Technological University Dublin
ARROW@TU Dublin

Dissertations

School of Computing

2018

Automation of Authorisation Vulnerability Detection in Authenticated Web Applications

Niall Caffrey
Technological University Dublin

Follow this and additional works at: https://arrow.tudublin.ie/scschcomdis

 Part of the Computer Sciences Commons

Recommended Citation

Caffrey, Niall (2018). *Automation of authorisation vulnerability detection in authenticated web applications.* Masters dissertation, DIT, 2018.

This Dissertation is brought to you for free and open access by the School of Computing at ARROW@TU Dublin. It has been accepted for inclusion in Dissertations by an authorized administrator of ARROW@TU Dublin. For more information, please contact yvonne.desmond@tudublin.ie, arrow.admin@tudublin.ie, brian.widdis@tudublin.ie.

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License

Automation of authorisation vulnerability detection in authenticated web applications

Niall Caffrey

C11508443

A dissertation submitted in partial fulfilment of the requirements of
Dublin Institute of Technology for the degree of
M.Sc. in Computing (Security & Forensics)

2018

DECLARATION

I certify that this dissertation which I now submit for examination for the award of MSc in Computing (Security & Forensics), is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the test of my work.

This dissertation was prepared according to the regulations for postgraduate study of the Dublin Institute of Technology and has not been submitted in whole or part for an award in any other Institute or University.

The work reported on in this dissertation conforms to the principles and requirements of the Institute's guidelines for ethics in research.

Signed: 

Date: 14/06/18

ABSTRACT

In the beginning the World Wide Web, also known as the Internet, consisted mainly of websites. These were essentially information depositories containing static pages, with the flow of information mostly one directional, from the server to the user's browser. Most of these websites didn't authenticate users, instead, each user was treated the same, and presented with the same information. A malicious party that gained access to the web server hosting these websites would usually not gain access to confidential information as most of the information on the web server would already be accessible to the public. Instead, the malicious party would typically modify the files that are on the server in order to deface the website or use the server to host pirated materials.

At present, the majority of websites available on the public internet are applications; these are highly functional and rely on two-way communication between the client's browser and the web server hosting the application. The content on these applications is typically generated dynamically, and is often tailored towards each specific user, with much of the information dealt with being confidential in nature.

A malicious party that compromises a web application, and gains access to confidential information which they normally should not be able to access, may be able to steal personal client information, commit financial fraud, or perform other malicious actions against those users whose personal information has been leaked.

This thesis seeks to examine the access controls that are put in place across a variety of web applications that seek to prevent malicious parties from gaining access to confidential information they should not be able to access. It will test these access controls to ensure that they are robust enough for their purpose, and aims to automate this procedure.

Key words: *Web Application Security, Security Requirements, Access Controls, Authorisation, Automation*

ACKNOWLEDGEMENTS

I would like to first express my thanks to my thesis supervisor Dr. Diana Carvalho E Ferreira of the School of Computing at Dublin Institute of Technology, who has been an incredible source of knowledge and support throughout the process.

I would also like to thank Brían Ward for his help in developing the scanner which was used for this experiment, and David Kennefick and Dearbhail Kirwan for their advice throughout the process.

Finally I would like to thank my friends and family for their support throughout the masters, without who I would never have gotten this far.

GLOSSARY OF TERMS

The following is a list of some of the terms that will be used throughout this paper, along with their meanings:

API – Application Programming Interface is a set of defined interfaces through which interactions happen between an enterprise and applications which uses its assets.

CSP – Cloud Service Provider is a company that offers network services, infrastructure, or business applications in the cloud.

SLA – Service Level Agreement is an agreement between a service provider and a client.

Request – A request is sent whenever a user visits a URL, or performs an action on a website or web application.

Response – A response is received for every request sent, this contains the data which was being requested, such as the page being requested in a URL.

URL – Uniform Resource Locator, these are the unique identifiers by which all pages on a website are known, and through which these pages can be retrieved for viewing.

Method – These are the different ways requests can be sent to interact with a web server, the two most common are GET, which is designed to retrieve resources from a web page, and POST which is designed to perform actions on the web server such as sending a contact us form, or transferring money from one account to the next.

Cookie – Cookies are a piece of data which is given to a client session once they visit a website or web application, cookies can have many uses such as tracking a user on a particular website, storing information regarding the session such as a client's basket in an online shop, or tracking the session of a user who has logged into the website or web application.

Header – These are parts of the requests and response which provide both the client and web server with information about each other.

GUI – Graphical User Interface

JavaScript – JavaScript is a simple programming language which can be used to extend the functionality of a website or web application.

XML – Extensible Markup Language is a specification for encoding data in a way that a machine can easily read.

Proxy – A proxy is a server which mediates access between the client browser and the web server hosting a web application or website.

Virtual Machine – a virtual machine is an emulation of a computer system, it sits on top of the normal operating system of a computer, allowing the user to access multiple different operating systems. For example a Windows machine could run a Linux virtual machine, or vice versa.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

1.1 Background

Currently, web applications form an incredibly important and crucial part of many businesses worldwide asset portfolio. The most recent estimates by Internet Live Stats[1] are that there is between 1.8 and 1.9 billion live websites, on the internet at the time of writing.

With the constantly increasing reliance on technology such as, smart phones, mobile apps and web applications, there has been a change in the attack surface for hackers. Some of these web and mobile applications handle incredibly valuable data and important, sensitive information about a company's clients.

It is due to the sensitive information that they hold, the lack of secure knowledge by the developers of said applications, and the increased demand for businesses to have an online presence, which makes websites, and web applications, prime targets for malicious attackers.

The increased risks has led to the rise of web application security, which is the practice of discovering, validating and remediating security vulnerabilities and issues within any given website. This has been strongly championed by the Open Web Application Security Project (OWASP), a not for profit organisation seeking to improve the state of web application security worldwide.

In their recent update to the OWASP Top Ten[2] – a list of the top ten security risks faced by a web application, OWASP listed Broken Access Control, as the fifth most serious risk.

Broken Access Control refers to the improper configuration of restrictions on what actions authenticated users are allowed to perform. This can allow a malicious user of the web application access to unauthorised functionality or data, which can include other users' personal information; including details such as usernames, passwords, and bank account or credit card details, among many others.

[1] http://www.internetlivestats.com/total-number-of-websites/

[2] https://www.owasp.org/index.php/Top_10-2017_Top_10

As part of a penetration test carried out on a web application by a computer security expert, the testing for Broken Access Control, more commonly known as authorisation testing, can be time consuming and repetitive.

There are many ways in which the testing for broken access controls may be done, the most common of which is to identify application functionality which only certain user roles should have access to, and then to attempt to access this functionality using a user role which should not have access, typically through browsing to the URL which hold the functionality.

1.2 Research Description

The purpose of this project is to examine whether or not the process of performing authorisation tests – as described above in the background, can be automated, resulting in an accurate picture of where any possible authorisation issue may occur.

For the purpose of this experiment, three plug-ins for the commercial tool Burp Suite – which will be discussed in further detail in Chapter 3, were identified that are designed to aid computer security experts in identifying authorisation vulnerabilities in authenticated web applications.

In addition to these plug-ins, a custom python script was developed that aims to automate the testing that the identified plug-ins perform.

Both the custom python script and the identified commercial plug-ins will be used to test a group of websites for the purpose of finding authorisation vulnerabilities. The results from both the plug-ins and the custom python script will then be manually validated by a group of experts, in order to determine how much of the identified vulnerabilities are real.

The manually validated results will then be compared, in order to determine if the custom python script – representing the automation of detection for these vulnerabilities, can provide results which are as accurate as the identified plug-ins – representing the manual testing for these vulnerabilities.

1.3 Research Objectives

The aim of this experiment can be summarised into the following question:

Will a custom developed web application scanner, provide a more accurate analysis than plug-ins developed for commercial grade software in the detection of authorisation vulnerabilities?

This can be further expanded on by the following statement:

A custom developed web application scanner focused on testing for authorisation vulnerabilities in web applications, will provide a more accurate (in terms of the number of false positives – detected vulnerabilities which after review are not real vulnerabilities, false negatives – vulnerabilities that aren't detected which after review are real vulnerabilities, and true positives – detected vulnerabilities which after review are determined to be real vulnerabilities - of detected authorisation vulnerabilities) analysis of authorisation vulnerabilities than plug-ins developed for commercial grade software designed to test for authorisation vulnerabilities (the AuthMatrix, Authz, and Autorize plug-ins for Burp Suite Pro).

The objectives of this dissertation can be broken down into:

1. Identify other previous work in this area, and determine whether there is a potential avenue of experimentation, which has not been considered previously.
2. Use the identified plug-ins and the custom python script to identify potential authorisation vulnerabilities in a group of websites.
3. To manually validate the results of the tools, through the use of experts in the field of web application security, in order to identify which of the results are valid authorisation vulnerabilities, and which are in fact false positives.
4. To compare the manually validated results, in order to answer the question stated above.

1.4 Research Methodologies

There are multiple research methods that will be employed for the duration of this experiment. Initially a literature review will be performed of the previous research in this area of study. This will provide an overview of what has previously been done in this area, as well as identify any possible avenues not followed in previous experiments.

The next method which will be employed can be considered *Primary Research*, instead of utilising existing data sets in order to perform the experiment, the data sets needed will be created by achieving the second objective stated above during the experiment.
The results generated during the initial portion of the experiment are considered *quantative* in nature, as there will be a large number of detected authorisation vulnerabilities which are classified as, false positives, true positives, false negatives, or true negatives.
Finally *Empirical Research* and *Deductive Reasoning* will be performed on the gathered data, in order to determine whether the overall aim of the experiment has been achieved.

1.5 Scope and Limitations

The literature review which took place was limited to five topics of research, which are related to, and have some association with the main topic of the experiment.
The web applications – as will be discussed in more detail in Chapter 3, was limited to thirty-five authenticated applications, which were then split further into three classifications:

1. Applications with known authorisation vulnerabilities
2. Applications which are known to not contain authorisation vulnerabilities
3. Applications where it is unknown whether or not they contain authorisation vulnerabilities

The identified plug-ins are additions for the popular Burp Suite – a combined web proxy and web application vulnerability scanner, which are available for both free and professional versions. The version of Burp Suite which was used for the purposes of this experiment was the paid professional version.
The custom script was written using python in the PyCharm Python IDE (Integrated Development Environment), using a student license.

1.6 Document Outline

The dissertation that follows is organised under the following chapters:

Chapter 2 – Literature Review and Related Work

This chapter details the initial phase of this experiment, an examination of existing experiments that cover similar areas of research. These have been broken down into five topics of research; Information Accountability and Usage Control, Web Application Security, Web Application Security Requirements, Web Application Access Control, and finally Web Application Authorisation.

Chapter 3 – Design and Methodology

This chapter details the experiment that was performed, how it was designed as well as detailing the tools used, and some of the web applications tested during the experiment.

Chapter 4 – Implementation and Results

This chapter details the implementation of the experiment, how the custom scanner works, and also providing details regarding the applications where it is unknown if they contain authorisation vulnerabilities. This chapter will also detail the results which were retrieved during the course of the experiment.

Chapter 5 – Evaluation, Analysis, and Discussion

This chapter will evaluate, analyse, and discuss the results of the experiment. This will be done in the context of comparing the results of the tests performed on the vulnerable web applications, as well as comparing the results of the tests performed on the applications where it is unknown if they contain vulnerabilities. The overall results will then be used to compare the different scanning tools that were used in the experiment: the three plug-ins for the commercial tool – Burp Suite, and the custom written scanner developed for use in this experiment.

Chapter 6 – Conclusion

The final chapter of the dissertation will contain a brief overview of the experiment to date, along with any conclusions that that have been reached. This chapter will also outline recommendations for future work in this area.

2. LITERATURE REVIEW AND RELATED WORK

2.1 Introduction

In this chapter an examination will be performed of existing experiments, and research that was previously performed in this area of research. To further this examination, the area of research was broken down into five related areas, as follows:

- Information Accountability and Usage Control
    - This portion focuses on one of the rationales behind web application access control, and the growing relevance of this in the current climate. Both the upcoming General Data Protection Regulation and the recent scandal involving Facebook and Cambridge Analytica, highlight the growing need for strong web application access control
- Web Application Security
    - This portion focuses on providing a general overview of web application security and its importance, of which access control or authorisation is large component.
- Web Application Security Requirements
    - This portion discusses the security requirements that web applications today should be meeting.
- Web Application Access Control
    - This portion discusses web application access control, the different types, and current flaws in access controls that may provide a malicious person with the means to recover information which they shouldn't have access.
- Web Application Authorisation
    - This portion discusses web application authorisation

2.2 Information Accountability and Usage Control

Accountability can be said to be formed of the procedures and processes which a party uses to justify and take responsibility for its actions. In terms of information accountability, this can be interpreted as the users of the information are held liable to

explain, justify, or answer for their use of information when requested to by the party that information belongs to (Gajanayake, Iannelle, & Sahama, 2011).

One of the original reasons behind web application access control was to provide a means of accountability and control over sensitive information, which many web applications store as a necessary part of their lifecycle about their users. Information accountability and usage control has grown increasingly relevant in the current climate, especially following on from the recent Cambridge Analytica controversy, and from the upcoming General Data Protection Regulation (GDPR) which will come into force from 25th May 2018.

The General Data Protection Regulation or GDPR for short was approved by the European Union Parliament on 14th April 2016. It was designed to replace the outdated Data Protection Directive 95/46/EC which was established in 1995, and is intended to harmonise data privacy levels throughout Europe. The key principles remain the same between the Data Protection Directive and GDPR;

- that data is fairly and lawfully process,
- that it is process for limited purposes,
- that the data is adequate, relevant, and not excessive
- that the data is kept accurate
- that the data is not kept for longer than is required
- that the data is processed within the client's rights
- that the data is kept secure
- And finally that the data is not transferred to other countries that do not have adequate protection.

However there are some changes which will be highlighted that will have far reaching consequences for companies across the globe. Three key changes are in regards to increased territorial scope, penalties, and consent[3]. The most significant of these changes are the increased territorial scope and, regulatory landscape of the regulation.

As part of GDPR all companies processing the personal data of clients residing within the European Union, regardless of the company's physical location, are subject to GDPR. The Data Protection Directive was ambiguous in this regard and the issue arose

[3] https://www.eugdpr.org/key-changes.html

in many high profile court cases. Another key change within the legislation concerns the penalties which companies failing to comply with GDPR will be hit with. Under GDPR, an organisation found to be in breach of legislation can be fined up to 4% of annual global turnover, or €20 million, whichever is found to be higher. Significantly these penalties apply to both controllers of the personal data, and the processors of it, meaning that cloud hosting providers will not be exempt from this regulation. The last key change which will be touched on in this dissertation, is regarding consent. Under GDPR, informed consent regarding the processing of personal data must be obtained from all users, as a result data and privacy policy must be easily accessible and presented in such a manner as to be understood by the average user. Each user must also be able to easily withdraw consent at any time.

A key aspect in ensuring that a company is complying with the incoming rules under GDPR is enacting proper access control mechanisms in regards to client information. It is important for security and data protection that only company officers who need access to client data have this access and ensuring that they only have access to the personal data of clients needed for their job.

Pato, Paradesi, Jacobi, Shih, & Wang (2011) state that access controls models falls short in two ways when dealing with potential damage from information leakage. These two ways are:

1. It relies on the ability to anticipate who should have access
2. It assumes that an authorised user will make appropriate and compliant use of the information.

The second of these short comings can be seen in the recent Cambridge Analytica controversy, wherein they were given access through Facebook to the personally identifiable information of up to 87 million Facebook users[4]. This data was then allegedly used to influence voting opinions in the countries of politicians that hired Cambridge Analytica[5].

[4] https://www.theguardian.com/technology/2018/apr/04/facebook-cambridge-analytica-user-data-latest-more-than-thought

[5] https://www.theguardian.com/uk-news/2018/mar/19/cambridge-analytica-execs-boast-dirty-tricks-honey-traps-elections

There has recently been a worrying trend of large data breaches being disclosed, from the recent Yahoo security breaches which occurred in 2013 and 2014 but were not publicly disclosed until 2016, to the recent Verizon breach which occurred in 2017, more and more companies are being attacked by malicious users. In fact this problem has been ongoing for the last eleven years, as can be seen in the 2007 paper *Information Accountability*, in it the author Daniel Weitzner states that disclosure of sensitive personal data, revelation of corporate secrets, distribution of copyrighted material, the sharing of confidential records among organizations in violation of regulation and policy, are the kind of breaches of established social norms and laws that are now considered to have become a part of everyday life. (Weitzner, *et al.*, 2007)

2.3 Web Application Security

With web applications, and indeed mobile applications, becoming ever increasingly integrated into the daily activities of the typical internet user, the security of these applications has increased in importance as well. As far back as 2008, the trend of an increasing reliance on web applications was noticed, as can be seen in the 2008 paper *Dynamic Test Input Generation for Web Applications*, wherein the author Gary Wasserman states that web applications continue to offer more features, handle more sensitive data, and generate content dynamically based on more sources as users increasingly rely on them for daily activities. (Wasserman, Yu, Dhurjati, Inamura, & Su, 2008)

Additional challenges have also been introduced to web applications, through the proliferation of cloud computing among modern web applications. Cloud computing brings with it its own unique challenges, which in their 2012 paper *State-of-the-art Cloud Computing Security Taxonomies - A classification of security challenges in the present cloud computing environment* (Srinivasan, Sarukesi, Rodrigues, M, & P, 2012) the authors split into two categories: Architectural and Technological aspects, and Process and regulatory related aspects. The challenges associated with the architectural and technological aspects of cloud computing includes: logical storage segregation and multi-tenancy security issues, identity management issues, insider attacks, virtualisation attacks, and cryptography and key management. While similarly the

challenges associated with the process and regulatory aspects of cloud computing includes: governance and regulatory compliance gaps, insecure APIs, cloud and CSP migration issues, and SLA and trust management gaps.

Alongside the growing reliance on web applications on the public internet, web applications have also been widely adopted inside organisations to support important functions. Many of these functions give the web application access to highly sensitive data, examples of which are:

- Human resources applications
    - Applications included in this classification can have access to information such as payroll information, employee performance feedback, recruitment information such as applicant's CVs, and even disciplinary information.
- Administrative interfaces
    - Applications included in this classification can have access to important infrastructure which the company uses, such as web and mail servers, and user workstations.
- Collaborative software
    - Applications included in this classification can have access to internal documents, can be used to manage internal workflow and projects, and can also be used to track issues internally.
- Business applications
    - Applications included in this classification such as enterprise resource planning software were previously developed as thick client applications – applications which work exclusively on the client machine with little interaction with the server, but have now been moving more towards access through a web browser.
- Software services
    - Applications included in this classification can include e-mail clients
- Traditional desktop applications
    - Applications included in this classification can include applications such as word processors or spreadsheet processors, which are also

seeing a migration to the web browser through applications such as Google Docs, and Microsoft Office 365.

However despite all this organisations continue to be unprepared for the threat that exists to their systems. In the 2013 *CREST Penetration Test Procurement Buyers Guide* (Creasey & Glover, 2013) the authors state that organisations often suffer from a lack of budget, resources, technology, or recognition of the type and magnitude of the problem. They also state that they don't have the software, testing, process, technology, or people to handle sophisticated cyber security threats.

