Pentti Laitinen

VULNERABILITIES IN THE WILD: DETECTING VULNERABLE WEB APPLICATIONS AT SCALE

UNIVERSITY OF JYVÄSKYLÄ
DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS
2018

ABSTRACT

Laitinen, Pentti
Vulnerabilities in the wild: Detecting vulnerable web applications at scale
Supervisor: Semenov, Alexander
Jyväskylä: University of Jyväskylä, 2018, 75 p.
Information Systems, Master's Thesis

Web applications are a popular target for malicious attacks. Common web applications can have multiple different security flaws discovered within a timespan of a year. It is important and useful practice to keep these applications up to date to avoid possible exploitation of these flaws, but rarely these systems have great automatic update systems built in, so the maintenance tasks fall to the users. If system is hacked by a malicious party it might not only be used to harm the owner of the system but to also harm other parties. Knowing the current installation base of specific web applications allows reacting to possible problems within the patching practises.

This study aims to construct a method for collecting meta information regarding vulnerable web applications at Internet-wide scale. Web content management system WordPress has been chosen for the testing application of this method as it is one of the most popular open source web application used today. Construction process of this information gathering method followed the six steps of the Design Science Research Methodology. Web content management system (WCMS) security literature has been reviewed within this study, to gain knowledge of vulnerabilities and risks that WCMS applications face. These results are then compared to the vulnerabilities and risks facing other common web applications. Second literature review covers previous reputable studies comparing and discussing vulnerability scanning. The information gained from this second literature review allows us to understand how applicable these methods presented in vulnerability scanning literature are to large scale scanning.

With knowledge gained from these literature reviews a scanning method was created and tested. The testing proved that new kind of extendable open source scanning tools created by The ZMap Project are fast and efficient for internet wide web application information gathering. The Censys project actively uses ZMap to gather research data from internet. This study uses the research data collected by Censys for testing of the constructed method. The data gained from the testing showed that there are still quite many hosts which had over a year old versions of WordPress running. The results allowed exploration of the installation age differences between continents, but these differences were quite small. Web applications which had digital certificate installed had slightly more recent versions of WordPress installed, compared to the sites which had no certificate.

Keywords: vulnerability, web crawling, information security, web applications, vulnerability scanning, web crawling, design science, WCMS

TIIVISTELMÄ

Laitinen, Pentti
Vulnerabilities in the wild: Detecting vulnerable web applications at scale
Ohjaaja: Semenov, Alexander
Jyväskylä: Jyväskylän yliopisto, 2018, 75 s.
Tietojärjestelmätiede, Pro gradu -tutkielma

Web-sovellukset ovat suosittu kohde pahansuoville hyökkäyksille. Yleisissä web-sovelluksista voi löytyä useita haavoittuvuuksia vuoden aikana, joten on tärkeää päivittää sovelluksia aktiivisesti, jos niihin tulee tietoturvapäivityksiä. Harvoin näissä sovelluksissa on kuitenkaan automaattisia päivityksiä, joten järjestelmien päivittäminen on usein käyttäjän harteilla. Jos järjestelmä joutuu hyökkäyksen kohteeksi, sitä ei pelkästään saateta käyttää sivuston omistajaa vastaan, vaan myös aiheuttamaan haittaa sen käyttäjille. Mikäli web-sovellusten päivitystavat olisivat paremmin tiedossa, voitaisiin päivityskäytäntöjä parantaa tämän tiedon pohjalta.

Tutkielman tavoitteena on muodostaa menetelmä internetin laajuiseen web-sovellusten haavoittuvuuteen liittyvän metainformaation tiedonkeruuseen. Metodia tullaan testaamaan WordPress-sovellusta vastaan, joka on yksi suosituimmista avoimen lähdekoodin web-sovelluksista. Menetelmä on artefakti, joka kehitetään noudattaen kuusi askelta käsittävää suunnittelutieteen (Design Science) metodologiaa.

Tutkimuksen yhteydessä tehdään kaksi kirjallisuuskatsausta. Ensimmäinen kirjallisuuskatsaus on toteutettu web-sovelluksia käsittelevän tietoturvakirjallisuuden pohjalta ja se keskittyy yleisemmällä tasolla web-sovelluksiin. Tämän katsauksen avulla pyritään hahmottamaan, millaisia riskejä ja hyökkäyksiä vastaavat sovellukset yleensä kohtaavat. Toinen kirjallisuuskatsaus keskittyy web-sovellusten haavoittuuksien skannaukseen, minkä avulla on mahdollista arvioida paremmin ovatko nykyiset ratkaisut sopivia koko verkon kattavaan tiedonkeruuseen.

Kirjallisuuskatsausten pohjalta tutkimuksessa muodostetaan menetelmä Internetin laajalle web-sovellusten informaation keruulle. Metodin testauksen ja arvioinnin tuloksena voidaan todeta, että modernit laajennettavat ZMap projektin luomat avoimeinlähdekoodin työkalut ovat nopeita ja tehokkaita laaja-alaiseen skannaukseen ja informaation keruuseen. Censys projekti käyttää ZMap-työkalua aktiivisesti datan keruuseen tutkimuksia varten. Tässä tutkimuksessa käytetään Censys projektin keräämää dataa apuna metodin testauksessa. Testeissä saatujen tuloksien perusteella on pääteltävissä, että varsin suurella osalla WordPress-asennuksista oli käytössä yli vuoden vanha versio sovelluksesta. Asennettujen versioiden tuoreudessa oli havaittavissa pieniä viitteitä siitä, että joillain mantereilla sijaitsevat asennukset olivat astetta tuoreempia kuin toisilla. Sillä oliko web-sovelluksen web-sivulle asennettu sertifikaatti, ei näyttänyt olevan juurikaan vaikutusta sovelluksen version tuoreuteen.

Asiasanat: haavoittuvuus, tietoturva, web-sovellukset, haavoittuvuusskannaus, web-indeksointi, suunnittelutiede, web-sisällön hallintajärjestelmä

FIGURES

FIGURE 1. DSRM Process Model 10
FIGURE 2. Framework of Security in WCMS Applications 21
FIGURE 3. Potential WCMS attacks 24
FIGURE 4. WCMS metamodel excerpt 26
FIGURE 5. Secubat Attacking Architecture 30
FIGURE 6. ZMap Architecture 39
FIGURE 7. The proposed method for scanning 44
FIGURE 8. WordPress installation counts 51
FIGURE 9. WordPress installation counts before 2015 52
FIGURE 10. Certificate installation numbers after 2015 53
FIGURE 11. Certificate installation numbers before 2015 54
FIGURE 12. Total installations for each continent 54
FIGURE 13. Linear regression of release age and total number of installations . 57

TABLES

TABLE 1. OWASP Top 10 2013 List 18
TABLE 2. OWASP Top 10 similarities in articles 27
TABLE 3. Recommended Scanning Practises 42
TABLE 4. WordPress versions with most installations 51
TABLE 5. Descriptive statistics 57
TABLE 6. Certificate correlations 58
TABLE 7. Continent correlations 58

TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
FIGURES
TABLES

1 INTRODUCTION 7
1.1 Motivation 8
1.2 Objectives 8
1.3 Research methods 10
1.4 Method implementation 11
1.5 Expected results 12

2 WEB APPLICATION SECURITY 14
2.1 Security in web application context 14
2.2 Software testing 16
2.3 Common web application threats 17

3 LITERATURE OVERVIEW 20
3.1 Web Content Management Systems 20
3.1.1 Security in dynamic web content management systems applications 21
3.1.2 Security in Open Source Web Content Management Systems 23
3.1.3 Towards an Access-Control Metamodel for Web Content Management Systems 25
3.1.4 Conclusions on WCMS security 27
3.2 Vulnerability Scanners 28
3.2.1 SecuBat: A Web Vulnerability Scanner 28
3.2.2 State of the art: Automated black-box web application vulnerability testing 30
3.2.3 Why Johnny Can't Pentest: An Analysis of Black-BoxWeb Vulnerability Scanners 32
3.2.4 Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner 34
3.2.5 Conclusions on vulnerability scanning tools 35

4 CONSTRUCTION OF THE ARTEFACT 36
4.1 Requirements 36
4.2 Methods of conducting internet wide scanning 37
4.2.1 ZMap 38
4.2.2 Application detection 39
4.2.3 Vulnerability databases 40
4.2.4 Ethics 41
4.3 The proposed method 43

5 DEMONSTRATION 46
5.1 Testing method 46
5.2 Choosing database 48
5.3 Information collection 49
5.4 Results 50
5.5 Validation 55

6 CONCLUSION 59

DEFINITIONS 61

REFERENCES 63

APPENDIX 1. First appendix 67

1 INTRODUCTION

Importance of information security has grown during the past decades as more and more of our services and information has moved online. Security is important in all information systems but it is especially important with systems that are connected to the Internet as flaws and vulnerabilities can create serious consequences to both users and operators of the system (Meike, Sametinger and Wiesauer, 2009). We daily use different web applications (See Definitions) which range from regular news sites to banking. Some of these applications are closed source applications, meaning that the source code used to build the application isn't open for public. There are also open source applications which publish their source code. As our technology usage grows the news about new software vulnerabilities or risks have become more common. Sometimes these vulnerabilities are even branded to gain more media coverage, take Heartbleed or Shellshock vulnerabilities as an example (MITRE Corporation, 2013, 2014).

Keeping applications up-to-date is an effective way of mitigating known vulnerabilities and hence avoid possible attacks. Usually software vendors aim to release patches for vulnerabilities as soon as possible, however users don't apply patches immediately after release (Shahzad, Shafiq and Liu, 2012). Vendors react faster to publicly disclosed vulnerabilities and vulnerabilities with high severity (Arora, Krishnan, Telang and Yang, 2010). Open source vendors supply patches noticeably faster than their closed sourced counterparts (Arora et al., 2010). User and application management behaviour of postponing patching increases the life cycle of the vulnerabilities (See Definitions) even past the patch days (Shahzad et al., 2012). In a way securing application environments is shared responsibility between users and vendors, but depending on user base of the applications, the importance of easy or automated patching grows. In the context of web applications this can mean automated patching systems, continuous integrations or active maintenance by the user.

There has been discussion in information security literature on how users are the major risk for security of informations systems. Arce (2003) calls users *The Weakest Link* of information systems and Bulgurcu, Cavusoglu and Benbasat (2010) discuss how previous information systems research has pointed out that employees' can be both considerable information security risk and asset at the

same time. Technology is used and managed by people and there is always chance of human errors which may cause software security issues.

1.1 Motivation

Unpatched software is a risk to both individuals and organizations. Vulnerabilities in even small applications within the network can work as a stepping stone for malicious parties to gain access to the systems and even extend vertically within these private networks.

Popular web applications can have multiple patches released within a time span of a year, which fix serious vulnerabilities within the application. Patching these applications isn't usually automated and hence it requires human interaction with the system. For individuals this can mean that some applications are left unpatched for long time. It is also possible that management of web applications can fall through the cracks also in organizational environments. By being able to gather statistical information regarding the running web application versions would give insight on how regularly people update their hosted web applications and what kind of security and patching differences there are between applications such as WordPress and Drupal which are being used for similar purposes. Knowing version information would also enable information gathering regarding the running vulnerable web application instances.

To the best of my knowledge studies regarding the process of large scale detection of web applications which have known vulnerabilities haven't been published. Known in this context means that there has been a public vulnerability disclosure on specific application version. By constructing a robust way of application information collection and vulnerability cross-referencing at scale would allow us to gain better knowledge about the application management behaviour and how well web applications are kept up-to-date. This data collection method should also be able to collect other metadata related to the content of web application as this would allow more elaborate analysis of the management behaviour. For example detecting whether a site is hosted and owned by a larger organization might mean that management operations such as updating are carried out more regularly than for a site which is owned and managed by a hobbyist.

1.2 Objectives

This thesis aims to construct a method of collecting web application meta information regarding vulnerabilities at scale. This method should be able to fingerprint the web application, gather related variables such as version information and cross-reference it to known vulnerabilities related to that application. Data collection method should also be able to gather or search keywords mentioned on the web application with which the sites can be further categorized. We can summarize this into our main research question:

- How to collect web application vulnerability information at large scale?

One of the most popular types of web applications are Web Content Management Systems (WCMS). According to survey done by W3Tech, about 52.9% of the websites use some kind of web content management systems (WCMS) for websites (W3Techs, 2017b). W3Techs surveys are based on three months average ranking of the Alexa top 10 million websites and the survey doesn't include subdomains or redirects (W3Techs, 2017b). The most popular open source WCMS are WordPress, Drupal and Joomla. Technology survey done by BuiltWith Pty Ltd states that about 37% of Alexa top hundred thousand and 46% of top million websites use WordPress as of April 2017 (BuiltWith Pty Ltd, 2017). Meike et al. (2009) explain that WCMSs allow users even without in depth knowledge to deploy and use these systems. Due to the popularity of these systems they've become an interesting target for malicious attackers, therefore one could say that importance of the security features and overall security of these applications has risen. This is why non-technical users should take extra precautions when using these systems and keep their systems always up-to-date. (Meike et al., 2009)

As this thesis aims to construct base for a method of the web application vulnerability information collection at internet wide scale, it isn't feasible to examine multiple different web applications, but rather construct the method and explain the steps needed for usage of this method. That is why this study focuses on collecting vulnerability information of most popular web applications, WCMS and specifically WordPress which has the largest user base out of different WCMS applications. To validate our choice of inspecting and evaluating the method on WordPress, we first need to know how flaws and weaknesses related to WCMS differ from general web applications. For vulnerability information collection it is also required to know the best practises and limitations of vulnerability scanners. We can condense these into the two sub research questions.

- Which web application flaws and weaknesses especially relate to WCMS?
- What are the best practises for application vulnerability information gathering?

This thesis attempts to answer these sub questions by conducting a systematic literature review on the subjects of WCMS security and Vulnerability Scanning tools. The knowledge gained from the literature review will be used for the vulnerability information collection method. The method will then be used for proof of concept data collection of sites running popular web application WordPress. The collected data will be then evaluated against other data sets such as the one collected by F-Secure corporations web crawler called Riddler.io which also collects version information of web applications. Research methods and structure of this thesis is presented in the following section 1.3.

1.3 Research methods

Design science has been discussed and used in the field of Information Systems Science during past couple decades and multiple different researchers have presented papers on the subject of IS research and Design Science (DS). At its core it is a problem solving paradigm that has its roots in engineering (Hevner, March, Park and Ram, 2004).

A notable Information Systems Science publication on Design Science is by Hevner et al. (2004), paper which presents seven guidelines for effective Design Science research in the field of Information Systems. Since then Design Science has slowly gained popularity in our field and Peffers, Tuunanen, Rothenberger and Chatterjee (2007) presented specific methodology called Design Science Research Methodology (DSRM) which builds upon previous Design Science research in Information Systems Science and demonstrates how DS research should be conducted in Information System Science. Main goals of the Methodology are to increase quality and validity of research done within Design Science. (Peffers et al., 2007)

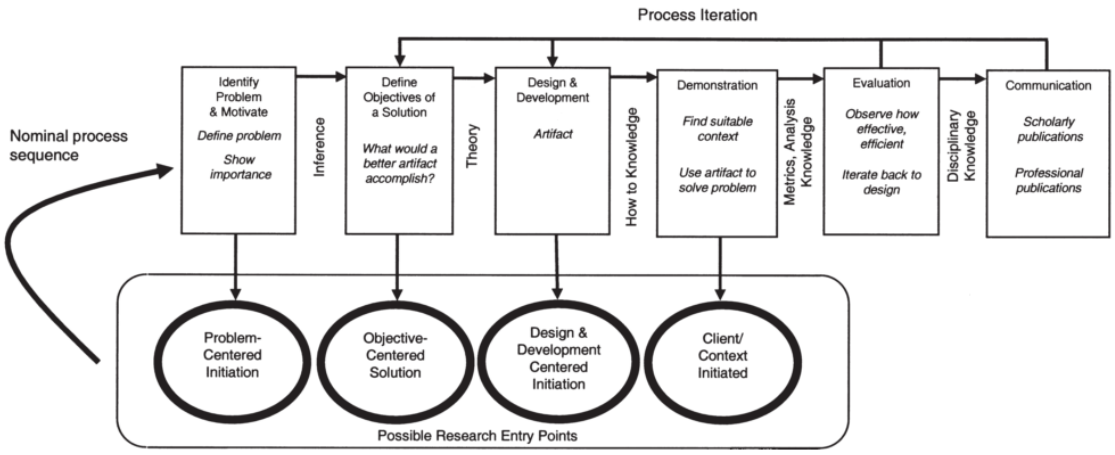

FIGURE 1: DSRM Process Model (Peffers et al., 2007)

This thesis follows the DSRM framework presented in the paper by Peffers et al. (2007). DSRM gives four different entry points for research. They are Problem-Centered Initiation, Objective-Centered Solution, Design and Development Centered Initiation and Client or Context Initiated research. Process model consists of six activities which are presented in sequential order but researcher isn't restricted to follow this structure from the beginning to the end as the point of entry might mean that the process starts from the middle and moves outwards from there (Peffers et al., 2007). Research questions and the motivation behind this thesis are problem centric so the Problem-Centered Initiation in the framework is a suitable entry point for this research into the process model. The following

sections of this chapter discuss each of the activities related to this thesis and present how these activities will be applied in this thesis.

