Towards Improved Security Criteria for Certification of Electronic Health Record Systems

Andrew Austin　Ben Smith　Laurie Williams
North Carolina State University
890 Oval Drive
Raleigh, NC 27695-8206 USA
+1 (919) 515-7926

andrew_austin@ncsu.edu　ben_smith@ncsu.edu　laurie_williams@ncsu.edu

ABSTRACT
The Certification Commission for Health Information Technology (CCHIT) is an electronic health record certification organization in the United States. In 2009, CCHIT's comprehensive criteria were augmented with security criteria that define additional functional security requirements. *The goal of this research is to illustrate the importance of requiring misuse cases in certification standards, such as CCHIT, by demonstrating the implementation bugs in an open source healthcare IT application.* We performed an initial evaluation of an open source electronic health record system, OpenEMR, using an automated static analysis tool and a penetration-testing tool. We were able to discover implementation bugs latent in the application, ranging from cross-site scripting to insecure cryptographic algorithms. Our findings stress the importance that certification security criteria should focus on implementation bugs as well as design flaws. Based upon our findings, we recommend that CCHIT be augmented with a set of misuse cases that check for specific threats against EMR systems.

Categories and Subject Descriptors
K.6.5 [**Management of Computing and Information Systems**]: Security and Protection.

General Terms
Security, Verification, Measurement.

Keywords
CCHIT, Security, Software Quality, Healthcare, EHR, Web Application, Static Analysis, Penetration Testing

1. INTRODUCTION
The Certification Commission for Health Information Technology (CCHIT) is a nonprofit organization founded in 2004 to certify electronic health record (EHR) systems in the United States. CCHIT maintains and periodically publishes criteria that are necessary for obtaining certification for both ambulatory (outpatient) and inpatient EHR systems. The purpose of CCHIT certification is to confirm that certified EHR systems maintain a minimum level of correctness, reliability, security, and interoperability. These characteristics are known as software quality factors [1].

In this research, we focus on one software quality factor, security, because of the inherent requirement of privacy associated with the sensitive and personal information contained within EHRs. In the United States, the American Recovery and Reinvestment Act of 2009 (ARRA) provides monetary incentives to medical providers for using EHR systems rather than paper counterparts [2]. Since 2006, the U.S. Department of Health and Human Services (HSS) has designated CCHIT as a Recognized Certification Body [3]. CCHIT is currently modifying its criteria to fullfill the ARRA defined requirements and is working closely with HSS, the HIT Policy Committee, and the Office of the National Coordinator for Health Information Technology to influence national EHR certification policy. In October 2009, CCHIT augmented its certification criteria with additional security criteria and provided corresponding black box test scripts[1]. These criteria provide specific recommendations intended to establish a minimum level of securty of an EHR system.

McGraw divides security faults into two important groups: **design flaws**, which are high-level problems associated with the architecture of the software; and **implementation bugs**, which are code-level software problems [4]. Security faults from each group generally occur with the same frequency as the other in any given software project [4]. After analyzing the test scripts and the associated security criteria, we have determined that the CCHIT criteria only address some design flaws and ignore possible implementation bugs completely. For example, one CCHIT criteria states that "When passwords are used, the system shall support case-sensitive passwords that contain typeable alpha-numeric characters in support of ISO-646/ECMA-6 (aka US ASCII)." Another CCHIT criteria states that "[t]he system shall provide the ability for authorized administrators to assign restrictions or privileges to users/groups". None of the other 54 non-documentation related CCHIT security criteria test for any potential implementation bugs.

[1]http://www.cchit.org/sites/all/files/CCHIT%20Certified%202011%20Security%20Test%20Script%2020091019.pdf

The goal of this research is to illustrate the importance of requiring misuse cases in certification standards, such as CCHIT, by demonstrating the implementation bugs in an open source healthcare IT application. In this paper, we present the results of our initial evaluation of OpenEMR[2], an open source EHR application that is seeking CCHIT certification. Our implementation level evaluation consists of analyzing the results of two web application security tools: IBM's Rational AppScan[3], which performs automated penetration testing; and Fortify 360[4], which performs security-focused static analysis.