In their 2014 paper *An Automated Approach to Vulnerability Assessment and Penetration Testing using Net-Nirikshak 1.0* (Shah & Mehtre, 2014, p. 707) the authors state that "*identification of vulnerabilities and remediation of the same has become one of the prime concerns of every web facing organisation in the last few decades."*

This can be seen further in the 2014 paper *A Survey and Vital Analysis of Various State of the Art Solutions for Web Application Security* (Thankachan, Ramakrishnan, & Kalaiarasi, 2014) where the authors state that "*a high percentage of web applications deployed on the internet are exposed to security vulnerabilities*". This is further back up by the Verizon 2017 Data Breach Investigations Report (Verizon, 2017, p. 24), where they found that "*almost 60% of breaches involved web applications either as the asset affected, and/ or a vector to the affected asset*."

To combat the rising threats to web applications, and the organisations to which they belong, a form of testing called penetration testing (informally known as pen testing) is performed. This form of testing is perhaps best defined in the 2016 paper *Manual and Automated Penetration Testing. Benefits and Drawbacks. Modern Tendency* (Stefinko, Piskozub, & Banakh, 2016, p. 488) where they define it as "*pen testing is used to search for vulnerabilities that might exist in a system. The goal of a penetration test is to increase data security."* They also state that the attacks performed in a penetration test "*probe a system's defences, these defences are then breached to evaluate the impact of any weaknesses, the results of these tests are used to improve a system's security, making them resilient to further attacks.*"

There are three types of penetration tests that can be performed, white hat, black hat, and grey hat. White hat is where the test is performed with full co-operation from the target network, and sometimes includes access to the source code. Black hat is where the test is performed with only a few administrators knowing that the test is underway, this allows for a simulation of a real world attack, wherein the defenders can learn how their internal security incident response performs. Finally grey hat is a combination of both white and black hat testing. However, in their 2015 paper *Penetration Testing Automation Assessment Method Based on Rule Tree* (Zhao, Shang, Wan, & Zeng, 2015, p. 1829) the authors state that "*Penetration testing requires higher confrontation and customization generally, and penetration test team needs analyse constantly in penetration process. But many penetration testing tools can`t be used automatically, and it is difficult to develop with other different commercial software."*

In 2008 the U.S. National Institute of Standards and Technology (NIST) proposed a four stage penetration testing model which consisted of planning, discovery, attack, and reposting. This model has gone on to become the basis for penetration tests performed globally.

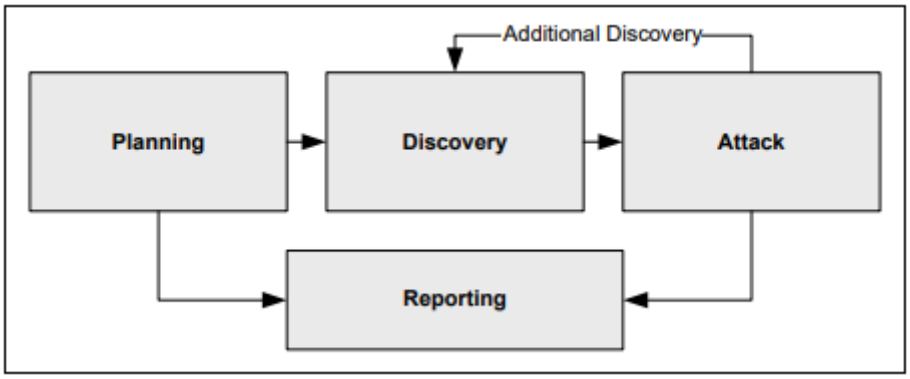

Figure 1 - Penetration Testing Model as proposed by NIST (NIST, 2008)

There are many reasons for penetration testing to be performed, some of the main reasons include: security issues – to identify any potential vulnerabilities before they can be used against the organisation or their systems, to protect information – with constant new attacks being developed to take advantage of vulnerabilities it's become increasingly difficult to protect a user's information, to prioritise security risks – issues identified during the penetration test can be prioritised based on the severity of the identified issue, and to secure against financial loss – they can help prevent or reduce

fines and lawsuits resulting from security malpractices - particularly with the incoming GDPR regulations.

2.4 Web Application Security Requirements

Security requirements for web applications can be derived from some of the basic tenants of web application security:

- Know the players – know who is logged into the application at any given time, and ensure that those who are logged in should have access
- Maintain confidentiality – this can be done through what is known as end to end encryption, no decryption or payload inspection while the data is in transit from client machines to the server
- Separate access and privilege from identity – associate the what access and privilege a user has in the web application to a role, instead of directly associating them to the user
- Maintain integrity – knowing that what you receive at any given moment is exactly what was sent
- Require explicit integrity – monitoring and logging all transactions done through the web application

There are however many different opinions on the security requirements which a web application should meet, but as stated in the 2015 paper *Analysis of Key Critical Requirements for Enhancing Security of Web Applications* (Kumar, 2015) they can be reduced down to nine key requirements. These nine requirements include: recognise and overthrow encrypted application attacks, observe all application communications, scrutinise and defeat zero-day attacks, adapt and implement policies for dynamic application environments, comprehensively screen application infrastructure elements, properly inspect benign traffic, install consistent security for all security sensitive applications, prevent the leakage of sensitive data related to stakeholders – stakeholders in this instance refers to anyone with an involvement with the application including any users, and secure infrastructure and protect the application users.

Similarly the 2016 paper *A Hybrid Threat Model for Software Security Requirement Specification* (Omotunde & Ibrahim, 2016) recommends some steps to take to secure a web application, this includes: changing GET methods to POST during client server interactions, use of parameterised statements instead of dynamically generated queries, server side validation of user input, zero level trust for all user input, use of the principle of least privilege, and the use of a white list – a list of allowed inputs for parameters.

Variations of these requirements have been stated at various times over the past decades since the public internet became so widely accessible. In their 2008 paper *Eliciting Security Requirements through Misuse Activities* (Braz, Fernandez, & VanHilst, 2008), the authors say that security is a matter of constraints, but in many others security is a more general need, and therefore it cannot be treated as a simple set of restrictions. They also say that typically, systems need some combination of authentication, authorization, transaction integrity, accountability, and message secrecy.

This is also visible in *Authorization and Privacy for Semantic Web Services* (Kagal, et al., 2004) where the authors describe authorization policies, as policies that constrain the provider to only accept requests for service from certain clients.

A paper back in 2011 (Kumar, Mitigating the Authentication Vulnerabilities in Web Applications through Security Requirements, 2011) stated that many organisations invested considerable resources to safeguard system security but are unprotected against security attacks. This may be due to a reliance on firewalls, network intrusion detection and prevention software, anti-virus software, anti-spyware, and Secure Sockets Layer or SSL in shorthand. However while these are critical systems to ensure the security of an organisation, they are no longer enough.

Additionally new challenges arise when attempting to meet requirement such as these, including the popularity of multi-tenancy in cloud computing service providers for example – where in a single instance of software runs on a server and serves multiple tenants, these tenants are groups of users who share a common access with specific privileges to the software instance running on the server. While multi-tenancy can help

improve quality of service, and help reduce service customisation and maintenance time, it invites additional challenges to securing these tenants. A part of this arises from each tenant potentially needing different security requirements, this means that the service provider needs to comply with each tenant's requirements, while at the same time ensuring that it remains compatible with the other tenants using the application (Alqahtani, He, & Gamble, 2017).

To understand the need for security for an application six questions can be asked (Assad, Tarciana, Ferraz, Ferreira, & Meira, 2010), these questions include:

1. What are the common security requirements for web applications?
2. What are the vulnerabilities found on each requirement?
3. What are the solution methods available?
4. What action is to be taken to mitigate?
5. What programming language will be used?
6. What are the impacts of the action in terms of architectural choices, frameworks and components?

2.5 Web Application Access Control

There are many different definitions for access control, with perhaps the best and most clear definition given in the 2011 paper *Access Control of Web and Java Based Applications* (Tso, Pajevski, & Brian, 2011) as seeking "to restrict access to protected resources by enforcing policies that state which subjects can perform which actions under which circumstances on which resources." This paper also state that it works on a number of levels including hardware, operating system, middleware, and application.

Access control is normally required to provide four main functions (Afolaranmi, et al., 2017), namely identification, authentication, authorisation, and accountability. These functions are further expanded on below:

- *Identification* – The process where the user selects an identity or a tag, typically in the form of a username, by which they will be known by the access control model.

- *Authentication* – This extends the previous function, by verifying the identity of the user. This normally involves the use of a password or a personal identification number.
- *Authorisation* – This function follows on from the authentication function, where by the access control model grants the user its rights to the corresponding application resources and functionality.
- *Accountability* – This incorporates all three previous functions and ensures that every activity in accessing resources or functionality is closely tracked and monitored, to ensure that only those authorised have gained access.

There are many different models which can be used as a basis for implementation of access controls in a web application. These models include, but are not limited to, Discretionary Access Control, Role Based Access Control, Attribute Based Access Control, and Emotion Based Access Control. These models are described best in the 2016 paper *Hybrid Authentication and Authorization Model for Web based Applications* (Indu & Rubesh Anand, 2016) where they describe Discretionary Access Control or DAC in short hand, as being based around an employee's identity and membership of certain user groups. They also state that this is typically based around Windows Active Directory. Role Based Access Control or RBAC in short hand, is based on different levels of authorisation controlled by a set of pre-defined of access privilege rules or application roles, these are normally associated with each user identity and is stored in the application database. Finally, they describe Attribute Based Access Control or ABAC in short hand, as taking into consideration user attributes, resource attributes, environmental conditions, and organisation policies to determine the user access privilege. Emotion Based Access Control or EBAC in short hand was first described in the 2012 paper *Authorized! Access Denied, Unauthorized! Access Granted* (Almehmadi & El-Khatib, 2013), where the authors describe this method as being based on the non-invasive acquisition of users' physiological signals, which the system then bases the decision to allow access to the application on.

However, the current popularity of social media has increased the complexity of implementing access controls in these applications. There are two main obstacles for implementing access controls for social media, first the diversity of the user base and the subsequent diversity of data contributed to the application from these users.

Second, that there still exists a large disparity between a user's mental model of the access controls provided by the application, and the current access control models used by the application (Singh, 2012).

Implementation of access controls in web applications is further complicated with the growing popularity of web application frameworks such as Ruby on Rails and Django. These frameworks increase programmer productivity and also help to prevent common errors in design and implementation. However Ruby on Rails doesn't implement access control mechanisms, instead it depends on third party libraries called gems, or the developer can implement it ad-hoc into their applications. A 2016 paper found numerous access control flaws in web applications developed using Ruby on Rails, which implemented one of these third-party libraries (Bocic & Bultan, 2016).

There are various types of authorisation flaws that derive from the above models, as seen in the 2011 paper *Detecting Insufficient Access Control in Web Applications* (Noseevich & Petukhov, 2011) where the authors identify three types of access control flaws, based on their root cause, which are as follows:

1. *Privileges under user control* – when the application makes assumptions about a user's privileges based on invalidated input
2. *Missing access control list entry* – when the application wide access control is implemented by use of a blacklist, missing entries will normally create authorisation vulnerabilities
3. Insufficient access control on certain execution paths

These flaws were at one stage quite common in web applications. In their 2011 book *The Web Application Hackers Handbook Second Edition* (Stuttard & Pinto, 2011) the authors state that between 2007 and 2011 they performed penetration tests on hundreds of web applications, and out of all these web applications 71% of web applications tested contained broken access controls. The authors also identify what they believe are the three main types of attacks which are used against access controls, which can be broken down into the following three categories:

- *Vertical privilege escalation* – This occurs when a user can perform functions which their assigned role should not allow. For example, a normal user accessing the administrator functionality to add a new user account.

- *Horizontal privilege escalation* – This occurs when a user can view or modify resources which they should not be able to. For example, viewing another user's emails.
- *Business logic exploitation* – This occurs when a user can exploit a flaw in the application's state machine to gain access to a resource which they normally shouldn't.

It is also not uncommon for these vulnerabilities to lead from one into another, for example a horizontal privilege escalation results in the malicious entity being able to launch a vertical privilege escalation attack.

The abundance of these vulnerabilities can be attributed to the fact that, as stated by the authors of the 2016 paper *Toward Exploiting Access Control Vulnerabilities within MongoDB Backend Web Applications* (Wen, *et al.*, 2016, p. 143) *"correct implementation of access control within a web application is non-trivial, since all feasible privileges of roles and users have to be thoroughly evaluated against the complex data models at the backend database.*"

The modern environment with the incoming GDPR and the Health Insurance Portability and Accountability Act (HIPAA) for healthcare companies and institutions and Gramm Leach Bliley Act (GLBA) for financial institutions in the United States highlights the growing need for secure access control to be implemented, in order to ensure that only authorised parties can access the sensitive information. In 2010 the authors of *Privacy-aware Role Based Access Control* (Ni, et al., 2010) state that privacy has been acknowledged to be a critical requirement for many business (and non-business) environments. They also state that legislative acts require enterprises to protect the privacy of their customers. In this they are referring to the private personal information which a customer would give to the organisation, trusting that only authorised users could access it.

2.6 Web Application Authorisation

Most modern web applications protect content from those unauthorised to access it, whether that is done by authentication or access authorisation. Web applications

currently use authentication techniques to validate and verify a user's identity; this often takes the form of a username and password. An alternative is to use an identity provider such as Shibboleth, Microsoft Passport, or Google Accounts. However this raises other problems, in the 2010 paper *DAuth: Fine-grained Authorization Delegation for Distributed Web Application Consumers* (Schiffman, Zhang, & Gibbs, 2010) the authors state that while these methods are effective for identifying users, they are less useful for applications working on their behalf. They also state that providing third party applications with login credentials raises the risk of identity theft through leaked secrets, and an increased attack surface for a malicious party.

This can be seen further in *The OAuth 2.0 Authorization Framework* (Hardt, 2012), where the creator of the framework Hardt states that in order to provide third-party applications access to restricted resources, the resource owner shares its credentials with the third party. He further goes on to state that this creates several problems, revolving around the need for the third-party application to store and use the credentials of the resource owner.

These problems aren't limited to traditional web applications but can also affect cloud applications. Bernd Grobauer et al (Grobauer, Walloschek, & Stocker, 2010) states in their 2010 paper that unauthorized access to the management interface is an especially relevant vulnerability for cloud systems: the probability that unauthorized access could occur is much higher than for traditional systems where the management functionality is accessible only to a few administrators.

In the 2017 White Hat Application Security Statistics Report (WhiteHat Security, 2017) they report that information leakage – of which authorisation vulnerabilities are one of the main root causes, is the most prevalent vulnerability with 37 percent likelihood. This is even more damning, when compared with one of the most common vulnerabilities that exists today Cross-Site Scripting, which has a likelihood of 33 percent. However, they further state that insufficient authorisation vulnerabilities only have an 8 percent likelihood of happening. They break this down even further by comparing the most serious vulnerabilities, such as SQL Injection, Cross-Site Scripting, Cross-Site Request Forgery, among others. When compared to these vulnerabilities, authorisation vulnerabilities can be seen in a new light. SQL Injection

was found to of a critical risk 94 percent of the time it's detected, while Cross-Site Scripting was only found to be of a critical risk 38 percent of the time it's detected. In comparison to this insufficient authorisation vulnerabilities, were found to be of a critical risk 40 percent of the times it was found to be present in an application. Furthermore, WhiteHat analysed the remediation rates of these vulnerabilities, or in other words the rate at which these vulnerabilities are fixed and found that only approximately 40 percent of insufficient authorisation vulnerabilities that were detected during 2016 were fixed.

One of the causes of this type of vulnerability lies in the higher privilege level which most web applications run at on the server, compared to the individual users who may be using the web application. This means that the developers must implement their own checks into the web application to enforce the authorisation levels of any given user. However, this can be problematic for developers, as a single missed check could result in unauthorised access. In addition to this most developers focus on the components that they themselves have written, however in the 2015 paper *FlowWatcher: Defending against Data Disclosure Vulnerabilities in Web Applications* (Muthukumaran, et al., 2015) the authors state that missing checks often exist in third party plug-ins and extensions. They further state that in 2013 16 percent of security bugs reported in CVE - Common Vulnerabilities and Exposures a repository of all publicly known vulnerabilities - for Drupal were related to unauthorised data disclosure in plug-ins.

The continued rise of social network web applications in today's markets clearly demonstrates the complexity and challenge that modern developers face when implementing authorisation and access control in modern web applications. Social network applications are constantly growing, adding more and more features, often in a form of arms race with each other to attract new users and keep existing ones. These new features involve improving experiences when using the application, and including more ways to express oneself, which often means new ways of interacting with other users. The complexity associated with these applications could perhaps best be described by examining common features, such as messaging – either one to one or group. In this situation only, the participant should be able to send and receive messages associated with the conversation, and only participants should be able to add

new people to the conversation. In addition to this, messages can be edited for a short time after they've been sent, but this can only be done by the user that sent the message. Another common feature involves a user posting content to their personal page, which is usually unrestricted, or posting content to a group – commonly only after being accepted into that group. There is also a way to create connections with other users, and in doing so opens even more features and ways to interact. Failure to properly implement authorisation checks for any one of these features mentioned above can lead to irreparable reputational data for the organisation, and on a personal level for the affected user – this could include serious life altering event such as loss of family, employment, and could even in some case lead to identity theft.

This has led to challenges (Marinescu, et al., 2017) which could be considered unique to social network web applications. Some of these challenges stem from the daily operations which the application must perform; many of the popular social networks have over 100 million active users, while an estimated 1.8 billion people have a social network account. This results in a large number of requests which need to be checked in real-time and is further complicated by the interconnected nature of the data being requested. This scale of request also impacts on the number of acceptable false positive, which may arise from authorisation checks, even a rate as low as 0.00001 percent can be considered unacceptable, due to the large number of users which would be affected, and the alarms that would be generated requiring human validation. Some of the other challenges are derived from the complicated nature of the code base required to operate a social network application.

2.7 Conclusion

In this chapter a range of literature related to web application authorisation was explored. The literature explored was broken down into five different topics, each of which are associated with, and have an impact on web application authorisation. These topics were information accountability and usage control, web application security, web application security requirements, web application access control, and finally web application authorisation itself.

Information accountability and usage control was explored in order to provide some background about the need for strong authorisation in web applications, and its importance in today's environment with legislation such as GDPR coming into enforcement.

Web application security and web application security requirements were explored due to the large role that authorisation plays in securing web applications. This also demonstrated the importance of secure web applications, and how that the level of security implemented in web applications, though improving, still leaves a lot to be desired.

Web application access control and web application authorisation were explored, in order to explain how access control and authorisation works, while at the same time displaying the complexities of these systems and how a malicious entity can exploit them.

3. DESIGN AND METHODOLOGY

3.1 Introduction

Following on from the review of the current related literature in the previous chapter, this chapter will focus on the presenting an outline of the experiment which was performed, and the technologies involved. This will be done under the following topics: experiment overview, technologies used, and experiment design. The experiment overview provides a brief examination of the experiment that was performed. Technologies used provide an introduction and brief overview of the technologies used including those used in the creation of the custom written scanner, a brief introduction and overview of the commercial tool and its plug-ins, and a brief overview of the vulnerable web application which were tested as part of the experiment.

3.2 Experiment Overview

The key objective of this experiment was to show that the task of performing authorisation testing on authenticated web applications can be automated, without the use of access control models, machine learning algorithms, or access to the source code of the application. In order to achieve this objective, various ways of performing authorisation testing was examined, with two key methods being chosen for examination. These two methods are as follows: one, to test the applications by attempting to directly browse to web application functionality in the browser, and two, to test the applications by proxying the browser through a web application proxy tool, in order to modify the cookies and/or headers.