The construction part of this thesis requires knowledge about previous scientific publications related to the subject of vulnerability testing tools or scanners and web applications. As literature review in this thesis serves as one of the main data collection methods for the construction, the eight guidelines for conducting a systematic literature review by Okoli and Schabram (2010) are going to be used for the review. These are *Defining purpose of the literature review*, *Protocol and training*, *Searching for the literature*, *Practical screening*, *Quality appraisal*, *Data extraction*, *Synthesis of studies* and *Writing the review* (Okoli and Schabram, 2010).

The first step of the guideline is similar to the first step of the DSRM where we defined the motivation and purpose of this thesis. Second step according to Okoli and Schabram is essentially meant for reviews where multiple reviewers are employed and therefore it can be excluded for this thesis. In the following third step reviewer needs to explicitly define the details of the literature search and describe the comprehensiveness of the search. During the search process reviewer also needs to conduct *practical screening* which is the fourth guideline. This requires clear presentation of the studies which were considered for the review and which were eliminated without further examination. *Quality appraisal* is the step where reviewer needs to define the criteria for judging which articles are insufficient quality to be included in the review synthesis. After this step, reviewer may extract the data from each of the chosen studies. This is followed with the *synthesis of studies* or analysis where reviewer combines the facts extracted from the studies. Finally, reviewer will write the review. (Okoli and Schabram, 2010)

This thesis consist of two literature review parts as literature related to both WCMS security and vulnerability scanning tools will be discussed. Practical screening explanations and boundaries of quality appraisals will be defined in chapter 3 where the review will be conducted.

1.4 Method implementation

This study follows the sequential order of activities presented in Design Science Research Methodology framework:

1. **Problem identification and motivation:** Introduction chapter lightly introduced the problem and the research questions. However, in following chapter called Web application security and literature review section Web Content Management Systems we further shed light on the foundation of the technological field behind research problem and the questions.

2. **Define the objectives for a solution:** This step consists of studying the problem definition and deducing what is possible and inferring the objectives of the solution. These objectives can either be quantitative or qualitative. For qualitative definitions this can mean for example rationally inferring why the artefact would better address the problem than possible previous research

artefacts created. (Peffers et al., 2007) Constructing these object definitions will be carried out in the chapters following the definition chapters as the further study of previous research and the research area is needed for the base of better solution objective definitions.

3. **Design and development:** Valid artefacts in design science can for example be constructs, models, methods or instantiations (Peffers et al., 2007). For this study the design and development part of the research consists of construction of vulnerability information collection method for web applications. This construction is based on the previous research done of web application security and vulnerability scanning in web content management systems. The method created in this step is the artefact from which proof of concept data collection is done for the fourth demonstration step.

4. **Demonstration:** Concludes the testing of the data collection method at larger scale for WordPress applications.

5. **Evaluation:** Validity and accuracy of the data collection method will be evaluated in this step by comparing the results of data collection to other data sets, such as Riddler.io and statistics of web technology survey companies such as W3Techs and BuiltWith. Quantitative analysis on version detection rate and other application variables will be done in this step.

6. **Communication:** In the last step the problem and its importance will be discussed in section 5.5. This includes discussion about the findings and results of the evaluation as well as utility and novelty of the artefact (Peffers et al., 2007). In conclusions chapter the applicability to other web application will be further discussed and suggestions on how similar studies could be improved in future will be given.

1.5 Expected results

The main artefact from this research will be the method of conducting large scale scanning of web application version information. This method should describe the approach which is most suitable for large scale web application security information gathering and provide steps on how this can be done. Building such approach requires knowledge of web application vulnerability detection, hence following chapters should give insight into how web application scanning has been done and how it can be conducted at a larger scale. The resulting method or artefact can be considered to be a guideline of conducting vulnerability information collection with testing conducted on popular web application WordPress.

The demonstration and testing of the artefact will consist of building a small application that can gather this data and allow us to see the installation bases between different WordPress versions. The hypothesis is that there are noticeable percentage of sites running versions of WordPress which are older than one year.

Noticeable in this sense means anything above 10%. As WordPress had its initial release back in 2003, there are likely still versions running which are over ten year-old. Still there is likely a negative correlation with release age and number of installations, which means that newer versions have larger number of active installations. It is also expected that sites which have some kind of certificate installed are more likely running newer versions compared to the sites which have no certificate installed. Data collection should also be able to collect the IP addresses of the sites and these addresses can be then geolocated. This data most likely shows little difference between different continents regarding installed versions. List bellow shows these five hypothesis condensed into short statements.

1. Over 10% of the installations are running versions which have been released over a year before the scan.

2. There are still some installations running early versions.

3. Older the version is, less it has active installations.

4. Sites having certificates are more likely running new versions.

5. Data shows little difference between continents.

2 WEB APPLICATION SECURITY

World Wide Web Consortium (W3C) the international standards organisation for World Wide Web doesn't clearly specify the definition for Web Application. W3C itself states that the subject referred as Web Applications hasn't been clearly addressed in HTML documentation (W3C, 2014). There seems to be no consistent definition for this term. Stuttard and Pinto (2011) define web applications as those applications that we access and communicate with by using a web browser. Generally a web application consist of server-side and client-side programs. Client side code is sent to the client from the server and the client then executes this code in the browser. Usually for web applications this means logic that server side doesn't need to observe and validate and which is more efficient to be handled on the client side such as changes in user interface. Server side of the application is the part of the program that handles for example the validation, authorization, data access and controller logic.

Public web applications are usually accessible from anywhere in the word but there are a lot of corporate web applications which have restricted the network area where they are able to be accessed. We use different web applications daily that handle very sensitive information, all ranging from banking, electronic prescriptions to accessing backups of our photos. Even applications which don't handle sensitive or valuable data are interesting targets for attackers as they can be used for other malicious gains such as serving malicious content.

Meike et al. (2009) classifies Web content management system as web applications. This chapter discusses the security of web applications and presents the terms and concepts that are relevant to this research. In the latter part of this chapter common risk and vulnerabilities related to web applications and software in general are also presented.

2.1 Security in web application context

Writing computer programs can be a complex task and the modern software development flow usually incorporates using many libraries together. According to

Shirey (2007) a flaw is *"An error in the design, implementation, or operation of an information system. A flaw may result in a vulnerability"*. Same Internet Security Glossary defines vulnerability as *"A flaw or weakness in a systems design, implementation, or operation and management that could be exploited to violate the systems security policy"*. (Shirey, 2007). Therefore, it is always a possibility that program has one or more flaws and these flaws may sometimes inflict a vulnerability in the program but a flaw doesn't necessarily mean that there is a vulnerability.

Design errors are a risk especially when security of the program hasn't been taken into consideration from the beginning of the design process. The architecture and design of the system needs to be coherent and take the security principles into account. Assumptions should be documented during the design process and there should also be clear risk analysis at both specifications- and class-hierarchy design stages. (Mcgraw, 2004). Possible design errors can range from varying error handling or not taking into account possible infrastructure weaknesses in the architecture.

Implementation errors or bugs may result for example into buffer overflows, race conditions or authentication system faults (See Definitions). Code reviews, unit testing and static analysis tools may be useful for identifying these implementation errors. Implementation errors can be as harmful as design errors, but faulty design is usually much harder to correct. (Mcgraw, 2004) Open source advocate Eric Raymond for example believes that simpler implementations and algorithms result into fewer faults and better working software. For example the KISS principle (Keep It Simple, Stupid!) has been advocated in his book *The Art of Unix Programming*. (Raymond, 2003)

Persons responsible of operation and management should actively monitor production systems for attacks. Information about these attacks and break in attempts should be collected and passed forward to development teams so that the systems security can further be improved and threat models could be updated. (Mcgraw, 2004) Developers and operations should also monitor third party libraries that are included in the system, for possible vulnerabilities and updates.

Definition of attack by Shirey (2007) is *"An intentional act by which an entity attempts to evade security services and violate the security policy of a system. That is, an actual assault on system security that derives from an intelligent threat"*. So an attack is a purposeful act where attacker, person who is attacking, tries to gain access or make the system execute something that he/she shouldn't be able to do.

Web applications can be hosted within the internal network of a company and outside access can be blocked by a firewall. Usually web applications face the public internet and thereby they are access-able in theory by anyone. Even in the cases when web applications are hosted in internal networks they might face attacks from outside by for example proxying the attack. Access restrictions like firewalls provide therefore mitigations, but if an application works in some kind of networked environment, the security should have high priority throughout the application life cycle.

The field of application security and development is constantly evolving and moving forward. For example the field of virtualization has changed a lot in a couple years with different operating system level virtualization implementations

like Docker. These methods also offer possibility to restrict access to a host operating system and make elevating privileges to other systems harder for attackers in theory.

Application security also extends in a way to people managing and using these systems. Attackers might try to gain access with different social engineering attacks such as phishing. Social engineering attacks are either non-technical or use low-technology methods to gain attack information by tricks or fraud. Phishing is a term for technique of trying to acquire sensitive data from users through a fraudulent solicitation in email or on a website (Shirey, 2007). In a phishing attack the malicious party masquerades the website or email as a legitimate business or other reputable source (Shirey, 2007).

Penetration testing (See Definitions) can be a useful way of evaluating security and vulnerabilities in a system. Shirey (2007) defines penetration test as a system test, which is often a part of system certification in which evaluator attempt to circumvent the security features of a system. Black-box testers can be a useful assistance tool for penetration- and other testers. Analysing program by running it with various inputs and without use or knowledge of the source code is called black-box testing. White-box analysis, on the other-hand, means analysing the source code and understanding the design of the program. (Potter and McGraw, 2004)

2.2 Software testing

Amman and Offutt define software testing as the process of evaluating software by observing its execution. Testing software consist of designing tests, then executing them and evaluating the output of the tests. These tests can be derived from specifications, design artefacts, requirements or from the source code of the program. Amman and Offutt (2008) describe five different software testing levels or testing activities. *Acceptance testing* is the activity where software is assessed with respect to the requirements. *System testing* assesses it against architectural design and *Integration testing* with respects to subsystem design. Assessments against detailed design are called *Module testing* and *Unit testing*. *Module testing* consists of assessing isolated individual modules where as *unit testing* tests parts of the source code and can be considered the lowest level of testing. (Amman and Offutt, 2008)

One limitation of software testing is that it only shows the presence of failures (See Definitions) but not actually the absence of them (Amman and Offutt, 2008). Validation and verification are common terms used in conjunction with software testing. Verification is usually the activity that requires more technical knowledge about the individual software artefacts, requirements and specifications whereas validation has more domain specific knowledge requirements (Amman and Offutt, 2008). Amman and Offutt define Validation as *The process of evaluating software at the end of software development to ensure compliance with intended usage*. Verification process is more related to determining whether software fulfils the requirements es-

tablished for it in a previous phase of development (See also Definitions). (Amman and Offutt, 2008)

Security testing focuses on testing software for undesirable and malicious behaviour (Amman and Offutt, 2008). Arkin, Stender and McGraw (2005) explain that security testing is in a way testing software for negatives when normal feature testing tests that functions properly perform specific tasks. This kind of testing for negative effects is a hard task as merely enumerating possible malicious actions only uncover that those specific faults (See Definitions) are not in the software under specified test conditions. Because new flaws (See Definitions) can always be discovered, these tests do not actually prove that tested systems are immune to all possible attacks. (Arkin et al., 2005)

Penetration testing is a form of security testing. Penetration testers use commonly static and dynamic analysis tools and fuzzers (Arkin et al., 2005). Black-box testing tools are also used for penetration testing Bau, Bursztein, Gupta and Mitchell (2010). Amman and Offutt (2008) defines black-box testing as tests which are derived from external descriptions of the software. For black-box testing tools this generally means that the tools have no internal knowledge of the system prior to scan. Commonly organizations have used penetration testing in the latter part of projects as a final acceptance regimen (Arkin et al., 2005). However, organisations such as HackerOne (hackerone.com) and Bugcrowd (bugcrowd.com) have made penetration testing more accessible for organizations of all sizes with public bug and security bounties.

Black-box testing tools such as the Burp suite are common for penetration testing where one wants to gather more information regarding the target web application by fuzzing (See Definitions) the applications parameters. This information can then help vulnerability discovery process. (Seitz, 2015) Term vulnerability scanner is sometimes used for these tools, but static analysis tools or white-box scanners can also be considered to be vulnerability scanners, but with access to source code.

2.3 Common web application threats

The Open Web application Security Project (OWASP) is a not-for-profit organization focused on improving software security and increasing security awareness. OWASP Top 10 is well-known project which aims to improve web application security by identifying ten most critical risks facing them. Most recent reversion of the project is OWASP Top 10 2013 (Table 1) and it is widely referenced. (OWASP Foundation, 2013b) Project is a great resource for anyone interested in web application security and the common threats and pitfalls.

The Top 10 list functions as a reference for application security and should not be used only by developers. Listing has a severity rating. The first entry is more serious than the second one, that is more severe than the third one, and so on. This however doesn't mean that A10 isn't critical or serious, as there are also other risks that are left outside the list. The list should be used as a guideline for managing web application security risks.

A1	Injection
A2	Broken Authentication and Session Management
A3	Cross-Site Scripting (XSS)
A4	Insecure Direct Object References
A5	Security Misconfiguration
A6	Sensitive Data Exposure
A7	Missing Function Level Access Control
A8	Cross-Site Request Forgery (CSRF)
A9	Using Components with Known Vulnerabilities
A10	Unvalidated Redirects and Forwards

TABLE 1: OWASP Top 10 2013 List (OWASP Foundation, 2013b)

Injection flaws are usually related to database such as SQL, NoSQL or Lightweight Directory Access Protocol. Injection flaws can also affect XML parsers and program arguments. Injection flaws can be easier to find out by examining code than testing. Fuzzing or scanning the application can help discover these faults. (OWASP Foundation, 2013b)

Broken Authentication and Session Management is the second on the OWASP Top 10 list. Building custom authentication and session management schemes securely and correctly is hard and this is why these parts of applications have frequently flaws in them and finding these flaws can be hard due to the unique implementation. (OWASP Foundation, 2013b)

The third item on the list is XSS or Cross Site Scripting. It is probably the most common fault that affects web applications. Cross site scripting usually happens when application doesn't correctly validate and escape the user input and allows injecting scripts to within its content. XSS flaws can be categorized into two different types, reflective- and stored attacks. Reflective attacks are attacks where the injected script is reflected off the server via error message or some other response. Stored attacks inject scrip permanently to applications database or some other permanent store location from which victim then retrieves the malicious script. (OWASP Foundation, 2013b)

Insecure direct object references is the fourth item on the list. Direct object references may compromise all the data that is referenced by the object, that is why direct name or key references should be avoided for example when web pages are being generated. Insecure direct object references are common although static code analysis and testing are usually able to pinpoint these flaws easily but exploiting these flaws is also fairly easy. (OWASP Foundation, 2013b)

The fifth point on the list is Security Misconfiguration which is also a common flaw related to web applications. Misconfiguration may happen on any layer of the application stack. This means that the configuration flaws may be present at a platform, web server, application server, database, framework and in any custom code related to the application. Communication between developers and system administrators plays key part in avoiding and fixing these problems according to OWASP Foundation (2013b). Automated scanners are also useful for detecting problems such as outdated systems, misconfiguration and use of default accounts. (OWASP Foundation, 2013b)

Sensitive data exposure is the sixth item on the list and the most common cause for this flaw is not encrypting sensitive data or using weak key generation and management for encryption algorithm. According to OWASP Foundation (2013b) weak algorithms are unfortunately common for password hashing but exploiting these flaws is hard since external attackers usually have limited access. Severity of these attacks is however high as this data may contain valuable information such as credit card- and personal data. (OWASP Foundation, 2013b)

These attacks seem to have gained popularity during the year 2016 as multiple huge breaches were disclosed such as the huge Yahoo data breach of approximately one billion accounts (Thielman, 2016) and the breach of Homeland Security of United States (Lichtblau, 2016). There seems to have been a quite noticeable trend of these kinds of attacks becoming more common especially when looking at the Data Breach report of Identity Theft Resource Center where year 2016 was the all-time high number of data breaches (Identity Theft Resource Center, 2017).

Missing Function Level Access Control the next risk on the OWASP list. Missing function level access control manifests itself either as result of system misconfiguration when function protection is managed with configuration or as forgotten access right checks in applications code. Again this flaws like this have moderate impact as they may allow unauthorized access to functionality and the exploitation can be fairly trivial. (OWASP Foundation, 2013b)

The eight risk on the list is the Cross site Request Forgery (CSRF) that is common vulnerability within web applications. Exploitation of this type of a flaw can leverage the fact that web applications allow attackers to predict all the details of particular action in the application. Applications often use session cookies for authentication which allows attackers then use forged malicious cookies for authentication in cases where the token is predictable. (OWASP Foundation, 2013b)

The ninth item is using components with known vulnerabilities. This is a very widespread problem as almost all applications have dependencies like common libraries to aid the development process. Detection of these problems is hard according to the OWASP as many development teams don't focus on keeping all the components and libraries used in the application up-to-date. Often it is even hard to know all the components which are being used in the application. (OWASP Foundation, 2013b) This is especially true when these libraries may themselves have multiple different dependencies and noticing use of possible vulnerable libraries within these may be very hard for development teams to keep track on.

