developers to think like an attacker. The test attack environment included OpenEMR and ProprietaryMed applications, hacking scripts with additional server, and the researchers' computer with WebScarab and Firebug. WebScarab, a Java-based application, was used as a proxy to execute testing attacks and record any traffic between the computers and the test servers. Firebug was used as JavaScript debugger to monitor the attacks. Firebug is a web development plug-in integrates with Firefox and enable users to edit HTML, JavaScript, and Cascading Style Sheets. Firebug is also able to executes any script live. Therefore, the researchers did not need an additional webpage for storing attacks.

In code vulnerable situations, the following problems occurred while none of them had previously been exposed by CCHIT test script: SQL injection, cross-site scripting, session hijacking, phishing, PDF exploits, denial of service (file uploads), and authorization failure. Table 1 shows the misuse case(s) of vulnerabilities that has not been addressed by CCHIT. Results show that CCHIT certification process has two failures: First, when an application meets the security requirements, CCHIT test scripts do not test the application for implementation vulnerabilities. Second, some security items about patient's health records are not considered at all. It has been suggested that misuse cases are added to the manual test script to simulate the attacks. Moreover, the test scripts can be more comprehensive by launching various attacks on the host application. The manual test scripts should also include the most current list of threats.

Austin et al. [11] discussed insufficient vulnerability discovery techniques from EHR applications. Four discovery techniques were applied to EHR applications to understand when to use each type of discovery techniques. The evaluated EHR applications were OpenEMR and Tolven eCHR. The techniques were systematic and exploratory manual penetration, automated penetration, and static analysis. Penetration testing looks at the security of an application from a user perspective and examines the functionality of an application. In manual penetration testing, no automated tools are used. Exploratory manual penetration is testing an application based on tester's prior experience and it has no test plan. Systematic penetration testing is a test based on a predefined test plan. Automated penetration testing uses automated tools to speed up the process of scanning. Static analysis testing examines the code without executing the program. It can be examination of the source code, the machine code, or the object code.

Table 1: List of vulnerabilities missing in CCHIT criteria

Implementation Vulnerabilities	
Vulnerability	**Misuse Case(s)**
SQL injection	Attacker obtains every user's username and password.
Cross-Site Scripting	Attacker causes a denial of service by rendering the home page to be blank for all future users. Attacker injects scripts that execute additional malicious code.
Session Hijacking	Attacker spoofs another user's identity. Attacker obtains unauthorized access to the system.
Phishing	Attacker obtains the victim's username and password.
PDF Exploits	Attacker executes applications on the client's computer. Attacker executes embedded applications.
Denial of Service: File Uploads	Attacker renders the web server slow or unresponsive.
Authorization Failure	Attacker creates a new user account with any access privileges the attacker desires.
Design Flaws	
Repudiation	Attacker modifies data in an untraceable fashion thus making fraud an unperceivable event to the EHR.
Lack of Authorization Control	Attacker views patient's confidential health records and personal identification information.

The authors [11] first collected the vulnerabilities detected by each technique, then classified vulnerabilities based on being true or false positive (False positive: mistakenly label code as contain fault. True positive: when faults are correctly identified). The techniques that generated false positives were static analysis and automated penetration testing. The developers need to manually examine each potential false report to recognize if they are false positives. Table 2 shows the true and false positive vulnerabilities identified by these two techniques.

Table 2: True and false positive vulnerabilities detected by static analysis and automated penetration testing

Static Analysis			
EHR Application	True Positive	False Positive	False Positive Rate
Tovlen eCHR	50	2265	98%
OpenEMR	1321	3715	74%
Automated Penetration			
Tovlen eCHR	22	15	40%
OpenEMR	710	25	3%

Some of the detected vulnerabilities were SQL injection, cross-site scripting, system information leak, hidden fields, path manipulation, dangerous function, no HTTP Only attribute, dangerous file inclusion, file upload abuse, and header manipulation. The authors classified vulnerabilities as either implementation vulnerabilities or design flaws. They empirically proved that no single technique is sufficient for discovering every type of vulnerability and also there is almost no vulnerabilities that can be detected by several techniques. Results showed that systematic manual penetration is more efficient than exploratory manual penetration in terms of detecting vulnerabilities. Systematic manual penetration was effective at finding design flaws. Static analysis detected the largest number of vulnerabilities. Automated penetration was the fastest technique, while static analysis, systematic penetration, and manual penetration were ranked in the next order respectively.

It is suggested that in case of time constraint, automated penetration is used to detect implementation vulnerabilities, and systematic manual penetration for discovering design flaws. The study has the following limitations: The selected tools for representing static analysis and automated penetration are not a representative of other tools. The authors used just one tool for measuring each detection technique, while other tools might detect other types of vulnerabilities. The examined EHR applications are not representative of all other software. The classification of errors (true positive and false positive) were time consuming and error prone. Human errors can cause vulnerabilities to be neglected.

2.3 Access Control in EHR

In EHR applications, access control is one of the necessary security requirements in terms of protecting patient information from being compromised [12]. Below is a highlight of some of the studies that has been done in this area.

Helms et al. [12] claimed that there has been little effort to evaluate access control vulnerabilities in EHR applications. Four EHR applications were evaluated based on 25 criteria related to access control. The evaluated EHRs were OpenEMR, OpenMRS, iTrust, and Tolven. The criteria were retrieved from HIPAA, Certification Commission for Health Information Technology (CCHIT) Criteria, National Institute for Standards and Technology (NIST) Meaningful Use, and NIST Flat role-based access control (RBAC). CCHIT criteria was met by iTrust but other applications were configuration dependent. OpenEMR and openMRS are able to create super user roles, which make them target of insider attacks. Among all evaluated applications, none of them addressed access control during the emergency time. Moreover, the EHR applications failed to allow creating roles with separation of duty. Separation of duty prevents a task to be done by just a single user. In addition, none of the certification criteria covered the implementation standards.

Oladimeji et al. [21] discussed that traditional access controls are insufficient in ubiquitous applications. They proposed a goal-oriented and policy-driven framework to mitigate the security and privacy risks of ubiquitous applications in healthcare domain. The framework captures application's security and privacy requirements and decreases the threats against those requirements. In the proposed framework, these items are modeled: (1) security and privacy objectives, (2) threats against those objectives, (3) mitigation strategies in the form of safeguards or countermeasures. The authors used emergency response scenario to show the efficiency of the framework.

It is mentioned that issues such as untimely arrival of ambulance is a real problem that could happened as a result of verbal misinformation, GPS misleading, or imprecise policies guiding. Introducing some automated mediation may lead to significant improvement. The eHealth security and privacy issues are described in 4 categories: confidentiality, privacy, integrity, and availability. The authors [21] mentioned that there are no universal solutions to these issues that fit all ubicomp applications. Therefore, each ubicomp application needs context-sensitive evaluation of what threats need to be addressed.

The work of Tuikka et al. [24] is based on systematic literature review of previous studies about patients’ involvement in EHR applications. Based on this work, patients' opinion has not been properly considered in EHR development. It is suggested that ethical values be considered in designing EHRs, and patients’ access to all their records and even able to add some information to them. The paper concluded that the best representatives for the patients’ needs are the patients themselves not the organizations or advocates.

In 2016, Grunwel et al. [19] discussed the delegation of access in EHR applications and proposed an Information Accountability Framework (IAF) to balance the requirements of both healthcare professionals and patients in EHR applications. In the framework, patients have explicit control over who access and use their information and set usage policies. The IAF framework ensures that the right information is available to the right person at the right time. To operationalize the framework, it needs to provide for a diverse range of users and use cases. For example, the requirement to delegate access to another user on one’s behalf.

2.4 Privacy and Monitoring of EHR Activities

Privacy protections apply to patient's "individually identifiable health information" [31]. As medical records are digitized, patient privacy becomes a more challenging issue [32]. A number of studies have discussed the patient privacy and the required monitoring over patient records.
In order to improve accountability of EHR applications, Mashima et al. [25] presented a patient centric monitoring system that monitors all the updates and usage of health information stored in EHR/PHR repository. The proposed system uses cryptographic primitives, and allows patients to have control over their health record accessibility. However, in this system the monitoring agent is assumed trusted.

King et al. [22] discussed that in EHR applications, viewing protected data is often not monitored. Therefore, unauthorized views of PHI remain undetected. They proposed a set of principles that should be considered during developing logging mechanisms. They monitored the current state of logging mechanisms to capture and prove user's activities in the application. The authors supplemented the expected results of existing functional black- box test cases to include log output. They used an existing black-box test suite provided by the National Institute of Standards and Technology (NIST). They selected 10 certification criteria from the NIST black- box testing including demographics, medication list, medication allergy list, etc. The authors executed the 30 test cases on EHR applications. 67.8% of applicable executed test cases failed. Four of failed cases was related to viewing of critical data, showing that users may view protected information without being captured.

In order to meet HIPAA's privacy requirements, Reinsmidt et al. [23] proposed an approach that provides a secure connected mobile system in a mobile cloud environment. The connection between mobile systems takes place using authentication and encryption. The protocol execution includes encryption, decryption, and key generation time. After a mobile device opens a socket with the listening server, the server responds with its public part of the DH exchange. The mobile device hashes the results with SHA-256 to calculate the symmetric encryption/decryption key. This key is used in the advanced encryption standard (AES).

Kingsford et al. [26] proposed a mathematical framework to improve the preservation of patient's privacy in EMR applications during the collection of patient health data for analysis. The authors used an identity based encryption (IBE) protocol. In the proposed framework, patient's identity is delinked from the health data before submitting to health workers for analysis. Health data is encrypted before submitting. The administrator then decrypts the submitted data. Patient’s identity is delinked from submitted data in this stage. The administrator checks that only the health data (not the identity of the patient) is sent to health worker for analysis. Therefore, the identity of the patient will not be disclosed and PU’s privacy is preserved.

Gaps between security policies and real breaches always exist in healthcare. Policies are often stated in an ambiguous manner [27, 28]. Therefore, in reality not all the breaches are addressed by policies. To measure the breach coverage percentage by HIPAA security policies, Kafali et al. [30] proposed a semantic reasoning framework to identify gaps between HIPAA policy and security breaches. The work revealed that only 65% of security breaches are covered by HIPAA policy rules. Moreover, HIPAA security policy is more successful in covering malicious misuse than accidental misuse.

Compared to previous literature works, our work contributed to checking the compliance of EHR applications with HIPAA (technical security requirements) and ONC (focused on security and privacy) criteria. In particular, none of the work has explored the issue of whether there is any link between potential security vulnerabilities and not complying with HIPAA and ONC.

CHAPTER 3

OpenEMR and OpenClinic Compliance with HIPAA

3.1 Overview

EHR applications are deployed over the web, leading to potential common security vulnerabilities such as SQL Injection, cross-site scripting, file inclusion, HTTP response splitting, and flow control alteration. In this chapter, we start with a review of these common security vulnerabilities in EHR applications. Then focus on the compliance of selected EHR applications with HIPAA technical requirements. The selected code analysis tools, named RIPS [16] and PHP VulnHunter [47], are used to examine the code of OpenEMR and OpenClinic [33]. The results obtained from the tools are then manually inspected for accuracy to form a basis for compliance with HIPAA technical requirements due to traditional web security vulnerabilities.