For the purposes of the experiment it was decided to write a custom script that would automate manual testing of authenticated web applications through the browsers. As well as this it was decided to pick three plug-ins for a commercial grade web application vulnerability scanner, which would perform the modification of cookies and/or headers testing. These would then be run against a number of web applications, which were divided into three categories: those with known authorisation vulnerabilities, those that are known to have no authorisation vulnerabilities and finally

those where it's unknown whether or not they have vulnerabilities. Once these web applications are finished being tested the results will be examined by a group of information security experts, who will validate whether any detected issues are real or whether they are false positives.

These results will then feed back into the overall hypothesis of the experiment, that being that a custom developed web application scanner focused on testing for authorisation vulnerabilities in web applications, will provide a more accurate (in terms of the number of false positives, false negatives, and true positives of detected authorisation vulnerabilities) analysis of authorisation vulnerabilities than plug-ins developed for commercial grade software designed to test for authorisation vulnerabilities.

3.3 Technologies Used

Over the course of the experiment there were a number of technologies used; these technologies include those used for the creation of the custom written scanner, the commercial tool and its plug-ins, and the vulnerable web applications that will be tested as part of the experiment. These technologies are explained in further detail below.

3.3.1 Developed Scanner

There were two technologies used for the development of the custom scanner, these being Python and Selenium. Both of these technologies are discussed below.

3.3.1.1 Python

Python is a widely used high level programming language-one with a large amount of abstraction from code that the machine understands; they use natural language elements to make the coding of a program easier for the developer. Python follows a philosophy of emphasising code readability and allows a programmer to develop their code in fewer lines than it would take in other languages.

Python supports many programming features including object oriented, dynamic typing, and automatic memory management and has a large set of libraries for many different uses. It comes pre-installed on Linux distributions, but needs to be

downloaded and installed for Windows. Python files do not need to be compiled but can be run immediately after being coded. It is also a free open source programming language.

3.3.1.2 Selenium

Selenium is a software testing framework for web applications; it also provides a domain-specific language known as Selenese that allows for tests to be written in a variety of popular programming languages. These tests can then be run using most modern browsers. It consists of two main parts: one, a complete integrated development environment that allows for the recording, editing, and debugging of tests. This is normally implemented as a browser add-on for either Firefox or Chrome. Scripts can be recorded and edited manually, and are generally stored as Selenese commands, which translate as commands for a browser. For example, a script may tell the browser to go to a certain website page, click the link to log in, enter the username and password, and then click the login button.

The second part, known as WebDriver, accepts commands which have been sent in Selenese or through the client API and sends them to the browser. This is done through a browser specific driver – e.g. a separate driver is needed for Chrome or Firefox, which sends the command to that browser and retrieves the results. Most of these drivers will launch an instance of the browser in question; browsers included are Chrome, Firefox, Safari, Microsoft Edge, Internet Explorer, and headless browsers such as PhantomJS. A client API is included in Selenium to allow tests be written in other programming languages, such as Java, Python, JavaScript, C#, or Ruby on Rails.

3.3.2 Burp Suite

Burp Suite[6] is a graphical tool written in Java for testing web applications. It comes in two versions; community edition available for free, which contains a stripped-down version with less features, or the professional version available for a yearly fee which contains all available features. The features which come with the professional version include: HTTP proxy – a web proxy server which sits between the browser and destination web servers allowing for the interception and modification of raw traffic,

[6] https://portswigger.net/burp

scanner – a web application vulnerability scanner allowing for the automated scanning of requests for vulnerabilities, intruder – can be used to automate web application attacks through either a user generated or provided list of vectors, spider – automates the crawling of a web site, repeater – allows for the manual modification of requests and provides the ability to resend these modified requests and view the responses, decoder – a tool for either encoding or decoding information, comparer – a comparison tool for viewing the differences between two requests/responses, extender – allows for new user created or third party created functionality to be added, and sequencer – analyses the randomness in a sample of data, this can be used to test the randomness of tokens such as session cookies or anti-CSRF tokens. Cross-Site Request Forgery or CSRF for short occurs when a malicious entity forces an end user of an application to execute unwanted actions on a web application in which they are currently authenticated. These attacks typically target state changing requests, such as changing account details, or transferring money from one account to another. One of the ways to defend against this kind of attack is to make use of anti-CSRF tokens, these are values added to either the request body, or as a header.

There are three plug-ins that can be added through the extender, which are of particular interest for the purpose of this experiment: AuthMatrix, Authz, and Autorize.

3.3.2.1 AuthMatrix

The below figure shows AuthMatrix[7], an extension for Burp Suite, which is designed to improve the process of verifying authorisation policies in web applications and services. It allows testers to define a set of roles, users, and requests that satisfactorily covers the capabilities of the target application. These are displayed in a table that is in a similar format to an access control matrix that's commonly found in various threat modelling methodologies. Testers can then just click one button, and the extension will test all combinations of roles and requests. The results are then displayed back in an easy to read colour coded interface indicating any possible authorisation vulnerabilities detected.

[7] https://portswigger.net/bappstore/30d8ee9f40c041b0bfec67441aad158e

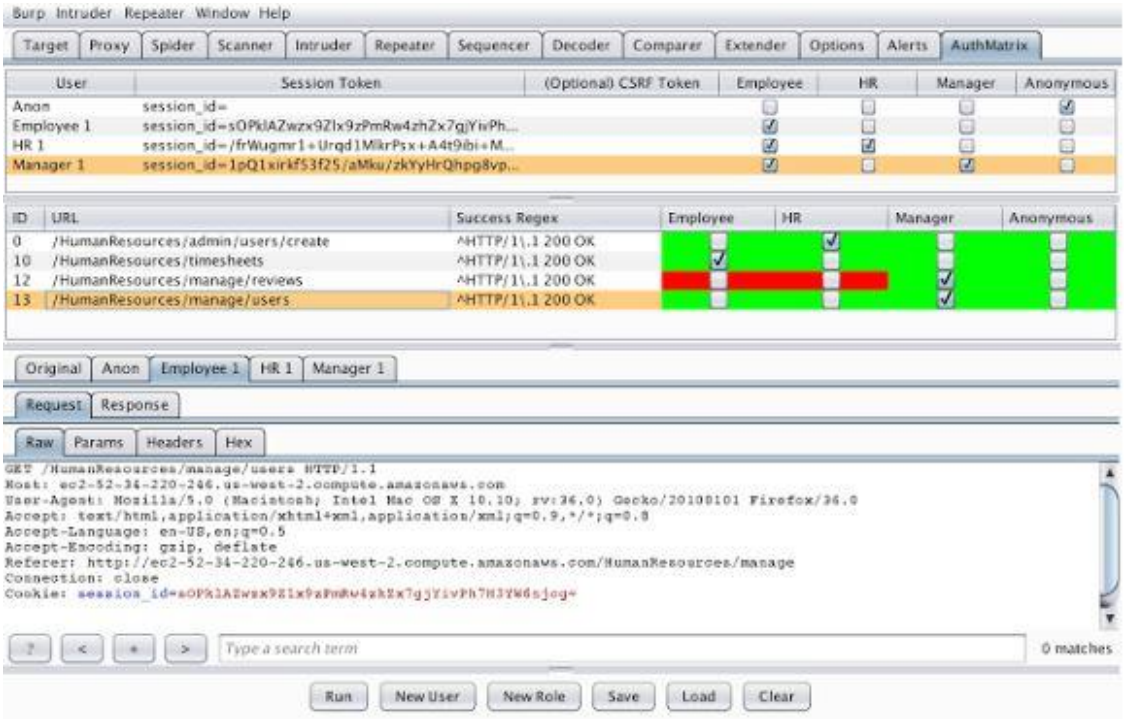

Figure 2 - Screenshot of AuthMatrix

3.3.2.1 Authz

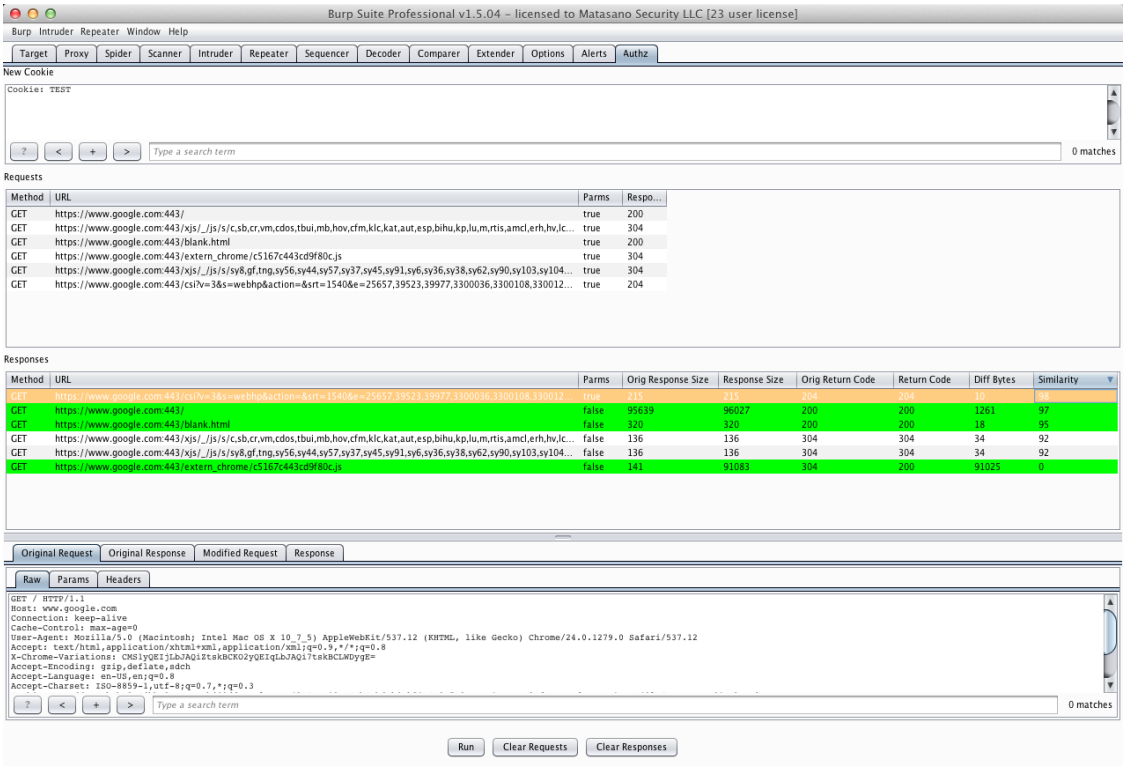

Figure 3 - Screenshot of Authz

The above screenshot shows Authz[8], an extension for Burp Suite, which is designed to help the pen tester performing authorisation testing. Testers can send requests to it, create a new cookie – presumably that of a different user, and send the requests with the created cookie. The extension then notes whether there is a difference in the responses, between the original response, and the modified response.

3.3.2.1 Authorize

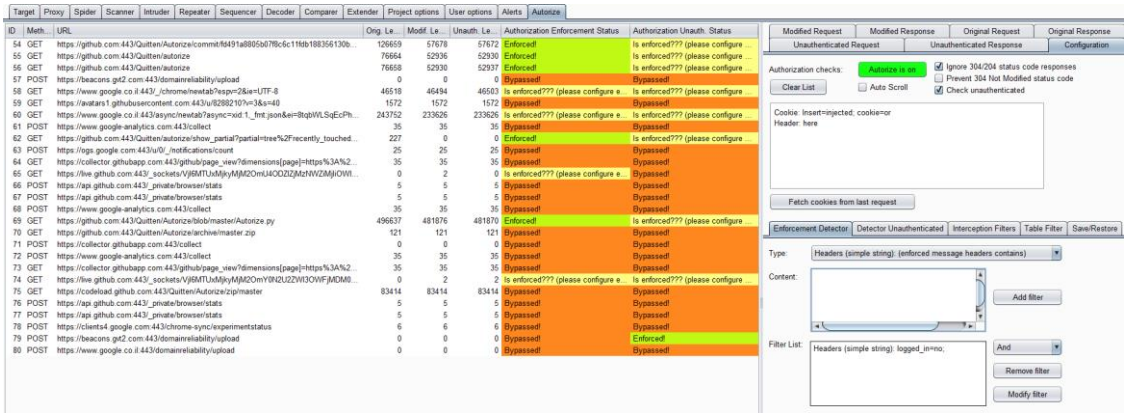

Figure 4 - Screenshot of Authorize

The above screenshot shows Authorize[9], an extension for Burp Suite, which attempts to automate the detection of authorisation enforcement. Testers can set the cookies and other potential authorisation headers which they want to use in the extension, and then turn it on. The extension works in the background while the tester is navigating the target application, recording requests sent, and repeating them with the specified cookies and headers.

3.3.3 Vulnerable Web Applications

As part of the experiment vulnerable web applications were tested, in an attempt to discern whether or not the custom written script or the plug-ins for Burp Suite could detect authorisation vulnerabilities. The vulnerable web applications were selected on the basis of whether or not they had authorisation vulnerabilities and are described in more detail below.

[8] https://portswigger.net/bappstore/4316cc18ac5f434884b2089831c7d19e

[9] https://portswigger.net/bappstore/f9bbac8c4acf4aefa4d7dc92a991af2f

3.3.3.1 HacMe Bank

The below screenshot shows HacMe Bank, a vulnerable web application designed to teach application developers, programmers, architects and security professionals how to create a secure web application. It simulates an online banking application, with a number of known vulnerabilities. The purpose is to allow users to test the exploitation of vulnerabilities in a safe and secure manner, and thus learn the specifics and how best to remediate any such vulnerability. The application is designed to work with Windows .NET Framework, and Microsoft SQL Server. For the purposes of this experiment, this application was run using Microsoft Internet Information Services (IIS) version 7, and was hosted in a virtual machine running windows 7 using VirtualBox – a program designed to manage and operate virtual machines. The vulnerabilities coded into it include, but are not limited to:

- SQL Injection
- Cross-Site Scripting
- Horizontal Privilege Escalation
- Vertical Privilege Escalation
- Directory Listing
- Session Hijacking
- Cookie Manipulation

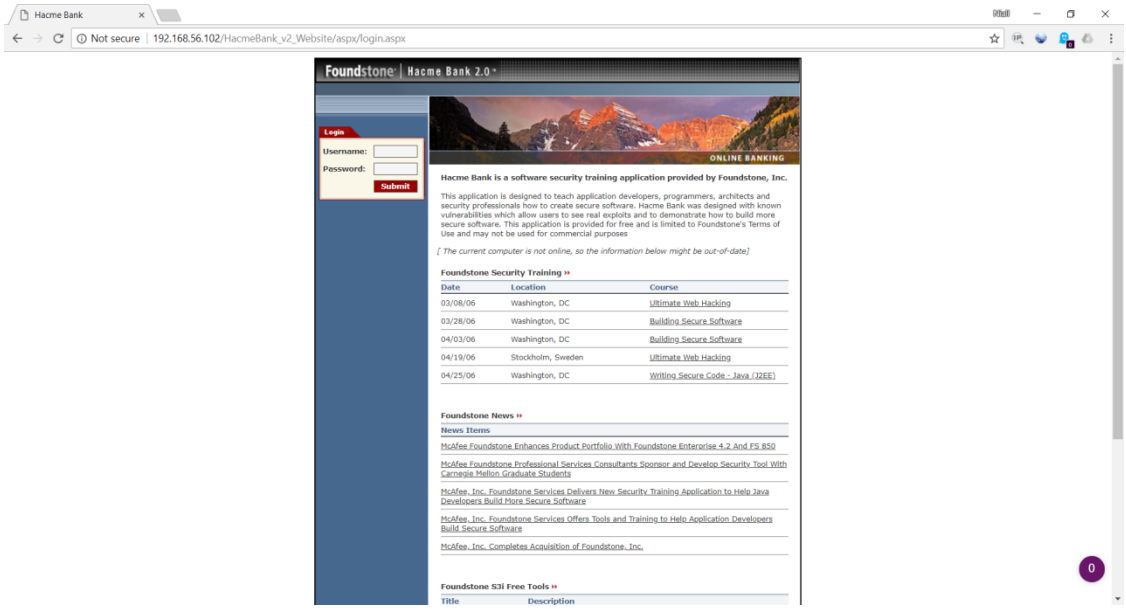

Figure 5 - Screenshot of HacMe Bank

3.3.3.2 OWASP WebGoat

The below screenshot shows OWASP WebGoat, a vulnerable web application, designed and maintained by the Open Web Application Security Project (or OWASP for short). The purpose is to allow users to test the exploitation of vulnerabilities in a safe and secure manner, and thus learn the specifics and how best to remediate any such vulnerability. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. The vulnerabilities coded into it include, but are not limited to:

- Cross-Site Scripting
- Insufficient Access Control
- Thread Safety
- Hidden Form Manipulation
- Parameter Manipulation
- Weak Session Cookies
- SQL Injection

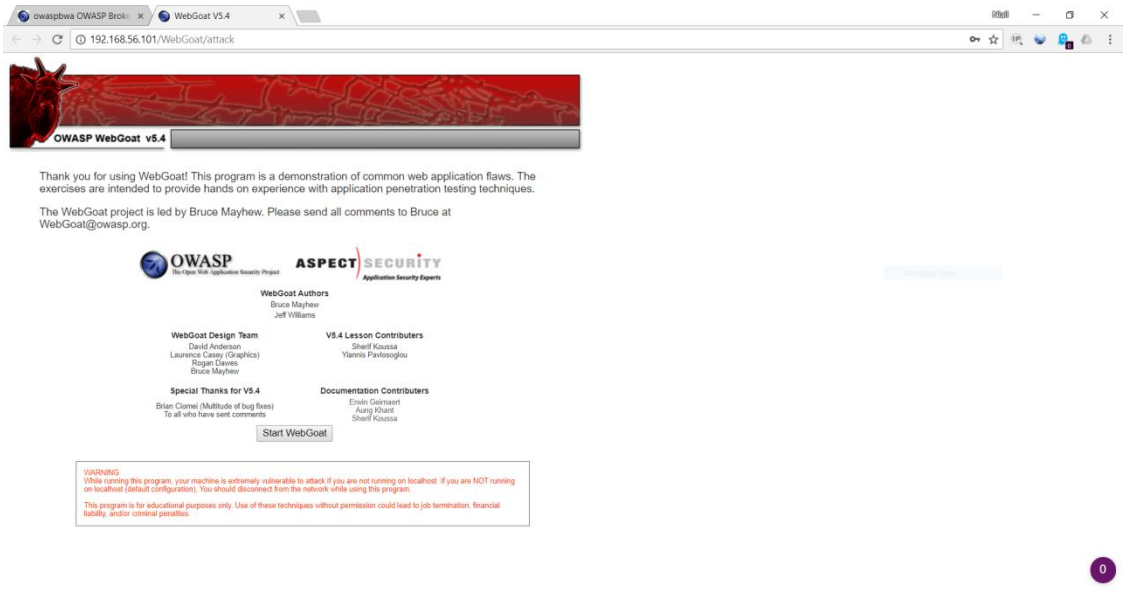

Figure 6 - Screenshot of OWASP WebGoat

3.3.3.3 OWASP Juice Shop

The below screenshot shows OWASP Juice Shop, a vulnerable web application, written in NodeJS, Express and AngularJS. The purpose is to allow users to test the exploitation of vulnerabilities in a safe and secure manner, and thus learn the specifics and how best to remediate any such vulnerability, while also allowing web application vulnerability scanners to test how well they cope with a JavaScript heavy application frontend and REST API backend. This application is hosted publicly, and is available at the following URL https://juice-shop.herokuapp.com. The vulnerabilities coded into it include, but are not limited to:

- Broken Authentication
- XML External Entities Injection
- Cross-Site Scripting
- Broken Access Control
- Insecure Deserialization
- SQL Injection

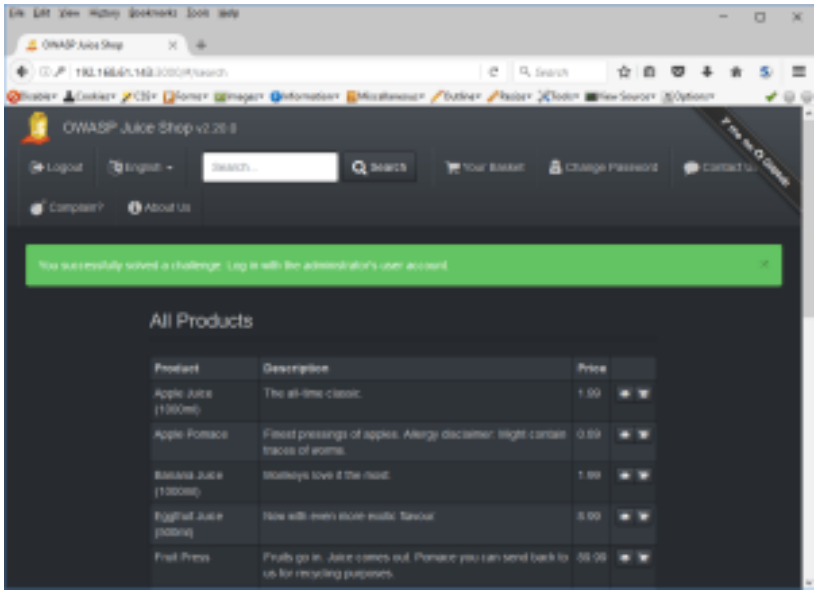

Figure 7 - Screenshot of OWASP Juice Shop

3.3.3.4 Pentester Lab 2

The below screenshot shows Pentester Lab 2, a vulnerable web application. The purpose is to allow users to test the exploitation of vulnerabilities in a safe and secure manner, and thus learn the specifics and how best to remediate any such vulnerability.