The last item on the OWASP Top 10 list is Unvalidated Redirects and Forwards. This risk manifests itself as a possibility of manipulating redirects with the help of a parameter that isn't being validated within the application. This allows an attacker to choose the destination page of the forward or redirect action. As an example an attacker may use this to evade access control or redirect victims to disclose passwords or other sensitive information. (OWASP Foundation, 2013b)

3 LITERATURE OVERVIEW

This chapter consists of two separate literature reviews which have been separated in their own sections. The first section covers the literature related to web content management systems and how these systems have been studied previously. Second section discusses the literature which presents vulnerability scanners or comparisons between different vulnerability scanners.

3.1 Web Content Management Systems

A Web Content Management System (WCMS) is a system that supports creating and publishing content in structured web formats. These systems usually include possibility of approving, reviewing and archiving of the content. Most often they are used for building corporate websites, online shops or community portals. Meike et al. (2009) According to April 2017 survey by W3Techs (2017b), about 52.9% of the websites use some kind of WCMS. Most popular WCMSs according to both W3Techs (2017b) and BuiltWith Pty Ltd (2017) are Wordpress, Joomla and Drupal. All three of these systems are open sourced and thereby free of use. This partly explains popularity of WCMSs as open source Web Content Management System is a low cost alternative to corporate software. They allow small business and organizations to create websites, blogs and web-stores for relatively low cost. WCMS are designed to be easy to use and allow users with little development knowledge to create customized web sites with broad functionality (Meike et al., 2009). This however rises a question on how well administrative tasks such as updating of the system are taken care of.

In this section we review WCMS security research and articles. Each paper chosen for review has been covered in its own section. Google Scholar and IEEE Xplore and ScienceDirect were used for searching articles. Keywords *Web content management system*, *WCMS*, *Security*, *Vulnerability Testing* and *Black box testing* were used. Articles were chosen based on the citation count and reputation of the journal or publisher. Extra value was given for articles which appeared in journals of information science, computer science and newer articles were preferred.

With these keywords the search engine results showed hundreds of results. However large part of these results were not actually related to web content management systems security or vulnerabilities relating to WCMS. Further restricting search criteria presented us with nine articles that relate to WCMS security. Tree articles were chosen out of these nine and selected to be presented here based on the number of citations the papers had and where they were published. Following sections present these three articles.

3.1.1 Security in dynamic web content management systems applications

Vaidyanathan and Mautone in their journal article published in Communications of the ACM discuss security in WCMS from organisation's point of view. They note that some organisations are adopting information technology like WCMS without understanding the security concerns which relate to it (Vaidyanathan and Mautone, 2009).

Writers state that there are five attributes of information security when talking about WCMS. *Confidentiality* attribute means that information isn't accessed by unauthorized user. Second attribute *integrity* is ensuring that applications and data cannot be modified by unauthorized users. *Authentication* is ensuring that content origin is identified correctly and that identities are not falsified. *Availability* for WCMS means that systems are available when authorized users need them to be and last attribute *non-reputation* is security assurance of sender and receiver not being able to deny transmission of content. (Vaidyanathan and Mautone, 2009) With the help of these attributes they formulate "*Framework of Security in WCMS Application*" (figure 2) which integrates eight functional dimensions of WCMS with the five goals of security (Vaidyanathan and Mautone, 2009).

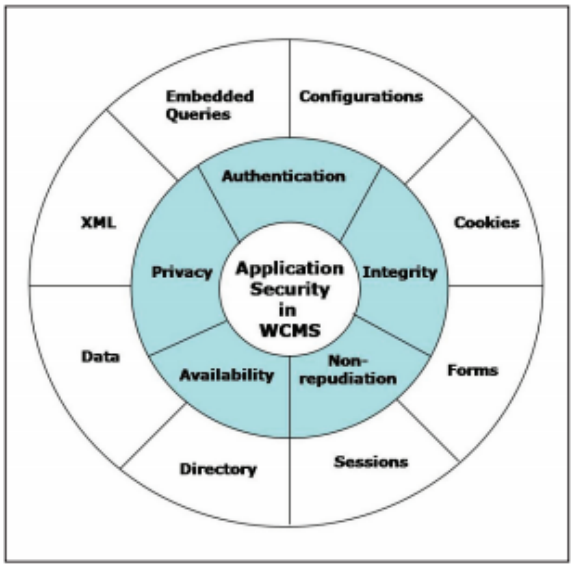

FIGURE 2: Framework of Security in WCMS Applications (Vaidyanathan and Mautone, 2009)

Vaidyanathan and Mautone (2009) point out that if security isn't one of the main goals when building a web system, it will more likely require more patches and updates to continually fix newly found flaws. They state that many vulnerabilities of WCMS are introduced at the application level. These include interruption, interception, fabrication and modification. Interruption means denial off access to assets of the system via deletion of them or making them unusable or unavailable. If unauthorized party gains access to an asset, it has been intercepted. Modification is the attack on the integrity of the data when attacker gains access to it. Attacker may also fabricate or counterfeit objects on the network. (Vaidyanathan and Mautone, 2009)

Configurations is one of the eight functional dimensions that Vaidyanathan and Mautone present and it means that WCMS should be configured properly on the server to ensure top-level security (Vaidyanathan and Mautone, 2009). Second dimension is *cookies* if handled improperly they may allow malicious user to hijack web sessions. *Forms* when poorly designed in a web application, may contain hidden fields that contain private information of users, accounts and sessions. Improper implementation may also allow attacker to execute code and this is why validation of form inputs and doing referrer checks on the server side should be done. According to the writers *Embedded Queries* in the framework means that malicious user may input additional field in hijacked forms to receive confidential information. To combat this Vaidyanathan and Mautone suggest careful examination of database queries for wild characters, validating inputs and taking care that proper permissions exist on the database objects accessed by web application. *Sessions* are often target of an attack and properly ensuring consistent and appropriate session timeouts for the application can be used as a security measure. *Directory* on the framework is the risk of having important application files accessible by malicious users. This should be handled by disabling directory browsing, having good file management process and removing unwanted files from document root entirely. Lastly *XML* or XML communication on the framework means not properly hiding or handling XML file transfer authentications. Encryption of XML files, enforcing security policies authorizing access and by generating log of these activities is proposed for tracking potential hackers wanting to use XML in malicious way. (Vaidyanathan and Mautone, 2009)

Vaidyanathan and Mautone used the framework to evaluate security of two different WCMS tools, Mambo and vBulletin. At the time of the research Mambo was leading WCMS (Vaidyanathan and Mautone, 2009). Based on the evaluation both systems had most of the security features in place, but some features needed to be placed with third party software. Writers also noted that there is no automatic update capabilities in these systems. Five goals of security were still mostly covered by both applications. (Vaidyanathan and Mautone, 2009) Writers concluded that constructed framework and evaluation can be insightful for academicians, information technology managers and practitioners of electronic business.

3.1.2 Security in Open Source Web Content Management Systems

Meike et al. take general approach of covering WCMS security in their article published in IEEE Security & Privacy Magazine. Paper defines that WCMS are commonly used for web sites, online shops and community portals (Meike et al., 2009). The paper also gives definitions to programming and computer security terms.

The authors maintain that especially open source WCMS are lucrative targets for attackers due to their popularity and the openly available source code that makes seeking out and locating possible flaws in the applications easier (Meike et al., 2009). With the help of defining key web application threats: *Data manipulation*, *Accessing confidential data*, *Phishing*, *Code execution* and *Spam* the authors conclude that "*Web applications in general and WCMSs in particular, operate in hostile environment*" (Meike et al., 2009).

The authors also point out that attackers often use various attack patterns which are blueprints for creating attack of specific type. There are many phases for each attack, and they consist of discovery and exploitation itself. Motivation for attack ranges from gaining confidential user data such as credit card information from e-commerce sites hosted with WCMS, gathering user information such as addresses, to damaging company reputation by altering content of the website in damaging way (Defacement). Meike et al. also list four qualities, which web application needs to have to be considered secure. *Authentication* is needed to ensure that the entities or people are who they pretend to be. *Confidentiality* according to authors, means hiding information from unauthorized people. *Integrity* is prevention of unauthorized people the right of modifying, withholding and deleting information. *Availability* means performing the operations according to their purpose over time. (Meike et al., 2009)

Meike et al. (2009) conducted analysis on two open sourced WCMSs Drupal (version 5.2) and Joomla (Version 1.0.13) that is derivative of Mambo. Goal was to get sense of systems security status and if users can trust this security without further ado. (Meike et al., 2009) The security analysis was conducted in multiple steps. The first step was to evaluate how different configuration settings influence security issues. Secondly they performed simple penetration test sending various malicious inputs using different penetration testing tools like OWASP WebScarab and TamperData. In the third step the source code was inspected and reviews for security issues produced. In the following fourth step this gained information of the source code was used to issue more focused malicious requests to systems. As the final step the writers evaluated community support for security issues by browsing project websites and forums. (Meike et al., 2009)

As a result to the analysis communities around both systems paid adequate attention to security aspects, systematically tracking vulnerabilities and providing patches with security fixes. Meike et al. explains that installation process should be as automated as possible for WCMSs as many users are non experts. This is also why default configuration settings should be secure. Both systems had plenty of room for improvement in installation process according to analysis. Both Joomla and Drupal were prepared for parameter manipulation, but they also had

deficiencies and neither of them sufficiently filtered HTTP headers and Web form data. Systems in analysis also were adequate in preventing cross-site scripting and both also performed checks to prevent SQL injection from happening. In authentication management Joomla and Drupal had weakness related to password security and unauthorized access to functions, however both system had proper security in login mechanisms and session data were handled correctly. All aspects of spam prevention had been covered in Joomla but Drupal had yet to make full effort in this field and malicious files weren't detected by either of the systems during upload process. Both systems had also weaknesses in privilege elevation. Drupal and Joomla have systems in place that warn user when trying to add questionable modules (plugins) to the system. (Meike et al., 2009)

From the results of the analysis Meike et al. form summary of possible attacks (figure 3) that WCMS might face. Application might encounter database level attacks such as SQL Injection or web server level attacks that attempt parameter manipulation, malicious file upload, authentication bypass, elevation of privilege, spam relay or session hijacking. User can also become a victim of XSS attacks or spam. (Meike et al., 2009)

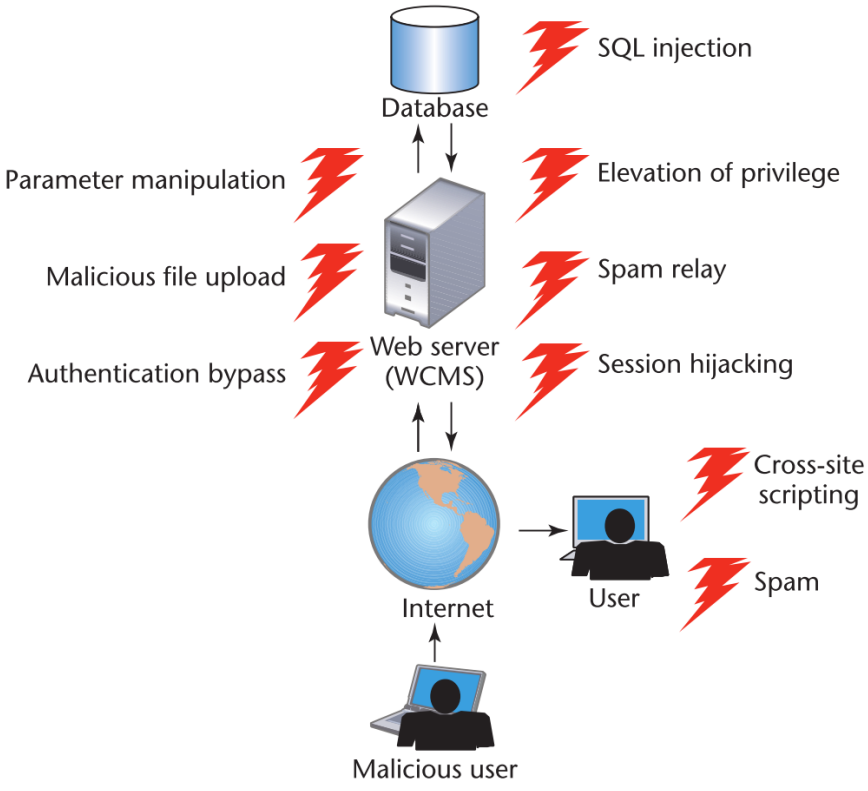

FIGURE 3: Potential WCMS attacks (Meike et al., 2009)

Meike et al. (2009) conclude that given expert knowledge eliminating vulnerabilities in both systems is possible. However they note that these systems are targeted to a non-expert audience and both systems had vulnerabilities that attackers can easily exploit. To minimize threats users should take precautions.

Especially non-technical users should always use the latest available version and technically skilled users can stick to older versions if they are aware of versions security status and regularly follow vulnerability and countermeasure updates provided by the communities. (Meike et al., 2009)

3.1.3 Towards an Access-Control Metamodel for Web Content Management Systems

Martínez, Garcia-Alfaro, Cuppens, Cuppens-Boulahia and Cabot (2013) take more focused approach in their paper by studying access-control (AC) in WCMS. Writers describe Access-control as *"a mechanism aiming at the enforcement of the confidentiality and integrity security requirements"* (Martínez et al., 2013). This means that access-control defines the subjects, objects and actions that system has and makes describing the assignments of permissions to subjects possible. These permissions then assert which actions the subject is allowed to preform on the objects. (Martínez et al., 2013)

Authors explain that integrated security mechanisms play an important role in WCMS as they usually manage sensitive information and users of WCMS's often lack an in-depth technical and security expertise. Martínez et al. therefore create a meta-model for WCMS access-control which allows AC implementation to be represented and analysed vendor-independently. This proposed access-control meta-model (figure 4) allows analysis of the access-control information disregarding the specificities of the concrete WCMS security features and implementation. (Martínez et al., 2013)

The meta-model is inspired by RBAC (Role-based access control) and contains all the basic concepts from it. Model consist of four elements, which are *Contents*, *Actions*, *Permissions* and *Subjects*. *Content* in the model is data that WCMS manages. Each of the *Content* elements has a *ContentType* which identifies the type. This presentation style allows both fine- and coarse-grained access-control. Users in the meta-model are provided with predefined content such as *Node*, which represents the principal contents of the WCMSs. There is also *Page* which means the full content pages of WCMS and *Post* that represents individual blog posts. *CustomNode* meta-class is also included in the model so that additional node types can be integrated into model. Representing the comments that can be added to other content elements is the *Comments* class. Martínez et al. have decided not to include the back-end administrative pages in the model as their behaviour is represented with the administrative operation execution permissions in the model. (Martínez et al., 2013)

All the actions that can be performed over the WCMS are called *Operations*. In the model these operations are divided into two types, *content operations* and *administration operations*. In addition, there is a third operation type called *custom operation* so that model can be extended (figure 4). *Content* operations are able to use all CRUD actions, that means creating, reading, updating and removing actions. There is also search operation available in the model since it is common for WCMS applications to have. In addition, there are *Publish* and *Unpublish* actions which Subjects of WCMSs can perform. (Martínez et al., 2013)

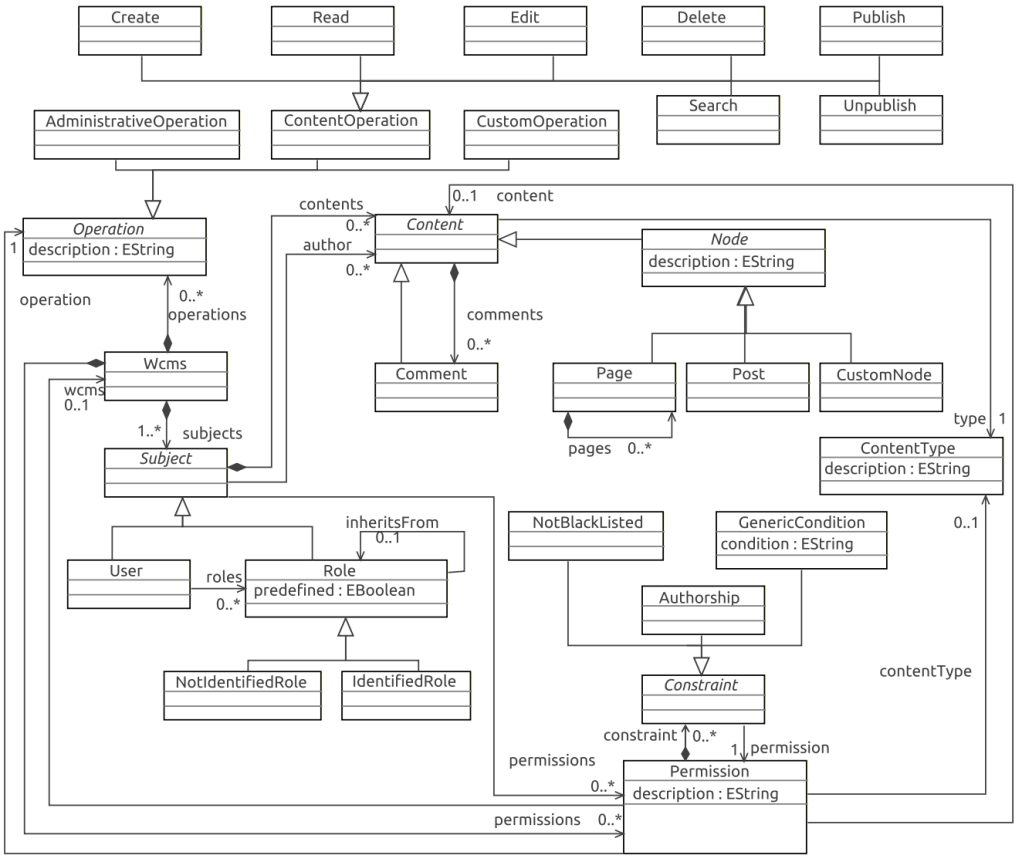

FIGURE 4: WCMS metamodel excerpt (Martínez et al., 2013)