The rest of the paper is organized as follows. Section 2 describes background pertaining to our paper. Section 3 outlines our methodology. Section 4 describes our results. Section 5 discusses some limitations with our approach. Section 6 provides a summary of our conclusions.

2. BACKGROUND

This section defines several key terms that are useful throughout the reading of our paper.

An **electronic health record** is a collection of medical information about individual patients and populations stored in an electronic manner, which may contain anything from a patient's home address and social security number to the fact that a patient has been diagnosed with a certain disease [5].

Static analysis examines software in an abstract fashion by looking at the code without executing it [6]. This examination can be performed by evaluating either source code, machine code, or object code of an application to obtain a list of potential faults found within the source. Static analysis can be done manually, or with the help of automated tools. Since programming languages are all quite different, a variety of tools are commonly used, although many modern tools provide support for a range of languages. Unfortunately, these tools are not perfect and they sometimes incorrectly label code as containing a fault. This mislabeling is called a **false positive,** as opposed to a **true postive**, when faults are correctly identified. Therefore, developers must manually examine each potential fault reported by these tools in order to determine if they are false positives. We call potential faults that have security implications **potential vulnerabilities.**

Penetration testing is a process in which a set of malicious tests are created to ensure that a software system does not violate any of the security policy's requirements [7]. Penetration testing asks that the tester "think like an attacker" and devise test cases which try to break the system's integrity by exposing vulnerabilities such as SQL injection, cross-site scripting, cross-site request forgeries, and error message information leakage.

Application security is frequently assessed and specified using **misuse cases [8]**. Similar to use cases, misuse cases specify "negative" use cases, that is: behavior that is *not* wanted in the proposed system. Examples of misuse cases include an unregistered user attempting to access the system; a patient trying to steal another patient's records; or a healthcare administrator trying to steal a patient's personal information. Misuse cases cause developers to ask questions such as "Who should have access to a patient's records?" as well as "What parts of a doctor's personal information should be available to patients?"

[2] http://www.oemr.org

[3] http://www-01.ibm.com/software/awdtools/appscan/

[4] http://www.fortify.com/products/fortify-360/

3. METHOD

This section describes our methodology for selecting and evaluating the target system OpenEMR. We performed the evaluation using Windows Vista Business, Service Pack 2, on a virtual machine with a 2.65Ghz Intel Core Duo and 1.00GB of RAM. OpenEMR was configured to run using Apache 2.2, MySQL v5.1.8, and PHP v5.2.11.

3.1 Subject Selection

OpenEMR is an open source EHR system licensed under the GPL[5]. In June 2009, OpenEMR was listed as one of the top ten community based open source health care projects, according to Black Duck Software[6]. The project has a community of 17 contributing developers[7] and at least 11 companies providing commercial support within the United States[8]. OpenEMR is also actively pursing CCHIT certification[9]. These facts make OpenEMR an ideal candidate to evaluate because of the ease in which one can access both the source code and support resources. Table 1 lists some additional characteristics of OpenEMR.

Table 1. Characteristics of OpenEMR

Language	PHP
Version Evaluated	3.1.0 (8/29/2009)
Lines of Code (counted by CLOC1.08[10])	277,702

3.2 Static Analysis

We performed automated static analysis on OpenEMR using the static analysis tool Fortify 360 v5.7. Fortify 360 is a tool focused on security and is able to analyze a variety of languages, including both PHP and Java, which is why it was selected over other static analysis tools. The application was analyzed with the options "Show me all issues that may have security implications" and "No, I don't want to see code quality issues" to only detect potential vulnerabilites. Once the automated analysis was completed, two researchers independently examined each potential vulnerability and its corresponding source code to classify it as either a true positive or a false positive. Once each potential vulnerability was independently categorized, the two researchers compared their findings. In the event of a disagreement, the researchers examined the potential vulnerability's source together and debating their opinion until a consensus was reached on the validity of each potential

[5] http://www.gnu.org/licenses/gpl.html

[6] http://www.marketwire.com/press-release/Black-Duck-Software-1002070.html

[7] http://sourceforge.net/project/memberlist.php?group_id=60081

[8] http://www.openmedsoftware.org/wiki/OpenEMR_Commercial_Help

[9] http://www.openmedsoftware.org/wiki/OpenEMR_Certification

[10] http://cloc.sourceforge.net/

vulnerability. Once this consensus was reach, researchers compared the vulnerability against the CCHIT security tests scripts to assess whether a test script could have surfaced the identified vulnerability.