This application can be downloaded as an iso file, which can then be used to run a live USB. For the purposes of this experiment, this application was run from the iso file on a virtual machine using VirtualBox – a program designed to manage and operate virtual machines. The vulnerabilities coded into it include, but are not limited to:

- SQL Injection
- Broken Authorisation
- Broken Authentication
- Insecure Captcha

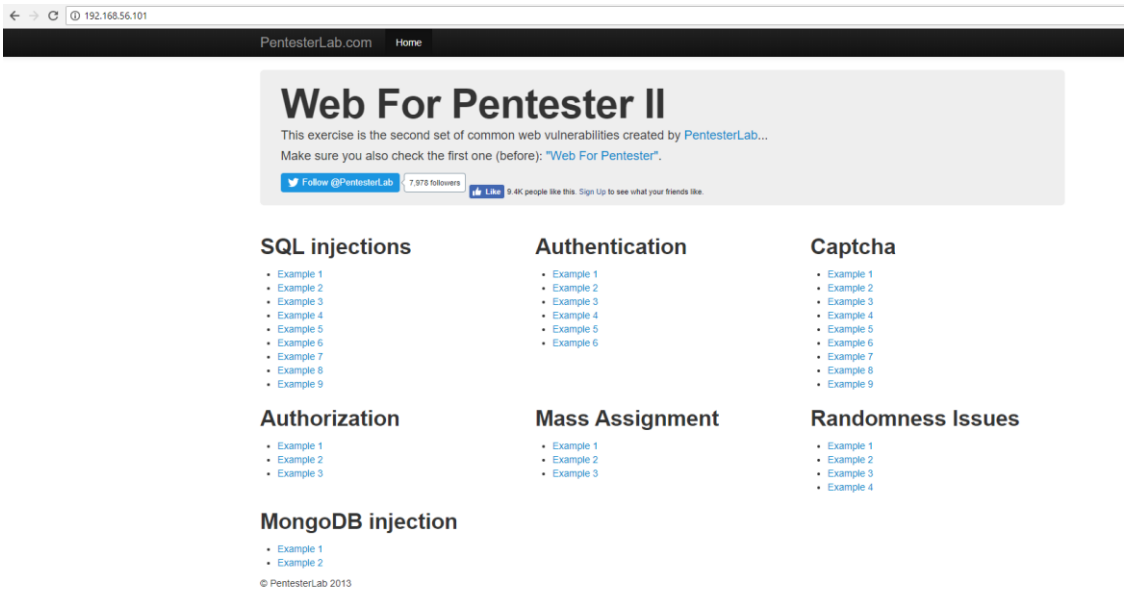

Figure 8 - Screenshot of Pentester Lab 2

3.3.3.5 Damn Vulnerable Web App

The below screenshot shows Damn Vulnerable Web App, a vulnerable web application, written in php and MySQL. The purpose is to allow users to test the exploitation of vulnerabilities in a safe and secure manner, and thus learn the specifics and how best to remediate any such vulnerability. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using

VirtualBox – a program designed to manage and operate virtual machines. The vulnerabilities coded into it include, but are not limited to:

- Brute Force Login
- Command Execution
- Insecure Captcha
- Cross-Site Scripting
- File Inclusion
- SQL Injection

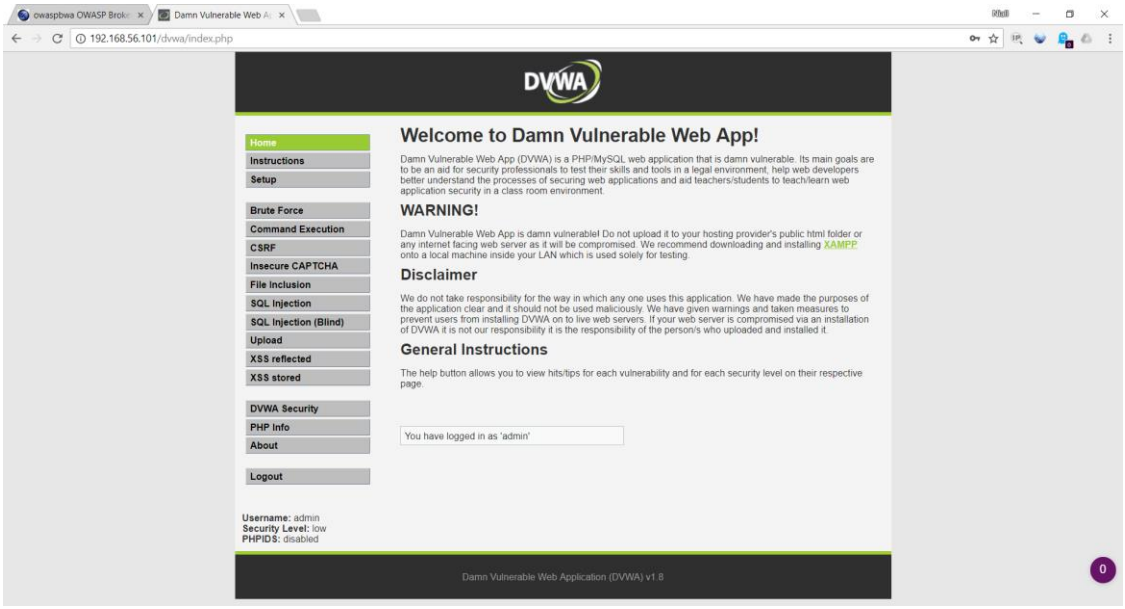

Figure 9 – Screenshot of DVWA

3.3.3.6 Bwapp

Bwapp is a free and open source deliberately insecure web application, developed in PHP and MySQL. It is designed to help security enthusiasts, developers and students to discover and to prevent web vulnerabilities. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. The vulnerabilities coded into it include, among many others:

- SQL Injection
- OS Command Injection
- Cross-Site Scripting

- Cross-Site Request Forgery
- Broken authentication and authorisation
- XML External Entity attacks

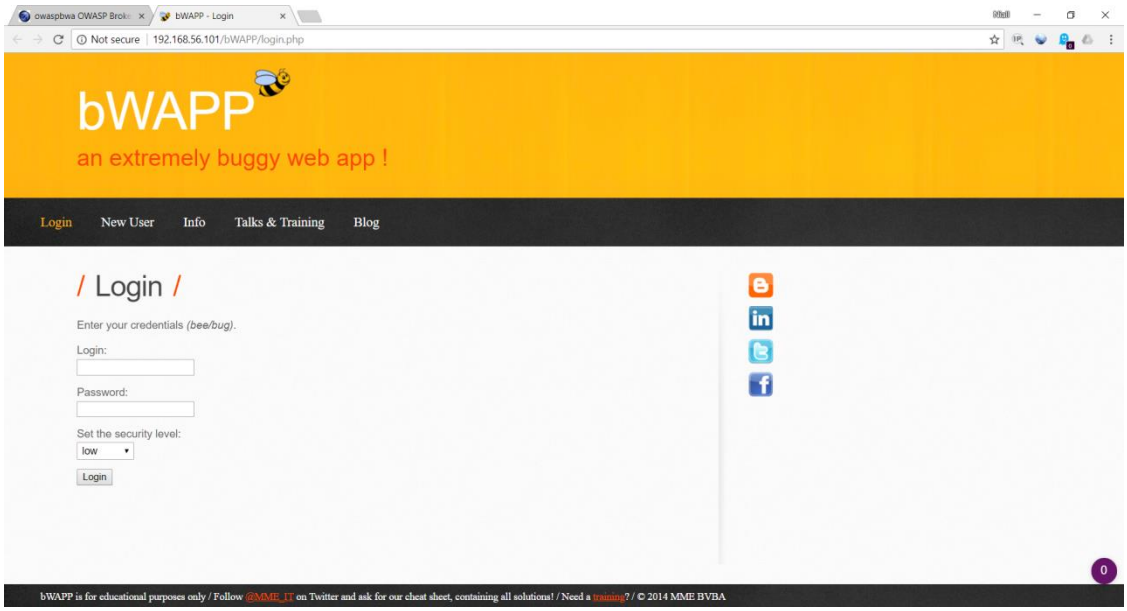

Figure 10 – Screenshot of Bwapp

3.3.3.7 Altoro Mutual

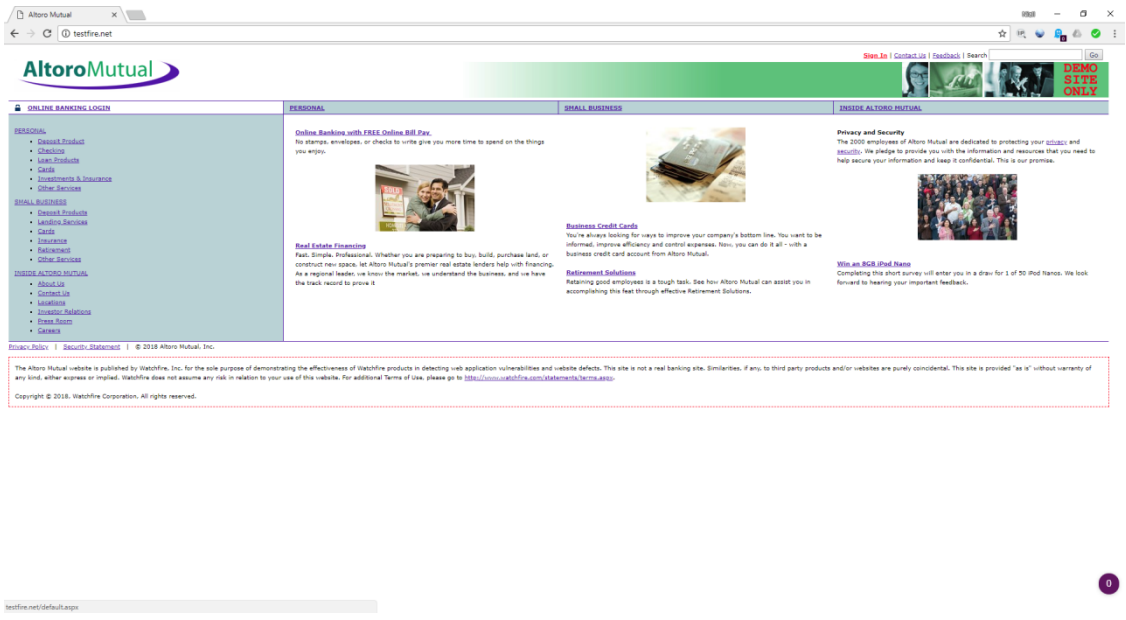

Figure 11 – Screenshot of Altoro Mutual

Altoro mutual, seen in the screenshot above, is a vulnerable web application, created by WatchFire – now owned by IBM – as a demo application for their black box

scanner. This application is publicly hosted, and is available at the following URL http://testfire.net/. The vulnerabilities coded into it include, among many others:

- SQL Injection
- Cross-Site Scripting
- Directory Listing
- Cross-Site Request Forgery

3.3.3.8 Mutillidae 2

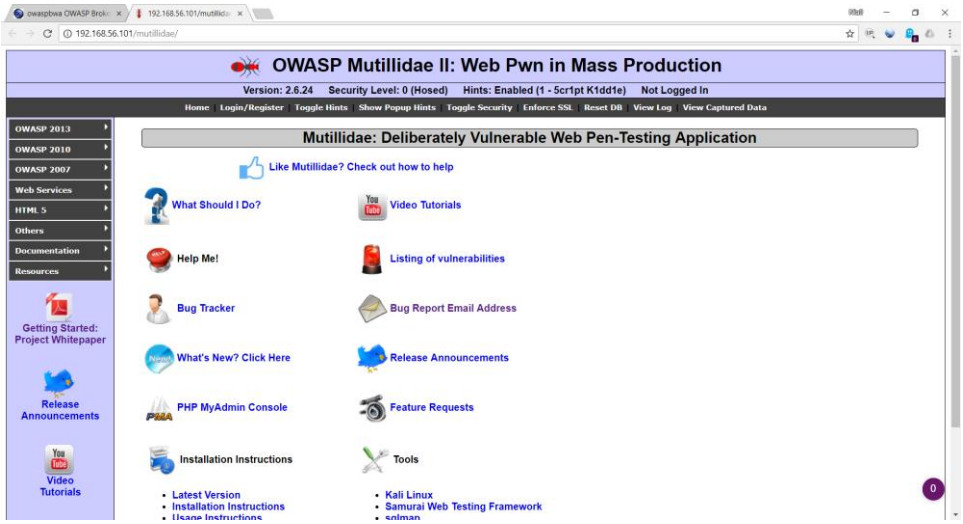

Figure 12 - Screenshot of Mutillidae

Mutillidae 2, as seen in the screenshot above, is a vulnerable web application, and has been created to include vulnerabilities from multiple versions of the OWASP Top Ten. It has been used for multiple different purposes, including: graduate security courses, corporate training, an "assess the assessor" target – wherein the client provides the prospective penetration testing team with a vulnerable web application in an effort to test the effectiveness of the team, and as a target for web application firewall testing. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. The following vulnerabilities are among the many that are coded into it:

- SQL Injection
- Cross-Site Scripting

- Privilege Escalation
- XML External Entity Injection
- Remote File Inclusion
- Cross-Site Request Forgery
- Broken Authorisation

3.3.3.9 OWASP Bricks

OWASP Bricks is a deliberately insecure web application, written in PHP and MySQL, and maintained by the Open Web Application Security Project (or OWASP for short). This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. The following is a small list of some of the vulnerabilities it contains:

- SQL Injection
- File Upload Vulnerabilities

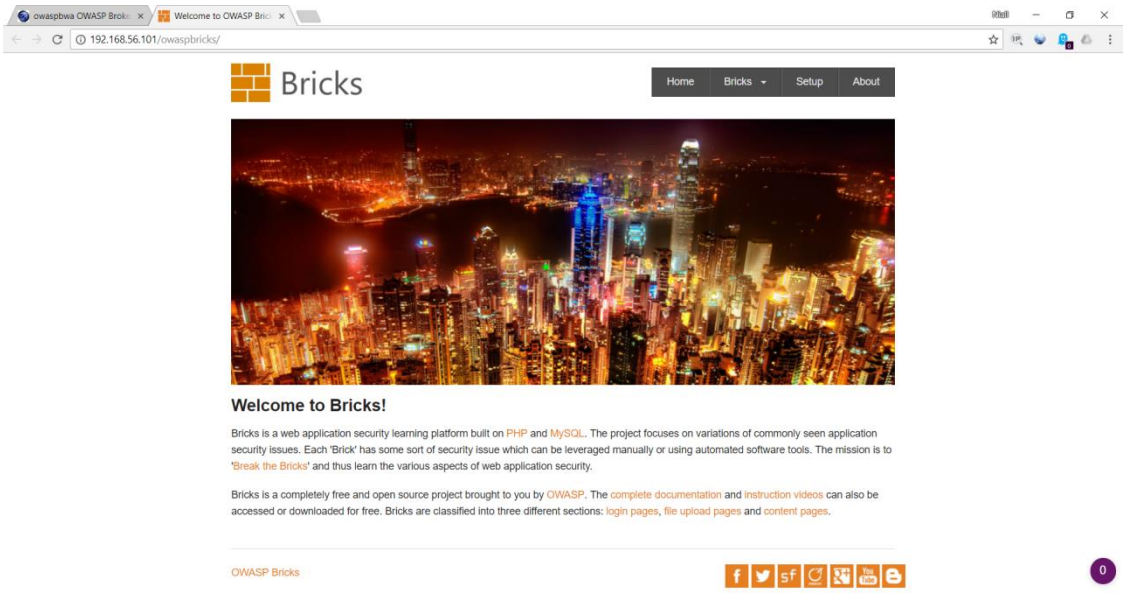

Figure 13 - Screenshot of OWASP Bricks

3.3.3.10 OWASP Rails Goat

RailsGoat is a vulnerable version of the Ruby on Rails Framework. It includes vulnerabilities from the OWASP Top 10. This project is designed to educate both

developers, as well as security professionals. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. Some of the vulnerabilities included are:

- SQL Injection
- OS Command Injection
- Cross-Site Scripting
- Broken Authentication
- Missing Access Controls
- Insecure Direct Object Reference
- Cross-Site Request Forgery
- Invalidated Redirects

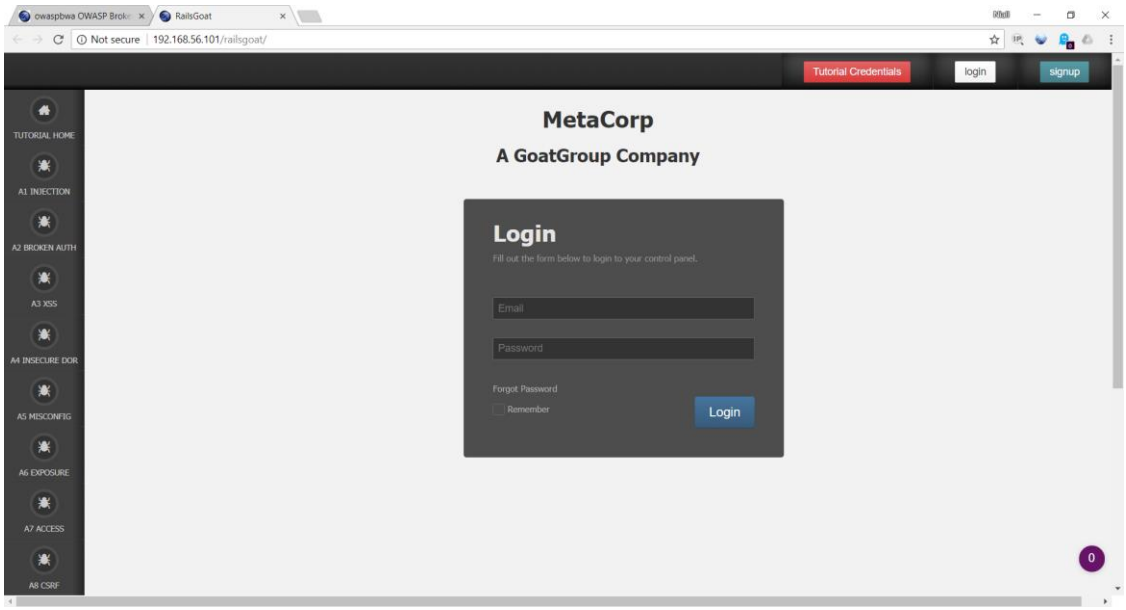

Figure 14 - Screenshot of OWASP Rails Goat

3.3.3.11 OWASP Vicnum

OWASP Vicnum is a project containing different vulnerable web applications based on games commonly used to waste time. These applications demonstrate common web security problems such as Cross-Site Scripting, SQL Injections, and Session Management issues. This application is one of many included in the OWASP Broken

Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines.