3.2 Common Web Security Vulnerabilities

SQL Injection: SQL injection is a code injection attack that occurs when an application does not sanitize untrusted input properly. Therefore, an attacker can inject reserved words into input fields, executes malicious SQL statements, and change the logical structures of SQL statements [4, 18].

Cross-Site Scripting: In a cross-site scripting attack, an attacker injects malicious code into the trusted web application that does not properly validate the user input. Therefore, a victim can run the malicious code into the browser. Cross-site scripting is a threat for users, not the application itself [4, 29].

Reflection Injection Attack: Reflection is the computer ability to inspect itself and describe the properties, methods and types of objects that the user is working with. In reflection injection attack, tainted data is used as a function name which may lead to execution of arbitrary functions and unexpected behavior of the application. Reflection injection may also lead to a specific code injection [34, 35].

File Inclusion: This vulnerability occurs due to poor input validation when an attacker is allowed to exploit external file inclusion functionality that dynamically includes local or external files. It will lead to the execution of malicious code or unauthorized access to sensitive files [20, 36].

HTTP Response Splitting: In HTTP response splitting attack, malicious data is embedded in HTTP response headers. HTTP response is split into two responses instead of one. This attack can lead to other vulnerabilities including cross-site scripting [37, 38].

Flow Control: Flow control attacks occur as a result of injecting malicious user input into the program counter. Flow control attacks are the subversion of the program execution due to tampering with program code. This vulnerability may result in execution of unintended code [39, 40].

Protocol Injection: It occurs when attacker change the connection handling parameters. User tainted data may be used when selecting those parameters [16].

File Disclosure: In File Disclosure vulnerability, an attacker can read local files. User tainted data is used when creating the file name that is supposed to get opened. Therefore, the attacker can read the source code and other files which might lead to other attacks [16].

File Manipulation: An attacker can inject code into a file or write to arbitrary files. User tainted data is used when creating the file name that is supposed to get opened or when creating the string that will be written to the file. An attacker can try to write arbitrary PHP code in a PHP file allowing to fully compromising the server [16].

3.3 HIPAA Security Requirements

Although EHRs resulted in better care, concerns of security and privacy breach always exist among digital formats. HIPAA (Health Insurance Portability and Accountability Act) was established in 1996 to protect health care coverage for individuals with lower income [41]. It provides federal protections for patient health information [14] by specifying measures to ensure EHR confidentiality, integrity, and availability [13]. HIPAA security requirements are divided into three types: Administrative, Physical, and Technical (See Appendix E). The administrative safeguards are described as policies and procedures designed to manage the selection, development, implementation, and maintenance of security measures. The physical safeguards are physical measures, policies, and procedures to protect an equipment, from natural and environmental hazards, and unauthorized intrusion. Finally, the technical safeguards are the technology and the policy-related requirements that protect electronic protected health information (EPHI) and control access to it [54, 55, 56].

Table 3 shows a set of HIPAA technical requirements. The first column of the table refers to applicable sections of HIPAA law on security requirements. For example, Section 164.312 specifies all technical safeguards that a covered entity must comply with. These include implementing access control through unique user identification, emergency access to PHI procedure, automatic logoff after a time of inactivity, encrypt and decrypt PHI, audit the access and usage of PHI, detection of improper access and alteration of PHI, verification of the integrity of received PHI electronically, and authentication (verify that an individual seeking access to PHI is recognized based on provided credential information) [42].

The second column shows high-level requirements to ensure safeguards. The requirement can be referring to other guidelines. The third column (status) refers whether a safeguard is required or addressable. The required rules are mandatory for all EHR applications and not implementing them leads fines and penalties [43]. The addressable rules are optional features that should be implemented in EHR so that PHI remains secured.

In this research, we focus on issues related to Technical Safeguards. If technical safeguards are complied with in an implemented EHR, then it enables not only meeting some of the other types of safeguards (e.g., providing tools and applications to check and monitor administrative security policies), but also can prevent unwanted incidents (due to not complying physical safeguards). For example, if a laptop having an EHR application gets lost or stolen, it would be very difficult to hack PHI data if authentication mechanism is established and encryption of data is applied into databases.

Table 3: HIPAA technical safeguard checklist

Security Rule Reference	Technical Safeguard	Status
164.312(a)(1) (Access Control)	Access Controls: Implement technical policies and procedures for electronic information systems that maintain EPHI to allow access only to those persons or software programs that have been granted access rights as specified in Sec. 164.308(a)(4).	■
164.312(a)(2)(i) (Identify and track user identity)	Assign a unique name and/or number for identifying and tracking user identity.	REQUIRED
164.312(a)(2)(ii) (Emergency access)	Establish (and implement as needed) procedures for obtaining necessary EPHI during and emergency.	REQUIRED
164.312(a)(2)(iii) (Automatic Log-Off)	Implement procedures that terminate an electronic session after a predetermined time of inactivity.	ADDRESSABLE
164.312(a)(2)(iv) (Encryption)	Implement a mechanism to encrypt and decrypt EPHI.	ADDRESSABLE
164.312(b) (Audit Controls)	Implement Audit Controls, hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use EPHI.	REQUIRED
164.312(c)(1) (Integrity)	Integrity: Implement policies and procedures to protect EPHI from improper alteration or destruction.	■
164.312(c)(2) (Integrity)	Implement electronic mechanisms to corroborate that EPHI has not been altered or destroyed in an unauthorized manner.	ADDRESSABLE
164.312(d) (Authentication)	Implement Person or Entity Authentication procedures to verify that a person or entity seeking access EPHI is the one claimed.	REQUIRED
164.312(e)(1) (Transmission Security)	Transmission Security: Implement technical security measures to guard against unauthorized access to EPHI that is being transmitted over an electronic communications network.	■
164.312(e)(2)(i) (Transmission Security)	Implement security measures to ensure that electronically transmitted EPHI is not improperly modified without detection until disposed of.	ADDRESSABLE
164.312(e)(2)(ii) (Encryption)	Implement a mechanism to encrypt EPHI whenever deemed appropriate.	ADDRESSABLE

3.4 Evaluation of OpenEMR and OpenClinic

To uncover potential web security vulnerabilities, we have used RIPS and PHP VulnHunter [16] static analysis tools on OpenEMR and OpenClinic [1], two open source EHR applications currently being adopted and used in the real world. We used Windows operating system to install the tools and EHR applications, and deployed Apache server and MySQL database.

Note static analysis tools do not run applications, instead scan source of the applications for known pattern of vulnerabilities. The tools are restricted to their application based on the implementation languages of the EHR applications (which was PHP in our study). Dynamic analysis tools have the capability of running applications with inputs, and inspect various states of applications to identify anomalies or security bugs. In our study, we found few to no support for dynamic analysis tool support at this time for PHP-based web applications. Given that our focus was using static analysis tools.

OpenEMR is a PHP-based application that uses Apache as the web server and MySQL as the database server. OpenEMR is under the GNU Public License (GPL). It can be installed on UNIX, Microsoft, and Mac OS X platforms. It contains many essential features for clinical practices such as feeding data of patient (e.g., biographic data, diagnostic results, medication history) as EHR, disease management, scheduling, and electronic billing. OpenEMR is one of the most popular free EHR systems with over 7000 downloads per month. It has been downloaded more than 500,000 times since March 2005 [44, 45, 46]. In this work, we scanned all 5594 files in OpenEMR.

OpenClinic is a platform independent, PHP-based EHR application which has been mainly used in private clinics and private doctors. It requires MySQL and a web server for executing PHP code (like Apache). OpenClinic is under GNU General Public License (GPL) [33]. In this project, we scanned all 170 files in OpenClinic.

RIPS is a tool for automated identification of vulnerabilities in PHP applications. In open-source version of RIPS, PHP code is tokenized and transformed into a program model for scanning. RIPS then detects vulnerable functions that an attacker can provide with malicious inputs. RIPS detects a number of vulnerability types including cross-site scripting, SQL injection, and local file inclusion. RIPS is capable of identifying a large number of vulnerabilities, including Code Execution, Command Execution, Cross-Site Scripting, Header Injection, File Disclosure, File Inclusion, File Manipulation, LDAP Injection, SQL Injection, Unserialize with POP, and XPath Injection [16, 17].

PHP VulnHunter is a static analysis tool which scans php vulnerabilities automatically. In fact, it uses a combination of static and dynamic analysis to automatically map the target application. It scans a large number of vulnerabilities in PHP web applications. PHP Vulnerability Hunter can detect the following classes of vulnerabilities: Arbitrary command execution, Arbitrary file read/write/change/rename/delete, Local file inclusion, Arbitrary PHP execution, SQL injection, User controlled function invocation, User controlled class instantiation, Reflected cross-site scripting (XSS), Open redirect, Full path disclosure [47, 48].

3.5 Results

The results of applying static analysis tools to OpenEMR and OpenClinic are shown in Tables 4, 5. Table 4 shows identified vulnerabilities by RIPS and PHP VulnHunter, while table 5 provides some code examples of the detected vulnerable files. Finally in table 6, we show the compliance of the two evaluated EHRs (OpenEMR and OpenClinic) with the HIPAA technical safeguards based on our observation. OpenEMR has 5000+ source files, and it took more than 3 hours of time to be scanned by RIPS and VulnHunter. In contrast, OpenClinic has much less source files and took about half an hour of time by the tools.

TABLE 4: Detected vulnerabilities in OpenEMR and OpenClinic

Vulnerability	**OpenEMR**		**OpenClinic**	
	RIPS	**VulnHunter**	**RIPS**	**VulnHunter**
File Inclusion	✓	✓	-	-
Cross-Site Scripting	✓	✓	✓	✓
SQL Injection	-	-	-	✓
HTTP Response Splitting	✓	-	✓	-
Reflection Injection	✓	-	-	-

Flow Control	✓	-	-	-
Protocol Injection	✓	-	-	-
File Disclosure	✓	✓	✓	-
File Manipulation	✓	-	✓	-

According to Table 4, Cross-Site Scripting was detected in both applications by both tools, while SQL Injection, Reflection Injection, Flow Control, and Protocol Injection were least detected vulnerabilities.

In Table 5, we show code example for vulnerabilities and describe how the presented code can lead to the associated vulnerability.

Table 5: Code examples of vulnerable files in OpenEMR and OpenClinic

Vulnerability	Code Example	Description
File Inclusion	require_once "sites/$site_id/sqlconf.php"; $site_id = 'default'; $site_id = $_SERVER['HTTP_HOST'];	The function require_once and $_SERVER together can cause File Inclusion vulnerability.
Cross-Site Scripting	echo $controller->act($_GET); $_GET = undomagicquotes ($_GET);	The function echo and $_GET together can cause Cross-Site Scripting vulnerability.
SQL Injection	mysql_query('USE ' . $_POST['dbName']) or die(sprintf(_("Instruction: %s Error: %s"), $sql, mysql_error()));	The function mysql_query and $_ POST together can cause SQL Injection vulnerability.
HTTP Response Splitting	header("Location: interface/login/login.php?site=$site_id"); $site_id = 'default'; $site_id = $_SERVER['HTTP_HOST'];	The function header and $_SERVER together can cause HTTP Response Splitting vulnerability.

Reflection Injection	call_user_func_array (array(&$obj, $func), array($var, $_POST));
	
	function populate_object (&$job);
	
	$func = "set_" . $varname;
	
	$varname = preg_replace("/[^A-Za-z0-9_]/", "", $varname);
	
	foreach ($_POST as $varname=>$var)
		The function call_user_func_array and $_POST together can cause Reflection Injection vulnerability.