Figure 1 presents an example of a SQL injection vulnerability (bolded) that Fortify 360 detected and labeled as "SQL Injection (Input Validation and Representation, Data Flow) ". Figure 1 is an example of a static analysis true positive.

```
<?
     $name = $_POST['name'];
     $query = "SELECT id, amount FROM users
WHERE name = '$name'";
     $result = mysql_query($query);
?>
```

Figure 1. SQL Injection Example True Positive (PHP)

Figure 2 shows an example of what Fortify 360 labeled (line bolded) as a "Password in Comment – Hardcoded passwords can compromise security in a way that cannot be easily remedied." Figure 2 is a false positive because there is no hardcoded password contained within the code comments, instead the tool simply detects the usage of the word 'password' in the code comments.

```
") VALUES ( "                                      .
     "''', "                                       .
// username
     "''', "                                       .
// password
```

Figure 2. SQL Injection Example False Positive (PHP)

3.3 Automated Penetration Testing

To conduct automated penetration testing, we used IBM Rational AppScan v7.8. Rational AppScan performs security testing of web applications, regardless of implementation language or platform. As with the Fortify analysis, two researchers independently examined each potential vulnerability and its corresponding source code to classify it as either a true positive or a false positive. Once each alert was independently categorized, the two researchers compared their findings. In the event of a disagreement, the researchers examined the potential vulnerability's source together and debated their opinion until a consensus was reached.

AppScan was set to scan starting from the OpenEMR's login page. AppScan allows the tester to configure a login policy, which essentially consists of a series of recorded HTTP exchanges to "teach" the tool how to gain authorized access to the system. We configured AppScan to check for "Application Only" tests, which excludes "Infrastructure Tests," which are targeted directly at specific application servers or frameworks, such as Apache Tomcat or Wordpress. We informed AppScan prior to the scan that the application under test was written using PHP, and used MySQL on the backend.

Although AppScan uncovered security issues with OpenEMR, such as the **Directory Listing Pattern** vulnerabilitiy type, not every vulnerability AppScan reported was a true positive. One example we encountered of a false positive was the **Email Address Pattern** vulnerability type, which AppScan uses to search for anything in HTTP responses coming from a web application that may resemble e-mail addresses. A webform used in OpenEMR for controlling batch communication contained an example email address of `your@example.com`, which was not actually a security vulnerability.

4. RESULTS

This section describes the results of our evaluation. A more updated version of our results containing studies on other open source healthcare applications and using a more varied selection of tools can be found on our publicly-accessible wiki[11].

4.1 Static Analysis

Using Fortify 360, we discovered 1,210 potential vulnerabilities related to security with OpenEMR. After the removal of false positives, we determined that there were 440 true positive implementation flaws that would not be detected by the CCHIT certification security test scripts. Table 2 presents a summary of the data we collected.

Table 2. Static Analysis Summary of OpenEMR

Measure	Value
Total Alerts	1210
True Positives	440
False Positives	770
False Positive Rate	63.64%

These 440 true positive vulnerabilities were broken down into the following types, appearing in order of frequency of occurrence:

- **Cross-Site Scripting (215)** – input is taken from a user and not correctly validated, allowing for malicious code to be injected into a web browser.
- **Nonexistent Access Control (129)** – access to a particular URL is not protected, granting anyone access.
- **Dangerous Function (24)** – methods used within the code are inherently insecure or deprecated and they should not be used.
- **Path Manipulation (20)** – input from users is directly passed to the filesystem allowing attackers to manipulate or read normally inaccessible files.
- **Error Information Leak (19)** – system or error information containing sensitive data is displayed to the user.
- **Global Variable Manipulation (9)** – attackers can manipulate the application's global variables.
- **Insecure Upload (8)** – file uploads are not properly validated, allowing attackers to upload malicious files.
- **Improper Cookie Use (7) –** sensitive information is stored in a persistent cookie, or the cookie failed to set the HttpOnly flag to mitigate Cross-Site scripting.
- **HTTP Header Manipulation (4)** – attackers can manipulate the HTTP response headers.