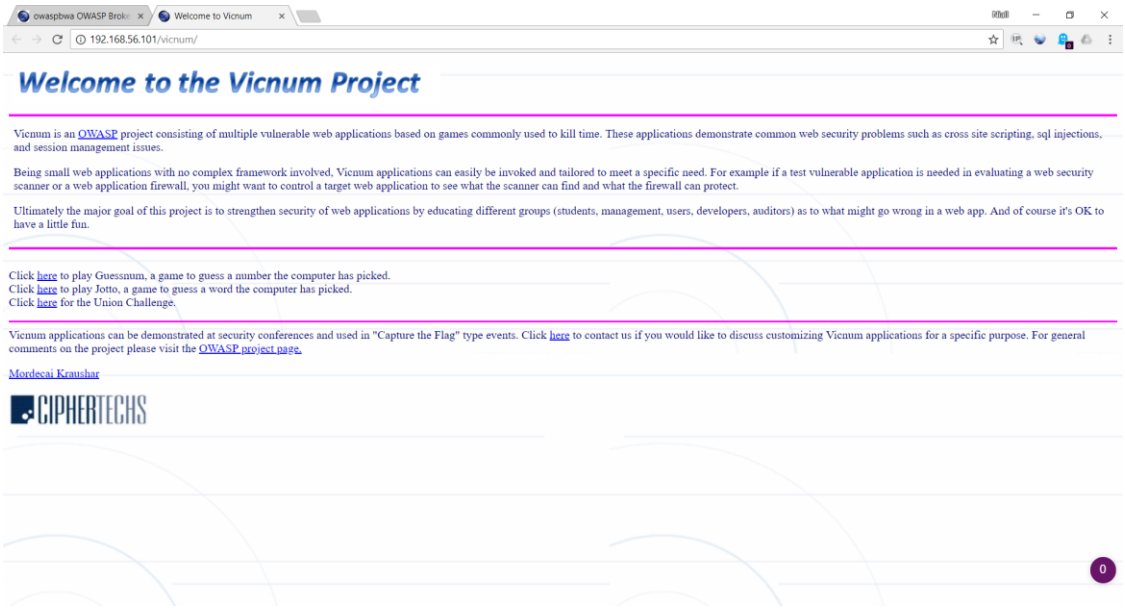

Figure 15 - Screenshot of OWASP Vicnum

3.3.3.12 OWASP WebGoat.NET

OWASP WebGoat.NET is a deliberately vulnerable asp.net, designed for use in classroom environments. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. The vulnerabilities included in it are:

- SQL Injection
- Cross-Site Scripting
- Broken Authentication
- Broken Access Controls
- Cross-Site Request Forgery
- Session Management, and many others

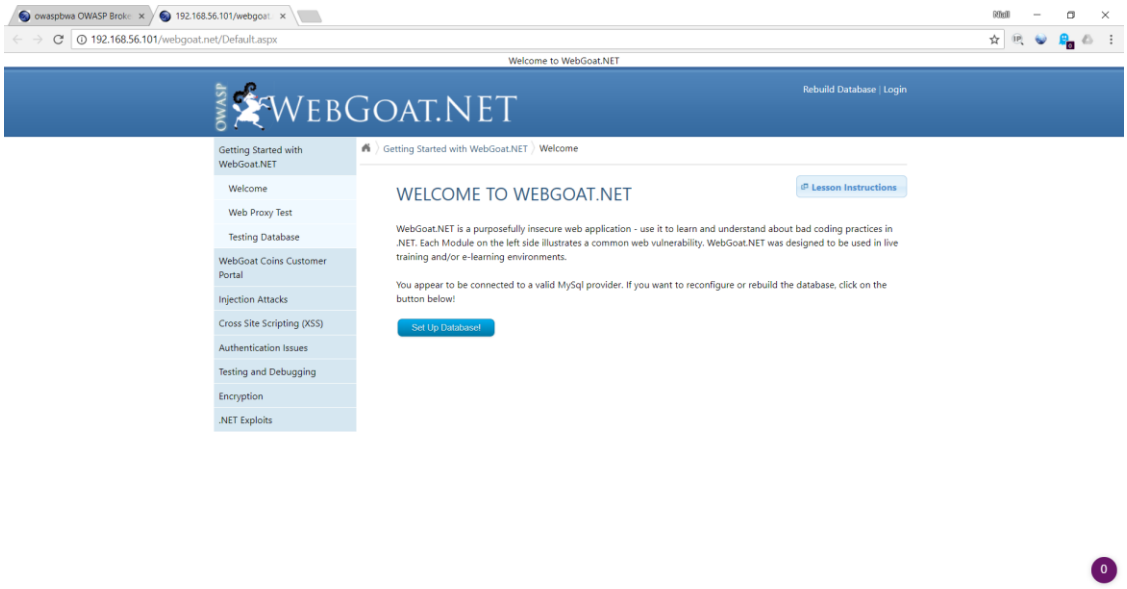

Figure 16 - Screenshot of OWASP WebGoat.NET

3.3.3.13 WackoPicko

WackoPicko is a vulnerable web application written by Adam Doupé for use in his 2010 paper *Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners.* This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines. Some of the vulnerabilities it includes are:

- Cross-Site Scripting
- SQL Injection
- OS Command Injection
- Directory Traversal
- File Inclusion

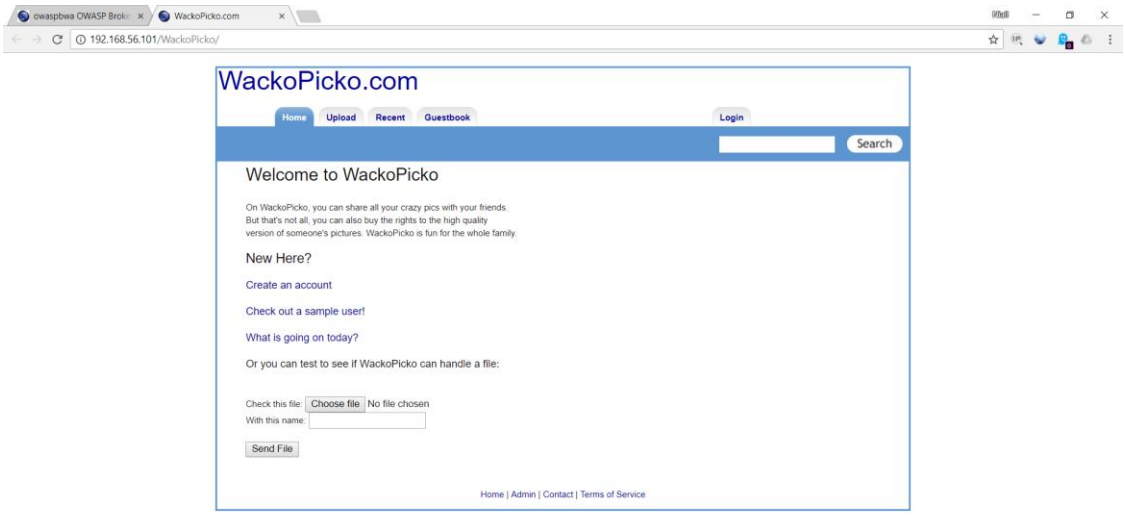

Figure 17 - Screenshot of WackoPicko

3.3.3.14 BodgeIt

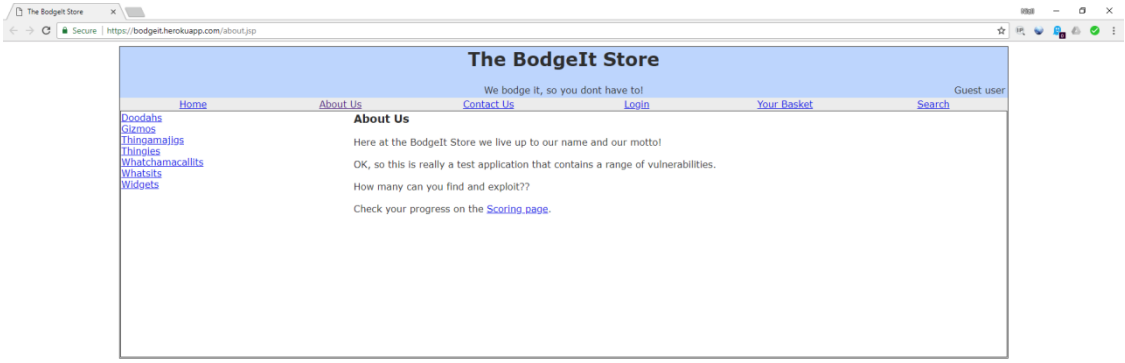

Figure 18 - Screenshot of BodgeIt

BodgeIt is a vulnerable web application, designed for those that are new to web application security, and want to learn about exploiting vulnerabilities. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of

this experiment using VirtualBox – a program designed to manage and operate virtual machines. It includes a few of the common vulnerabilities, including:

- Cross-Site Scripting
- SQL Injection
- Cross-Site Request Forgery
- Insecure Direct Object References
- Application Logic Vulnerabilities

3.3.3.1 OWASP AppSensor Demo Application

The OWASP AppSensor Demo Application was designed to demonstrate the OWASP AppSensor project, which defines a framework that offers guidance on implementing intrusion detection and automated response into an existing application. This application is one of many included in the OWASP Broken Web Application virtual machine, a UNIX based operating system hosting a number of vulnerable web applications, this virtual machine was used to host the application for the purposes of this experiment using VirtualBox – a program designed to manage and operate virtual machines.

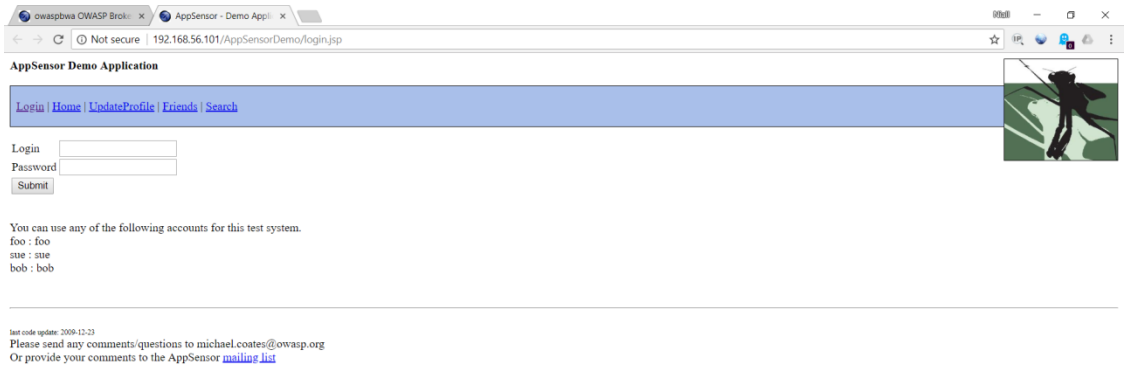

Figure 19 - Screenshot of OWASP AppSensor Demo Application

3.4 Experiment Detail

Figure 19 below shows a brief outline of the experiment which took place. There are eight stages to the experiment that was performed; each of these stages will be described in detail below.

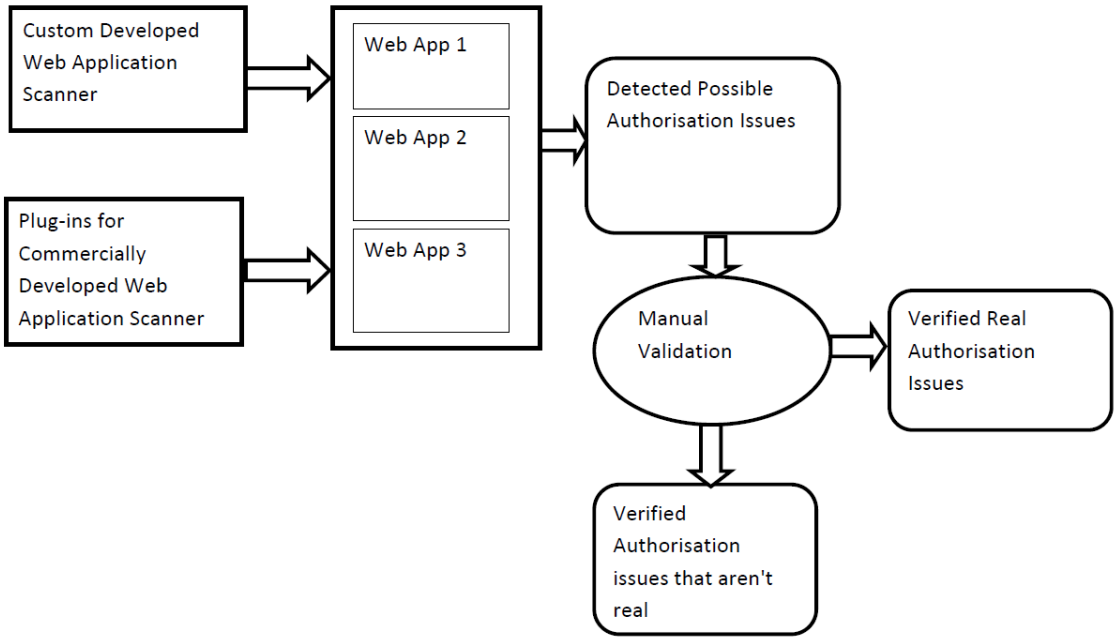

Figure 20 - Diagram showing the experiment which took place

First Stage

The first of the experiment was to create a web application scanner – a program that would check a web application for vulnerabilities. This web application scanner would focus on detecting broken access control issues. Similarly in their 2016 paper (Near & Jackson, 2016) *Finding Security Bugs in Web Applications using a Catalogue of Access Control Pattern*, the authors developed a program they called SPACE – Security Pattern CheckEr, an automated program for detecting access control issues in web applications. However, SPACE has a significant flaw when considering it as a program for widespread testing of security issues as; it requires access to the source code, and for the developers to provide a mapping from the application to the catalogue types – which it tests against. This is unfeasible when considered for testing on a larger scale, due to two main factors, one: that the personnel performing a security

review of the web application would have access to the source code – most security reviews take place in a black box environment, i.e. without knowledge of the underlying infrastructure. Two: that the personnel would then have enough knowledge of the application to provide the mapping.

The web application scanner that was developed for this experiment however, requires no knowledge of the application source code. This scanner is comprised of two technologies which have been discussed above; python – the language used to develop the scanner, and selenium – designed to automate browsers. The user will still need to supply some information before running the program against a web application, however this information is easy to obtain, and does not require any knowledge of back end infrastructure. An xml file, containing the selenium instructions on logging into the application as a user, is all the application will need before it can be run against the web application.

Second Stage

The second stage of the experiment was to run the scanner developed in the previous stage against web applications with known vulnerabilities. The web applications chosen for this stage are purposely designed to be vulnerable, in order to instruct the users on web application security and vulnerability detection. The chosen vulnerable web applications include Pentester Lab 2, OWASP WebGoat, HacMe Bank, and others.

Similarly in their 2012 paper (Buchler, Oudinet, & Pretschner, 2012) *Semi-Automatic Security Testing of Web Applications from a Secure Model*, the authors tested their model against WebGoat, and were able to successfully detect a role based access control vulnerability. Also in their 2010 paper (Doupé, Cova, & Vigna, 2010) *Why Johnny Can't Pentest: An Analysis of Black-box Web Vulnerability Scanners*, the authors developed a vulnerable application called WackoPicko, in order to test the web vulnerability scanners they were using in their experiment.

Third Stage

The third stage of the experiment was to run the scanner against web apps in the public domain. To achieve this twenty authenticated web applications were chosen at random. Prior to the start of the experiment it was unknown as to whether or not these applications had broken access control. The pool which the web applications will be

drawn from contains web applications belonging to a wide variety of businesses, ranging from eSports to venture capital firms. The owners of the twenty web applications were contacted prior to the beginning of this stage, and permission was granted to perform the testing required as part of the experiment, with the provision that they be alerted to potential vulnerabilities, and that the returned results are anonymous.

Fourth Stage

The fourth stage of the experiment was to manually validate the results of the previous two stages. This manual validation would be performed by a group of four full time information security consultants. The manual validation will be done by examining the returned results to identify the locations of the detected authorisation vulnerabilities, followed by an attempt to exploit these detected vulnerabilities. The exploitation of these vulnerabilities will only proceed as far as necessary to confidently state that a vulnerability exists in the detected location. This stage will show how accurately the developed scanner has detected web application vulnerabilities.

The authors of the two papers mentioned above in stage two, chose to perform the experiment with web applications that were specifically designed to be vulnerable. However while that may be good enough to prove that they can detect vulnerabilities, many of which are simple in their execution, a truer test would be to compare the results from the vulnerable websites to those of websites that may not have vulnerabilities. One of the issues with web application vulnerability scanners occurs when the scanner believes it has detected vulnerability, when in truth it has detected a false positive.

By comparing the results of web applications with known vulnerabilities, with the results of web applications that may or may not have the vulnerabilities, a greater awareness of potential false positives returned by the scanner may be gained.

Fifth Stage

The fifth stage was to repeat the second stage tests, but this time instead of the custom developed scanner, the plug-ins for the commercial tool will be used. These plug-ins as detailed above, are AuthMatrix, Authz, and Authorize. The web applications chosen for this will be the same as those used in the second stage.

The reason the same web applications will be used again for the commercial tool, is for an accurate analysis of whether the custom written scanner or the plug-ins are more accurate in the detection of authorisation vulnerabilities.

Sixth Stage

The fifth stage was to repeat the third stage tests, but this time instead of the custom developed scanner, the plug-ins for the commercial tool will be used. These plug-ins as detailed above, are AuthMatrix, Authz, and Authorize. The web applications chosen for this will be the same as those used in the third stage.

Seventh Stage

The fourth stage of the experiment was to manually validate the results of the previous two stages. This manual validation would be performed by a group of three or four full time information security consultants. This stage will show how accurately the commercial tool has detected web application vulnerabilities.

Eight Stage

The eight stage will be where the results of the developed scanner, and the results of the commercial tool, are compared to determine which is more accurate at detecting authorisation/broken access control issues.

3.5 Conclusion

This chapter examined the design of the experiment which took place, covering the following topics: experiment overview, technologies used, and experiment detail.