Flow Control	extract ($_GET, EXTR_SKIP); $_GET = undomagicquotes ($_GET);	The function extract and $_GET together can cause Possible Flow Control vulnerability.
Protocol Injection	fsockopen fsockopen(($this->ssl"ssl://" : "") . $ip, $port, $errno, $error, $this->timeout) : $ip = gethostbyname($domain), $domain)) ⇓ Function connecttohost($domain, $port, $resolve_message)	The function fsockopen and $_GET together can cause Protocol Injection vulnerability.
File Disclosure	fread $filetext = fread($tmpfile, $_FILES['file']['size'][$key]); $tmpfile = fopen($_FILES['file']['tmp_name'][$key], "r");	The function fread and $_FILES together can cause File Disclosure vulnerability.
File Manipulation	chmod chmod($destinationFile, 0644); $destinationFile = $destinationDir . '/' . $destinationName; function upload(&$file, $destinationDir = "", $destinationName = "", $secure = true) $destinationName = $file['name']; function upload(&$file, $destinationDir = "", $destinationName = "", $secure = true)	The function chmod and $_GET together can cause File Manipulation vulnerability.

File inclusion may lead to unauthorized access and consequently alternation of PHI [20, 36]. This vulnerability violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

In cross-site scripting, an attacker can cause a denial of service by rendering the home page to be blank for all future users or inject scripts that execute additional malicious code [29]. Therefore, authorized users may fail to access the PHI during emergency time. This vulnerability violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

In SQL injection attack, an attacker injects reserved words into input fields, executes malicious SQL statements, and change the logical structures of SQL statements [18]. This vulnerability violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

HTTP response splitting leads to other vulnerabilities including cross-site scripting, which may disrupt authorized access to the system and cause denial of service [37, 38]. This vulnerability violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

Reflection injection may allow an attacker to gain access to the PHI and result in specific code injection [34, 35]. This violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

Flow control attacks may result in execution of unintended code, which could result in a denial of service [39, 40]. This can cause problem especially during emergency time. Flow control attacks violate HIPAA rule for access during emergency time (Security rule 164.312(a)(2)(ii)).

In protocol injection vulnerability, user tainted data may be used when selecting and changing the connection handling parameters [16]. This violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

In file disclosure vulnerability, an attacker can read files. User tainted data is used when creating the file name that is supposed to be opened. Therefore, the attacker can read the source code and other files, which might lead to other attacks [16]. This vulnerability violates HIPAA rules for

Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

In file manipulation vulnerability, an attacker can inject code into a file or write to arbitrary files [16]. This violates HIPAA rules for Access Control (Security rule 164.312(a)(1)) and Integrity of the EHR application (Security rule 164.312(c)(1)).

OpenEMR also failed to address encryption and decryption of PHI. There were no functions such as crypt() for encryption and md5() for data hashing. This obviously violates HIPAA rule for encryption and decryption of PHI (Security rule 164.312(a)(2)(iv)).

Table 6 shows the compliance of evaluated EHRs (OpenEMR and OpenClinic) with the HIPAA technical safeguards based on our observation.

Table 6: EHRs compliance with HIPAA technical requirements

HIPAA Technical Requirements	OpenEMR	OpenClinic
164.312(a)(1) Access Controls	No	No
164.312(a)(2)(i) Identify and track user identity	Yes	No
164.312(a)(2)(ii) Emergency access	No	No
164.312(a)(2)(iii) Automatic Log-Off	Yes	Yes
164.312(a)(2)(iv) Encryption	No	No
164.312(b) Audit Controls	Yes	No
164.312(c)(1) Integrity	No	No
164.312(c)(2) (Integrity)	N/A	N/A
164.312(d) (Authentication)	N/A	N/A
164.312(e)(1) Transmission Security	No	No

164.312(e)(2)(i) (Transmission Security)	N/A	N/A
164.312(e)(2)(ii) (Encryption)	N/A	N/A

In the next chapter, the resulted obtained from the scanning tools are manually inspected for accuracy to form a basis for compliance with ONC certificate due to traditional web security vulnerabilities.

CHAPTER 4

Compliance Checking for ONC Security and Privacy Requirements

4.1 Overview

In chapter 3, we applied two popular code analysis tools, named RIPS and PHP VulnHunter to examine the code in OpenEMR and OpenClinic [16, 33]. Then we inspected the obtained results and manually checked the accuracy to form a basis for compliance with HIPAA technical requirements. In this chapter, we inspect the obtained results from chapter 3, and check the accuracy to form a basis for compliance with ONC privacy and security-focused requirements [49]. The initial results show that both evaluated EHRs may violate some required and optional HIPAA and ONC privacy and security requirements.

It is important to note that in some instances, HIPAA and ONC criteria may appear to overlap. For example, both HIPAA and ONC define a unique identifier for each user as well as automatic log-off after a predetermined period of inactivity. However, they were examined on an individual basis due to the severity of ramifications associated with non-compliance. Non-compliance with the HIPAA criteria has far worse implications than not meeting ONC criteria. In some cases, the two criteria address different requirements. For example, ONC requires EHR applications to allow a user to capture a patient's request for an amendment to their electronic health information.

4.2 Compliance with ONC Certification

In this section, we discuss ONC Certification Program, which is required for eligible providers seeking to demonstrate "meaningful use" [49]. Meaningful use is the minimum government standards for using and exchanging patients' health information. ONC certification has several criteria including Clinical Processes, Care Coordination, Clinical Quality Measurement, Privacy and Security, Patient Engagement, Public Health, Health IT Design and Performance, and Electronic Exchange. In this work, we considered Privacy and Security criteria and studied the compliance of the two evaluated EHRs (OpenEMR and OpenClinic) with Privacy and Security Criteria (Tables 7 and 8).

Table 7 shows the 2015 Edition of ONC certification criteria for EHR privacy and security.

Table 7: Privacy and security-related of ONC certification criteria

ONC Certification Criterion	**Short Description**
45 CFR §170.314(d)(1)	**Authentication, Access Control, and Authorization:** EHR must authenticate and authorize a user, and control user access to the information.
45 CFR §170.314(d)(2)	**Auditable Events and Tamper-Resistance:** EHR must: Record user actions in an audit log; Record audit log status or encryption status; Only enable specific users to disable an audit log, if possible; Protect actions and statuses; and Detect when the audit log has been altered.
45 CFR §170.314(d)(3)	**Audit Report(s):** EHR must allow a user to generate an audit report for a specific time period.
45 CFR §170.314(d)(4)	**Amendments:** EHR must allow a user to capture a patient's request for an amendment to their electronic health information.
45 CFR §170.314(d)(5)	**Automatic Log-Off:** EHR must log-off users automatically after a predetermined period of inactivity.
45 CFR §170.314(d)(6)	**Emergency Access:** EHR must permit specific users to have access to the electronic health information in the event of an emergency.
45 CFR §170.314(d)(7)	**End-User Device Encryption:** EHR must encrypt electronic health information on end-user devices.
45 CFR §170.314(d)(8)	**Integrity:** EHR must use secure hashing standards to ensure that electronic health information has not been altered.
45 CFR §170.314(d)(9)	**Optional – Accounting of Disclosures:** EHR must record treatment, payment, and health care operations disclosures.

Table 8 shows the compliance of the two evaluated EHRs (OpenEMR and OpenClinic) with the ONC Privacy and Security Criteria based on our observation.

Table 8: EHR compliance with ONC certification criteria

Certification Criterion	OpenEMR	OpenClinic
45 CFR §170.314(d)(1) Authentication, Access Control, and Authorization	Yes	Yes
45 CFR §170.314(d)(2) Auditable Events and Tamper-Resistance	Yes	No
45 CFR §170.314(d)(3) Audit Report(s)	Yes	No
45 CFR §170.314(d)(4) Amendments	Yes	No
45 CFR §170.314(d)(5) Automatic Log-Off	Yes	Yes
45 CFR §170.314(d)(6) Emergency Access	No	No
45 CFR §170.314(d)(7) End-User Device Encryption	No	No
45 CFR §170.314(d)(8) Integrity	No	No
45 CFR §170.314(d)(9) Accounting of Disclosures	N/A	N/A

According to Table 8, OpenEMR and OpenClinic failed to meet the ONC Criteria during emergency time. Also, End-User Device Encryption and Integrity were not addressed by the EHRs. OpenEMR supported other ONC criteria while OpenClinic failed to address most of the cases.

4.3 Improving Compliance

To improve compliance, we developed and implemented code solutions to address some of the security vulnerabilities found in evaluated EHR applications. We then rescanned the new code with the two static analysis tools and found no vulnerability. Table 9 provides some examples of identified unsanitized code in EHRs and how we fixed the code to improve the compliance with HIPAA and ONC security requirements. Below we briefly explain two examples of fixing vulnerable code in Table 9.

Table 9: Code examples of vulnerable EHR files and the solution code.

Vulnerability	Vulnerable Code Example	Fixed Code Example
Cross-Site Scripting: *$_POST + echo()*	echo echo " durations[" . $crow['pc_catid'] . "] = $duration\n"; $duration = $_POST['form_duration'];	print ("Hello " . htmlentities($_GET["name"], ENT_QUOTES, "utf-8"); code was changed to: $duration = htmlentities($_POST['form_duration'],ENT_QUOTES, "utf-8);

File Inclusion: *$_GET + Require_one()*	require_once require_once $GLOBALS['OE_SITE_DIR'] . "/sqlconf.php"; // sqlconf.php);
	$tmp = 'default' : $_GET['site'];
		$files = array("index.php", "main.php");
		if(!in_array($_GET["file"], $files)) exit ;
		code was changed to:
		$tmp = empty($_GET['site']) ? 'default' : if (!in_array($_GET['site'], $site))exit;

File Manipulation: *$_GET + mkdir()*	mkdir mkdir($GLOBALS['OE_SITE_DIR'] . '/documents/mpdf/pdf_tmp/', 0755); $tmp = 'default' : $_GET['site'];	$files = array("index.php", "main.php"); if(!in_array($_GET["file"], $files)) exit ; code was changed to: $tmp = empty($_GET['site']) ? 'default' : if (!in_array($_GET['site'], $site))exit;
File Disclosure: *$_FILES + fread()*	fread $filetext = fread($tmpfile, $_FILES['file']['size'][$key]); $tmpfile = fopen($_FILES['file']['tmp_name'][$key], "r");	$files = array("index.php", "main.php"); if(!in_array($_GET["file"], $files)) exit ;
Http response splitting: *$_GET + header()*	header header("Content-Length: " . filesize($tmpcouchpath)); $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; $tmp = 'default' : $_GET['site'];	if(!in_array($_GET["url"], $whitelist)) exit ;

In a cross-site scripting attack, an attacker injects malicious code into the trusted web application that does not properly validate the user input. Cross-site scripting may occur when function *echo()* comes with the *$_GET* (or *$_POST*). To fix this vulnerability, we used *htmlentities()* function, which converts all applicable characters to HTML entities.

File inclusion occurs due to poor input validation when an attacker is allowed to exploit external file inclusion functionality that dynamically includes local or external files. File inclusion may occur when function *require_once()* comes with the *$_GET* (or *$_POST*). Function *require_once()* works almost as function *include()*. Not limiting the file name to specific paths or extensions is one way to avoid file inclusion. To ensure that, we added *if (!in_array($_GET["file"], $files)) exit* to the code.