[11] http://agile.csc.ncsu.edu/healthcare

- **Hidden Field Manipulation (3)** – hidden fields are not properly validated, allowing attackers to manipulate application function.
- **Command Injection (2)** – input from users is directly executed, allowing malicious users to execute commands on the host.

As an example, one true positive was the lack of a "user specific secret in order to prevent an attack." The file admin.php, a source file belonging to the events calendar component, requires no authentication for the import and export of data.

Another example of a vulnerability found by fortify is the use of an insecure or deprecated method. One method, `mysql_escape_string()` was used four times throughout the OpenEMR codebase. This method does not properly escape input by taking into account the current character set and is deprecated and even removed in the most recent releases of PHP[12]. Figure 3 shows one example of OpenEMR using this deprecated code (bolded).

```
foreach ($_POST as $k => $var) {
      if (! is_array($var)) $_POST[$k] =
      mysql_escape_string($var);
      echo "$var\n";
    }
```

Figure 3. mysql_escape_string() deprecated method (PHP)

4.2 Automated Penetration Tests

Using Rational AppScan, we discovered 140 potential vulnerabilities with OpenEMR. After the removal of false positives, we determined that there were 130 implementation flaws that would not be detected by the CCHIT certification security test scripts. Table 3 presents a summary of the data we collected.

Table 3. Automated Penetration Test Summary of OpenEMR

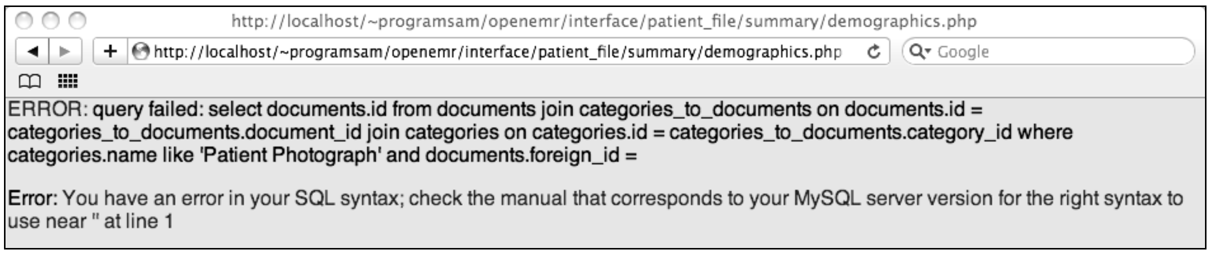
Measure	Value
Total Alerts	140
True Positives	130
False Positives	10
False Positive Rate	7.14%

These 130 true positive vulnerabilities were broken down into the following types, appearing in order of frequency of occurrence:

- **Cross-Site Scripting (50)** – similar to the first example above.
- **Phishing Through Frames (25)** – in which a parameter was used to inject an embedded frame with a request to an off-site URL.
- **Cross-Site Request Forgery (22)** – in which AppScan demonstrated it was possible to generate an unauthorized request to any page in the system by creating a mock version of an HTML form.
- **Error Message Information Leak (14)** – as in the example presented in Figure 4.
- **SQL Injection (4)** – input from users is directly used in SQL queries, allowing attacks to read and manipulate the database in unintended ways.
- **JavaScript Cookie References (6)** – in which portions of the application's logic were controlled via a JavaScript set cookie, which could easily be disabled if the user chooses to turn off JavaScript in their browser.
- **Directory Listing (6)** – attackers can view the contents of individual directories.
- **Password Not Encrpyted (2) –** password not sent over SSL, allowing attackers to more easily intercept them.
- **Path Disclosure (1)** – information about the system path is leaked to the attacker.