The third element is called *Permissions*, and they represent the right of performing actions at the WCMS. These permissions can define constraints obeying which actions can be executed. In the meta-model there are two constraints identified *Authorship* and *NotBlacklisting*. *Authorship* permissions are only effective if *Subject* is the author of the *Content*. *NotBlacklisting* restricts the applicability of the permission to the condition of not being blacklisted. In addition, there is also *GenericCondition* to allow extending of the model. (Martínez et al., 2013)

Last element presented by Martínez et al. is *Subjects*. These are the elements which interact with the contents of the WCMS by performing actions. Meta-model represents subjects in *Users* and *Roles* in which users get their roles assigned. Depending on the specific WCMS the *Roles* might be predefined by the developer of the system. If predefinitions have been done, they are represented in the model with *predefined* attribute. Martínez et al. identified two common roles *IdentifiedRole* and *NotIdentifiedRole* which are commonly present in WCMS applications and determinate whether user has logged in or not. The meta-model also supports role inheritance as seen in the Figure 4. (Martínez et al., 2013)

With the help of the constructed meta-model Martínez et al. examined open source WCMS Drupal by mapping the information from Drupal system to the meta-model. As a result there exists three main content types *Pages, Articles* and *Comments*. In Drupal there are three roles by default *Anonymous*, *Authenticated*

and *Administrator*. Meta-model was able to represent access-control information of Drupal. (Martínez et al., 2013)

The paper suggest three different applications for the constructed meta-model. The first is *Visualization* as the model makes it easier to analyse and visualize the access-control in WCMSs. The second application is *Queries*, where model eases the security queries when one wants to learn in more detail about specific details of the security policies in the evaluated system, especially in WCMS the information is scattered among number of databases. The last application *Migration*, ie. when a user wants to migrate the content from one WCMS to another. The model eases the migration process of the old access-control information to the other WCMS application As a conclusion the authors state that they successfully presented a meta-model for WCMSs access-control which is vendor-independent and that they currently have ability to extract this model data from Drupal installations. (Martínez et al., 2013)

3.1.4 Conclusions on WCMS security

Web content management applications have wide range of features and use cases as presented by the articles in this section. Meike et al. covered both corporate and individual users of WCMS and Vaidyanathan and Mautone's paper was targeted more towards helping organisations rank and evaluate security of competing systems. WCMS have also multiple different use cases all ranging from web-stores to simple websites, blogs and forums.

The framework for WCMS security assessment by Vaidyanathan and Mautone overlaps on many points with the OWASP Foundation Top 10 list as seen in Table 2 and only risks similar or related to *Unvalidated Redirects and Forwards (A10)* were not covered by their framework. Similarly, paper by Meike et al. covers or mentions almost all the top 10 risks in their paper. Martínez et al. specifically discussed access control and although their paper notices multiple similar risks related to WCMS, they cover mostly access control related risks such as authentication and session management.

	Vaidyanathan andMautone, 2009	**Meike et al.,2009**	**Martínez et al., 2013**
A1	X	X	X
A2	X	X	X
A3	X	X	
A4	X	X	
A5	X	X	X
A6	X	X	X
A7	X	X	X
A8	X		
A9	X	X	
A10			

TABLE 2: OWASP Top 10 similarities in articles

Risks that surround WCMS seem to be very similar to the ones that general web applications face. Importance of keeping these systems up to date and configured properly were mentioned by both Meike et al. (2009) and Vaidyanathan and Mautone (2009). As about 52.9% of websites use some kind of WCMS, there exists huge number of sites that need to have configured and managed their sites in security aware way (W3Techs, 2017b).

3.2 Vulnerability Scanners

In this section we take a look on the previous research done on the subject of vulnerability scanners and security scanning tools. Related papers were searched with Google Scholar, IEE Xplore and ScienceDirect by using following keywords *Vulnerability*, *Scanner*, *Black-box*, *Security* and *Testing*. Also the citations within the papers were checked for related articles. From this group we chose the papers which were from reputable sources and had great amount of quality citations. Newer articles were weighted higher on the scale. These papers were chosen for the following literature review.

3.2.1 SecuBat: A Web Vulnerability Scanner

Kals, Kirda, Kruegel and Jovanovic discuss construction and evaluation of a new open source black-box testing tool called SecuBat, which they created. Authors assume that many web developers are not security-aware and that many web sites are vulnerable. In their paper Kals et al. aim to expose how simple exploiting and attacking application-level vulnerabilities automatically is for attackers. (Kals et al., 2006)

Kals et al. discuss how most web application vulnerabilities result from input validation problems such as SQL injection and Cross-Site Scripting. Two main approaches exist for the bug and vulnerability testing software. One is *white-box* testing, in which the testing software has access to source code of the application and this source code is then analysed to track down defections and vulnerabilities in the code. Authors state that these operations are usually integrated into development process with the help of add-on tools in the development environments. Other approach is called *black-box* testing, where the tool has no access to source code directly, but instead tries to find vulnerabilities and bugs with special input test cases which are generated and then sent to the application. Responses are then analysed for unexpected behaviours that indicate errors or vulnerabilities. (Kals et al., 2006)

SecuBat is a black-box testing tool as it crawls and scans websites for the presence of exploitable SQL injection and cross-site scripting (XSS) vulnerabilities (Kals et al., 2006). The scanning component in SecuBat utilizes multiple threads to improve crawling efficiency as remote web servers have relatively slow response time. The attack component in SecuBat initiates after crawling phase is completed and list of targets has been populated (figure 5). The scanning component is

especially interested in presence of web forms at the web sites as they constitute our entry points to web applications. These web forms are then observed by the tool as it chooses type of attack which will be sent to the form. (Kals et al., 2006)

At the time when the paper was written the white-box testing hadn't experienced widespread use for finding security flaws in applications. Authors explain that the important reason for this has been limited detection capability of white-box testing tools. (Kals et al., 2006)

Kals et al. explain that then popular black-box vulnerability scanners such as Nikto and Nessus use large repositories of known software flaws for detection. Authors argue that these tools lack ability to identify previously unknown instances of vulnerabilities due to relying mainly on these repositories. SecuBat, the vulnerability scanner created by Kals et al. does not rely on known bug database but scans for general classes of vulnerabilities (SQL Injection, XSS and CSRF). Secubat attempts to generate proof-of-concept exploits in certain cases to increase the confidence of detections. (Kals et al., 2006)

SecuBat consists of *crawling component*, *attack component* and *analysis component*. The crawling component crawls the target site using queued workflow system to combat slow response times of web servers. This allows 10 to 30 concurrent worker threads to be deployed for a vulnerability detection run. The crawling component is given root target address (URL) from which SecuBat steps down the link tree. Authors note that the crawling component has been heavily influenced by crawling tools such as Ken Moody's and Marco Palomino's *SharpSpider* and David Cruwys' *spider*. (Kals et al., 2006)

The crawling phase is followed by the attacking phase, in which SecuBat processes the list of target pages. The component scans each crawled page for presence of web forms and fields as they are the common entry points to web applications. Action address and the method used to submit the content are then extracted from these forms. Depending on the attack being launched the appropriate form fields are chosen and then the content will be uploaded to server. The possible response from the server is then analysed by the analysis module that parses and interprets the response. Module uses attack-specific response criteria and keywords for confidence value calculations to decide whether the attack was successful.

Kals et al. implemented the components in SecuBat in the architecture seen in Figure 5. The architecture supports adding possible new analysis and attacking plugins into application. Secubat was implemented as Windows Forms .NET application that uses SQL server for saving and logging the crawling data. This also allows generation of reports from the crawling and attack runs. SecuBat uses a dedicated crawling queue for crawling tasks. The crawling tasks consist of web pages that are to be analysed for potential targets. Attacks are implemented with Attack queue that is handled with queue controller that periodically checks the queue for new tasks. These tasks are then passed to thread controller that selects free worker threads. Worker threads execute the analysis task and notify the workflow controller when the task has been completed. (Kals et al., 2006)

Researchers evaluated the effectiveness of the vulnerability scanner SecuBat by doing combined crawling and attack run. The crawling was started by using a Google search response page for word *login* as a seed page for the crawler. Total 25

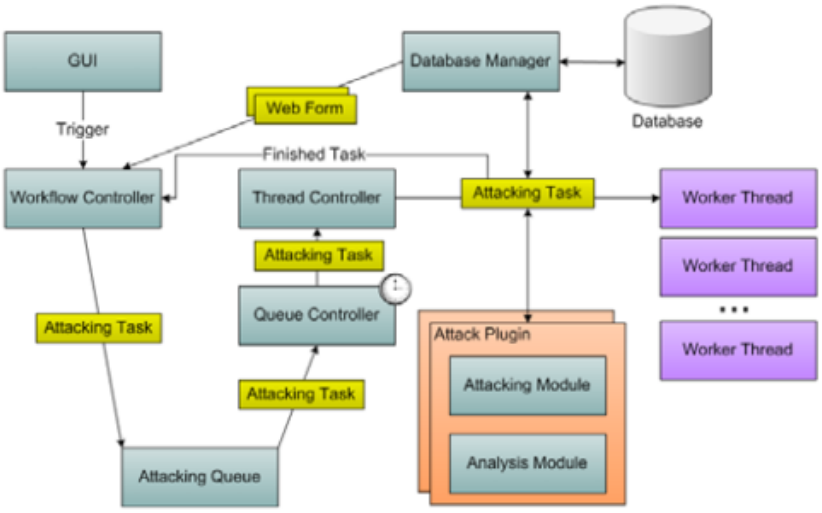

FIGURE 5: Secubat Attacking Architecture (Kals et al., 2006)

064 pages and 21 627 web forms were included in the crawl to which the automatic attacks were performed. Results indicated that the analysis module had found between 4 to 7 percent of the pages to be potentially vulnerable to attacks which were included in SecuBat. (Kals et al., 2006)

The authors further evaluated the accuracy of the tool by selecting a hundred interesting web sites from the potential victim list for further analysis. Kals et al. carried out manual confirmation of the exploitable flaws in the identified pages. Among these victims were well-known global companies. No manual SQL vulnerability verification were done based on ethical reasons as SQL attacks have risk of damaging the operational databases. Writers notified the owners of the pages about possible vulnerabilities. (Kals et al., 2006)

Kals et al. conclude that many web application vulnerabilities product of generic input validation problems and that many web vulnerabilities are easy to understand and avoid. However web developers are not security-aware and there are many vulnerable web applications on the web. Researchers predict that it is only matter of time before attackers start using automated attacks. (Kals et al., 2006)

3.2.2 State of the art: Automated black-box web application vulnerability testing

In their paper Bau et al. examine commercial black-box web application vulnerability scanners. Authors discuss how these black-box tools have become commonly integrated into compliance processes of major commercial and governmental standards such as Payment Card Industry Data Security Standard (PCI DSS), Health Insurance Portability and Accountability Act (HIPAA) and Sarbanes-Oxley Act. Bau et al. aimed to study current automated black-box web application scanners and evaluate what vulnerabilities these scanners test, how well these tested vul-

nerabilities represent the ones in the wild and how effective the scanners are. (Bau et al., 2010)

Researchers were unable to find competitive open-source tools in this area and therefore the study consists of eight well-known commercial vulnerability scanners WVS (Acunetix), HailStorm Pro (Cenzic), WebInspect (HP), Rational AppScan (IBM), McAfee Secure (McAfee), QA Edition (N-Stalker), QualysGuard PCI (Qualys) and NeXpose (Rapid7). Bau et al. explain that the study isn't aimed to be considered a purchase recommendation, as they provide no comparative detection data. (Bau et al., 2010)

Authors compare the vulnerability categories given by the scanning tools to the vulnerability incident rate data recorded by VUPEN security. VUPEN is an aggregator and validator of vulnerabilities reported by various databases such as National Vulnerability Database (National Institute of Standards and Technology, 2017). Bau et al. found that Cross-Site Scripting, SQL Injection and forms of Cross-Channel Scripting have been consistently the three of top four most reported web application vulnerability classes and Information Leaking being the one of the top four ones. Comparing these results with the commercial application scanning tests, the authors concluded that these were also the top four vectors that the scanners found. (Bau et al., 2010)

Their first phase of the experiments evaluated the scanner detection performance on established web applications. Authors chose previous versions of Drupal, phpBB and WordPress from around January 2006 as all of them had well-known vulnerabilities. Testing the scanning applications against these web applications showed that the scanners did well in Information Disclosure and Session Management vulnerability detection. Bau et al. hypothesise that adding effective test vectors to these categories is easier than to others. According to the tests the scanners did also reasonably well in detecting XSS and SQL vulnerabilities with approximately 50% detection rate. CSRF detection however was quite low. (Bau et al., 2010)

The second phase authors constructed a custom application that was used as a testbed. It contained set of contemporary vulnerabilities as well as vulnerabilities found in the wild. Application had also functionality to test all the vulnerabilities specified in the NIST Web Application Scanner Functional Specification as well as most of the vulnerability scanner detection capabilities specified in the Web Application Security Consortium. Scanners were also evaluated for how well they handled different encoding links in crawling of the testbed site. (Bau et al., 2010)

Running the vulnerability scanners against the testbed showed that scanning time between products varied from 66 minutes to 473 minutes. Also amount of network traffic range was quite large from 80 MB to nearly 1 GB. Coverage analysis by the researchers showed that the scanners had low comprehension of active technologies such as Java applets, Silverlight and Flash. Bau et al. speculate that some scanners might only perform textual analysis and this might be result of that. Detection results show that the scanners can detect over 60% of reflected XSS vulnerabilities. Most of the scanners also detected first-order SQL Injection vulnerabilities. Other vulnerability classification groups didn't fair so well in the results as no other group passed detection rating of more than 32.5%. (Bau et al., 2010)

Authors conclude that no scanner was top performer between vulnerability classifications and that for example the top performer in XSS and SQL Injection detection was in the bottom three in the Session Vulnerability detection. The writes state that the high detection rate scanners were able to control the number of false positives, while the low detection rate scanners produced many false positives. The study found that the vulnerability detection rates of the scanners were generally below 50%. Authors, however, note that black-box testing tools may prove to be very useful components in security-auditing when considering the factors of costs and time saved from manual review. (Bau et al., 2010)

3.2.3 Why Johnny Can't Pentest: An Analysis of Black-BoxWeb Vulnerability Scanners

Doupé, Cova and Vigna evaluate both commercial and open-source black-box web vulnerability scanners in their paper. The authors explain that popularity of web application scanners has risen because scanners have become automated, easy to use, and they are not restricted to specific web application technologies (Doupé et al., 2010). Writers point out that these tools however have their limitations as most testing tools, there is no provided guarantee of integrity of results and naive use of the scanners might results in false sense of security. Doupé et al. aimed to find out why these tools have poor detection performance and what are the root causes of the errors that these tools make. Custom web application called WackoPicko was build by the authors to evaluate black-box testing tools and to find out what are the root causes of these errors.