There was a brief overview of what the experiment would entail, a brief description of the technologies involved in the experiment – including the technologies for the custom written scanner, the commercial tool and its plug-ins, and the vulnerable web applications which were tested as part of the experiment. There was also a breakdown of the details of the experiment, detailing the stages, and how it will be performed.

4. IMPLEMENTATION AND RESULTS

4.1 Introduction

Following on from the design of the experiment in the previous chapter, this chapter will examine the implementation of the experiment and the results which were gathered. It has been broken down into two sections, experiment implementation – which will contain a description of the custom written scanner and detail how it works, and a brief description of the public domain websites which were assessed as part of the experiment. The second section, experiment results will contain the results from each of the custom written scanner, and the commercial tool plug-ins.

4.2 Experiment Implementation

This section covers the implementation of the experiment, it provides details about how the custom written scanner works, and discusses the public domain websites which were tested.

4.2.1 Custom Written Scanner

The custom written scanner was developed using Python and the client API for Selenium – which allowed the scanner to issue commands to a local instance of a browser, in this case Firefox was used. However there is no difference between what browsers are used for with the scanner, the only difference being what driver is present on the machine on which the scanner is running.

The scanner begins by requesting for the user to provide the base URL of the application to be tested, and one of more xml files containing the Selenium instructions on logging into the application. A check is performed initially to determine whether the user has supplied one or more xml files, if one xml file has been supplied the scanner will then check for direct browsing authentication vulnerabilities – accessing pages and functionality that only an authenticated user has access to while unauthenticated, however it two xml files have been supplied the scanner will then check for broken access controls otherwise known as authorisation vulnerabilities.

The program then parses through the xml Selenium commands, and builds a map of how to log into the application which it then follows. Once the application has

successfully authenticated, the program retrieves all the links on the home page, storing them in a list, to from a site map of the application. This list is then iterated through, visiting each URL in the list, and retrieving all the URLs on the page again – but only adding them to the site map if they aren't present already in the site map. Once a full site map has been built, the program then opens a new browser instance and repeats the process with the second set of login instructions, or if only one xml file was entered, it will proceed with the same steps but will no authenticate to the application first.

Once the two site maps are finished, the program then compares them to determine which URLs are present in both, and which are only present in one of the site maps. A new list is then formed of the URLs which are only present in one of the site maps, along with the details of which user session the URL was found in.

The program will then run through the new list, and attempt to access each of the URLs listed, in the open browser instance associated with the user that did not have access. If the URL is accessible in both browser instances, then it's reported to the user, otherwise the program will move onto the next URL.

4.2.2 Public Domain Websites

As part of the experiment, it was necessary to test a number of public domain websites, in order to use them as the actual experiment details. The vulnerable web applications described in the previous chapter were used as two control groups, one; to determine if the custom written scanner and the commercial tool plug-ins detected authorisation vulnerabilities when they existed on the application, and two; to determine if the custom written scanner and the commercial tool plug-ins would not detect authorisation vulnerabilities when they did not exist in the application.

Along with these two groups it was necessary in order to gain, an accurate assessment of both the custom written scanner and the commercial tool plug-ins, and in an effort to avoid bias, to assess a third control group where it was unknown whether or not the applications contained authorisation vulnerabilities.

The public domain websites chosen were selected from a pool of ten thousand public domain web applications, with the application owner's permission sought and gained before any testing of the web applications took place. This pool of ten thousand public

domain web applications included applications that ranged in purpose and technology, from electronic sports – commonly known as eSports are a form of competitions using video games - websites to online banking web applications, from online shops and power company websites to state broadcaster websites.

The twenty web applications that were chosen at random provided what could be considered a slice of life examination of web applications that are in the public domain. They reflected both web applications which the normal internet user may come across on a regular basis, as well as specialised web applications which only certain types of internet users would come across. These twenty web applications included:

- A mobile phone application marketing campaign web application – this web application allows the users to create and examine the results of a marketing campaign to be pushed to mobile applications,
- A forum website,
- A document storage web application,
- Two online gambling websites,
- A medical trial study investigator portal web application – this web application allows those organising a medical trial to examine and investigate those participating in the trial,
- An online merchandise shop,
- An online grocery shop,
- A website designer web application,
- An insurance broker website,
- Two power company websites,
- A web application to allow users remote access to a corporate network,
- A site inspection website – this web application allowed for the tracking of health and safety inspections of workplaces,
- A job listing website,
- A secure file transfer website,
- A corporate security training website,
- An insurance service website,

- A peer to peer lending website – this web application allows users to post loan applications and allows other users to crowd fund the loan, and
- A tax services accountants' website.

4.3 Experiment Results

This section details the results of the experiment that was performed, separating the results between the vulnerable web applications, and the public domain web applications. These results are further broken down for each of the custom written scanner, and the commercial tool plug-ins. It should be noted that the numbers displayed in the tables below reflect overall instances of vulnerabilities, rather than each individual instance of broken authorisation or broken access control. What this means is that in the example of a banking web application, if all the users are able to view the account information of the other users, then this has been counted as a single occurrence of the vulnerability, rather than as separate vulnerabilities, one for each of the users.

4.3.1 Results from Vulnerable Web Applications

The following table shows the overall results received when testing using both the custom written scanner, and the commercial tool plug-ins to test the vulnerable web applications that were described in the previous chapter. These results include those of two of the control groups of the experiment, the applications that are known to contain broken access controls or authorisation vulnerabilities, and the applications where it is known that they don't contain broken access controls or authorisation vulnerabilities. The results have been broken down into four values: total vulnerabilities – this number shows the total amount of authorisation vulnerabilities that were detected in the application, false negatives – this number shows the total amount of authorisation vulnerabilities that exist in the web application which were missed by one or more of the scanning tools, false positives – this number shows the total amount of detected authorisation vulnerabilities which were found to be non-existent after manual validation by a group of security experts, and real vulnerabilities – this number shows the total amount of detected authorisation vulnerabilities which were found to be valid issues by a group of security experts.

Application Name	Total Detected	False Negatives	False Positives	Real Vulnerabilities
HacMe Bank	3	3	0	3
OWASP WebGoat	0	2	0	2
OWASP Juice Shop	0	1	0	1
Pentester Lab 2	1	2	0	3
Damn Vulnerable Web App	0	0	0	0
Bwapp	0	1	0	1
Altoro Mutual	0	0	0	0
Mutillidae 2	0	1	0	1
OWASP Bricks	0	0	0	0
OWASP Rails Goat	1	0	0	1
OWASP Vicnum	0	0	0	0
OWASP WebGoat.NET	0	0	0	0
WackoPicko	0	0	0	0
BodgeIt	0	0	0	0
OWASP AppSensor Demo Application	1	0	0	1

Table 1 – Total Vulnerabilities detected in Vulnerable Web Applications by all three scanning tools

As can be seen in Table 1, there were no false positives detected by any of the scanning tools, when they were used to test each of the web applications for broken access controls or authorisation vulnerabilities. There was however a number of false negatives found, these occurred where there was no direct link to the vulnerable functionality in the application, instead the user needed to guess the URL in order to gain access.

The following table shows the overall results received when testing using the custom written scanner, and the commercial tool plug-ins. The results have been broken down into each of the commercial tool plug-ins, and the custom written scanner.

Application Name	AuthMatrix	Authz	Authorize	Custom Script
HacMe Bank	3	3	3	0
OWASP WebGoat	0	0	0	0
OWASP Juice Shop	0	0	0	0
Pentester Lab 2	1	1	1	1
Damn Vulnerable Web App	0	0	0	0
Bwapp	0	0	0	0
Altoro Mutual	0	0	0	0
Mutillidae 2	0	0	0	0
OWASP Bricks	0	0	0	0
OWASP Rails Goat	1	1	1	1
OWASP Vicnum	0	0	0	0
OWASP WebGoat.NET	0	0	0	0
WackoPicko	0	0	0	0
BodgeIt	0	0	0	0
OWASP AppSensor Demo Application	1	1	1	1

Table 2 - Vulnerabilities detected by each of the technologies

As can be seen in Table 2 above the custom script was at par with the three commercial tool plug-ins for each of the applications bar one. The web application in question is HacMe Bank, which uses JavaScript for navigation. This will be discussed more in the next chapter in the scanning tools evaluation.

4.3.2 Results from Public Domain Web Applications

The following table shows the overall results received when testing using both the custom written scanner and the commercial tool plug-ins. This table shows the results of received after testing the third control group, the applications where it is unknown whether or not they contain broken access controls or authorisation vulnerabilities. The results have been broken down in the same manner as those of Table 1.

Application Name	Total Detected	False Negatives	False Positives	Real Vulnerabilities
Mobile phone app marketing campaign app	3	3	0	3
Forum website	1	0	1	0
Document storage app	2	0	0	2
Betting Website	1	0	0	1
Medical Trial Study Investigator Portal	2	0	0	2
Online merchandising shop	3	0	0	3
Online grocery shop	2	0	0	2
Website designer	2	0	1	1
Betting Website	1	0	1	0
Insurance Broker Site	1	0	0	1
Power Company Website	2	0	0	2
Remote Access Website	0	0	0	0
Site Inspection Website	3	0	1	2
Job Listing Website	2	0	0	2
Power Company Website	0	0	0	0
Secure File Transfer	0	0	0	0
Corporate Security Training Website	1	0	0	1
Insurance Service Site	2	0	1	1
Peer to Peer Lending Website	1	0	0	1
Tax Services Website	2	1	0	2

Table 3 - Total Vulnerabilities detected in Public Domain Web Applications by all three scanning tools

As can be seen in Table 3 there were a number of false positives and false negatives found when testing the public domain web applications. The majority of false

negatives were found in the mobile phone application marketing campaign application, on reviewing the results from this application it was noted that the application uses what's known as anti-CSRF tokens as headers, and as so some of the scanning tools were unable to detect any vulnerabilities. This will be discussed more in the next chapter in the scanning tools evaluation.

The following table shows the overall results received when testing using the custom written scanner, and the commercial tool plug-ins. The results have been broken down into each of the commercial tool plug-ins, and the custom written scanner.

Application Name	**AuthMatrix**	**Authz**	**Authorize**	**Custom Script**
Mobile phone app marketing campaign app	0	2	0	3
Forum website	0	1	0	0
Document storage app	2	2	2	2
Betting Website	1	1	1	1
Medical Trial Study Investigator Portal	2	2	2	2
Online merchandising shop	3	3	3	3
Online grocery shop	2	2	2	2
Website designer	1	2	1	1
Betting Website	0	1	0	0
Insurance Broker Site	1	1	1	1
Power Company Website	2	1	2	2
Remote Access Website	0	0	0	0
Site Inspection Website	2	3	3	2
Job Listing Website	2	2	2	2
Power Company Website	0	0	0	0
Secure File Transfer	0	0	0	0
Corporate Security Training Website	1	1	1	1
Insurance Service Site	1	2	2	1
Peer to Peer Lending Website	1	1	1	1
Tax Services Website	2	1	2	2

Table 4 - Vulnerabilities detected by each of the technologies

As can be seen in Table 4 the custom script was again on par with the commercial tool plug-ins, and in some cases it performed better, notably in the case of the Mobile phone application marketing campaign application.

4.3 Conclusion

This chapter examined the implementation of the experiment and the results which were gathered. It was broken down into two sections, experiment implementation – which contained a description of the custom written scanner and details about how it works, and a brief description of the public domain websites which were assessed as part of the experiment. The second section, experiment results contained the results from each of the custom written scanner, and the commercial tool plug-ins.

It was found through the results of each of the scanning tools, that the custom script and AuthMatrix were the most accurate scanning tools with eighty-three percent accuracy, while Authorize was found to be eighty percent accurate, and Authz was found to only be sixty-six percent accurate.

5. ANALYSIS, EVALUATION, AND DISCUSSION

5.1 Introduction

Following on from the previous chapter wherein the implementation of the experiment was detailed, and the results from the experiment were provided, this chapter will discuss the results and the experiment itself. It has been broken down into two parts: the first part technologies evaluation will evaluate and discuss the two main technologies used in the experiment, the custom written scanner, and the commercial tool plug-ins. It will detail any strengths and limitations of these technologies as they were noted during the course of the experiment.

The second part, results evaluation, will evaluate and discuss the results received over the course of the experiment. This has also been split into two parts: the first focuses on the results derived from the vulnerable web applications, while the second part focuses on the results derived from the public domain web applications.

5.2 Technologies Evaluation

This section focuses on the evaluation of both the custom written scanner, and the commercial tool plug-ins, in regards to their strengths and weaknesses, as noted during the course of the experiment.

5.2.1 Custom Written Scanner

The custom written scanner that was developed as part of the experiment was able to accurately detect authorisation vulnerabilities without reporting any false positives, with a minimum of false negatives. There was only one application which was tested that the custom written scanner reported false negatives, that is where the custom written scanner missed real vulnerabilities. This application was called HacMe Bank, and it used JavaScript for navigation. Out of a total of thirty-five applications which were tested, the custom written scanner was able to find the correct number of authorisation vulnerabilities in twenty-nine of web applications tested, resulting in an accuracy of eighty-three percent. There were some strengths and limitations noted over the course of the experiment, as detailed below.

Strengths

The following strengths were noted during use of the custom written scanner over the course of the experiment:

- The custom written scanner was easy to use, with the user only needing to enter the base URL of the application, and specify the file containing the login instructions.
- The custom written scanner is able to handle requests from the application which include CSRF tokens in either the header or in the request body.
- The custom written scanner handles the session for each user automatically, allowing the user to "point and click" regarding testing of a web application.

Limitations

The following limitations were noted during use of the custom written scanner over the course of the experiment:

- During the crawling phase, where the scanner builds up the site maps of the application, the custom written scanner was unable to successfully parse links which use JavaScript. The scanner looks for all the URLs that are listed on a page in the application, and uses these URLs to build up the site map, this is the way that most web application scanners both commercially developed or otherwise builds a site map for a web application. However if the application uses JavaScript for navigation, then there are no URLs for the scanner to parse. One possible way of resolving this issue, would be to force the custom scanner to open any links that use JavaScript in a new tab or window, and check the whether the URL is of the same format that is expected for the application. However this would require the complete rewriting of the crawling functionality in the custom scanner, and due to time constraints this was unable to be implemented.

5.2.2 Commercial Tool Plug-ins

5.2.2.1 AuthMatrix

The AuthMatrix plug-in for Burp Suite, which was used to test web applications as part of the experiment was able to accurately detect authorisation vulnerabilities

without reporting any false positives, with a minimum of false negatives. Out of a total of thirty-five applications which were tested, AuthMatrix was able to find the correct number of authorisation vulnerabilities in twenty-nine of web applications tested, resulting in an accuracy of eighty-three percent. There were some strengths and limitations noted over the course of the experiment, as detailed below.

Strengths

The following strengths were noted during use of the AuthMatrix plug-in for Burp Suite over the course of the experiment:

- AuthMatrix had an easy to use and intuitive GUI

Limitations

The following limitations were noted during use of the AuthMatrix plug-in for Burp Suite over the course of the experiment:

- AuthMatrix is unable to handle requests from the application which include CSRF tokens in either the header or in the request body.
- AuthMatrix requires the user to build up a matrix of all the requests and user types of the target application, before testing can begin.
- AuthMatrix requires the user to keep two different user sessions active for the duration of the tests necessitating the use of two different browsers.
- AuthMatrix has no web application crawling functionality, instead it relies on the tester to manually build the target application site map, and send requests to AuthMatrix for testing.

5.2.2.2 Authz

The Authz plug-in for Burp Suite, which was used to test web applications as part of the experiment was unable to accurately detect authorisation vulnerabilities without reporting any false positives, and also reporting a number of false negatives. As noted in the results in the previous chapter, there were several instances where Authz reported an authorisation vulnerability, which after manual validation by a group of security experts was shown to be a false positive. Out of a total of thirty-five applications which were tested, Authz was able to find the correct number of authorisation vulnerabilities in twenty-three of web applications tested, resulting in an

accuracy of sixty-six percent. There were some strengths and limitations noted over the course of the experiment, as detailed below.

Strengths

The following strengths were noted during use of the Authz plug-in for Burp Suite over the course of the experiment:

- Authz is able to handle requests from the application which include CSRF tokens in either the header or in the request body.

Limitations

The following limitations were noted during use of the Authz plug-in for Burp Suite over the course of the experiment:

- Authz is able to handle requests from the application which include CSRF tokens in either the header or in the request body.
- Authz has no web application crawling functionality, instead it relies on the tester to manually build the target application site map, and send requests to Authz for testing.
- Authz requires the user to keep two different user sessions active for the duration of the tests necessitating the use of two different browsers.

5.2.2.3 Authorize

The Authorize plug-in for Burp Suite, which was used to test web applications as part of the experiment was able to accurately detect authorisation vulnerabilities without reporting any false positives, with a minimum of false negatives. Out of a total of thirty-five applications which were tested, Authorize was able to find the correct number of authorisation vulnerabilities in twenty-eight of web applications tested, resulting in an accuracy of eighty-three percent. There were some strengths and limitations noted over the course of the experiment, as detailed below.

Strengths

The following strengths were noted during use of the Authorize plug-in for Burp Suite over the course of the experiment:

- Authorize has no web application crawling functionality, instead once the user activates the plug-in Authorize then passively monitors the Burp Suite proxy, and repeats all the requests with the different user specified session tokens.

Limitations

The following limitations were noted during use of the Authorize plug-in for Burp Suite over the course of the experiment:

- Authorize is unable to handle requests from the application which include CSRF tokens in either the header or in the request body.
- Authorize requires the user to keep two different user sessions active for the duration of the tests, usually through the use of two different browsers.
- Authorize requires the user to specify a string that indicates that proper access controls are in place, however if the wrong string is used, this can lead to false positives

5.2.3 Technologies Evaluation Summation

The following table summarises the evaluation of the technologies that were detailed above, detailing the relative strengths and weaknesses that were found between the custom developed scanner, and the three commercial tool plug-ins. It has been divided into six categories:

- Crawler – this category details how the crawling of the scanning tool is handled, whether it is automated, or performed manually by the user
- JavaScript – this category details whether or not the tool was able to handle navigation which uses JavaScript
- Anti-CSRF Tokens – this category details whether or not the tool was able to handle applications which send anti-CSRF tokens as part of the request body, or in the headers
- Sessions – this category details how the tool handles sessions, whether it is performed by the tool itself, or performed manually by the user
- GUI – this category details whether or not the tool had a GUI
- Ease of Use – this category details how easy it was to use the scanning tool

Tool	Crawler	JavaScript	Anti-CSRF Tokens	Sessions	GUI	Ease of Use
AuthMatrix	Manual	Yes	No	User	Yes	User required to build user matrix before beginning
Authz	Manual	Yes	Yes	User	Yes	User required to manually send each request
Authorize	Manual	Yes	No	User	Yes	User required to provide tool with string denoting no access to restricted functionality
Custom Script	Automated	No	Yes	Tool	No	User required to provide XML file containing login instructions, and base URL

Table 5 - Comparison of Scanning Tools

5.3 Results Evaluation

This section focuses on the evaluation of the results derived over the course of the experiment from the custom written scanner, and the commercial tool plug-ins, in regards to detected authorisation vulnerabilities in the tested applications.

5.3.1 Results from Vulnerable Web Applications

Figure 20 as seen below shows the comparison of total vulnerabilities found in each of the vulnerable web applications that were tested prior to the experiment, broken down into total vulnerabilities – this number shows the total amount of authorisation vulnerabilities that were detected in the application, false positives – this number

shows the total amount of detected authorisation vulnerabilities which were found to be non-existent after manual validation by a group of security experts, and real vulnerabilities – this number shows the total amount of detected authorisation vulnerabilities which were found to be valid issues by a group of security experts. These applications were tested by a group of web application security experts before the experiment occurred, in order to determine how much vulnerabilities each application contained – this could be done since those applications with vulnerabilities were purposely designed that way, and so the amount of vulnerabilities contained in each application could easily be determined.