The third row (second column) shows an example of file manipulation where user input (*$_GET['site']*) could lead to a default file creation within the local file server of a website. The fixed code (third column) shows one solution where acceptable file names are whitelisted into an array (*$files*). If user input is empty, a default file is created, otherwise user input is matched with known filenames. Similarly, we have addressed file disclosure issue during file reading in the fourth row of Table 9.

In file disclosure attack, an attacker can read local files. User tainted data is used when creating the file name that is supposed to get opened. Therefore, the attacker can read the source code and other files which might lead to other attacks. File disclosure may occur when function *fread()* comes with *$_FILES.* To fix this vulnerability, we added *if(!in_array($_GET["file"], $files)) exit;* to make sure that the attacker cannot read the source code.

The last row of Table 9 shows HTTP response splitting. In HTTP response splitting attack, malicious data is embedded in HTTP response headers. HTTP response is split into two responses instead of one. This vulnerability can occur when function *header()* comes with *$_GET* (or *$_POST*). To address this issue, we used *if(!in_array($_GET["url"], $whitelist)) exit;* which prevents from spilt the HTTP response into two responses.

Note our suggested fixed examples highlights the need of using specific APIs available in the implementation language (e.g., htmlentities() in PHP), or adding the needed code for checking of entities from white or black list, to prevent vulnerabilities. It may not be possible to provide one common solution for multiple vulnerabilities due to implementation of input filtering can vary from application to application.

The fixing of code took much less time once we have a common API pattern (e.g., htmlentities()) identified for finding and replacing with the correct code. However, for checking of whitelisting or blacklisting (e.g., if (!in_array(..)..), it took few hours to identify the location and fix the code.

CHAPTER 5

Conclusion

Electronic Health Record applications are digital versions of paper-based patient's health information. They are increasingly being implemented in many countries. They have resulted in better healthcare, lower costs, easier follow ups, and more precise medical decisions. EHR applications are guided by measures of HIPAA to ensure confidentiality, integrity, and security. However, concerns of security breach always exist in digital formats. In many reported breaches, improper use of EHRs has resulted in disclosure of patient's protected health information. Therefore, more awareness in existing EHRs capability of protecting patient's healthcare data is needed.

We conducted a survey of literature search focused on relevant works about EHR security and privacy vulnerabilities, complying with HIPAA and ONC security and privacy requirements; identified key security requirements of HIPAA and ONC framework; evaluated selected open source EHR applications for security vulnerabilities using open source scanner tools; compared and contrasted security vulnerabilities identified by scanner tools with HIPAA and ONC security requirements; and proposed an approach to secure personal health identifier information within applications.

We identified security and privacy requirements for HIPAA technical requirements, and identified a subset of ONC criteria related to security and privacy. Then we evaluated EHR applications for security vulnerabilities, and finally proposed mitigation of security issues towards better compliance. The proposed approach helps practitioners to reuse the open source tools towards certification compliance.

Our future work includes the evaluation of other open-source EHR applications and using additional and different static analysis tools. We also plan to evaluate the effectiveness of our proposed solution by using dynamic analysis tools.

Appendix A: RIPS Results for OpenEMR

Vulnerability	Vulnerability Concept	File(s)

File Disclosure	Source sink vulnerability
	$_GET + require_once() = File Inclusion
	
	19: require_once require_once $GLOBALS['OE_SITE_DIR'] . "/sqlconf.php"; // sqlconf.php
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
		ccr/createCCR.php

File Disclosure	Source sink vulnerability
	$_GET + opendir() = File Disclosure
	
	57: opendir $dh = opendir($templatedir);
	54: $templatedir = $GLOBALS['OE_SITE_DIR'] . '/documents/doctemplates';
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	requires:
	56: if(file_exists($templatedir))
	45: ⇓ function upload_action($patient_id, $category_id)
		controllers/C_Document.class.php

File Disclosure	source sink vulnerability $_GET + readfile() = File Disclosure	custom/qrda_download.php

File Disclosure	73: readfile readfile($finalZip);
	63: $finalZip = $qrda_file_path . $fileList[0]; // if(count($fileList) > 1) else ,
	55: $qrda_file_path = $GLOBALS['OE_SITE_DIR'] . "/documents/cqm_qrda/"; // qrda_category1.inc
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	43: $fileList = explode(",", $fileName);
	39: $fileName = $_GET['fileName'] : "";
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	requires:
	42: if($fileName)
		custom/ajax_download.php

File Disclosure	source sink vulnerability
	$_POST + parse_ini_file() = File Disclosure
	
	45: parse_ini_file $config = parse_ini_file($config_file); // gacl_admin.inc.php
	40: $config_file = dirname(__FILE__) . '/../gacl.ini.php'; // gacl_admin.inc.phpif(!isset($config_file)),
	59: $config_file = ''; // gacl.class.php
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	requires:
	44: if(file_exists($config_file))
		gacl/admin/about.php

Protocol Injection	source: $_GET $_POST + sink: mail() = vulnerability: Protocol Injection
	
	45: mail mail($pt_email, $email_subject, $email_body, $headers)) // batchEmail.php
	32: $pt_email = $row['email']; // batchEmail.php
	28: $row = sqlfetcharray ($res)){ // batchEmail.php
	125: $res = sqlstatement ($sql);
	122: $sql .= ' ORDER BY ' . $_POST['sort_by'];
	117: $sql .= " $and patient_data.email IS NOT NULL "; // switch($_POST) : , case $choices : ,
	111: $sql .= " $and patient_data.hipaa_mail='YES' "; // if($_POST != 'NO'),
	105: $sql .= " $and patient_data.sex='" . $_POST['gender'] . "' "; // if($_POST != 'Any'),
	99: $sql .= " $and DATEDIFF(CURDATE( ), patient_data.DOB)/ 365.25 <= '" . $_POST['age_upto'] . "' "; // if($_POST != 0 AND $_POST != ''),
	95: $sql .= " $and DATEDIFF(CURDATE( ), patient_data.DOB)/ 365.25 >= '" . $_POST['age_from'] . "' "; // if($_POST != 0 AND $_POST != ''),
	89: $sql .= " $and forms.date > '" . $_POST['seen_since'] . "' "; // if($_POST != 0 AND $_POST != ''),
	85: $sql .= " $and forms.date > '" . $_POST['seen_since'] . "' "; // if($_POST != 0 AND $_POST != ''),
	80: $sql .= $sql_where_a;
	70: $sql = "select patient_data.*, cal_events.pc_eventDate as next_appt,cal_events.pc_startTime as appt_start_time,cal_date.last_appt,forms.last_visit from patient_data left outer join openemr_postcalendar_events as cal_events on patient_data.pid=cal_events.pc_pid and curdate() < cal_events.pc_eventDate left outer join (select pc_pid,max(pc_eventDate) as last_appt from openemr_postcalendar_events where curdate() >= pc_eventDate group by pc_pid) as cal_date on cal_date.pc_pid=patient_data.pid left outer join (select pid,max(date) as last_visit from forms where curdate() >= date group by pid) as forms on forms.pid=patient_data.pid";
	78: $sql_where_a .= " $and cal_events.pc_endDate < '" . $_POST['app_e'] . "'"; // if($_POST != 0 AND $_POST != ''),
	74: $sql_where_a = " $and cal_events.pc_eventDate > '" . $_POST['app_s'] . "'"; // if($_POST != 0 AND $_POST != ''),
	73: $and = where_or_and ($and); // if($_POST != 0 AND $_POST != ''),
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	508: extract($_GET, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	508: extract($_GET, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
		interface\batchcom/batchcom.php

	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	508: extract($_GET, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	77: $and = where_or_and ($and); // if($_POST != 0 AND $_POST != ''),
	73: $and = where_or_and ($and); // if($_POST != 0 AND $_POST != ''),	

File Disclosure	source sink vulnerability
	$_GET + fgets() = File Disclosure
	33: fgets $line = fgets($fh)){
	31: $fh = fopen($filename, 'r');
	28: $filename = $GLOBALS['OE_SITE_DIR'] . '/edi/process_bills.log';
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
		interface\billing/customize_log.php

File Disclosure	source sink vulnerability
	$_GET + fpassthru() = File Disclosure
	39: fpassthru fpassthru($fp);
	29: $fp = fopen($fname, 'r');
	22: $fname = $claim_file_dir . $fname;
	11: $claim_file_dir = $GLOBALS['OE_SITE_DIR'] . "/edi/";
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	16: $fname = preg_replace("[\\\\]", "", $fname);
	15: $fname = preg_replace("[\.\.]", "", $fname);
	14: $fname = preg_replace("[/]", "", $fname);
	13: $fname = $_GET['key'];
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
		interface\billing/get_claim_file.php

File Disclosure	source sink vulnerability
	$_GET + scandir() = File Disclosure
	98: scandir $files_array = scandir($mainPATH);
	61: $mainPATH = $GLOBALS['fileroot'] . "/contrib/" . strtolower($db);
	152: $GLOBALS['fileroot'] = "$webserver_root"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	60: $db = $_GET['db'] : '0';
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
		interface\code_systems/list_staged.php

File Disclosure	source sink vulnerability
	$_GET + file_get_contents() = File Disclosure
	228: file_get_contents $file_checksum = md5(file_get_contents($file));
	110: $file = $mainPATH . "/" . $file;
	61: $mainPATH = $GLOBALS['fileroot'] . "/contrib/" . strtolower($db);
	152: $GLOBALS['fileroot'] = "$webserver_root"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	60: $db = $_GET['db'] : '0';
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	108: foreach($files_array as $file)
	98: $files_array = scandir($mainPATH); // , trace stopped
		interface\code_systems/list_staged.php

File Disclosure	source sink vulnerability $_GET + scandir() = File Disclosure	interface\code_systems/standard_tables_manage.php
File Disclosure	source sink vulnerability $_POST + opendir() = File Disclosure	interface\fax/faxq.php
File Disclosure	source sink vulnerability $_GET $_POST + fread() = File Disclosure	interface\fax/fax_dispatch.php
File Disclosure	source Sink Vulnerability $_GET + fgets() = File Disclosure	interface\fax/fax_view.php
File Disclosure	-	interface\forms/CAMOS/admin.php

File Disclosure	source sink vulnerability
	$_GET + imagecreatefrompng() = File Disclosure
	
	539: imagecreatefrompng $im = imagecreatefrompng($chartpath . $chart);
	36: $chartpath = $GLOBALS['fileroot'] . "/interface/forms/vitals/growthchart/";
	152: $GLOBALS['fileroot'] = "$webserver_root"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	197: $chart = "2-20yo_girls_BMI.png"; // elseif($charttype == "2-20"), elseif(preg_match('/^female/i', $patient_data)),
	190: $chart = "2-20yo_boys_BMI.png"; // elseif($charttype == "2-20"), if(preg_match('/^male/i', $patient_data)),
	143: $chart = "birth-24mos_girls_HC.png"; // if($charttype == 'birth'), elseif(preg_match('/^female/i', $patient_data)),
	136: $chart = "birth-24mos_boys_HC.png"; // if($charttype == 'birth'), if(preg_match('/^male/i', $patient_data)),
	417: extract(getpatientageymd ($dob, $date)); // register_globals implementationif($_GET == 1),
	51: $dob = $patient_data['DOB']; // if(isset($pid) && is_numeric($pid)),
	49: $patient_data = getpatientdata ($pid, "fname, lname, sex, DATE_FORMAT(DOB,'%Y%m%d') as DOB"); // if(isset($pid) && is_numeric($pid)),
	39: $pid = $_GET['pid'];
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	407: list($date, $height, $weight, $head_circ) = explode('-', $data); // list() if($_GET == 1),
	406: foreach($datapoints as $data) // if($_GET == 1),
	59: $datapoints = explode('~', $_GET['data']);
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
		interface\forms/vitals/growthchart/chart.php