According to Doupé et al. web application scanners commonly consist of three different main modules *a crawler*, *an attacker* and *an analysis* module. The WackoPicko web application was designed to asses black-box web application scanners and these modules. WacoPicko is fully functional application that contains sixteen vulnerabilities that represent the vulnerabilities found in wild, as reported by the OWASP Top 10 project. (Doupé et al., 2010)

Researchers ran 11 web application scanners against their WackoPicko application. Scanning tools tested were Acunetix, AppScan, Burp, Grendel-Scan, Hailstorm, Milescan, N-Stalker, NTOSpider, Paros, w3af and Webinspect. Three of these were open source programs (Grendel-Scan, Paros and w3af) and others had a commercial licence. Three different configuration modes were used when running the scanners. In *initial* configuration mode the scanner was just directed to initial page of the application and told to scan all the vulnerabilities. *Config* configuration gave scanner valid username and password combination or login macro before a scan and in *manual* configuration most of the work was done by the user as scanners were put into proxy mode. (Doupé et al., 2010)

The authors noticed that the time span that the scanners used to scan the application was quite large; Burb was able to scan the application in 74 seconds while N-Stalker used 6 hours. Most of the scanners however completed their scan under 30 minutes. Authors gave their students a task to detect all the vulnerabilities in the application and only forceful browsing vulnerability was not found by the students. This result was compared to the scanning results where

no scanner was able to detect Session ID, Parameter Manipulation, Stored SQL Injection, Directory Traversal, Multi-Step Stored XSS, Logic Flaw and Forceful Browsing vulnerabilities. Only one scanner was able to exploit weak passwords in the system and login into administrator page. (Doupé et al., 2010)

All scanners except Milescan, generated false positive results. Majority of these false positives were due to supposed information leakage vulnerability where application leaks local file paths. Authors explain that two main reasons for false positives seemed to be that the scanners passed file name parameters in file traversal testing, which were then stored to some pages such as guest book, and caused scanner in later run detect these paths as information leakage. Other reason for false positive generation was that WackoPicko uses absolute paths for href attribute anchors and scanners mistook this as disclosure of paths in the local file system.(Doupé et al., 2010)

Doupé et al. studied how each of the scanners attempted to detect vulnerabilities and found that scanners would first crawl the site looking for injection points. After detecting these points the scanners would then try injecting values into each of the parameter and observe the responses that the web application returns. If pages had multiple inputs, scanners would generally try each of them in turn. This impacted some scanners as they left some fields empty in WackoPicko comment form and were unable to post comment as some required fields were left empty. (Doupé et al., 2010)

Crawling capabilities also varied between scanners. Some scanners had over 1000 accesses for each vulnerable URLs where Grendel-Scan never had accessed URLs more than 50 times. Two scanners had defective HTML parsing that caused them to miss stored XSS attack. The main feature for WackoPicko application was uploading of pictures. Three of the scanners were unable to upload successfully any pictures to application, where some uploaded 324 pictures. Scanners had also problems of running all dynamic JavaScript challenges in the page. Only one successfully completed all of them. No scanner found Flash vulnerability on applications onclick-event. Infinite web sites (pages that generate sites based on user input) proved to be problematic for Grendel-Scan as the WackoPickos calendar caused it to run out of memory while trying to access all the pages. (Doupé et al., 2010)

Doupé et al. conclude that scanning modern web applications was a serious challenge for vulnerability scanners. They point out two types of problems that affect web application vulnerability scanners. The first are the problems consists of implementation errors such as faulty HTML parsers or lack of support for commonly-used technologies such as JavaScript or Flash. The second are the problems cripple the crawling of these scanners. Modern applications with input validation and complex forms seem to effectively block scanning and crawling of the pages. The cause for this seems to be that the scanners do not model and track the state of the application. Doupé et al. suggest that more intelligent algorithms are needed for modern application "deep" crawling and that scanners need to be state aware. (Doupé et al., 2010)

Doupé et al. conclude that in order for scanners to be effective they require a sophisticated understanding of the applications they are running the test on and the limitations of the tool itself. Scanners detect certain kinds of well-established

vulnerabilities but not well-understood vulnerabilities cannot be detected by these scanners. (Doupé et al., 2010)

3.2.4 Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner

Doupé, Cavedon, Kruegel and Vigna introduce state-awareness to vulnerability scanners in their research. Writers claim that black-box scanners often operate in point-and-shoot manner when testing web applications and this has limitations as application complexity increases and when multiple actions within application change its state. This classic black-box scanning approach crawls web application to enumerate all reachable pages and then fuzzes the input data within sites. Classical approach completely ignores the different states that modern web applications may have which causes the scanner to likely test only fraction of the application. Doupé et al. aim to improve black-box scanning by constructing a partial model of the web application's state machine using automation. (Doupé et al., 2012)

State-awareness in black-box scanning allows scanner to detect pages that have their functionality change based on different states of the application. An example of a state change is a login page of a web application that is in the state zero when user is not logged in, and when a login has been completed, the page has a different functionality and is in the state one. After logging in the page might show links to other pages within application that were previously unknown to the scanner. (Doupé et al., 2012)

Doupé et al. create a state-change detection algorithm that detects state changes based on the applications outputs on identical inputs. When identical inputs cause different outputs the applications state has likely changed. Researchers explain that the algorithm first crawls the application sequentially by making requests based on a link in the previous response. It assumes that the state stays the same but when two identical requests following each other receive different responses, the algorithm presumes that one of the requests has changed the state of the web application. (Doupé et al., 2012)

The state-aware vulnerability scanner also clusters similar pages together to handle possible infinite sections of web applications and to detect when response has changed. This is done by modelling the pages using links present on the page. When links differ on the page the state of the application has changed. To detect which request by the scanner introduced the change in the web applications state, the scanner uses heuristics. Their heuristic favours the newer request over older requests and *POST* requests over *GET* requests. (Doupé et al., 2012)

Common black-box scanners use concurrent requests to increase performance of the scanner. However scanner of Doupé et al. needs to browse the web pages in a sequential order as concurrent requests can influence application's state. As scanner progresses trough a web application it only moves to the next page when the last page contains no unvisited links. When selecting path, the scanner tries to select links that are less likely to cause state changes as it wants to explore as much of the application in the current state as possible before changing the state. It also selects next pages by favouring ones with the biggest number of links that haven't

been previously visited or links that have been infrequently visited. Authors used fuzzing plugins of open-source scanner called w3af as the fuzzing component of the scanner. Implementation of the scanner allowed any fuzzing component to be used with it. (Doupé et al., 2012)

Doupé et al. evaluated their scanner against other scanners by using two metrics, False Positives and Code Coverage of the scanners. The scanners compared were *wget*, *w3af*, *skipfish* and their own *state-aware scanner*. Scanners were tested against eight different applications including two different WordPress versions. State-aware scanner had the best code coverage of the scanners. This verifies the effectiveness of the state-aware scanner algorithms. Most importantly the state-awareness showed improvements against w3af scanner which used the same fuzzing component. Improved code coverage against other scanners varied between half a percent to 140.71 percent. The authors conclude that using the presented state-aware black-box scanner it would be possible to scan more of the web application's states and that all black-box tools wanting to understand the internal state machines of web applications should adopt a similar approach. (Doupé et al., 2012)

3.2.5 Conclusions on vulnerability scanning tools

Kals et al. (2006) and Doupé et al. (2012) both suggested improvements to general black-box scanning tools, whereas papers Bau et al. (2010) and Doupé et al. (2010) gave more in-depth comparison of available tools, their features and detection rates. Kals et al. (2006) noted that popular black-box scanners use large software flaw and vulnerability repositories instead of trying to find new vulnerabilities. As of 2010 tools seemed to have moved more towards the scanning methodology of SecuBat as scanners were able to find custom flaws from custom web application created by Doupé et al. (2010). However scanning time seemed to vary quite a bit between different scanners as some were able to complete scans under two minutes and some scanners took 6 hours to scan applications (Doupé et al., 2010). Similar large variance of scanning times were detected by Bau et al. (2010) when running commercial scanners against their testbed as scanning time ranged between 66 and 473 minutes.

Only Kals et al. (2006) carried out tests to live production applications in internet as others constructed a lab environment that ran different web applications. SecuBat evaluation run scanned about 25 thousand pages and Kals et al. (2006) found that about 5% of these sites were vulnerable to SQL injections and differing XSS attacks. Interestingly it seems like the tools mentioned in Bau et al. (2010) were not using up to date or on the fly cross-referencing vulnerability databases, as applications had generally about 50% detection rate of XSS and SQL vulnerabilities. Seemingly this kind of cross-referencing vulnerability databases when checking non custom web applications could work as a quick way of detecting possible known vulnerabilities in specific web applications.

4 CONSTRUCTION OF THE ARTEFACT

In this chapter we define the requirements for an internet wide vulnerability detection method and explain the steps taken while creating the construct itself. We also discuss best practises when doing large scale scanning and ethical dilemmas surrounding it.

4.1 Requirements

The aim of this study is to construct an effective way of conducting internet wide vulnerability information gathering on web applications. The method should be able to scan most of the IPv4 address space. Scanning IPv6 range would be cumbersome as its specification makes large vertical scans very slow compared to IPv4 scans. This is because IPv6 increases IP address size from 32 bits (2^{32}) to 128 bits (2^{128}) (IETF, 2017). This means that entire IPv4 address space can fit into IPv6 address space 79 octillion times. It would be preferable that the scanning could be done within a time span of one day as this would mean that the scan data can be considered a snapshot of that one day and that is why scanning only IPv4 range is preferable. Fast scanning would make it possible to conduct monthly or even weekly scanning with the method. It is realistic to restrict scanning to most common web application ports TCP/80 (HTTP) and TCP/443 (HTTPS) as scanning all open ports from available IP addresses would exponentially increase the time required for scanning (See Definitions).

Detection of WCMS versions should then be done against the IP addresses which respond to HTTP GET requests. The method should be able to detect major and minor versions of the application, in this study this means detecting WordPress versions such as 4.7.3. If the version detection fails but the scanner is able to detect with high accuracy that site is running WordPress, the site (source data) should then be saved so that the detection can be further improved. After version detection, it should be possible to list related vulnerabilities on basis of versions.

Collection of the related metadata, such as server certificates and other HTTP headers would also increase usability of the scanner as this data can serve as an important piece of information when data is to be analysed. We can form these functional requirements and constraints into the list bellow.

- Theoretical ability of scanning entire IPv4 address space.
- Able to conduct fast scanning.
- Ability to detect IPv4 addresses which respond to HTTP GET requests on port 80 or 433.
- Ability to gather HTTP header and body.
- Ability to gather HTTP/HTTPS related information such as certificate information.
- Reliably detect most versions of specified web application.

Scanners covered in the previous chapter are not designed for scanning whole IP range, and they are also meant for new vulnerability detection. Using even the fastest scanner covered in the literature review, it would take much longer than a day to go tough the vast number of websites. Following section discusses methods of conducting internet wide scanning.

4.2 Methods of conducting internet wide scanning

Internet wide scanning isn't an easy task. A common approach for website scanning is to use web crawler in combination with a search engine, such as Google. Web crawlers however have multiple different problems. Doupé et al. (2010) noted that some crawling components of vulnerability scanners had over 1000 accesses to same URLs (See Definitions). Detecting infinite loops and actual depth of crawling in web applications can be difficult for crawlers when sites are not static, as the case is often with web applications. The other problem is related to the large amount of pages in the web and the amount of data included (Castillo, 2005). This means that crawling can take a long time and that the result isn't actually a snapshot as multiple pages may have changed during the scan (Castillo, 2005).

There has been, however, changes in the internet wide scanning during the past couple of years, as new tools, such as ZMap and Masscan, have been released (Durumeric, Wustrow and Halderman, 2013 and Graham, 2014). These tools aren't however crawlers but rather ports scanners that can be extended or their results passed forwards. They are similar to the Nmap that is an open source network exploration and security auditing utility (Lyon, 2011). With a tool like Nmap it requires multiple machines and weeks of time to complete horizontal scan of public address space Durumeric et al. (2013). Masscan is an internet port scanner that can scan entire IPv4 range under 6 minutes and uses custom TCP/IP stack in order to achieve this (Graham, 2014). ZMap on the other hand promises to be able to scan entire public IPv4 address range in under 45 minutes by using single mid-range machine and gigabyte Ethernet connection (Durumeric et al., 2013).

Before these tools, internet wide horizontal scans took a lot of time or a botnet (See Definitions). Legitimate way of doing internet wide scanning and research related to it has been previously very difficult for researchers whereas it has been known that malicious parties do this with the help of stolen network access e.g. larger botnets. (Durumeric et al., 2013) Both malicious parties and researches have now taken into these new tools as likely malicious scans are conducted with them from bullet-proof hosting companies and legal academic research is being done by researchers. Almost 80% of the single port scan traffic is originating from large scans that target over 1% of IPv4 address space and 30% of scans target more than 50% of the IPv4 address space. (Durumeric, Bailey and Halderman, 2014)

ZMap and Masscan are the most popular scanners when these horizontal scans target over 10% of IPv4 address space. Seemingly ZMap is being used more and more when the coverage of the scan increases as it was the tool of choice for 21.7% of scans that targeted more than 50% of the address space whereas Masscan was used only for 3.4% of these scans. The same study found that academic scholars clearly identify themselves when conducting scanning and that 30% out of ZMap scans targeting over 10% of the address space scanned HTTP and HTTPS ports. These scans were done by academic institutions. (Durumeric et al., 2014)

Due to the academic background, extendibility and multiple publications related to ZMap it seems like an obvious choice for conducting academic internet wide vulnerability research. ZMap is also still under active development and it is being used for multiple different ongoing scanning projects such as Rapid7's Project Sonar and Censys project (Durumeric, Adrian, Mirian, Bailey and Halderman, 2015 and Rapid7, 2017b). In the following subsection we take a closer look on how ZMap operates.

4.2.1 ZMap

ZMap was created by researchers working at University of Michigan to improve internet wide network scanning. ZMaps architectural choices allow it to be 1300 times faster than Nmap with most aggressive settings, without sacrificing accuracy. (Durumeric et al., 2013)

ZMap has *Optimized probing*, which means that ZMap assumes that source network is well provisioned and that targets are randomly ordered and widely dispersed. ZMap skips TCP/IP stack altogether and generates Ethernet frames directly. Nmap on the other hand adapts its transmission rate so that it won't saturate source or target networks. (Durumeric et al., 2013)

ZMap also has *No per-connection state* compared to Nmap, which maintains the state for each connection. ZMap can skip storing the connection states as it selects addresses according to a random permutation generated by a cyclic multiple group. ZMap accepts response packets with correct state fields for the duration of the scan, and thereby it is able to extract as much data as possible from the responses it receives. (Durumeric et al., 2013)

There is also *No retransmission* like in NMap where connection retransmits and timeouts are handled. ZMap skips this step by sending always a fixed number of probes per target. Although this can cause variation in results due to packet

loss, it has been shown that ZMap still manages to reach 98% network coverage when using only single probe per host even when running at maximum scanning speed. (Durumeric et al., 2013).

Scanner consists of three parts *scanner core*, *probe modules* and *output handlers* as seen in Figure 6. *Scanner core* consists of command line, configuration, performance monitoring, address generator, address exclusions, progress monitoring, progress reading and writing networks packages. *Probe modules* are extensible and can be customized for different types of probes. They are also used for generating probe packets and for interpretation of response packets for validity. *Output handlers* are modular and make it possible for scanner results to be piped to other processes, added to databases or passed straight to user code. (Durumeric et al., 2013)

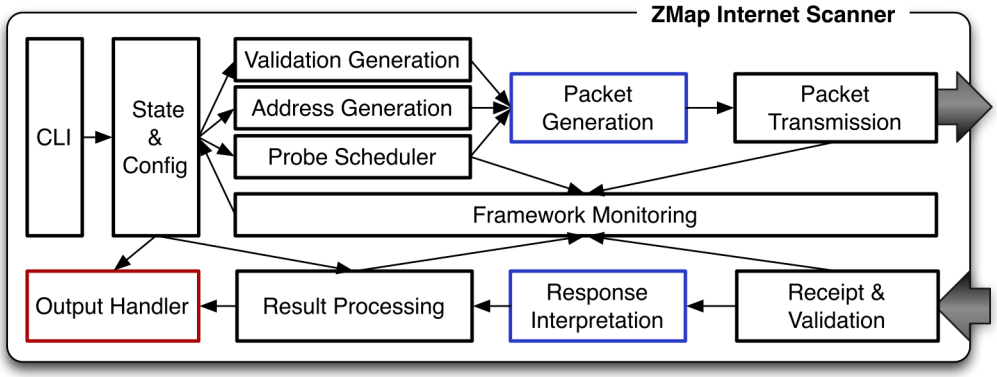

FIGURE 6: ZMap Architecture (Durumeric et al., 2013)

Limitation of ZMap is that as of writing this it only works for IPv4 address range. Approximately 9.9% of web sites have IPv6 enabled (W3Techs, 2017a). This however doesn't mean that these pages or hosts are inaccessible via IPv6 range as many sites such as Google and Wikipedia have enabled IPv6 support but, they are still accessible via IPv4 addresses.

Creators of ZMap have also created companion tools for it such as ZTee, ZGrab. ZTee is the tool recommended being used when piping ZMap to other programs such as application scanners. It is similar to regular Unix tee-command which is output buffer and splitter, but ZTee also has buffering. ZGrab is a TSL banner fetcher which also gathers other information (ZMap Team, 2017). For example ZGrab can gather HTTP request bodies and server metadata such as web server version information.

4.2.2 Application detection

Running regular application vulnerability scanners against numerous websites would be very time-consuming, illegal or in most case would at least require permission from the owners. Other way of detecting possible vulnerabilities is to detect application and then fingerprint its version.

Detecting if a site is running specific web application varies between applications. In the case of WordPress there are a couple ways of checking if the site is running it. For example, if the site has */wp-login.php* file available, */wp-content/* path is accessible or doesn't return error, *readme.html* exists and states that the site is running WordPress, or a HTML meta tag with the attribute *name* containing a *generator* and the *content* attribute containing text *WordPress*. It is possible for a host to have configured their installation in a way that removes or changes these paths, so its not always possible to detect whether the site is running WordPress with the help of these. WPScan is a popular WordPress black box vulnerability scanner which is included in Kali Linux and also uses these methods for detection (WPScan Team, 2017a).

Similarly, there are multiple possibilities for version detection. WordPress states its version in Readme.html file with default configuration. Version information is also stated in HTML meta tag within the content attribute (content="WordPress 4.x.x") with default configuration. Version information is also listed in WordPress RSS and Atom feeds.

For other web applications this version detection and application pinpointing approach of course differs as different applications give varying amounts of information regarding their versions to end users. Default installations of both Joomla and Drupal use meta that includes generator name information. Drupal also includes the major version number in the generator field.

There exist also an option of calculating hashes for supplied style sheets, scripts and other files for each version of the software and comparing these hashes to the hashes of the files that the site is hosting. This however can also be unreliable and time-consuming. Choosing the lookup for HTML Tag of WordPress versions seems like a good baseline since it is shown with the default WordPress configuration. Using it for the detection will give quite a reliable way of examining the installation version and it is possible to use the other methods on top of this to increase the reliability of the scan results.