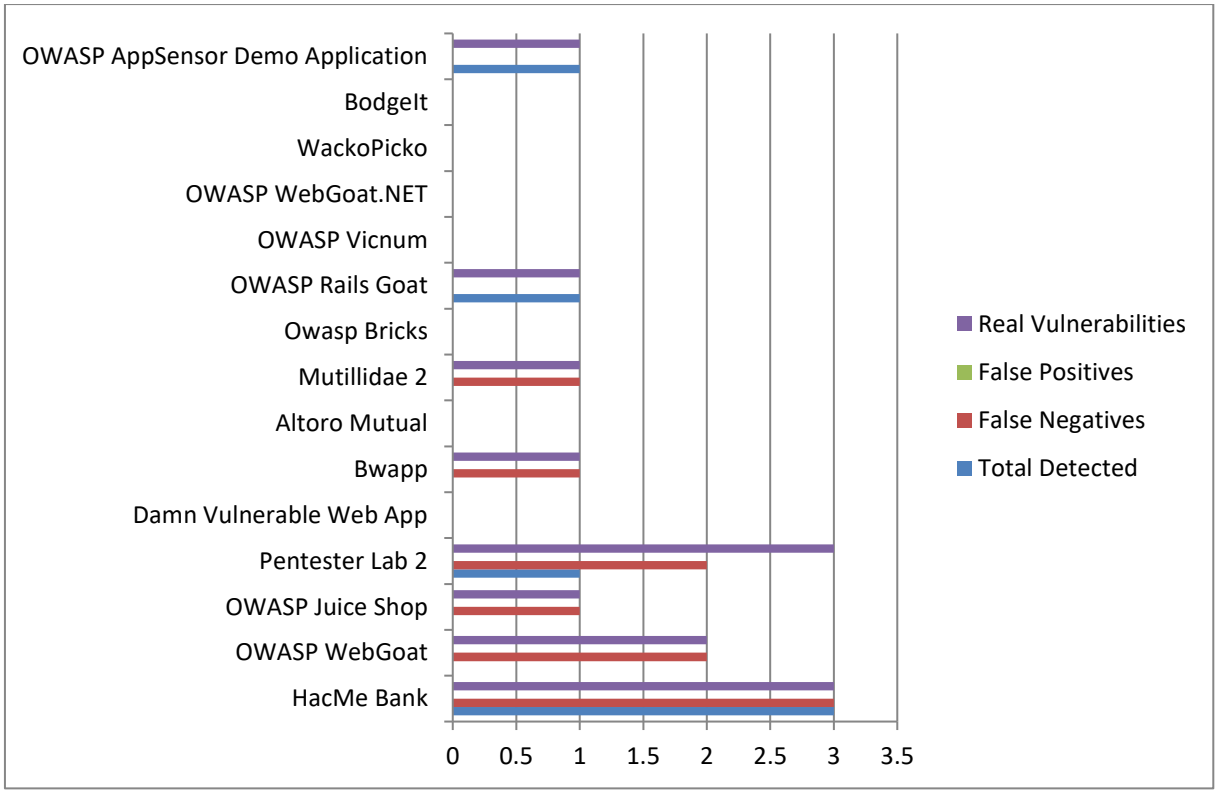

Figure 21 - Comparison of Vulnerabilities found in Vulnerable Web Applications

As can be seen in Figure 21, eight of the vulnerable web applications tested contained broken access controls or authorisation vulnerabilities. However when the web applications were tested using the custom written scanner and the commercial tool plug-ins only four of the vulnerable web applications were found to contain broken access controls or authorisation vulnerabilities, as can be seen in figure 22. Another point of interest is that in figure 22 there was only one broken access control or authorisation vulnerability detected by each of the scanning tools, however in figure 21 there were three issues detected.

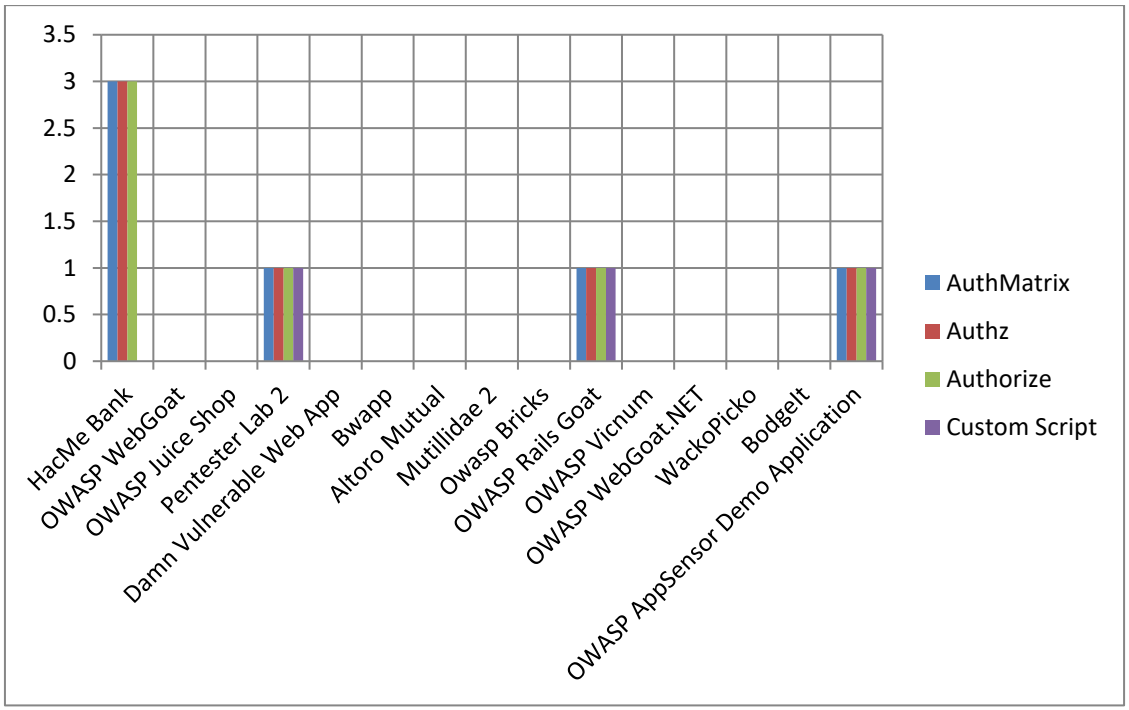

Figure 22 - Comparison of Vulnerabilities found in each vulnerable web application broken down by scanning tool

The disparities between these two figures have a simple explanation, each of these vulnerable web applications contain a vulnerability wherein the user can directly browse to administrator functionality, through the use of appending a string to the URL. The scanning tools used during the experiment work, in essence, by using two different user accounts, and comparing the site maps in the case of the custom written scanner, and by replaying requests associated with each role in the case of the commercial tool plug-ins.

Another interesting point is that in one of the vulnerable web applications HacMe Bank, the commercial tool plug-ins all found three vulnerabilities, while the custom written scanner did not find any. The reasoning behind this however has already been touched upon in the evaluation of the custom written scanner, as the application uses JavaScript for navigation and the custom written scanner is unable to crawl a website that uses JavaScript for navigation.

5.3.2 Results from Public Domain Web Applications

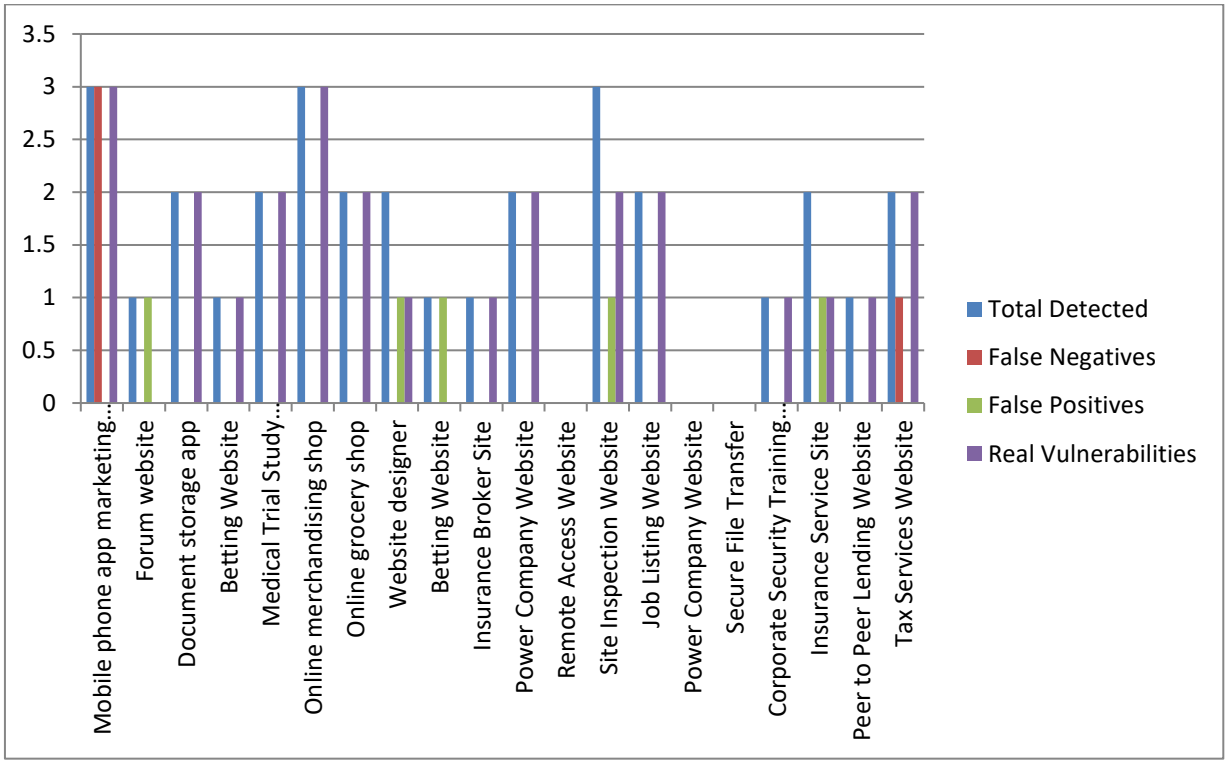

Figure 23 - Comparison of Vulnerabilities found in Public Domain Web Applications

Figure 22 as shown above displays a comparison of total vulnerabilities found in each of the public domain web applications that were tested during the experiment, broken down into total vulnerabilities – this number shows the total amount of broken access controls or authorisation vulnerabilities that were detected in the application, false positives – this number shows the total amount of detected broken access controls or authorisation vulnerabilities which were found to be non-existent after manual validation by a group of security experts, and real vulnerabilities – this number shows the total amount of detected broken access controls or authorisation vulnerabilities which were found to be valid issues by a group of security experts. The interesting point to note here is that out of the twenty web applications tested fifteen contained broken access controls or authorisation vulnerabilities; this means that seventy-five percent of the tested applications contained these vulnerabilities. Another interesting point that can be found in figure 22, is that five of the applications where broken access controls or authorisation vulnerabilities were detected, contained false positive findings, with two of these applications being found to contain no broken access controls or authorisation vulnerabilities after being tested by a group of computer security experts, in other words forty percent of the times false positives were detected

the web applications contained no broken access controls or authorisation vulnerabilities.

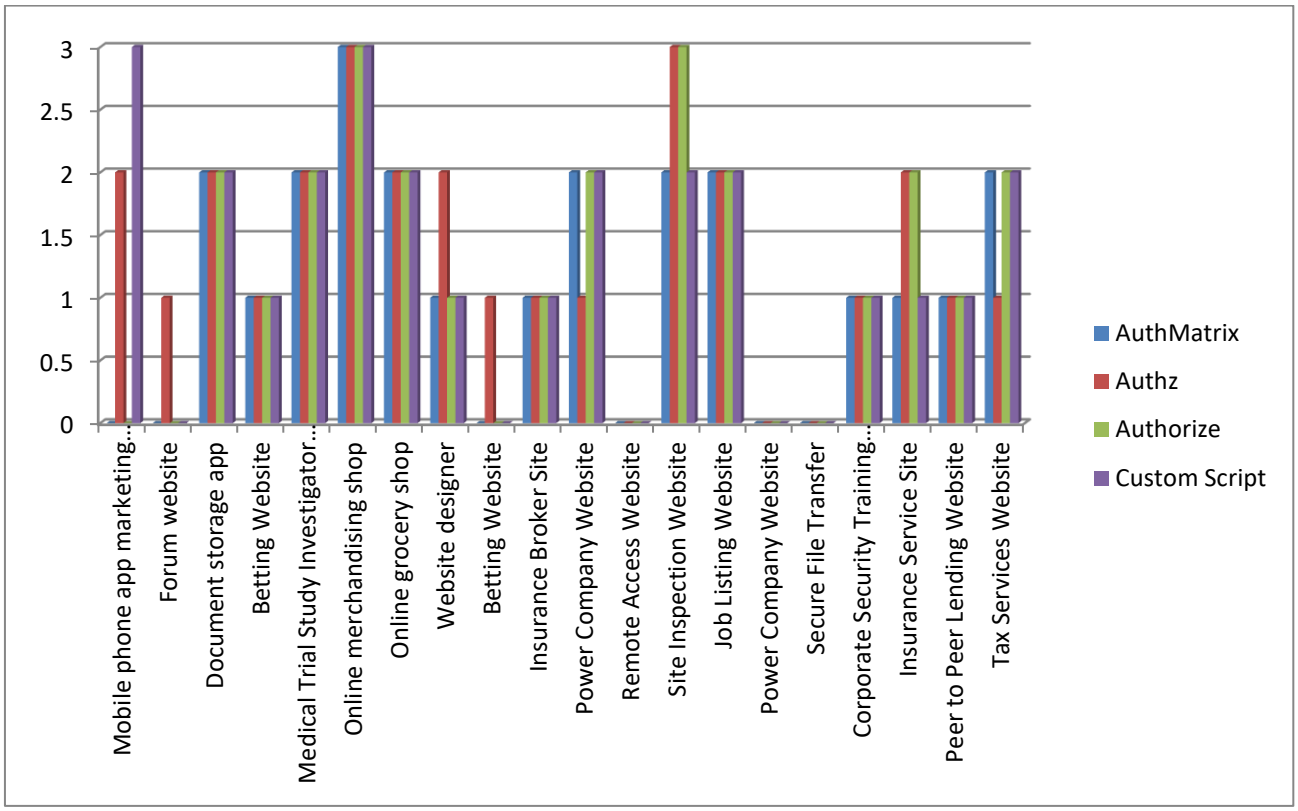

Figure 24 - Comparison of Vulnerabilities found in each public domain web application broken down by scanning tool

These numbers have been further broken down in figure 23, showing how much of the detected broken access controls or the authorisation vulnerabilities, were found by each of the scanning tools. Of particular note here is that for each of the applications where broken access controls or authorisation vulnerabilities were found, the custom written scanner was successfully able to identify the issue, without any false positives. One of the commercial tool plug-ins, AuthMatrix, was also able to successfully identify broken access controls or authorisation vulnerabilities in the majority of the applications. However, there was one application which AuthMatrix was unable to detect any issues, that being the Mobile phone app marketing campaign app. The reasoning behind this was touched upon in the evaluation of the commercial tool plug-ins, that being because the web application in question used anti-CSRF tokens in either the body of the request or in the request headers. As the AuthMatrix and Authorize plug-ins use session cookies when testing the application, neither of these were able to detect any issue for that particular application.

Another point of interest from these figures is that there is one commercial tool plug-in which consistently returned false positives for applications whenever a false positive was returned. This plug-in being Authz, with Authorize also returning some false positives, however Authz returned a far greater portion of the false positives than Authorize.

5.3.3 Comparison of custom written scanner and commercial tool plug-ins

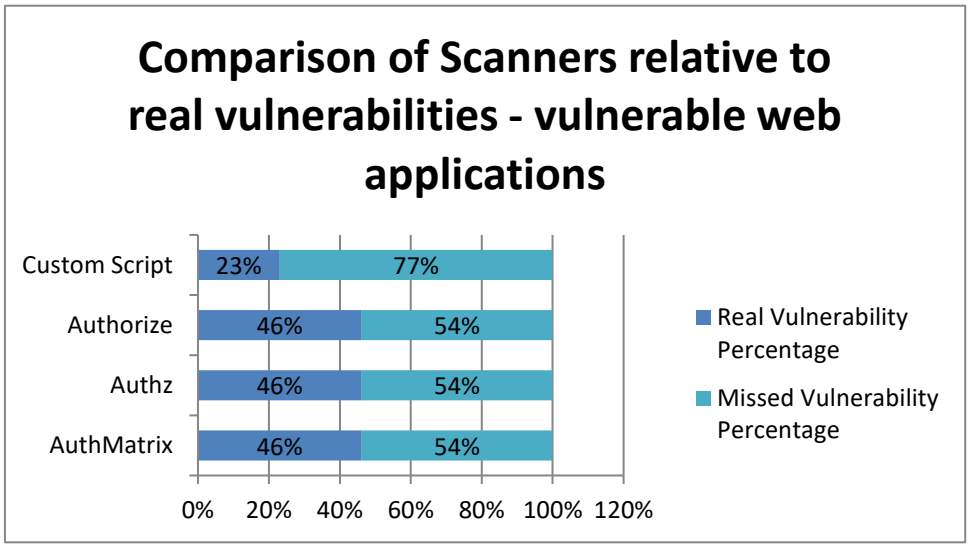

Figure 25 - Comparison of Scanners relative to real vulnerabilities - vulnerable web applications

The figure above shows the relative percentage of broken access controls or authorisation vulnerabilities which each of the scanning tools detected in the vulnerable web applications. Of note is that each of the commercial tool plug-ins AuthMatrix, Authz, and Authorize found fifty percent of the intended vulnerabilities while the custom written scanner only found twenty-five percent of the intended vulnerabilities. This disparity has been touch upon above in both the technology evaluation and the evaluation of the results from the vulnerable web applications. However it will be restated here, the reason behind the disparity is that one of the vulnerable web applications, which contained three instances of broken access controls or authorisation vulnerabilities, and which accounts for twenty-five percent of the total vulnerabilities across all the vulnerable web applications, uses JavaScript for navigation. The custom written scanner is unable to navigate a web application which

uses JavaScript, while the commercial tool plug-ins which requires the user to manually build up the site map of the web application, and are unconcerned with whether an application uses JavaScript for navigation.

However a difference can be seen, when the results of the public domain websites are examined, as seen in figure 26. The custom written scanner performed better than the commercial tool plug-ins, where one hundred percent of the broken access controls or authorisation vulnerabilities it detected, were found to be real vulnerabilities after manual validation by a group of computer security experts. This is compared to eighty-eight percent for AuthMatrix, ninety-six percent for Authorize, and one hundred and eight percent for Authz respectively – the percentage for Authz comes from the extra volume of false positives which it detected compared to real vulnerabilities. The high percentage of detected vulnerabilities for the Authz plug-in correlates to figure 27, wherein the relative percentage of each scanning tool in regards to false positives is shown. The Authz plug-in was found to have produced one hundred percent of the false positives, while forty percent of these were also detected by the Authorize plug-in. This is in contrast to the AuthMatrix plug-in and the custom written scanner, which both found no false positives during the course of the experiment.