File Disclosure	Source / sink / Vulnerability
	$_GET
	$_POST + glob() = File Disclosure
	320: glob glob($filepath . '/' . $filename) as
	319: $filepath = $GLOBALS['oer_config']['documents']['repository'] . $pid;
	235: $pid = $_SESSION['pid'];
	474: $_SESSION['pid'] = $_POST['pid']; // globals.phpelseif(!empty($_POST) && empty($_SESSION)),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	318: $filename = $pid . "_" . $encounter . ".pdf";
	235: $pid = $_SESSION['pid'];
	474: $_SESSION['pid'] = $_POST['pid']; // globals.phpelseif(!empty($_POST) && empty($_SESSION)),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	237: $encounter = date("Ymd");
		interface\forms/eye_mag/save.php

File Disclosure	source / sink / vulnerability
	$_POST + file() = File Disclosure
	177: file $form_title_file = file($GLOBALS['srcdir'] . "/../interface/forms/$fname/info.txt");
	426: $srcdir = $GLOBALS['srcdir']; // globals.php
	150: $GLOBALS['srcdir'] = "$webserver_root/library"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	160: foreach($inDir as $fname)
	148: $inDir[$i] = $fname; // if($fname != "." && $fname != ".." && $fname != "CVS" && $fname != "LBF" && (is_dir($dpath . $fname) \|\| stristr($fname, ".tar.gz") || stristr($fname, ".tar") \|\| stristr($fname, ".zip") || stristr($fname, ".gz"))),
	143: $fname = readdir($dp));
	141: $dp = opendir($dpath); // , trace stopped
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
		interface\forms_admin/forms_admin.php

File Disclosure	56: fread $bat_content = fread($fd, filesize($error_log_path . '/' . $filename));
	55: $fd = fopen($error_log_path . '/' . $filename, 'r')){
	35: $error_log_path = $GLOBALS['OE_SITE_DIR'] . '/documents/erx_error';
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	40: $filename = ''; // if(array_key_exists('filename', $_REQUEST)) else ,
		interface\logview/erx_logview.php

Session Fixation	Source sink vulnerability
	$_GET + setcookie() = Session Fixation
	74: setcookie setcookie("pc_facility", $_SESSION['pc_facility'], time () + (3600 * 365));
	73: $_SESSION['pc_facility'] = $_GET['pc_facility'];
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	requires:
	74: if($GLOBALS['set_facility_cookie'] && ($_SESSION['pc_facility'] > 0))
		interface\main/calendar/index.php

File Disclosure	source sink vulnerability $_FILES + fread() = File Disclosure 87: fread $filetext = fread($tmpfile, $file['size']); 86: $tmpfile = fopen($file['tmp_name'], "r"); 70: foreach($_FILES as $file) requires: 57: if($request->ispost()) 55: ⇓ function uploadaction()	interface\modules/zend_modules/module/Documents/src/Documents/Controller/DocumentsController.php
File Disclosure	source sink vulnerability $_FILES + fgetcsv() = File Disclosure 170: fgetcsv $acsv = fgetcsv($fhcsv, 4096); 147: $fhcsv = fopen($_FILES['userfile']['tmp_name'], "r"); requires: 136: if($form_step == 1) 138: if(is_uploaded_file($_FILES['userfile']['tmp_name'])) 149: if($fhcsv) 153: if($form_vendor == '1235186800') 155: if($form_action == 1)	interface\orders/load_compendium.php
File Disclosure	-	interface\patient_file/download_template.php
File Disclosure	-	interface\patient_file/education.php
File Disclosure	-	interface\patient_file/letter.php
File Disclosure	-	interface\patient_file/merge_patients.php

File Disclosure	-	interface\patient_file/transaction/print_referral.php

File Disclosure	Source + sink = vulnerability
	$_GET $_FILES + file() = File Disclosure
	60: file $Response271 = file($FilePath);
	41: $FilePath = $target;
	39: $target = $target . time () . basename($_FILES['uploaded']['name']);
	34: $target = $GLOBALS['edi_271_file_path'];
	177: $GLOBALS['edi_271_file_path'] = $GLOBALS['OE_SITE_DIR'] . "/edi/"; // globals.php
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	requires:
	36: if(isset($_FILES) && !empty($_FILES))
	53: if(!isset($message))
	55: if(move_uploaded_file($_FILES['uploaded']['tmp_name'], $target))
		interface\reports/edi_271.php

File Disclosure	source + sink = vulnerability
	$_POST + opendir() = File Disclosure
	568: opendir $dh = opendir($themedir);
	567: $themedir = "$webserver_root/interface/themes";
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	509: extract($_POST, EXTR_SKIP); // globals.phpregister_globals implementationif($fake_register_globals),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
	requires:
	397: if(!$userMode \|\| in_array($grpname, $USER_SPECIFIC_TABS))
	414: if(!$userMode \|\| in_array($fldid, $USER_SPECIFIC_GLOBALS))
	563: if($fldtype == 'css')
		interface\super/edit_globals.php

File Disclosure	source: $_GET $_POST + sink: readfile() = vulnerability: File Disclosure
	55: readfile readfile($templatepath);
	42: $templatepath = "$templatedir/$form_filename";
	36: $templatedir = "$OE_SITE_DIR/documents/doctemplates";
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] . '/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites"; // sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; // globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); // globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; // sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); // globals.phpif($sanitize_all_escapes),
	34: $form_filename = strip_escape_custom ($_REQUEST['form_filename']);
	45: $_REQUEST = undomagicquotes ($_REQUEST); // globals.phpif($sanitize_all_escapes),
	requires:
	41: if(!empty($_POST['bn_download']))
		\interface\super/manage_document_templates.php

File Disclosure	source: $_POST $_FILES + sink: fgets() = vulnerability: File Disclosure
	94: fgets $line = fgets($eres)) !==
	81: $eres = fopen($tmp_name, 'r'); // if($zipin->open($tmp_name) === true) else ,
	60: $tmp_name = $_FILES['form_file']['tmp_name'];
	requires:
	57: if(!empty($_POST['bn_upload']))
	66: if(is_uploaded_file($tmp_name) && $_FILES['form_file']['size'])
		interface\super/load_codes.php

File Disclosure	source: $_POST + sink: file_get_contents() = vulnerability: File Disclosure
	56: file_get_contents $data = file_get_contents("https://rxnav.nlm.nih.gov/REST/interaction/list.json?rxcuis=" . $rxcui_list);
	55: $rxcui_list = implode("+", $nameList);
	41: $nameList[] = $rXn['rxcui'];
	40: $rXn = sqlquery ("SELECT \`rxcui\` FROM \`" . mitigatesqltableuppercase ('RXNCONSO') . "\` WHERE \`str\` LIKE ?", array("%" . $drug[0] . "%"));
	39: $drug = explode(" ", $name['drug']);
	38: $name = sqlfetcharray ($medList)){
	36: $medList = sqlstatement ("SELECT drug FROM prescriptions WHERE active = 1 AND patient_id = ?", array($pid));
	476: $pid = 0 : $_SESSION['pid']; // globals.php
	474: $_SESSION['pid'] = $_POST['pid']; // globals.phpelseif(!empty($_POST) && empty($_SESSION)),
	43: $_POST = undomagicquotes ($_POST); // globals.phpif($sanitize_all_escapes),
		interface\weno/drug-drug.php

File Disclosure	source sink vulnerability
	$_GET + file_get_contents() = File Disclosure
	76: file_get_contents $content = file_get_contents($temp_url);
	72: $temp_url = $GLOBALS['OE_SITE_DIR'] . '/documents/' .
	$from_pathname . '/' . $from_filename;
	17: $GLOBALS['OE_SITE_DIR'] = $GLOBALS['OE_SITES_BASE'] .
	'/' . $tmp; // sqlconf.phpif(empty($GLOBALS)),
	15: $GLOBALS['OE_SITES_BASE'] = dirname(__FILE__) . "/../sites";
	// sqlconf.phpif(empty($GLOBALS)),
	84: $GLOBALS['OE_SITES_BASE'] = "$webserver_root/sites"; //
	globals.php
	56: $webserver_root = str_replace("\\", "/", $webserver_root); //
	globals.phpif(IS_WINDOWS),
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	53: $webserver_root = dirname(dirname(__FILE__)); // globals.php
	16: $tmp = 'default' : $_GET['site']; //
	sqlconf.phpif(empty($GLOBALS)),
	42: $_GET = undomagicquotes ($_GET); //
	globals.phpif($sanitize_all_escapes),
	71: $from_pathname = implode("/", $from_pathname_array);
	70: $from_pathname_array = array_reverse($from_pathname_array);
	68: $from_pathname_array[] = array_pop($from_all);
	64: $from_all = explode("/", $url);
	63: $url = preg_replace("|^(.*)://|", "", $url);
	42: $url = $d->get_url();
	65: $from_filename = array_pop($from_all);
	64: $from_all = explode("/", $url);
	63: $url = preg_replace("|^(.*)://|", "", $url);
	42: $url = $d->get_url();
	requires:
	39: if($_REQUEST['ccr_ajax'] == "yes")
	62: if($storagemethod == 1) else
	75: if(!file_exists($temp_url)) else
		library\ajax/ccr_import_aja
		x.php

Protocol Injection	source sink vulnerability $_GET + fsockopen() = Protocol Injection 236: fsockopen fsockopen(($this->ssl"ssl://" : "") . $ip, $port, $errno, $error, $this->timeout) : 228: $ip = gethostbyname($domain), $domain)) // if(preg_match('/^[0- 9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$/', $domain)) else , 210: ⇓ Function connecttohost($domain, $port, $resolve_message)	library\classes/smtp/smtp.p hp

File Disclosure	source sink vulnerability
	$_GET + file_get_contents() = File Disclosure
	1919: file_get_contents $ret = file_get_contents($path); // tcpdf_static.php
	1918: foreach($alt as $path) // tcpdf_static.php
	1916: $alt = array_unique($alt); // tcpdf_static.php
	1903: $alt[] = htmlspecialchars_decode(urldecode($tmp)); // tcpdf_static.phpif(preg_match('%^(https?)://%', $url) && empty($_SERVER) && empty($_SERVER)), if(empty($urldata)), if(strpos($url, $host) === 0),
	1902: $tmp = str_replace($host, $_SERVER['DOCUMENT_ROOT'], $url); // tcpdf_static.phpif(preg_match('%^(https?)://%', $url) && empty($_SERVER) && empty($_SERVER)), if(empty($urldata)), if(strpos($url, $host) === 0),
	1899: $host = $protocol . '://' . $_SERVER['HTTP_HOST']; // tcpdf_static.phpif(preg_match('%^(https?)://%', $url) && empty($_SERVER) && empty($_SERVER)), if(empty($urldata)),
	1883: $protocol .= 's'; // tcpdf_static.phpif(!empty($_SERVER) && (strtolower($_SERVER) != 'off')),
	1881: $protocol = 'http'; // tcpdf_static.php
	1881: $protocol = 'http'; // tcpdf_static.php
	1890: $url = htmlspecialchars_decode($url); // tcpdf_static.php
	1888: $url = $protocol . ':' . str_replace(' ', '%20', $url); // tcpdf_static.phpif(preg_match('%^//%', $url) && !empty($_SERVER)),
	1883: $protocol .= 's'; // tcpdf_static.phpif(!empty($_SERVER) && (strtolower($_SERVER) != 'off')),
	1881: $protocol = 'http'; // tcpdf_static.php
	1886: $url = $file; // tcpdf_static.php
	1866: ⇓ function filegetcontents($file)
	requires:
	1866: ⇓ function filegetcontents($file)
		library\html2pdf/vendor/tecnickcom/tcpdf/tools/tcpdf_addfont.php