4.2.3 Vulnerability databases

Probably the most well-known vulnerability database is the National Vulnerability Database that is hosted by National Institute of Standards and Technology. *NVD is the U.S. government repository of standards based vulnerability management data represented using the Security Content Automation Protocol* (National Institute of Standards and Technology, 2017). NVD includes databases all ranging from security checklists to impact metrics. NVD also has a vulnerability search engine which allows one to search for known software flaws (National Institute of Standards and Technology, 2017). NVD uses Common Vulnerabilities and Exposures (CVE) identifiers for vulnerability naming and standardized CVE style description style for each vulnerability (MITRE Corporation, 2017 and National Institute of Standards and Technology, 2017). These vulnerabilities are scored with help of Common Vulnerability Scoring System (CVSS) which is a framework for assessing and quantifying the impact of software vulnerabilities (Mell, Scarfone and Romanosky, 2006).

There exists also other sites that use the same data provided by NVD and combine it with other databases. One of these is CVE Details which uses NVD feeds and combines other additional information such as possible Metasploit modules that use vulnerability in question and other related exploits listed by the Exploit Database (Exploit Database, 2017 and Özkan, 2017 and Rapid7, 2017a). There is also vulnerability database only for WordPress and its plugins called WPScan Vulnerability Database which is also searchable by WordPress version and lists all possible vulnerabilities related to that specific version as a result (WPScan Team, 2017b).

4.2.4 Ethics

Crawling the web causes traffic and might cause financial costs to the owners of web sites which are crawled. There are also other ethical problems related to crawling. These things should be taken into consideration before using a web crawler.

The robots.txt protocol which governs the way how web crawlers should operate has been quite widely adopted. Robots.txt protocol allows website owners to implement mechanism for controlling how crawlers scan their pages or if they are even allowed to do so. (Thelwall and Stuart, 2006) However this method only works if crawler respects this protocol. According to Thelwall and Stuart (2006) there are four types of issues that web crawlers may raise for society or individuals. These are denial of service, cost, privacy and copyright. (Thelwall and Stuart, 2006)

Denial of service here doesn't mean purposeful denial of service attack but an unwanted one where websites design causes crawler to access the same location multiple times. It is also possible that crawler might slow down the traffic for other users of the website. Cost issue here means that the extra traffic that crawler causes could cause increased cost due to a more excessive bandwidth usage. Privacy issue arises if the crawled information is used in a non-ethical way, for example, if the crawler collects email addresses from websites and adds them to a spam list. Thelwall and Stuart (2006) claims that the biggest issue with crawlers is that they ostensibly do illegal things, that is making copies of copyrighted material. (Thelwall and Stuart, 2006)

Scanning done with ZMap however isn't strictly web crawling as ZMap operates with IP addresses and doesn't crawl links in possible web sites. ZMap can of course be extended with modules to do so or the results can be piped to a web crawler. Still, the creators of ZMap discuss good internet citizenship in regard to scanning. ZMap tries to avoid stressing target network by accessing addresses according to a random permutation (Durumeric et al., 2013). A regular web crawler tries to go through specific website in a sequential order and this has a higher possibility of causing traffic peaks. ZMap is unable to honour robots.txt standard as it is a port scanner and there doesn't exist similar standard for port scanning software currently.

However, there is still a small change that any interaction with remote systems may cause problems for the owners, or they might become alarmed by the

abnormal traffic (Durumeric et al., 2013). Durumeric et al. suggest that researchers should prepare a website that informs about the intention of the scan and includes possible contact details. In their case the website was hosted on same address as the scans originated from so it was easy to find. ZMap creators also present seven guidelines for good scanning practises as seen in the Table 3 bellow.

1. Coordinate closely with local network administrators to reduce risks and handle inquiries
2. Verify that scans will not overwhelm the local network or upstream provider
3. Signal the benign nature of the scans in web pages and DNS entries of the source addresses
4. Clearly explain the purpose and scope of the scans in all communications
5. Provide a simple means of opting out and honour requests promptly
6. Conduct scans no larger or more frequent than is necessary for research objectives
7. Spread scan traffic over time or source addresses when feasible

TABLE 3: Recommended Scanning Practises (Durumeric et al., 2013)

4.3 The proposed method

Previous chapters have discussed the approaches for vulnerability detection and flaws related to WCMS applications. Common vulnerability testing scanners are designed for small scale scans where one application or server is scanned per scan. The fastest scanners in Doupé et al. (2010) paper managed to scan their test application in 74 seconds. This would mean that even in the best scenario scanning Alexa Top 1 million sites would take over 856 days if scans averaged 74 seconds for each page and the connection speeds would be similar to the ones Doupé et al. (2010) had in their lab environment and if the scans would be conducted in sequence.

Black-box scanning tools are designed for assessing a single application at a time for vulnerabilities. Using these tools which are mainly designed for scanning single application for parallel scanning would most likely be inappropriate for the solution, but possible. Some of the programs mentioned by Doupé et al. (2010) can be executed in parallel like the Burp suite. Using Burp suite for parallel scanning huge amount of websites would require building an application to execute Burp scans and handle the list of target websites, for example which of the Alexa Top 1 million sites have been successfully scanned. Scanning time for such an experiment is hard to estimate as adding more simultaneous scanners would result into diminishing returns quite fast due to limitations of network bandwidth and computing power. There are also ethical problems regarding this type of scanning, not to mention the legal ones which vary between different jurisdictions because running black-box scanner against unknown website might reduce websites performance or affect it in some other way.

With the help of vulnerability databases we can see which versions likely have vulnerabilities, so detecting applications version is in most cases enough for detecting if installation of the application has a flaw. Detecting applications and their versions should give us approximation of the number of vulnerable applications in the wild. The proposed method of conducting internet wide application vulnerability scanning consists of following steps depicted in Figure 7.

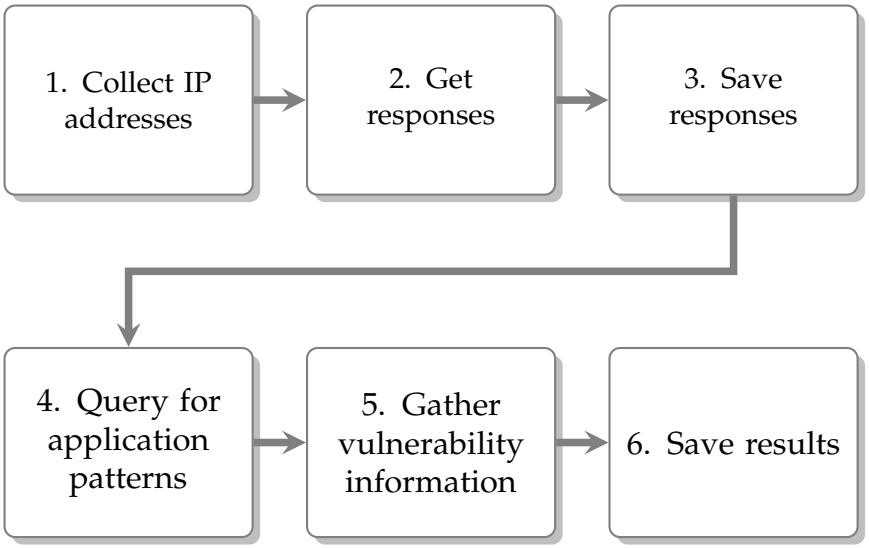

FIGURE 7: The proposed method for scanning

1. **Collect IP addresses.** All IP addresses and response bodies of addresses which respond to port scans on HTTPS or HTTP will be collected and saved for the following steps. Since ZMap doesn't have multi-portscan support, it is required to conduct two scans. The following two commands save IP addresses which respond on ports 80 or 443.

```
# zmap -p 80 --output-file=http_results.csv
# zmap -p 443 --output-file=https_results.csv
```

2. **Get responses.** ZMap companion tool ZGrab which is a TSL banner grabber with other functionality included. It can be used to get TSL banners from IP addresses but It can also gather other information such as HTTP body and server headers. Piping the addresses from ZMap to ZGrab can be done with ZTee output buffer and splitter which is included with ZMap. (ZMap Team, 2017) Following commands will run ZMap and pass the port scan results via ZTee to ZGrab which will then grab server related information and HTTP body from root of the address (See Definitions). Data will then be saved in JSON-format.

```
# zmap -p 80 --output-fields=* | ztee http_results.csv | zgrab --
    port 80 --http="/" --output-file=http_banners.json
# zmap -p 443 --output-fields=* | ztee https_results.csv | zgrab
    --port 443 --tls --http="/" --output-file=https_banners.json
```

3. **Save responses.** The Resulting file will be large and parsing huge JSON files can be inefficient. Importing data into more manageable form will improve its usability. Importing the JSON data into a database allows querying the data with relative ease.

4. **Query for application patterns.** As the data mass resulting from large scale scanning is huge, we have only collected the HTTP body responses from the sites which we have discovered. Web applications commonly still have patterns in their landing page which reveal a version related information (subsection 4.2.2). In case of a default WordPress installation, version information is stored in every generated page within HTML meta tag. Example query for approximate number of sites running WordPress version 4.7.3 could for example be following in pseudo-code.

   ```
   SELECT count(*) from db WHERE db.httpbody CONTAINS 'content="
       WordPress 4.7.3' AND NOT db.httpbody CONTAINS 'content="
       WordPress 4.7.3.'
   ```

   In the pseudo query we discard results of the versions which match the version string, but where the following character is dot as this can mean that we count other versions also (e.g. version 4.7 and version 4.7.3).

   In case full version detection for web application requires additional information from other application path, it is possible to run ZGrab or other application scanner again as we have stored the IP addresses. For example if we can detect that website is running Web Application "A" based on the html tags, but the version information is usually stored in some JavaScript file or Readme.html and the path is guessable we can run the ZGrab with different *–http* parameter and save this information to our database.

5. **Gather vulnerability information.** When the number of installations for specific versions has been determined, we can gather version related vulnerability information from vulnerability database of our choice and add this information to our database.

6. **Save results.** Query results and vulnerability information should be saved or exported for further analysis.

By following these steps it is possible to collect vulnerability information regarding web applications at scale. Tools like ZMap also allow us to gather other metadata such as server information, certificates and possibly location information during the scan. ZMap, ZTee and ZGrab related commands presented above and their results were tested in a small lab environment. The following chapter demonstrates use of this method.

5 DEMONSTRATION

This chapter demonstrates the use of the six steps (Figure 7) for collecting vulnerability information at large a scale and presents the findings regarding WordPress versions in the wild. Demonstration is done with the help of Censys database of University of Michigan which uses ZMap and ZGrab to collect Internet-wide data for research purposes (Durumeric et al., 2015).

The previous chapter discussed methods of conducting Internet-wide scanning and presented a method for doing detection of vulnerable web applications at larger scale. A small scale testing of the method will be done in a small lab environment to validate that the tools would output useful data for version fingerprinting. This testing will be presented in the next section. Conducting large scale scanning is problematic in Finland due to Chapter 38, Section 8 of The Criminal Code of Finland called *Computer break-in* (Ministry of Justice of Finland, 2015). There is prejudice (KKO:2003:36) related to this section where port scanning of address space of a Finnish bank was considered a crime and penalties were given (Supreme Court of Finland, 2003). Because of these reasons Internet-wide data collection will not be done within this these, but rather ready collected data will be used for analysis. Luckily there are open databases which collect data with the same or similar tools. The next section presents how lab environment testing was conducted and section after that discusses available databases for the demonstration.

5.1 Testing method

Section 4.3 presented method which could be used for an Internet-wide scanning of web applications. Small home lab environment was build to examine how these steps could be used for gathering the required information. The environment consisted of eight different IP addresses which were hosting pages on HTTP port. One of these addresses was hosting WordPress website with default configurations of version 4.7.3 and other addresses had either static sites of other web applications running on them.

The first step of the method is the IP address collection. The address range of ZMap scans can be restricted by specifying scanning subnet address for the tool. The lab environment used here was hosted under subnet address of 192.168.0.0/16 and ZMap has restricted scans to specific subnets with a blacklist as these are not usually the preferred targets of the scans. Unblocking the local network subnet was therefore needed to conduct this scan and this was done by editing the blacklist configuration file. After unblocking the desired subnet, following command was run from the scanning machine to check that desired amount of addresses was returned from the ZMap scan.

```
# zmap -p 80 -o result.csv 192.168.0.0/16
```

Scanning the subnet for responding addresses took around six seconds, but running scan in the lab environment with these settings seemed to give an incomplete list as the results. Dropping the default scanning rate 10 000 packets per second down to 300 packets per second seemed to fix the problem of dropped packages. Most likely the consumer grade router in the environment couldn't handle the average rate of 8 000 packages per second and dropped most of them during the scan. This might have been a security measure in the router. Scanning with the following command showed the all the eight addresses desired in the results.

```
# zmap -p 80 -r 300 -o result.csv 192.168.0.0/16
```

Rate limiting shouldn't be needed in large scale scanning as ZMap dispenses the scanning probe so that addresses will not be scanned in sequential order. However, as the subnet of the lab environment is so small the router in the environment seemed to suffer from the large number of packets. With the rate limits we can proceed to the second step of the method which is the actual data collection. Collection of http information from the subnet can be done with the following command.

```
# zmap -p 80 -r 300 --output-fields=* 192.168.0.0/16 | ztee
    http_results.csv | zgrab --port 80 --http="/" --output-file=
    http_results.json
```

Scan produces results into a file which is formatted into JavaScript Object Notation (JSON). The File consists of an array of objects which can be parsed through. Each IP address in the file has information regarding time stamp of the scan and data of the response. In this test case it means information regarding HTTP response, such as status code, protocol, HTTP headers and HTTP body. File from the lab environment scan is so small that importing the results into database would be inefficient. Instead, each HTTP body data object will be parsed through with following the regular expression (RegEx) for WordPress site matches.

```
RegEx: content\x3D\x5C\x22WordPress.([0-4]\.\d+\.?\d?\.?\d?)
```

This regular expression allows us to match the HTML body content tag version information as presented in the method proposal. For example, it is possible to match the following escaped HTML string.

```
content=\"WordPress 4.7.1
```

It is also possible to use regular expression capture grouping (round brackets in the above expression) to gather the matching versions into a list of the matched

versions or count the matched versions with it. In our test results we have one matching site which has the WordPress version information in the body content tag and the sites IP address matches our WordPress hosts address. This rudimentary testing has proven that detection is possible with the method presented in section 4.3.

As the testing environment was so small with limited hardware, it is hard to estimate how long scanning of all the available addresses in the internet would take. However, even gathering the available addresses is 1300 times faster with ZMap compared to Nmap (Durumeric et al., 2013). It is also possible to download HTTP responses during the scan by using ZMap and ZGrap at the same which makes data gathering quite fast. The hardest thing is to estimate how long parsing such a data mass would take. However, this parsing could be done with help of virtual machines or databases provided by huge cloud providers with relatively small cost. In the next section we will discuss the possibility of using data collected in a similar way for analysis part of this thesis.

5.2 Choosing database

Internet-Wide Scan Data Repository (Scans.io) is an archive of public research data which has been collected by active scans of the internet. It lists multiple different datasets of different scans ranging from different port scans, certificate scans to HTTP scans. Both Rapid7 and University of Michigan have multiple datasets listed on the site which have been collected with the help of ZMap. (Censys Team, 2017b)

Rapid7's Project Sonar conducts both HTTP and HTTPS scans weekly and provides these as a compressed package of JSON via the Internet-Wide Scan Data Repository. These packages include the HTTP GET responses from servers and SSL certificate information if it is available. Rapid7 approximates that each HTTP scan is a snapshot of a maximum of 8 hours. The scan of March 14th 2017 for HTTP consists of 75GB of compressed data and for HTTPS 67GB of data. In the uncompressed format these would be over 1.5TB each. (Rapid7, 2017b) There are multiple tools that can be used to ease the analysis of this data, but there are not public query-able databases for Project Sonar, so using SQL syntax for searching fingerprints would require us to import the data into a database.

Other option is the Censys project by the Censys Team and University of Michigan which is aimed at enabling researchers to conduct research regarding internet-wide scanning. Censys offers search engine to scans of IPv4 address space, Alexa top million domains and X.509 Certificates. The scan data is also provided in a raw format if one wants to analyse it locally. (Durumeric et al., 2015) ZMap being able to scan IPv4 range under 45 minutes and ZGrab taking about 6 hours and 20 minutes for HTTPS handshakes, Censys scans also are a snapshot of under 8 hours (Durumeric et al., 2015, 2013).

Censys team uses exclusion list for the scans so that organisations and individuals have option to opt-out from the scan range. This exclusion list consisted of 0.11% of the public IPv4 addresses in 2013 and ZMap was able to find 97%

of the theoretical maximum number of IPv4 addresses (Durumeric et al., 2013). Current exclusion list isn't public and it is not known how many organizations have blocked scans in other way. However, there are currently over 116 million hosts which have HTTP enabled and are included in the Censys scan. Authors of Censys also compared the tool to other similar tool called Shodan, which is a similar service but closed source and requires paying for results of more than 50 hosts. According to their results Censys found 222% more HTTP hosts than Shodan.

However, limitation of the Censys data is that there is currently no HTTPS Get results within the data compared to Project Sonar. Censys and Project Sonar also only collect the root path of the server. If for example same server hosts WordPress on a different path (e.g. /blog/ or /wordpress/) the scan data doesn't show these. Censys offers web search engine for the scan data and REST API that allows programmatic access to the same data. It is also possible to get SQL like query access as a researcher to the raw data which is being hosted on Google's BigQuery. There are multiple snapshots for almost every moth dating back to October 2015 and it is also possible to query them via the SQL access. (Durumeric et al., 2015). Due to the access to historical data, possibility of using SQL queries to crawl trough the HTTP bodies of large amount of hosts and transparency Censys dataset was chosen to be used in this thesis.