One vulnerability type AppScan discovered was **Error Message Information Leakage** that displayed the entire structure of a SQL query (see Figure 4). The page a practitioner would use to view a patient's personal information, demographics.php, is one example. AppScan was able to set the parameter `set_pid`, which controls the logic of the select query used to browse through patient information, to `null`, which caused OpenEMR to display the result in Figure 4. Such a fault is a dangerous result because it allows the attacker to know what the structure of the query looks like, which makes future SQL injection attacks easier.

Another vulnerability type discovered was **Cross-Site Scripting**. An extended demographics display page, demographics_full.php (Figure 5), has a parameter for `set_pid`,

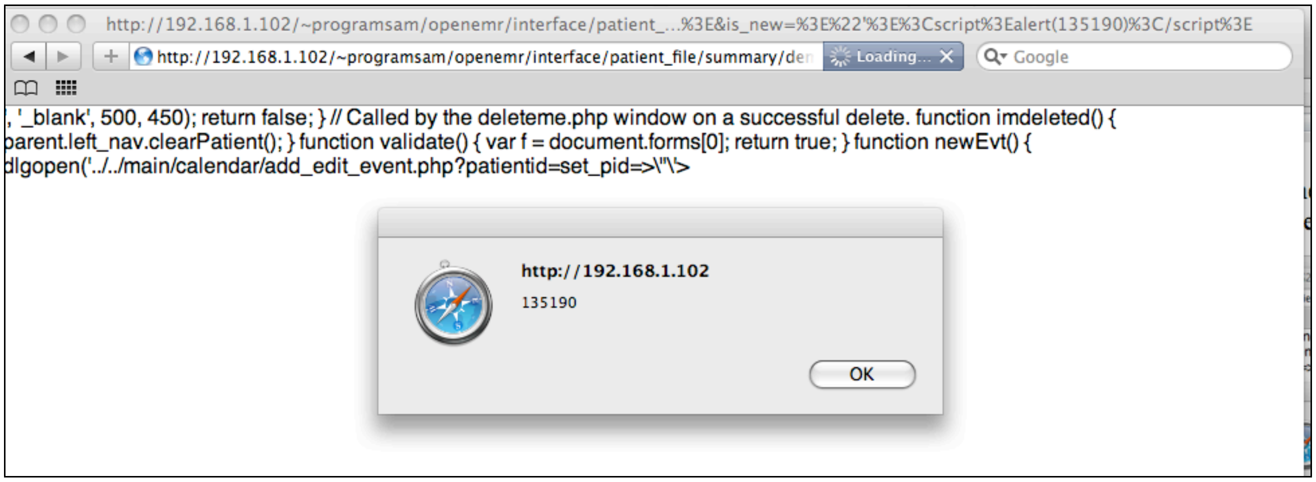

http://localhost/~programsam/openemr/interface/patient_file/summary/demographics.php
http://localhost/~programsam/openemr/interface/patient_file/summary/demographics.php
Google

```
ERROR: query failed: select documents.id from documents join categories_to_documents on documents.id =
categories_to_documents.document_id join categories on categories.id = categories_to_documents.category_id where
categories.name like 'Patient Photograph' and documents.foreign_id =

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to
use near '' at line 1
```

Figure 4. Error Message Information Leak in OpenEMR

[12] http://php.net/manual/en/function.mysql-escape-string.php



```
http://192.168.1.102/~programsam/openemr/interface/patient_...%3E&is_new=%3E%22'%3E%3Cscript%3Ealert(135190)%3C/script%3E
http://192.168.1.102/~programsam/openemr/interface/patient_file/summary/den
, '_blank', 500, 450); return false; } // Called by the deleteme.php window on a successful delete. function imdeleted() {
parent.left_nav.clearPatient(); } function validate() { var f = document.forms[0]; return true; } function newEvt() {
dlgopen('../../main/calendar/add_edit_event.php?patientid=set_pid=>\"\>
```

Figure 5. Cross-Site Scripting in OpenEMR

which is the parameter required for the patient's identifier. AppScan set this parameter to

```
>"'><script>alert(135190)</script>&is_new=>"'><s
cript>alert(135190)</script>
```

and was able to cause OpenEMR to execute this script, thus indicating that the application is vulnerable to Cross-Site Scripting.