File Disclosure	Source sink vulnerability
	$_GET + opendir() = File Disclosure
	138: fread $filetext = fread($tmpfile, $_FILES['file']['size'][$key]); // C_Document.class.php
	137: $tmpfile = fopen($_FILES['file']['tmp_name'][$key], "r"); // C_Document.class.php
	requires:
	122: if(count($_FILES['file']['name']) > 0)
	136: if($_FILES['file']['error'][$key] > 0 || empty($fname) || $_FILES['file']['size'][$key] == 0) else
	80: ⇓ function upload_action_process()
		myportal/soap_service/server_side.php

File Disclosure	Source sink vulnerability
	$_FILES + fread() = File Disclosure
	138: fread $filetext = fread($tmpfile, $_FILES['file']['size'][$key]); // C_Document.class.php
	137: $tmpfile = fopen($_FILES['file']['tmp_name'][$key], "r"); // C_Document.class.php
	requires:
	122: if(count($_FILES['file']['name']) > 0)
	136: if($_FILES['file']['error'][$key] > 0 || empty($fname) || $_FILES['file']['size'][$key] == 0) else
	80: ⇓ function upload_action_process()
		patients/get_patient_documents.php

File Disclosure	source sink vulnerability
	$_GET + readfile() = File Disclosure
	55: readfile readfile($script_name);
	49: $script_name .= DIRECTORY_SEPARATOR . $filename; // if(preg_match("@^[\w][\w\.-]+$@", $filename)),
	42: $script_name = 'js';
	45: foreach($path as $index=>$filename)
	44: $path = explode("/", $script);
	40: foreach($_GET['scripts'] as $script)
	38: $_GET['scripts'] = json_decode($_GET['scripts']);
	22: $_GET['scripts'] = json_encode($_GET['scripts']);
	22: $_GET['scripts'] = json_encode($_GET['scripts']);
	requires:
	39: if(!empty($_GET['scripts']) && is_array($_GET['scripts']))
	54: if(preg_match("@\.js$@", $script_name) && is_readable($script_name))
		phpmyadmin\js/get_scripts.js.php

Session Fixation	source sink vulnerability
	$_GET + session_id() = Session Fixation
	306: session_id session_id($_GET['session_to_unset']); // swekey.auth.lib.php
	requires:
	304: if(!empty($_GET['session_to_unset']))
		phpmyadmin\libraries/plugins/auth/AuthenticationCookie.class.php

Session Fixation	-	phpmyadmin\libraries/plugins/auth/AuthenticationSignon.class.php

File Disclosure	source sink vulnerability
	$_GET + readfile() = File Disclosure
	55: readfile readfile($script_name);
	49: $script_name .= DIRECTORY_SEPARATOR . $filename; // if(preg_match("@^[\w][\w\.-]+$@", $filename)),
	42: $script_name = 'js';
	45: foreach($path as $index=>$filename)
	44: $path = explode("/", $script);
	40: foreach($_GET['scripts'] as $script)
	38: $_GET['scripts'] = json_decode($_GET['scripts']);
	22: $_GET['scripts'] = json_encode($_GET['scripts']);
	22: $_GET['scripts'] = json_encode($_GET['scripts']);
	requires:
	39: if(!empty($_GET['scripts']) && is_array($_GET['scripts']))
	54: if(preg_match("@\.js$@", $script_name) && is_readable($script_name))
		phpmyadmin/js/get_scripts.js.php

File Disclosure	source / sink / vulnerability
	$_GET + file() = File Disclosure
	152: file $existing = file($output, FILE_IGNORE_NEW_LINES);
	93: $output = STDOUT; // if(isset($opts->o)), if('-' == $output),
	91: $output = $opts->o; // if(isset($opts->o)),
	89: $output = $path . DIRECTORY_SEPARATOR . 'plugin_classmap.php';
	84: $path = realpath($libraryPath); // if(isset($opts->l)),
	78: $libraryPath = rtrim($libraryPath, '/\\') . DIRECTORY_SEPARATOR; // if(isset($opts->l)),
	77: $libraryPath = $opts->l; // if(isset($opts->l)),
	77: $libraryPath = $opts->l; // if(isset($opts->l)),
	73: $path = $_SERVER['PWD']; // if(array_key_exists('PWD', $_SERVER)),
	71: $path = $libPath;
	31: $libPath = getenv('LIB_PATH') : __DIR__ . '/../library';
	requires:
	141: if($appending)
		vendor\zendframework/zendframework/bin/pluginmap_generator.php

Appendix B: PHP VulnHunter Results for OpenEMR

Vulnerability	Code Excerpt	File
XSS	Line 318 0309: <td><?php echo htmlspecialchars(xl('Context'),ENT_QUOTES);?></td> 0310: <td> 0311: <select name='filter_context' id='filter_context' onchange='javascript:document.myform.submit();'> 0312: <option value=''><?php echo htmlspecialchars(xl('Select a Context'),ENT_QUOTES);?></option> 0313: <?php 0314: $context_sql="SELECT * FROM customlists WHERE cl_list_type=2 AND cl_deleted=0"; 0315: $context_res=sqlStatement($context_sql); 0316: while($context_row=sqlFetchArray($context_res)){ 0317: echo "<option value='".htmlspecialchars($context_row['cl_list_slno'],ENT_QUOTES)."' "; 0318: echo ($_REQUEST['filter_context']==$context_row['cl_list_slno']) ? 'selected' : '' ; 0319: echo ">".htmlspecialchars($context_row['cl_list_item_long'],ENT_QUOTES)."</option>"; 0320: } 0321: ?> 0322: </select> 0323: </td> 0324: <td><?php echo htmlspecialchars(xl('Users'),ENT_QUOTES);?></td> 0325: <td> 0326: <select name='filter_users' id='filter_users' onchange='javascript:document.myform.submit();'> 0327: <option value=''><?php echo htmlspecialchars(xl('Select a User'),ENT_QUOTES);?></option>	library\custom_template\personalize.php

XSS	Line 333 0324: <td><?php echo htmlspecialchars(xl('Users'),ENT_QUOTES);?></td> 0325: <td> 0326: <select name='filter_users' id='filter_users' onchange='javascript:document.myform.submit();'> 0327: <option value=''><?php echo htmlspecialchars(xl('Select a User'),ENT_QUOTES);?></option> 0328: <?php 0329: $user_sql="SELECT DISTINCT(tu.tu_user_id),u.fname,u.lname FROM template_users AS tu LEFT OUTER JOIN users AS u ON tu.tu_user_id=u.id WHERE tu.tu_user_id!=?"; 0330: $user_res=sqlStatement($user_sql,array($_SESSION['authId'])); 0331: while($user_row=sqlFetchArray($user_res)){ 0332: echo "<option value='".htmlspecialchars($user_row['tu_user_id'],ENT_QUOTES)."' "; 0333: echo ($_REQUEST['filter_users']==$user_row['tu_user_id']) ? 'selected' : '' ; 0334: echo ">".htmlspecialchars($user_row['fname']." ".$user_row['lname'],ENT_QUOTES)."</option>"; 0335: } 0336: ?> 0337: </select> 0338: </td> 0339: </tr> 0340: </table> 0341: </fieldset> 0342: <table align="center" width="100%">	library\custom_template\pe rsonalize.php
XSS	Line 155 0146: 0147: <!-- Required for the popup date selectors --> 0148: <div id="overDiv" style="position:absolute; visibility:hidden; z- index:1000;"></div> 0149: 0150: <form id="new-encounter-form" method='post' action="<?php echo $rootdir ?>/forms/newpatient/save.php" name='new_encounter'> 0151: 0152: <div style='float:left'> 0153: <?php if ($viewmode) { ?> 0154: <input type=hidden name='mode' value='update'> 0155: <input type=hidden name='id' value='<?php echo (isset($_GET["id"])) ? attr($_GET["id"]) : '' ?>'> 0156: <span class=title><?php echo xlt('Patient Encounter Form'); ?></span> 0157: <?php } else { ?> 0158: <input type='hidden' name='mode' value='new'> 0159: <span class='title'><?php echo xlt('New Encounter Form'); ?></span> 0160: <?php } ?> 0161: </div> 0162: 0163: <div> 0164: <div style = 'float:left; margin-left:8px;margin-top:-3px'>	interface\forms\newpatient\ common.php

Insecure Extract Usage	Line 218
	
	0209:
	0210:
	0211: if (php_sapi_name() == 'cli') {
	0212: parse_str(implode('&', array_slice($argv, 1)), $_GET);
	0213: $_SERVER['REQUEST_URI']=$_SERVER['PHP_SELF'];
	0214: $_SERVER['SERVER_NAME']='localhost';
	0215: $backpic = "";
	0216: $ignoreAuth=1;
	0217: }
	0218: $get_count = extract($_GET, EXTR_OVERWRITE);
	0219: // Following breaks link to OpenEMR structure dependency - assumes phpseclib is subdir
	0220: $script_dir = dirname(__FILE__);
	0221: ini_set('include_path', ini_get('include_path') . PATH_SEPARATOR . "$script_dir/phpseclib");
	0222: require_once ("$script_dir/phpseclib/Net/SFTP.php");
	0223: function get_openemr_globals ($libdir) {
	0224: if (!isset($site)) $_GET['site'] = 'default';
	0225: require_once ("$libdir/../interface/globals.php");
	0226: }
	0227: function sftp_status($msg, $val) {
		library\edihistory\test_edih_sftp_files.php

Local File Inclusion	Line 10 0001: <?php 0002: 0003: $special_timeout = 3600; 0004: require_once("../../../interface/globals.php"); 0005: // if (!allowed("frmprint")){ msgDenied(); } 0006: 0007: // ensure the path variable has no illegal characters 0008: check_file_dir_name($_GET["formname"]); 0009: 0010: include_once($incdir . "/forms/" . $_GET["formname"]."/printable.php"); 0011: ?> 0012:	contrib\acog\print_form.php

Local File Inclusion	Line 26
	
	0017: $encounter = $_GET['encounter'];
	0018: }
	0019: if($_GET["formname"] != "newpatient"){
	0020: include_once("$incdir/patient_file/encounter/new_form.php");
	0021: }
	0022:
	0023: // ensure the path variable has no illegal characters
	0024: check_file_dir_name($_GET["formname"]);
	0025:
	0026: include_once("$incdir/forms/" . $_GET["formname"] . "/new.php");
	0027: }
	0028: ?>
	0029:
		interface\patient_file\encounter\load_form.php