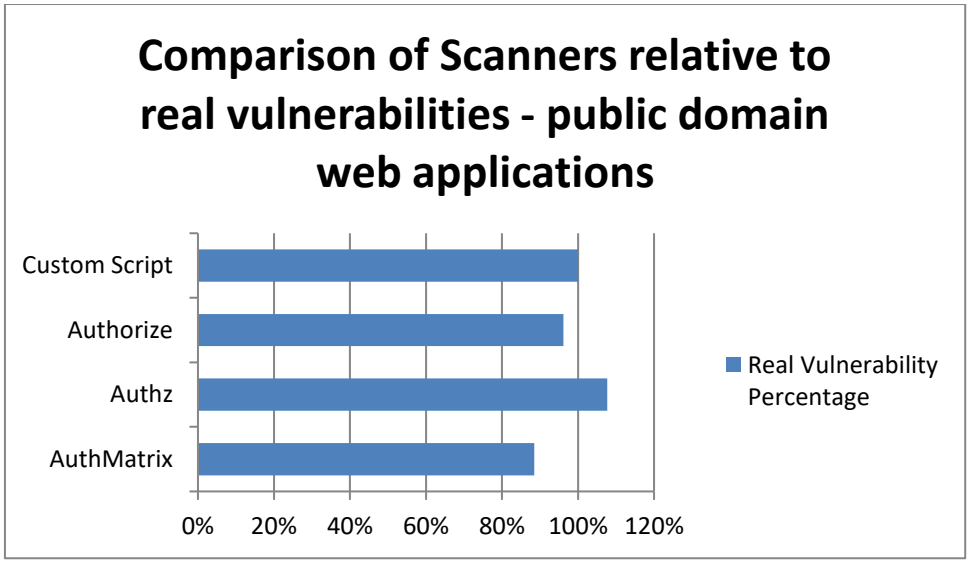

Figure 26 - Comparison of Scanners relative to real vulnerabilities - public domain web applications

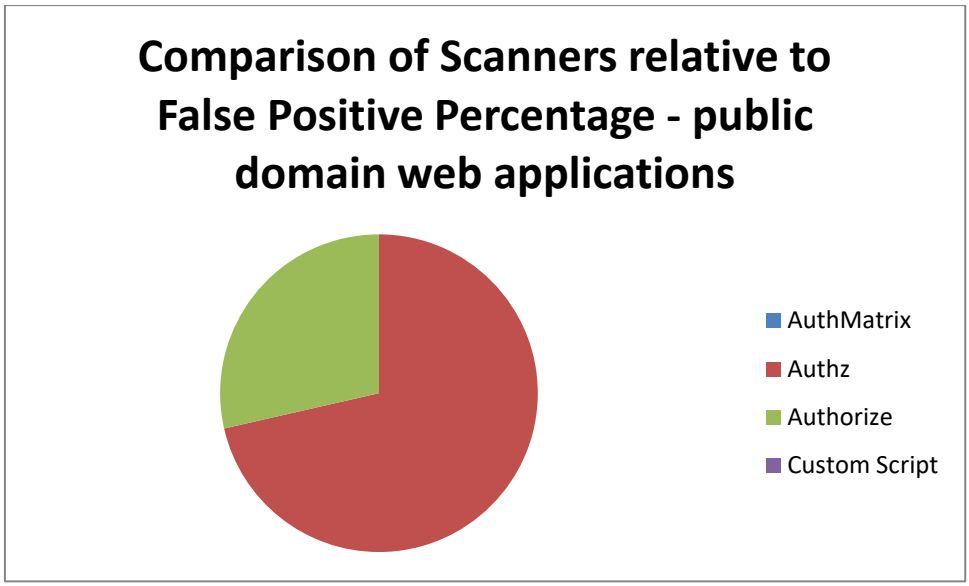

Figure 27 - Comparison of Scanners relative to False Positive Percentage - public domain web applications

Also of importance is to note how much broken access controls or authorisation vulnerabilities each of the scanning tools examined missed over the course of the experiment, this can be seen in figure 28. The three plug-ins for the commercial tool, AuthMatrix, Authz, and Authorize, each missed twelve percent of the total number of vulnerabilities, while the custom written scanner missed none.

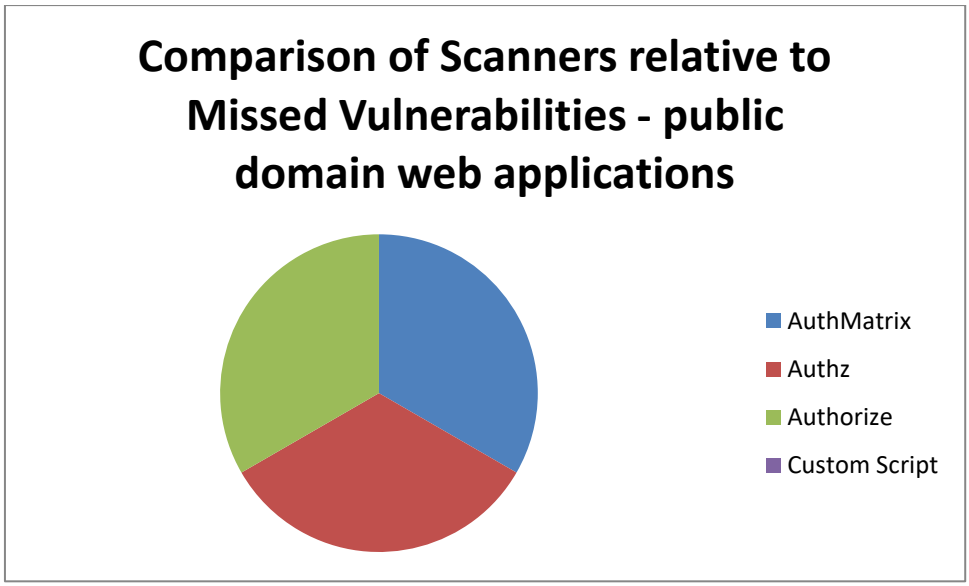

Figure 28 - Comparison of Scanners relative to Missed Vulnerabilities - public domain web applications

5.4 Conclusion

This chapter examined and evaluated the results generated during the experiment. It was broken down into two parts: first the technology evaluation, and second the results evaluation.

The technology evaluation looked at the three commercial tool plug-ins, and the custom written scanner. It examined the strengths and limitations of each of these scanning tools, as they were noted during the experiment.

The results evaluation looked at the results generated from the testing of the vulnerable web applications, and the public domain web applications. It examined the findings of the scanning tools in regards to both vulnerable web applications, and the public domain web applications which were tested over the course of the experiment. In addition to this the performance of each of the scanning tools, were compared in terms of real vulnerabilities detected, real vulnerabilities missed, and false positives detected. It is also clear to see that although lacking in some areas, the custom written scanner, provides an overall assessment of a target application that's as accurate as the commercial tool plug-ins in some regards, and in other regards is more accurate, despite its limitations.

The recommend approach in light of the evaluation would be for multiple tools to be used when attempting to automate the detection of these vulnerabilities, to validate the findings of one tool, with those of another.

6. CONCLUSION

6.1 Research Overview

The objective of this research was to examine whether the tests performed to test an authenticated web application could be automated, in such a way that the person performing a penetration test of the web application need not interact with the automated tool beyond initiation, and monitoring for errors.

In the introduction chapter this was summarised by the following question.
Will a custom developed web application scanner, provide a more accurate analysis than plug-ins developed for commercial grade software in the detection of authorisation vulnerabilities?
In order to best approach this question, the research was broken down into four avenues:

1. Identify other previous work in the this area, and determine whether there is a potential avenue of experimentation, which has not been considered before
2. Use the identified plug-ins and the custom python script to identify potential authorisation vulnerabilities in a group of website.
3. To manually validate the results of the tools, through the use of experts in the field of web application security, in order to identify which of the results are valid authorisation vulnerabilities, and which are in fact false positives.
4. To compare the manually validated results, in order to answer the question stated above.

6.2 Experimentation Design, Results and Evaluation

6.2.1 Design

The experiment was broken down into the following eight stages:

1. To create the web application scanner
2. Run the web application scanner against web applications with known vulnerabilities
3. Run the web application scanner against public domain web applications

4. Have a group of computer security experts manually validate the results of the previous two stages
5. Run the commercial tool plug-ins against web applications with known vulnerabilities
6. Run the commercial tool plug-ins against public domain web applications
7. Have a group of computer security experts manually validate the results of the previous two stages
8. Compare the results of the developed web application scanner, with those of the commercial tool plug-ins

These eight stages were designed in order to best achieve the stated objective of the experiment. The custom written scanner that was created as part of the first stage was designed to minimise the effort the user needed to interact with the scanner in order to test the application. In this way it contrasted with the commercial tool plug-ins which required more user interaction in order to test the application.

6.2.2 Results

The results were split into two different categories: those that were derived from the vulnerable web applications and those that were derived from the public domain web applications. The vulnerable web applications were tested by a group of computer security experts prior to the beginning of the experiment, this was done in order to determine how many instances of broken access controls or authorisation vulnerabilities were present in each of the applications, a figure needed when comparing the results of the each individual scanning tool.

It was found through the results of each of the scanning tools, that the custom script and AuthMatrix were the most accurate scanning tools with both detecting twenty-nine vulnerabilities out of a maximum of thirty-nine giving them an overall result of seventy-four percent accuracy, while Authorize was found to have detected twenty-eight vulnerabilities out of a maximum of thirty-nine giving it an overall result of seventy-two percent accuracy, while finally Authz was found to only be fifty-nine percent accurate having only detected twenty-three out of a possible thirty-none vulnerabilities.

6.2.3 Evaluation

The results returned from the experiment were compared and evaluated based on the total number of vulnerabilities, both real and false positive found in each of the vulnerable web applications, and the public domain web applications. They were further broken down and compared based on each of the scanning tools.

The scanning tools were also compared and evaluated based on the relative amount of real vulnerabilities each tool detected, false positives each tool detected, and on the number of vulnerabilities each tool didn't detect but other tools did.

The discrepancies between the different scanning tools can be traced back to their functionality. The custom scanning tool was unable to parse the URLs of web applications which used JavaScript for navigation, while AuthMatrix and Authorize were unable to detect vulnerabilities when the web applications used anti-CSRF tokens in the request headers or bodies.

6.2.4 Limitations

There are some limitations to the experiment which were noted over the duration:

- The experiment took place using fifteen vulnerable web applications, and twenty public domain web applications.
- All of the web applications tested used cookies as a means to manage sessions; however some web applications do not use cookies for session management.
- No application programming interfaces or APIs were tested as part of the experiment
- Due to time constraints each of the public domain web applications were only tested once. This can have an effect on the results due to the possibility that different URLs may be available on different occasions for each of the web applications. For example, if the request for a URL timed out due the experiment, the result would not necessarily be accurate.

6.3 Future Work and Recommendations

There are a number of possible suggestions for where future work in this area could lead, or where additional experimentation could occur:

Due to the increasing trend of using application programming interface or API back ends to serve both web applications and mobile applications, APIs are becoming more common, and so is the need for proper access controls on API back ends.

With the increasing popularity of APIs, and the growing concern over cookies tracking users across multiple websites, there is a growing trend towards web applications that don't use cookies. The cookies can typically be replaced with a unique value in the request header or body similar to the anti-CSRF tokens which were described earlier.

6.4 Final Conclusion

The objective of this research as stated above in the research overview was to examine whether the tests performed to test an authenticated web application could be automated, in such a way that the person performing a penetration test of the web application need not interact with the automated tool beyond initiation, and monitoring for errors.

It is clear that while there was some limitations noted during the evaluation of the custom written scanner, that it is capable of matching and at time exceeding the capabilities of the commercial tool plug-ins. However further work will need to be done on the custom written scanner, in order to overcome the stated limitations.

BIBLIOGRAPHY

Afolaranmi, S. O., Ferrer, B. R., Mohammed, W. M., Lastra, J. L., Ahmad, M., & Harrison, R. (2017). Providing an Access Control layer to Web-Based Applications for the industrial domain. *2017 IEEE 15th International Conference on Industrial Informatics* (pp. 1096-1102). IEEE.

Almehmadi, A., & El-Khatib, K. (2013). Authorized! Access Denied, Unauthorized! Access Granted . *SIN'13* (pp. 363-367). Aksaray, Turkey: ACM.

Alqahtani, S., He, X., & Gamble, R. (2017). Adapting Compliance of Security Requirements in Multi-Tenant Applications. *2017 IEEE 2nd International Workshops on Foundations and Applications of Self* Systems (FAS*W)* (pp. 122-129). IEEE.

Assad, R. E., Tarciana, K., Ferraz, F. S., Ferreira, L. P., & Meira, S. R. (2010). Security Quality Assurance on Web-based Application Through Security Requirements Tests. *2010 Fifth International Conference on Software Engineering Advances* (pp. 272-277). IEEE.

Bocic, I., & Bultan, T. (2016). Finding Access Control Bugs in Web Applications with CanCheck. (pp. 155-166). Singapore: ACM.

Braz, F. A., Fernandez, E. B., & VanHilst, M. (2008). Eliciting Security Requirements through Misuse Activities. *19th International Workshop on Database and Expert Systems Application, 2008. DEXA '08.* . Turin: IEEE.

Buchler, M., Oudinet, J., & Pretschner, A. (2012). Semi-Automatic Security Testing of Web Applications from a Secure Model. *IEEE* (pp. 253-262). IEEE.

Creasey, J., & Glover, I. (2013). Penetration Test Procurement Buyers Guide. Great Britain: CREST.

Doupé, A., Cova, M., & Vigna, G. (2010). Why Johnny Can't Pentest:An Analysis of Black-box Web Vulnerability Scanners. Santa Barbara: University of California.

Gajanayake, R., Iannelle, R., & Sahama, T. (2011). Privacy by Information Accountability for E-Health Systems. *2011 6th International Conference on Industrial and Information Systems* (pp. 49-53). Sri Lanka: IEEE.

Grobauer, B., Walloschek, T., & Stocker, E. (2010, June 17). Understanding Cloud Computing Vulnerabilities. *IEEE Security & Privacy*, pp. 50-57.

Hardt, D. (2012, Octoer). The OAuth 2.0 Authorization Framework. Internet Engineering Task Force.

Indu, I., & Rubesh Anand, P. M. (2016). Hybrid Authentication and Authorization Model for Web based Applications . (pp. 1187-1191). IEEE.

Kagal, L., Finin, T., Paolucci, M., Srinivasan, N., Sycara, K., & Denker, G. (2004). Authorisation and privacy for semantic web services. *IEEE Intelligent Systems*, 50-56.

Kumar, R. (2011). Mitigating the Authentication Vulnerabilities in Web Applications through Security Requirements. *2011 World Congress on Information and Communication Technologies* (pp. 1294-1298). IEEE.

Kumar, R. (2015). Analysis of Key Critical Requirements for Enhancing Security of Web Applications . *2015 International Conference on Computers, Communications, and Systems* (pp. 241-245). IEEE.

Marinescu, P., Parry, C., Pomarole, M., Tian, Y., Tague, P., & Papagiannis, I. (2017). IVD: Automatic Learning and Enforcement of Authorization Rules in Online Social Networks. *2017 IEEE Symposium on Security and Privacy* (pp. 1094-1109). IEEE.

Muthukumaran, D., O'Keeffe, D., Priebe, C., Eyers, D., Shand, B., & Pietzuch, P. (2015). FlowWatcher: Defending against Data Disclosure Vulnerabilities in Web Applications . (pp. 603-615). Denver: ACM.

Near, J. P., & Jackson, D. (2016). Finding Security Bugs in Web Applications using a Catalog of Access Control Patterns. *2016 IEEE/ACM 38th IEEE International Conference on Software Engineering.* Austin: IEEE.

Ni, Q., Bertino, E., Lobo, J., Brodie, C., Karat, C.-M., Karat, J., & Trombeta, A. (2010). Privacy-aware role-based access control. *ACM Transactions on Information and System Security*.

NIST. (2008, September). Technical Guide to Information Security Testing and Assessment. Gaithersberg, U.S.A.: NIST.

Noseevich, G., & Petukhov, A. (2011). Detecting Insufficient Access Control in Web Applications. *SysSec Workshop (SysSec), 2011 First* (pp. 11-18). Amsterdam, Netherlands: IEEE.

Omotunde, H., & Ibrahim, R. (2016). A Hybrid Threat Model for Software Security Requirement Specification . IEEE.

Pato, J., Paradesi, S., Jacobi, I., Shih, F., & Wang, S. (2011). Aintno: Demonstration of Information Accountability on the Web. *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on* (pp. 1072-1081). Boston: IEEE.

Schiffman, J., Zhang, X., & Gibbs, S. (2010). DAuth: Fine-grained Authorization Delegation for Distributed Web Application Consumers. *2010 IEEE International Symposium on Policies for Distributed Systems and Networks* (pp. 95-102). IEEE.

Shah, S., & Mehtre, B. M. (2014). An Automated Approach to Vulnerability Assessment and Penetration Testing using Net-Nirikshak 1.0. *2014 IEEE International Conference on Advanced Communication Control and Computing Technologies* (pp. 707-712). IEEE.

Singh, K. (2012). xAccess: A Unified User-Centric Access Control Framework for Web Applications. *2012 IEEE Network Operations and Management Symposium (NOMS): Short Papers* (pp. 530-533). IEEE.

Srinivasan, M. K., Sarukesi, K., Rodrigues, P., M, S. M., & P, R. (2012). State-of-the-art Cloud Computing Security Taxonomies - A classification of security challenges in the present cloud computing environment. Chennai: ACM.

Stefinko, Y., Piskozub, A., & Banakh, R. (2016). Manual and Automated Penetration Testing. Benefits and Drawbacks. Modern Tendency . Lviv-Slavske, Ukraine.

Stuttard, D., & Pinto, M. (2011). *The Web Application Hackers Handbook Second Edition.* Indianpolis: John Wiley & Sons.

Thankachan, A., Ramakrishnan, R., & Kalaiarasi, M. (2014). A Survey and Vital Analysis of Various State of the Art Solutions for Web Application Security. Tamil Nadu: IEEE.

Tso, K. S., Pajevski, M. J., & Brian, J. (2011). Access Control of Web and Java Based Applications. *2011 17th IEEE Pacific Rim International Symposium on Dependable Computing* (pp. 320-325). IEEE.

Verizon. (2017). 2017 Data Breach Investigations Report. Verizon.

Wasserman, G., Yu, D., Dhurjati, D., Inamura, H., & Su, Z. (2008). Dynamic test input generation for web applicaitons. *08 Proceedings of the 2008 international symposium on software testing and analysis* (pp. 249-260). Seattle: ACM.

Weitzner, D. J., Abelson, H., Berners-Lee, T., Feigenbaum, J., Hendler, J., & Sussman, G. J. (2007, June 13). Information Accountability. Cambridge, Massachussetts, United States of America: Massachussetts Institute of Technology.

Wen, S., Xue, Y., Xu, J., Yang, H., Li, X., Song, W., & Si, G. (2016). Toward Exploiting Access Control Vulnerabilities within MongoDB Backend Web Applications . *2016 IEEE 40th Annual Computer Software and Applications Conference* (pp. 143-153). IEEE.

WhiteHat Security. (2017). Application Security Statistics Report. WhiteHat.

Zhao, J., Shang, W., Wan, M., & Zeng, P. (2015). Penetration Testing Automation Assessment Method Based on Rule Tree. *The 5th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems* (pp. 1829-1833). Shenyang: IEEE.