Of the 130 true positives found by AppScan, 61 were also found previously by Fortify. This overlap in results acts as additional confirmation of the validity of these true positives.

5. LIMITATIONS

The results reported by automated tools may not be entirely comprehensive. Future testing could use additional tools to discover faults not originally detected. Researchers examined each potential vulnerability and determined if they were either true or false positives. This examination could possibly introduce human error, which could affect our results. The application we studied, OpenEMR, is not currently CCHIT-certified. We only conducted one case study on one software project, which could potentially not be representative. Future studies should investigate the security posture of other open source healthcare applications and help to introduce new criteria for CCHIT certification.

6. CONCLUSIONS

Our results revealed that many of the errors, including numerous input validation vulnerabilities, are disregarded in the existing CCHIT criteria and test scripts.

Consider a misuse case that could be created for the situation where a patient creates his or her own web form to edit another patient's records using the page demographics.php within OpenEMR. The resultant security requirement from this misuse case would indicate that the page demographics.php should contain a user-specific secret, created at runtime, to prevent an attack of this type. This implementation bug is taken from an example that is detected by Fortify 360 and can be seen in our results.

A generalized form of this misuse case should be included in the CCHIT security criteria. Rather than a misuse case that specifically describes an attack on demographics.php, a black box, high-level misuse case can be written to capture an attack pattern that would occur for all electronic health records system. In this example, the misuse case would read something similar to "*A patient attempts to modify another patent's demographics that he or she is not authorized to view or edit*.".

This change to the security certification criteria for CCHIT would help instill secure coding practices by encouraging developers to build security in early in the development process, a philosophy supported by McGraw [4]. Rather than waiting until late in the development cycle to execute tools such as AppScan and Fortify, CCHIT misuse cases will encourage developers of EHR system think about security early in the software lifecycle, as well as throughout the lifecycle, to ensure that EHR systems protect our health records.

7. FUTURE WORK

Our future work will specify and analyze a set of black box test cases for the implementation bugs discovered and tested by AppScan and Fortify. We will also generate misuse cases that simulate attack patterns that have been common among both web and client applications in recent years. We also plan to further evaluate OpenEMR to determine how well it will score against CCHIT's test scripts.

8. ACKNOWLEDGMENTS

We would like to thank Niraj Deosthali and Jerrod Lankford for their assistance in eliminating false positives from our potenial security faults. This work is supported by the Agency for Healthcare Research Quality as well as an IBM Faculty award. Additionally, this work is supported by the National Science Foundation Research Experience for Undergraduates No. 0937692. Any opinions expressed in this material are those of the authors and do not necessarily reflect the view of the National Science Foundation.

9. REFERENCES

[1] J. Cavano, and J. McCall, “A framework for the measurement of software quality,” in Software quality workshop on functional and performance issues, 1978, pp. 133-139.

[2] E. Singer, "A Big Stimulus Boost for Electronic Health Records," *Technology Review*, MIT, 2009.

[3] H. P. Office. "HHS Officially Recognizes Certification Body to Evaluate Electronic Health Records," 1/18/2010, 2010; http://www.hhs.gov/news/press/2006pres/20061026a.html.

[4] G. McGraw, *Software Security: Building Security In*: Addison-Wesley Professional, 2006.

[5] T. Gunter, and N. Terry, “The Emergence of National Electronic Health Record Architectures in the United States and Australia: Models, Costs and Questions,” *Journal of Medical Internet Research,* vol. 7, no. 1, 2005.

[6] N. Ayewah, D. Hovermeyer, J. D. Morgenthaler *et al.*, “Using Static Analysis to Find Bugs,” *IEEE Software,* vol. 25, no. 5, pp. 22-29, 2008.

[7] B. Arkin, S. Stender, and G. McGraw, “Software penetration testing,” *IEEE Security & Privacy,* vol. 3, no. 1, pp. 84-87, 2005.

[8] G. Sindre, and A. Opdahl, “Eliciting requirements with misuse cases,” *Requirements Engineering,* vol. 10, no. 1, pp. 34-44, 2005.

[9] F. Swiderski, and W. Snyder, *Threat Modeling*, Redmond, WA: Microsoft Press, 2004.