Local File Inclusion	Line 18
	
	0009: if (substr($_GET["formname"], 0, 3) === 'LBF') {
	0010: // Use the List Based Forms engine for all LBFxxxxx forms.
	0011: include_once("$incdir/forms/LBF/view.php");
	0012: }
	0013: else {
	0014:
	0015: // ensure the path variable has no illegal characters
	0016: check_file_dir_name($_GET["formname"]);
	0017:
	0018: include_once("$incdir/forms/" . $_GET["formname"] . "/view.php");
	0019: }
	0020:
	0021: $id = $_GET["id"];
	0022: ?>
	0023:
		interface\patient_file\encounter\view_form.php

Appendix C: RIPS Results for OpenClinic

Vulnerability	Vulnerability Concept	File(s)
File Disclosure	source sink vulnerability $_POST $_FILES + file_get_contents() = File Disclosure	admin/theme_preload_css.php install/index.php
File Manipulation	source sink vulnerability $_GET + chmod() = File Manipulation chmod chmod($destinationFile, 0644); 124: $destinationFile = $destinationDir . '/' . $destinationName; 114: ⇓ function upload(&$file, $destinationDir = "", $destinationName = "", $secure = true) 122: $destinationName = $file['name']; 114: ⇓ function upload(&$file, $destinationDir = "", $destinationName = "", $secure = true)	medical/test_edit.php
Cross-Site Scripting	source sink vulnerability $_GET + echo() = Cross-Site Scripting	admin/dump_optimize_db.php layout/header.php doc/index.php admin/user_edit_form.php
HTTP Response Splitting	source sink vulnerability $_POST + header() = HTTP Response Splitting	admin/dump_process.php

Appendix D: PHP VulnHunter Results for OpenClinic

Vulnerability	**Code Excerpt**	**File**

SQL injection	Line 600, 602, 604
	
	0591:);
	0592: echo HTML::itemList($array);
	0593:
	0594: exit();
	0595: }
	0596:
	0597: /**
	0598: * Database creation
	0599: */
	0600: mysql_query('DROP DATABASE IF EXISTS ' . $_POST['dbName']) or die(sprintf(_("Instruction: %s Error: %s"), $sql, mysql_error()));
	0601: //mysql_create_db($_POST['dbName']);
	0602: mysql_query('CREATE DATABASE ' . $_POST['dbName']) or die(sprintf(_("Instruction: %s Error: %s"), $sql, mysql_error()));
	0603: //mysql_select_db($_POST['dbName']);
	0604: mysql_query('USE ' . $_POST['dbName']) or die(sprintf(_("Instruction: %s Error: %s"), $sql, mysql_error()));
	0605:
	0606: /**
	0607: * Database tables creation
	0608: */
	0609: require_once(dirname(__FILE__) . "/parse_sql_file.php");
	0610:
	0611: $tables = getTables();
	0612: foreach ($tables as $tableName)
	0613: {
		install\wizard.php

SQL injection	Line 634
	
	0625: exit();
	0626: }
	0627: }
	0628:
	0629: /**
	0630: * Database tables update (setting_tbl, staff_tbl, user_tbl)
	0631: */
	0632: //mysql_select_db($_POST['dbName']);
	0633: mysql_connect($_POST['dbHost'], $_POST['dbUser'], $_POST['dbPasswd']) or die(sprintf(_("Instruction: %s Error: %s"), $sql, mysql_error()));
	0634: mysql_query('USE ' . $_POST['dbName']) or die(sprintf(_("Instruction: %s Error: %s"), $sql, mysql_error()));
	0635:
	0636: $sql = "UPDATE setting_tbl SET ";
	0637: $sql .= "clinic_name='" . $_POST['clinicName'] . "', ";
	0638: $sql .= "clinic_hours='" . $_POST['clinicHours'] . "', ";
	0639: $sql .= "clinic_address='" . $_POST['clinicAddress'] . "', ";
	0640: $sql .= "clinic_phone='" . $_POST['clinicPhone'] . "', ";
	0641: $sql .= "language='" . $_POST['clinicLanguage'] . "', ";
	0642: $sql .= "session_timeout=" . intval($_POST['timeout']) . ", ";
	0643: $sql .= "items_per_page=" . intval($_POST['itemsPage']) . ", ";
		install\wizard.php

XSS	Line 363
	
	0354: if (!$single) // == false
	0355: {
	0356: $auxConn->close();
	0357: unset($auxConn);
	0358: }
	0359:
	0360: /**
	0361: * "Displays" the dump...
	0362: */
	0363: echo (isset($_POST['as_file']) ? $dumpBuffer : HTML::xmlEntities($dumpBuffer));
	0364:
	0365: /**
	0366: * Close the html tags and add the footers in dump is displayed on screen
	0367: */
	0368: if (empty($_POST['as_file']))
	0369: {
	0370: echo HTML::end('pre');
	0371: echo HTML::para(HTML::link(_("Back return"), "../admin/dump_view_form.php"));
	0372:
		admin\dump_process.php

Appendix E: HIPAA Security Checklist

Security Rule Reference	SAFEGUARD	STATUS
Administrative Safeguards		
164.308(a)(1)(i)	Security Management Process: Implement policies and procedures to prevent, detect, contain, and correct security violations.	■
164.308(a)(1)(ii)(A)	Have a Risk Analysis completed based on NIST Guidelines.	REQUIRED
164.308(a)(1)(ii)(B)	Complete Risk Management process based on NIST Guidelines.	REQUIRED
164.308(a)(1)(ii)(C)	Have formal sanctions or policies against employees who fail to comply with security policies and procedures.	REQUIRED
164.308(a)(1)(ii)(D)	Implement procedures to regularly review records of activities such as audit logs, access reports, and security incident tracking.	REQUIRED
164.308(a)(2)	Assigned Security Responsibility: Identify the security official who is responsible for the development and implementation of the policies and procedures required by this subpart for the entity.	REQUIRED
164.308(a)(3)(i)	Workforce Security: Implement policies and procedures to ensure that all members of its workforce have appropriate access to EPHI, as provided under paragraph (a)(4) of this section, and to prevent those workforce members who do not have access under paragraph (a)(4) of this section from obtaining access to electronic protected health information (EPHI).	■
164.308(a)(3)(ii)(A)	Implement procedures for the authorization and/or supervision of employees who work with EPHI or in locations where it might be accessed.	ADDRESSABLE
164.308(a)(3)(ii)(B)	Implement procedures to determine that access of employees to EPHI is appropriate.	ADDRESSABLE
164.308(a)(3)(ii)(C)	Implement procedures for terminating access to EPHI when an employee leaves you organization or as required by paragraph (a)(3)(ii)(B) of this section.	ADDRESSABLE
164.308(a)(4)(i)	Information Access Management: Implement policies and procedures for authorizing access to EPHI that are consistent with the applicable requirements of subpart E of this part.	■
164.308(a)(4)(ii)(A)	For clearinghouses, implemented policies and procedures to protect EPHI from the larger organization.	ADDRESSABLE
164.308(a)(4)(ii)(B)	Implement policies and procedures for granting access to EPHI, for example, through access to a workstation, transaction, program, or process.	ADDRESSABLE
164.308(a)(4)(ii)(C)	Implement policies and procedures that are based upon your access authorization policies, established, document, review, and modify a user’s right of access to a workstation, transaction, program, or process.	ADDRESSABLE
164.308(a)(5)(i)	Security Awareness and Training: Implement a security awareness and training program for all members of its workforce (including management).	■
164.308(a)(5)(ii)(A)	Provide periodic information security reminders.	ADDRESSABLE

164.308(a)(5)(ii)(B)	Develop policies and procedures for guarding against, detecting, and reporting malicious software.	ADDRESSABLE
164.308(a)(5)(ii)(C)	Develop procedures for monitoring login attempts and reporting discrepancies.	ADDRESSABLE
164.308(a)(5)(ii)(D)	Develop procedures for creating, changing, and safeguarding passwords.	ADDRESSABLE
164.308(a)(6)(i)	Security Incident Procedures: Implement policies and procedures to address security incidents.	
164.308(a)(6)(ii)	Develop procedures to identify and respond to suspected or know security incidents; mitigate to the extent practicable, harmful effects of known security incidents; and document incidents and their outcomes.	REQUIRED
164.308(a)(7)(i)	Contingency Plan: Establish (and implement as needed) policies and procedures for responding to an emergency or other occurrence (for example, fire, vandalism, system failure, and natural disaster) that damages systems that contain EPHI.	
164.308(a)(7)(ii)(A)	Establish and implement procedures to create and maintain retrievable exact copies of EPHI.	REQUIRED
164.308(a)(7)(ii)(B)	Establish (and implement as needed) procedures to restore any loss of EPHI data that is stored electronically.	REQUIRED
164.308(a)(7)(ii)(C)	Establish (and implement as needed) procedures to enable continuation of critical business processes and for protection of EPHI while operating in the emergency mode.	REQUIRED
164.308(a)(7)(ii)(D)	Implement procedures for periodic testing and revision of contingency plans.	ADDRESSABLE
164.308(a)(7)(ii)(E)	Assess the relative criticality of specific applications and data in support of other contingency plan components.	ADDRESSABLE
164.308(a)(8)	Establish a plan for periodic technical and nontechnical evaluation, based initially upon the standards implemented under this rule and subsequently, in response to environmental or operational changes affecting the security of EPHI that establishes the extent to which an entity's security policies and procedures meet the requirements of this subpart.	REQUIRED
164.308(b)(1)	Business Associate Contracts and Other Arrangements: A covered entity, in accordance with Sec. 164.306, may permit a business associate to create, receive, maintain, or transmit EPHI on the covered entity's behalf only of the covered entity obtains satisfactory assurances, in accordance with Sec. 164.314(a) that the business associate appropriately safeguard the information.	
164.308(b)(4)	Establish written contracts or other arrangements with your trading partners that documents satisfactory assurances required by paragraph (b)(1) of this section that meets the applicable requirements of Sec. 164.314(a).	REQUIRED
Physical Safeguards		
164.310(a)(1)	Facility Access Controls: Implement policies and procedures to limit physical access to its electronic information systems and the facility or facilities in which they are housed, while ensuring that properly authorized access is allowed.	