5.3 Information collection

Censys project REST API which offers search, view, report and data endpoints for any registered users and SQL query and export endpoints for verified researchers (Durumeric et al., 2015). There is also Python library which is also created by Censys project and is a lightweight Python wrapper for the API (Censys Team, 2017a).

WordPress has used similar HTML meta content field to inform its version atleast since version 0.71. List of WordPress versions since the version 0.71 were collected from WordPress GitHub page as well as from the release archive located on WordPress home page. In total 224 unique versions were found, ranging from June 9th 2003 with version 0.71 to the current version 4.7.3 which was released on March 6th 2017. (WordPress Foundation, 2017a, b)

SQL queries for Censys use Google BigQuery as a backend and the search queries use the specific BigQuery syntax for searches. (Durumeric et al., 2015 and Google, 2017) With the help of Censys python library, a small data collection application was constructed to query and extract information regarding different WordPress installations from the database. Search query was improved further to use regular expressions and minimize the number of queries required for information gathering. Example of the total installations gathering query used can be seen in bellow.

```
SELECT version, count(*) AS count FROM (
SELECT REGEXP_EXTRACT(i.p80.http.get.body, r'content="WordPress
    .([0-4]\.\d+\.?\d?\.?\d?)') AS version
FROM [ipv4.20170318] i)
WHERE version IS NOT NULL
GROUP BY version
ORDER BY version DESC
```

The results were exported from database into coma-separated values (CSV) format and extra information regarding version specific vulnerabilities were attached to this information. Vulnerability data was collected during the database scan from WPScan Vulnerability Database which closely matches the WordPress Vulnerabilities disclosed by National Vulnerability Database.

After these steps the results were sanitized from unknown versions with the help of known unique version listing. Total 20 unknown versions were removed from the results. There were 14 versions with less than two detected installations. The highest numbers of installations for non-listed versions were for version 4.8 (161 installations) and 4.7.4 (21 installations). These versions are most likely upcoming versions which are under testing. In total 215 unique known versions were left after the sanitization. The oldest version of these being 1.2 with the release date of May 2004.

Current data collection application uses single query for gathering installation numbers from all versions of WordPress. Further queries are required as data regarding server certificates or geographic continents. Gathering information regarding installation counts, certificate levels, geographic locations and up-to-date CVE vulnerability information for found versions takes approximately 3 minutes with the constructed application when querying the full IPv4 dataset.

5.4 Results

Data collection was done against the Censys IPv4 full scan data dated 01.04.2017. Total 692 039 WordPress installations with valid versions were found with the regular expression search method. Figure 8 shows installation counts based on the release dates of the found versions.

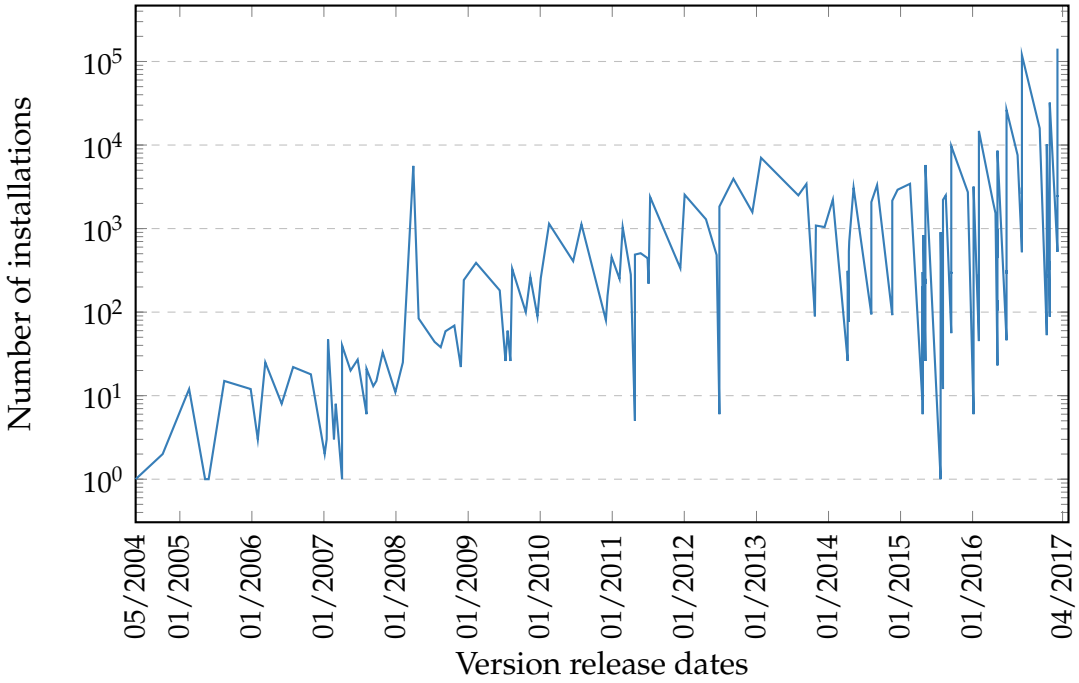

FIGURE 8: WordPress installation counts

Table 4 shows the top ten versions which had the highest number of installations. Because WordPress releases security updates for older minor versions also, there can be multiple different versions released on the same day. For example versions 4.6.1 and 4.5.4 include fixes for the same security issues (WordPress Foundation, 2016). Version 4.7.3 is the most common installation encountered with 142 555 identified pages. It was released on 6th of March 2017 and there are zero known disclosed vulnerabilities that affect the version as of 3rd of April 2017. Version 4.6.1 has the second most installations with 119 238 unique addresses detected and release date of 7th of September 2016. At the time of the scan there were 13 known disclosed vulnerabilities in that version.

Version	Total Installations	Disclosed Vulnerabilities	Release Date
4.7.3	142555	0	06.03.2017
4.6.1	119238	13	07.09.2016
4.5.4	33417	13	07.09.2016
4.7.2	32216	6	26.01.2017
4.6.4	27714	0	06.03.2017
4.5.3	26358	16	21.06.2016
4.4.5	23352	13	07.09.2016
4.5.7	17927	0	06.03.2017
4.7	15880	18	06.12.2016
4.4.8	15421	0	06.03.2017

TABLE 4: WordPress versions with most installations

Figure 9 allows us to see that there are a couple versions released before 2015 which still have over 5000 active installations. For example version 2.5 has 5 630 installations based on the results. Version 2.5 is almost 10 years old with the release date of March 29th of 2008 and there are 14 different vulnerabilities which haven't been patched in that specific version. Version 3.5.1 has the highest number of active installations from the versions released before 2015 with release date of 24.1.2013 and total 7 036 detected sites still running it.

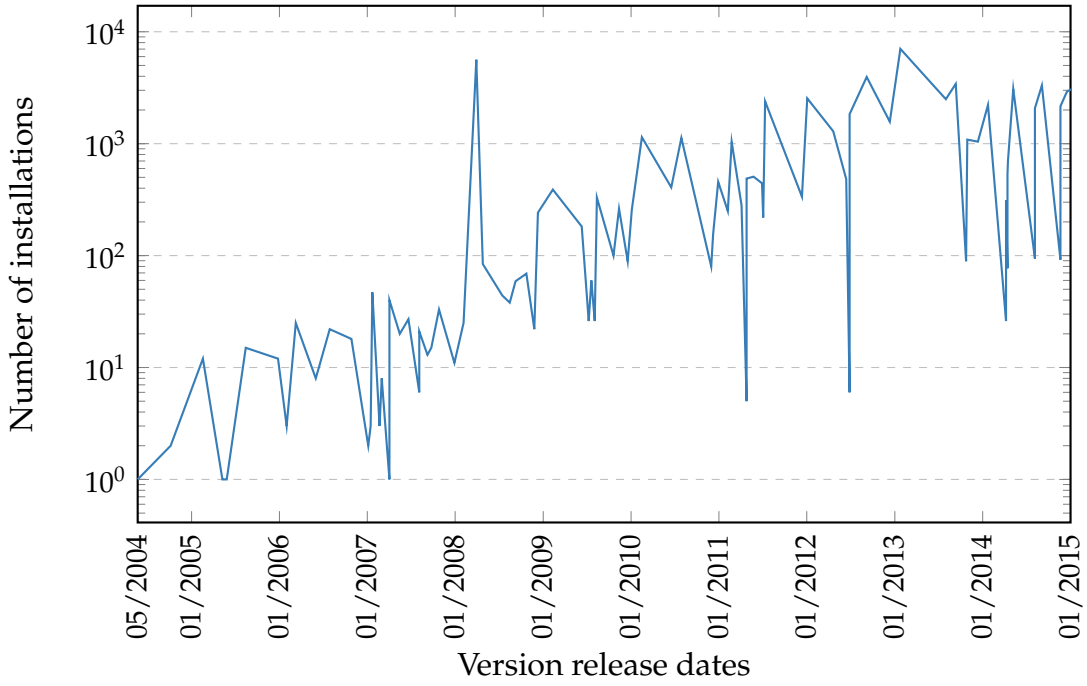

FIGURE 9: WordPress installation counts before 2015

Censys also has certificate information in their database as the ZGrab is able to collect it during the scan. There are three different validation levels for digital certificates (Leavitt, 2011). Levels are based on the degree of validation that the sender provides. Domain validated (DV) certificates base the validation on the fact that the requester has rights for the domain name. Organization validated (OV) certificates require verification of organization's formal name and the DNS names of organization. Certificate Authorities (CA) usually verify the formal name by asking copies of paperwork, like articles of incorporation. Extended validation (EV) certificates have the highest validation criteria (Leavitt, 2011). The criteria for issuing EV certificates is created and kept up to date by organization of leading Certificate Authorities, Browser makers and application vendors and it is called CA/Browser Forum (CA/Browser Forum, 2017 and Leavitt, 2011).

Figure 10 shows post-2015 release installation numbers for different certificate validation levels. Similarly, to the total installation values the Version 4.7.3 has the most installations with Domain Validated certificates totalling at 81 656. Non surprisingly it is also the most popular version for Organization Validated certificates with 5 198 installations and for Extended Validation certificates with 847.

Installation number drops in the Figure 9 and following Figure 10 are explained by the versions which have no or very few installations (see Appendix 1).

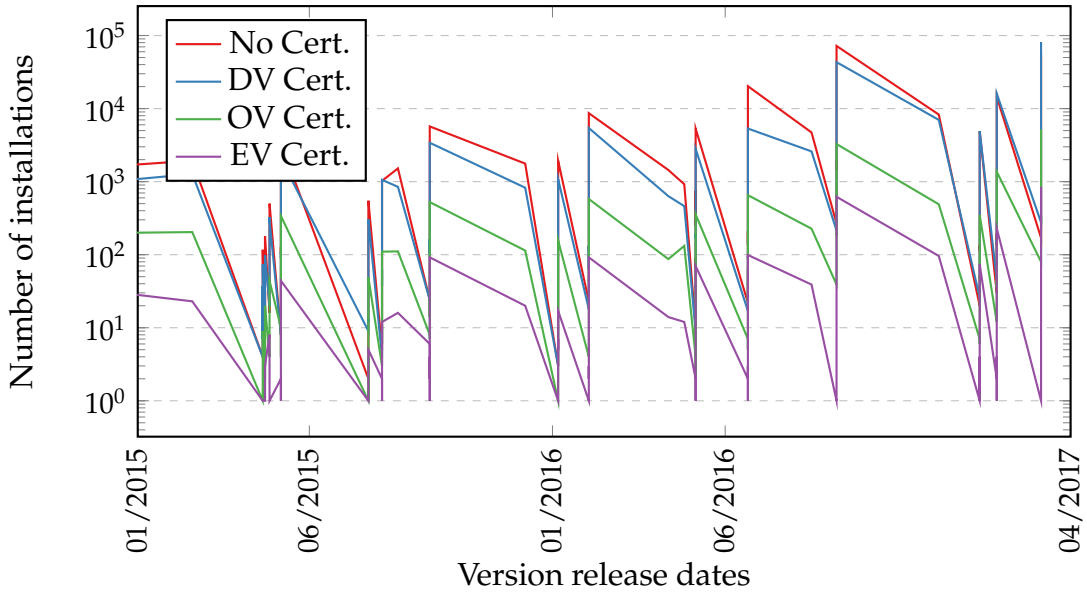

FIGURE 10: Certificate installation numbers after 2015

Certificate installation information for versions released before 2015 can be seen in Figure 11. Version 2.5 released on March 29th of 2008 has a notable number of 5 492 installations without certificates and this can be seen as the spike in early 2008 in Figure 11. There are installations with version 2.5 and Extended Validation certificate but there are 124 instances with Domain Validated certificates and 14 sites with Organization Validated certificates for that version.

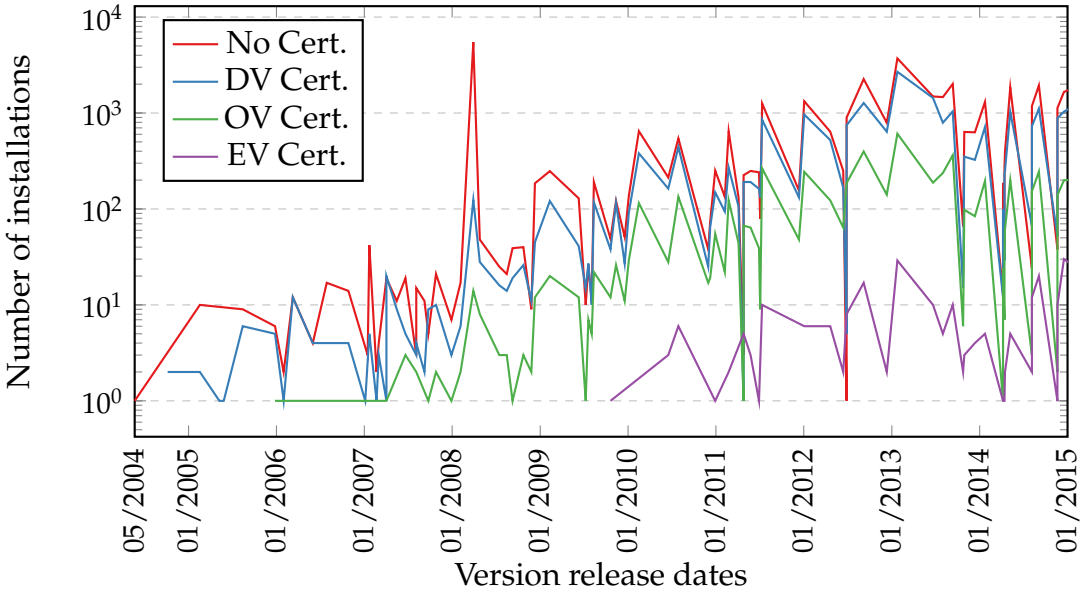

FIGURE 11: Certificate installation numbers before 2015

Geographic location information was also queried from the database. Figure 12 show the total installation numbers between continents. Majority of the installations are located in North America with total 392 360 detected installations. Second largest continent is Europe with 197 865.

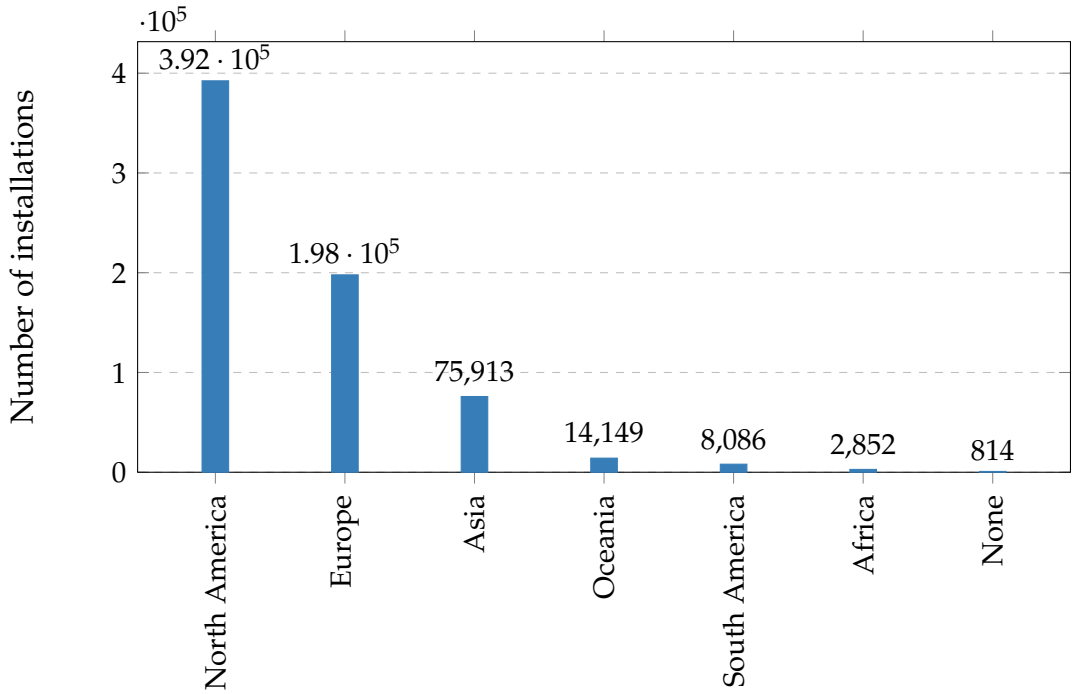

FIGURE 12: Total installations for each continent

The following section 5.5 will discuss the results presented in this chapter and compare them to the expected results which were hypothesised in section 1.5. Results will also be compared against other similar datasets and statistics presented of WordPress installations.