164.310(a)(2)(i)	Establish (and implement as needed) procedures that allow facility access in support of restoration of lost data under the disaster recovery plan and emergency mode operations plan in the event of an emergency.	ADDRESSABLE
164.310(a)(2)(ii)	Implement policies and procedures to safeguard the facility and the equipment therein from unauthorized physical access, tampering, and theft.	ADDRESSABLE
164.310(a)(2)(iii)	Implement procedures to control and validate a person's access to facilities based on their role or function, including visitor control, and control of access to software programs for testing and revision.	ADDRESSABLE
164.310(a)(2)(iv)	Implement policies and procedures to document repairs and modifications to the physical components of a facility, which are related to security (for example, hardware, walls, doors, and locks).	ADDRESSABLE
164.310(b)	Implement policies and procedures that specify the proper functions to be performed, the manner in which those functions are to be performed, and the physical attributes of the surroundings of a specific workstation or class of workstation that can access EPHI.	REQUIRED
164.310(c)	Implement physical safeguards for all workstations that access EPHI to restrict access to authorized users.	REQUIRED
164.310(d)(1)	Device and Media Controls: Implement policies and procedures that govern the receipt and removal of hardware and electronic media that contain EPHI into and out of a facility, and the movement of these items within the facility.	■
164.310(d)(2)(i)	Implement policies and procedures to address final disposition of EPHI, and/or hardware or electronic media on which it is stored.	REQUIRED
164.310(d)(2)(ii)	Implement procedures for removal of EPHI from electronic media before the media are available for reuse.	REQUIRED
164.310(d)(2)(iii)	Maintain a record of the movements of hardware and electronic media and the person responsible for its movement.	ADDRESSABLE
164.310(d)(2)(iv)	Create a retrievable, exact copy of EPHI, when needed, before movement of equipment.	ADDRESSABLE
Technical Safeguard		
164.312(a)(1)	Access Controls: Implement technical policies and procedures for electronic information systems that maintain EPHI to allow access only to those persons or software programs that have been granted access rights as specified in Sec. 164.308(a)(4).	■
164.312(a)(2)(i)	Assign a unique name and/or number for identifying and tracking user identity.	REQUIRED
164.312(a)(2)(ii)	Establish (and implement as needed) procedures for obtaining necessary EPHI during and emergency.	REQUIRED
164.312(a)(2)(iii)	Implement procedures that terminate an electronic session after a predetermined time of inactivity.	ADDRESSABLE
164.312(a)(2)(iv)	Implement a mechanism to encrypt and decrypt EPHI.	ADDRESSABLE

164.312(b)	Implement Audit Controls, hardware, software, and/or procedural mechanisms that record and examine activity in information systems that contain or use EPHI.	REQUIRED
164.312(c)(1)	Integrity: Implement policies and procedures to protect EPHI from improper alteration or destruction.	▨
164.312(c)(2)	Implement electronic mechanisms to corroborate that EPHI has not been altered or destroyed in an unauthorized manner.	ADDRESSABLE
164.312(d)	Implement Person or Entity Authentication procedures to verify that a person or entity seeking access EPHI is the one claimed.	REQUIRED
164.312(e)(1)	Transmission Security: Implement technical security measures to guard against unauthorized access to EPHI that is being transmitted over an electronic communications network.	▨
164.312(e)(2)(i)	Implement security measures to ensure that electronically transmitted EPHI is not improperly modified without detection until disposed of.	ADDRESSABLE
164.312(e)(2)(ii)	Implement a mechanism to encrypt EPHI whenever deemed appropriate.	ADDRESSABLE

References

[1] What is an electronic health record (EHR)? https://www.healthit.gov/providers-professionals/faqs/what-electronic-health-record-ehr

[2] What are the differences between electronic medical records, electronic health records, and personal health records? https://www.healthit.gov/providers-professionals/faqs/what-are-differences-between-electronic-medical-records-electronic, Nov 2015.

[3] The Importance of Data in Healthcare, May 2013, http://www.lumedx.com/the-importance-of-data-in- health-care-.aspx

[4] B. Smith, A. Austin, M. Brown, J. King, J. Lankford, A. Meneely, and L. Williams, "Challenges for Protecting the Privacy of Health Information: Required Certification Can Leave Common Vulnerabilities Undetected " In Proceedings of the second annual workshop on Security and privacy in medical and home-care systems, Pages 1-12, Chicago, Illinois, USA - October 08, 2010.

[5] M. Oliynyk, "Why is healthcare data security so important?", Mar 2016, https://www.protectimus.com/blog/why-is-healthcare-data-security-so-important/

[6] What is 2FA? An extra layer of security that is known as multi factor

[7] Emory Healthcare cyberattack affects 80,000 patient records, http://www.modernhealthcare.com/article/20170302/NEWS/170309983/emory-healthcare-cyberattack-affects-80000-patient-records

[8] 2016 Data Breach Investigations Report, https://regmedia.co.uk/2016/05/12/dbir_2016.pdf

[9] ThreatConnect, Security Operations and Analytics Platform, https://www.threatconnect.com

[10] Guide to Privacy and Security of Electronic Health Information, April 2015, Accessed from https://www.healthit.gov/sites/default/files/pdf/privacy/privacy-and-security-guide.pdf

[11] A. Austin and L. Williams, "One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques" Proceedings of the 5th International Symposium on Empirical Software Engineering and Measurement, ESEM 2011, Banff, AB, Canada — September 22-23, 2011.

[12] E. Helms, L. Williams, "Evaluating Access Control of Open Source Electronic Health Record Systems" Proceedings of the 3rd Workshop on Software Engineering in Health Care, Pages 63-70, Waikiki, Honolulu, HI, USA — May 22 - 23, 2011.

[13] D. Bowers, The Health Insurance Portability and Accountability Act: is it really all that bad?, October 2011, Accessed from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1305898/

[14] Guide to Privacy and Security of Electronic Health Information, April 2015, Accessed from https://www.healthit.gov/sites/default/files/pdf/privacy/privacy-and-security-guide.pdf

[15] D. Allan, Web application security: automated scanning versus manual penetration testing, IBM Rational Software, Somers, White Paper 2008`

[16] RIPS- A static source code analyzer for vulnerabilities in PHP scripts, http://rips-scanner.sourceforge.net/

[17] RIPS (Re-Inforce PHP Security), https://en.wikipedia.org/wiki/RIPS

[18] W. Halfond, and A. Orso, "AMNESIA: Analysis and monitoring for NEutralizing SQL injection attacks," In Proceedings of International Conference on Automated Software Engineering, pp. 174-183 , 2005.

[19] D. Grunwel, T. Sahama, "Delegation of access in an Information Accountability Framework for eHealth" Proceedings of the Australasian Computer Science Week Multiconference, Article No. 59, Canberra, Australia — February 01 - 05, 2016.

[20] File inclusion attacks: http://resources.infosecinstitute.com/file-inclusion-attacks/#gref

[21] EA. Oladimeji, HT. Jung, E. Chung, and J. Kim, "Managing Security and Privacy in Ubiquitous eHealth Information Interchange" Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication, February 2011.

[22] J. King and L. Williams, "Log Your CRUD: Design Principles for Software Logging Mechanisms" Proceedings of the 2014 Symposium and Bootcamp on the Science of Security.

[23] E. Reinsmidt, D. Schwab, and L. Yang, "Securing a Connected Mobile System for Healthcare" Proceedings of the 2016 IEEE 17th International Symposium on High Assurance Systems Engineering.

[24] A. Tuikka, M. Rantanen, and O. Heimo, "Where is Patient in EHR Project?" ACM SIGCAS Computers and Society - Special Issue on Ethicomp: Volume 45 Issue 3, Pages 73-78, September 2015.

[25] D. Mashima and M. Ahamad "Enhancing Accountability of Electronic Health Record Usage via Patient-centric Monitoring" Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium

[26] KM. Kingsford, F. Zhang, MD. Nii Ayeh, and A. MaryMargaret, "A Mathematical Model for a Hybrid System Framework for Privacy Preservation of Patient Health Records" Proceedings of the 2017 IEEE 41st Annual Computer Software and Applications Conference.

[27] E. Kamsties. Understanding ambiguity in requirements engineering. In Engineering and Managing Software Requirements, pages 245–266. Springer Berlin Heidelberg, 2005.

[28] D. Popescu, Spencer Rugaber, Nenad Medvidovic, and Daniel M. Berry. Reducing ambiguities in requirements specifications via auto- matically created object-oriented models. In Proceedings of the 14th Workshop on Innovations for Requirement Analysis, pages 103–124, 2007.

[29] G. Wassermann, and Z. Su, "Static detection of cross-site scripting vulnerabilities," In proceedings of International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 171-180.

[30] O. Kafali, J. Jones, M. Petruso, L. Williams, and M. Singh, "How Good is a Security Policy against Real Breaches?" Proceedings of the 39th International Conference on Software Engineering, Pages 530-540, Buenos Aires, Argentina - May 20, 2017.

[31] Protecting Your Privacy & Security, Your Health Information Privacy, https://www.healthit.gov/patients-families/your-health-information-privacy

[32] R. Kam, "Top 3 issues facing patient privacy", Jul 2012, http://www.healthcareitnews.com/news/top-3-issues-facing-patient-privacy

[33] What is OpenClinic project? http://openclinic.sourceforge.net

[34] Unsafe Use of Reflection, https://www.owasp.org/index.php/Unsafe_use_of_Reflection

[35] Reflection in PHP, http://culttt.com/2014/07/02/reflection-php/

[36] What is local file inclusion (LFI): https://www.acunetix.com/blog/articles/local-file-inclusion-lfi/

[37] "Divide and Conquer - HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics" by Amit Klein, http://www.packetstormsecurity.org/papers/general/whitepaper_httpresponse.pdf

[38] HTTP Response Splitting, http://projects.webappsec.org/w/page/13246931/HTTP%20Response%20Splitting
[39] Control-Flow Security by William Patrick Arthur, https://web.eecs.umich.edu/~taustin/papers/Arthur_dissertation.pdf
[40] Anti-Subversion Software, https://en.wikipedia.org/wiki/Anti-Subversion_Software
[41] HIPAA Background, http://hipaa.bsd.uchicago.edu/background.html Feb 2010
[42] Legal Information Institute, 1992, https://www.law.cornell.edu/cfr/text/45/164.308
[43] HIPAA Security Checklist, https://www.ihs.gov/hipaa/documents/IHS_HIPAA_Security_Checklist.pdf
[44] OpenEMR, http://www.open-emr.org/
[45] OpenEMR, https://en.wikipedia.org/wiki/OpenEMR#cite_note-6
[46] https://sourceforge.net/projects/openemr/files/stats/timeline
[47] PHP Vulnerability Hunter: https://thehackernews.com/2011/11/php-vulnerability-hunter-v1146.html
[48] PHP Vulnerability Hunter Overview, https://www.autosectools.com/PHP-Vulnerability-Scanner
[49] Certification Guidance for EHR Technology Developers Serving Health Care Providers Ineligible for Medicare and Medicaid EHR Incentive Payments, https://www.healthit.gov/sites/default/files/generalcertexchangeguidance_final_9-9-13.pdf
[50] Farhadi, M., Haddad, H., Shahriar, H. M. (2018). In Y. Malleh (Ed.), Compliance of Electronic Health Record Applications with HIPAA Security and Privacy Requirements (pp. 199-213). Security and Privacy Management, Techniques, and Protocols. https://www.igi-global.com/chapter/compliance-of-electronic-health-record-applications-with-hipaa-security-and-privacy-requirements/202045
[51] Farhadi, M., Haddad, H., Shahriar, H. M. (2018). Static Analysis of HIPAA Security Requirements in Electronic Health Record Applications (pp. 474-479). Proc. of 42nd IEEE Annual Computer Software and Applications Conference (COMPSAC). https://ieeecompsac.computer.org/2018/
[52] Farhadi, M., Haddad, H., Shahriar, H. M. (2019). Compliance Checking of Electronic Health Record Applications for Security and Privacy Requirements. (under review) Proc. of 43rd IEEE Annual Computer Software and Applications Conference (COMPSAC). https://ieeecompsac.computer.org/2019/
[53] Farhadi, M., Haddad, H., Shahriar, H. M. (2017). Mitigation of Security Risks of Electronic Health Record Applications. Scientific Computing Day, https://technology.gsu.edu/scientific-computing-day/conference-archive/
[54] Health IT Security - Administrative Safeguards, https://healthitsecurity.com/news/a-review-of-common-hipaa-administrative-safeguards
[55] Health IT Security - Physical Safeguards, https://healthitsecurity.com/news/looking-back-at-hipaa-physical-safeguard-requirements
[56] Health IT Security, https://healthitsecurity.com/news/hipaa-technical-safeguards-basic-review