5.5 Validation

This study aimed to construct a method of collecting web application vulnerability information at large scale. The proposed method for data collection was presented in chapter 4 and it was then applied to data collection in chapter 5. Censys database was used to streamline the data collection. The dataset for the Censys dataset of 1st of April 2017 contained total 63 297 814 unique IP addresses which responded with HTTP status code OK (200 OK) and had HTTP body content. The Data collection method presented in this thesis was able to find 692 039 unique IP addresses which responded to HTTP/HTTPS requests with metadata that matches WordPress installation. Hence, the detection found that approximately 1.08% of the addresses which had web server running with data were hosting WordPress in the root directory.

BuiltWith Pty Ltd estimates that there are total 18 308 117 WordPress sites in the entire web and 234 918 sites within Alexa top million sites (BuiltWith Pty Ltd, 2017). The total number of installations is hard to verify as BuiltWith Pty Ltd specifies only that they use combination of Alexa and Quantcast for data sources. This likely includes sites which run on same address and different subdomain or path. As Censys also offers dataset containing responses and information regarding Alexa top million sites, we can compare the BuiltWith Pty Ltd data with our search query. The same query as the one presented in chapter 5 was modified to run against Censys Alexa top million sites dataset of date 1st of April 2017. This resulted in total 122 343 sites which were running some version of WordPress and responded with status HTTP status OK. Whole Censys Alexa dataset had 789 564 sites which had HTTP body content and which responded with OK status code. This means that the simple application version detection in case of WordPress detected that approximately 15.5% of the sites in Censys Alexa top million sites were running WordPress. According to BuiltWith Pty Ltd 23.5% of the top million sites are running WordPress so the HTTP body based detection is able to detect 66% of the sites which BuiltWith Pty Ltd detects as WordPress (BuiltWith Pty Ltd, 2017). However, BuiltWith Pty Ltd can't detect other than the major version numbers (1.x, 2.x, 3,x, 4.x) of the WordPress sites, meaning it is unable to tell difference between the versions 4.0 and 4.1.

Riddler is a tool for web topology mapping, attack surface enumeration and web discovery (F-Secure, 2017). Riddler is a combination of fast web crawler and high performance custom database. Riddler also has an application detection functionality built in and it is possible to search for example WordPress sites with search query *keyword:wordpress* (F-Secure, 2017). Random sample of 200 addresses from the results of this study were chosen for comparison against F-Secures Riddlers application detection. Out of the 200 unique sites chosen, only

three sites had any information in Riddlers database and only one of them was detected to be running WordPress. However, the version detected by the scan of this study was 4.6.1 whereas Riddler thought that the site was running version 3.6.2. It was manually confirmed that the site was actually running version 4.6.1. Riddler data regarding the site also included HTTP response date which was dated 15th of September 2014 so we can deduce that the site information and the version information in Riddlers database regarding that specific site wasn't recent and that the site might have been running version 3.6.2 in 2014, but we are unable to confirm this.

Riddler doesn't rely on port scanners like ZMap due to the chance of source IP address of scans might become automatically blocked by firewalls with such tools. This might be the reason why only three out of 200 addresses had any information in Riddlers database. Not relying on link crawling but going trough all accessible IPv4 addresses has seemingly the advantage of detecting sites which haven't been referred by search engines or other sites.

Successful data collection done in chapter 5 also allows us to explore the five hypothesis presented in section 1.5. The first hypothesis stated that over 10% of the active installations are running versions which have been released over a year ago. As the scan was conducted on first of April 2017 this means that versions released on or before first of April 2016 are of interest to us. In total 122 622 sites out 692039 sites had an older than one year old version in the sample. The first hypothesis is thus correct as approximately 17.7% of the sites detected had older versions installed. Interestingly the scan only picked 183 unique sites which were running versions released before first of April 2007. In our sample these 183 sites are a minority but still these sites are at risk if the vulnerabilities in these versions haven't been mitigated in an other way by the owners. This means that there are still some instances of early the versions like the second hypothesis hypothesised.

The third hypothesis claimed that there is a negative correlation between release age and number of installations. When calculating linear regression with dependent value as total installation number for each version and having the release age as the covariate we get R-value of -0.182. This is illustrated in Figure 13 This indicates that there is weak negative correlation between the release age and the number of installations there are for that specific version. The R^2 for the pair is 0.033, so the model explains only a small part of the variation in number of installations and release age. The second hypothesis is therefore correct, but the model doesn't explain the variance well and making predictions on the installation numbers in the future is hard. Partly this variance can be explained with the fact that multiple versions have same release dates different versions get same security updates for example.

The fourth hypothesis presented in section 1.5 was that sites which have a certificate installed are likely to have more recent version of WordPress installed. Standard descriptive information relating certificates and WordPress installation numbers were calculated can be seen in Table 5. Descriptive statistics show that installations without a certificate have the highest mean and Domain Validated certificates having next highest mean of release age.

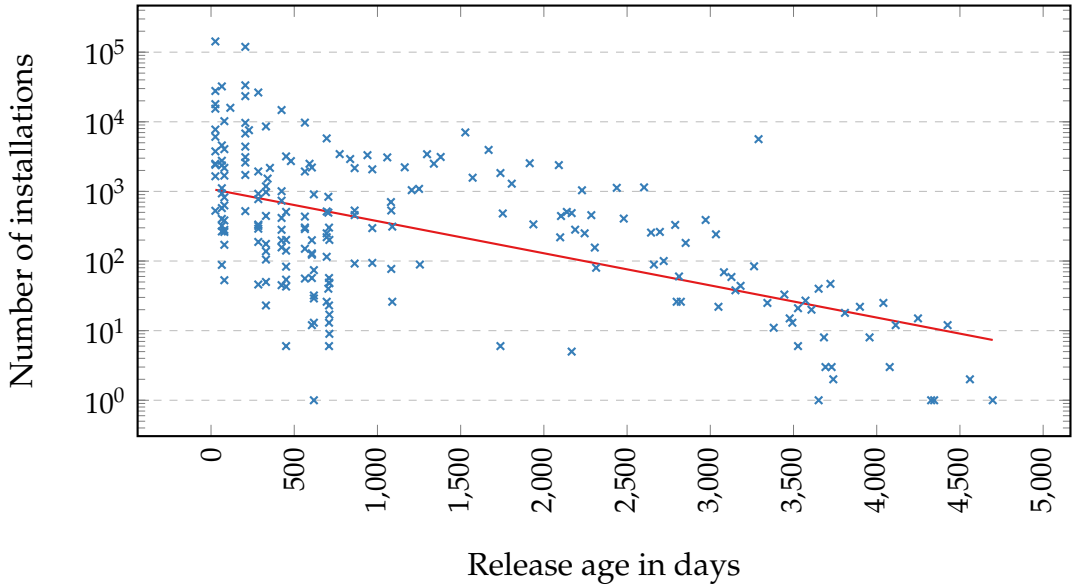

FIGURE 13: Linear regression of release age and total number of installations

	Total	No Cert	DV Cert	OV Cert	EV Cert	Release Age
Mean	3234	1686	1389	141.8	17.09	1298
Std. Deviation	1.344e+4	6696	6592	455.8	75.66	1313
Miniimum	1.000	0.000	0.000	0.000	0.000	26.00
Maximum	1.426e+5	7.212e+4	8.166e+4	5198	847.0	4697

TABLE 5: Descriptive statistics

Table 6 shows both Pearson's R and Spearman's RHO calculated for the release age and different certificate types. Dataset is more monotonic rather than just linear. Pearson's R shows quite similar negative correlation between all of the certificate classes, with more expensive (OV and EV) certificates having higher negative correlation than the sites with Domain Validated certificates or non validated sites. Still Spearman's coefficient can give better measure of the strength of the association between two variables whereas Pearson's correlation might give misleading information on the dataset (Hauke and Kossowski, 2011). Spearman's RHO shows that all certificate classes have strong strength of correlation to release age thus meaning that newer versions have likely more running instances. Seemingly there is small difference between the strength of correlation between the classes, but the difference itself seems to be non significant. Sites without certificate have Spearman's RHO of -0.526 whereas the Domain Validated sites have the highest strength of -0.592. There seems to be small but non significant difference sites which have a certificate and sites which don't, but based on the results it is hard to argue that the fourth hypothesis of sites with certificates having newer version has strong support based on the data.

The last hypothesis expected there to be very little difference between different continents and the ages of the running releases. However, Table 7 shows that there are much larger differences between different continents when comparing

	Total	No Cert	DV Cert	OV Cert	EV Cert
Pearson's r	-0.182**	-0.181**	-0.171*	-0.212**	-0.212
p-value	0.007	0.008	0.012	0.002	0.007
Spearman's rho	-0.557****	-0.526***	-0.592***	-0.551***	-0.582***
p-value	<.001	<.001	<.001	<.001	<.001

* p <.05, ** p<.01, *** p<.001

TABLE 6: Correlation of release age and total active installations and release age and different certificate installations. The first column shows values for total installations and the second column for installations without certificates. The third has the values for Domain Validated certificates, the fourth for organizational validated certificates and the last column for extended validated certificates.

the differences between certificate classes. South America has the lowest Spearman's rho value of -0.494 whereas Oceania has the highest value with -0.613. This proves the last hypothesis wrong as there is clear noticeable difference between age of the release and continents.

	South America	Oceania	North America	Europe	Asia	Africa
Pearson's r	-0.208**	-0.176**	-0.176*	-0.184**	-0.212**	-0.152*
p-value	0.002	0.010	0.010	0.007	0.002	0.026
Spearman's rho	-0.494***	-0.613***	-0.551***	-0.550***	-0.574***	-0.541***
p-value	<.001	<.001	<.001	<.001	<.001	<.001

* p <.05, ** p<.01, *** p<.001

TABLE 7: Correlation between release age and active installations on different continents. The first column shows values for South American installations and the following columns show the values for Oceania, North America, Europe, Asia and Africa in sequence.

Data showed that a surprisingly large number of the WordPress sites have recent versions of the software installed. This might be due to the automatic maintenance and security updates which were introduced into WordPress in version 3.7 (WordPress Foundation, 2013). It would be interesting to compare the state of running installations before release of 3.7, but unfortunately similar Censys data doesn't go that far back.

6 CONCLUSION

The purpose of this study was to find an effective way of collecting web application vulnerability information at large scale. This study was conducted by following a Design science research methodology (see section 1.4) where a vulnerability scanning method was the artefact resulting from the study.

Literature review was conducted to gain better understanding of risks that WCMS applications face and how vulnerability scanning should be conducted. Articles with good reputation were chosen for review on both subjects. Based on the literature review on WCMS application security, it seemed that the risks facing these applications are very similar to the ones that are listed as most common risk facing web applications in general. Review on articles relating to web application vulnerability scanning reinforced the idea that vulnerability scanning is seen as a useful tool to improve software security. However, the articles comparing different vulnerability scanners also presented the problem relating to the speed of general vulnerability scanners which make it infeasible to use these tools for large number of sites, not to mention the ethical and legal problems relating to running full feature vulnerability scanners against sites without permission from the owners.

Based on the literature review, an alternative approach was to be taken for large scale vulnerability information collection. In chapter 4 requirements for such method were laid out and available extendable tools which are able to do full IP range scans were further studied. After this a method for scanning was proposed. This collection method was then tested via demonstration scan and the results were compared to the hypothesis presented in the introductory chapter.

Demonstration showed that the method of scanning and detecting sites with vulnerable versions of application seems to yield fast and fairly accurate information on large scale. Tools such as ZMap and ZGrab allow conducting fast internet wide sweeps of open ports and capture important application information during this. Simple filtering of the data was enough in the case of WordPress to detect most of the running WordPress installations and their versions.

Resulting data from the scan showed that 17.7% of the WordPress sites detected older than one year old versions. There was also clear negative correlation between the release age and the number of installations that version has. Only small differences could be seen between different certificate type installations and

release age. There were minor differences between different continents on how recent versions were common within them.

Modern tools and access to high bandwidth allows collecting of snapshot like data of application versions and detecting possible sites which have unpatched vulnerabilities. Similar data has likely been collected by malicious parties for quite some time with help of botnets or other tools. New tools allow better understanding of the status of current installed web applications. This information might become useful as vulnerable applications are not only risk for the owners of the sites and machines where they are being ran on, but also for other people as they can be used for malicious attacks such as DDoS.

This study has worked as a proof of concept that vulnerability information collection of web applications is possible at the internet wide scale. The constructed WordPress detection is in itself quite simple but it seems to be quite effective. Extending the method to other applications would be a subject for further study and such scans could be used for monitoring current installation bases of web applications.

DEFINITIONS

Authentication in the computer security context means the process of verification of user or other entity who it claims to be. For example users commonly authenticate with web applications by submitting their user name and password to authenticate with the application.

Backdoor usually allows malicious user to connect to computer with little or no authentication and execute commands on the local system. Backdoor is usually gained with malicious code that installs itself on the computer. (Sikorski and Honig, 2012)

Botnet consists of multiple computers that have been infected with backdoor. These computers listen for instructions from command-and-control server and act upon them. (Sikorski and Honig, 2012)

Buffer overflow is an attack technique that leverages a storage bounds checking flaw in either software or hardware. (Shirey, 2007)

Content Management System (CMS) is an application that allows management and creation of digital content.

Failure is an event when a system is unable to perform within the specified limits or when it is altogether unable to perform the required tasks. (ISO, IEC and IEEE, 2017, pp. 178)

Fault is an error in software. (ISO et al., 2017, pp. 179)

Flaw can be an error in the implementation design or operation of an information system. (Shirey, 2007)

Fuzzing is the process of dynamically generating common attack string inputs or queries to find vulnerabilities in applications. (Stuttard and Pinto, 2011)

Hypertext Transfer Protocol (HTTP) is a TCP-based, internet protocol that is used to carry data requests and responses in World Wide Web (Shirey, 2007). Typically, the port number 80 is used for HTTP connections.

Hypertext Transfer Protocol Secure (HTTPS) is an adaptation of the HTTP protocol for secure communications. Transport Layer Security (TSL) is commonly used for securing the HTTPS connections. Previously this was done with the Secure Sockets Layer (SSL).

Penetration test is the practice where experts try to break in or abuse the system and this way find the flaws, so they can be fixed. Commonly penetration tests

closely resemble what real attacker would do. (Pfleeger, Pfleeger and Margulies, 2015)

Root address is the root path of an address. For example, www.example.org address is a root path but www.example.org/example/ isn't.

Uniform resource identifier (URI) is a compact sequence of characters that can be used to identify an abstract or physical resource available on the Internet. (ISO et al., 2017, pp. 485)

Uniform resource locator (URL) is a URI which describes both access method and location of information source object on the internet (e.g. http://example.org). (Shirey, 2007)

Validation is the process of evaluating that a system fulfils the requirements specified for the system. (ISO et al., 2017, pp. 495-496)

Verification is the process of confirming whether the specified requirements have been fulfilled by examining the objective evidence. (ISO et al., 2017, pp. 500)

Vulnerability is design, implementation or operation and management flaw or weakness which could be exploited to gain access or otherwise violate system's security policy. (Shirey, 2007)

Web Content Management System (WCM) is a system that allows users to create and manage web content.

Web application is an application which can be accessed and communicated with by using a web browser (Stuttard and Pinto, 2011).

REFERENCES

Amman P. and Offutt J. (2008). Introduction to Software Testing. Cambridge University Press.

Arce I. (2003). The weakest link revisited. IEEE Secur. Priv. Mag. 1(2), 72–76.

Arkin B., Stender S. and McGraw G. (2005). Software penetration testing. IEEE Secur. Priv. 3(1), 84–87.

Arora A., Krishnan R., Telang R. and Yang Y. (2010). An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. Inf. Syst. Res. 21(1), 115–132.

Bau J., Bursztein E., Gupta D. and Mitchell J. (2010). State of the art: Automated black-box web application vulnerability testing. Proc. - IEEE Symp. Secur. Priv. , 332–345.

BuiltWith Pty Ltd (2017). CMS technologies Web Usage Statistics, April. Retrieved 03.04.2017 from http://trends.builtwith.com/cms.

Bulgurcu B., Cavusoglu H. and Benbasat I. (2010). Information security policy compliance: An empirical study of rationality-based beliefs and information security awareness. MIS Q. 34(3), 523–548.

CA/Browser Forum (2017). Guidelines For The Issuance And Management Of Extended Validation Certificates Version 1.6.1. Technical report.

Castillo C. (2005). Effective web crawling. ACM SIGIR Forum 39(1), 55.

Censys Team (2017a). Censys Python Library. Retrieved 22.03.2017 from https://github.com/censys/censys-python.

Censys Team (2017b). Internet-Wide Scan Data Repository. Retrieved 22.03.2017 from https://scans.io/.

Doupé A., Cavedon L., Kruegel C. and Vigna G. (2012). Enemy of the State: A State-Aware Black-Box Web Vulnerability Scanner. USENIX Secur. Symp. , 523–538.

Doupé A., Cova M. and Vigna G. (2010). Why Johnny can't pentest: An analysis of black-box web vulnerability scanners. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics) 6201 LNCS, 111–131.

Durumeric Z., Adrian D., Mirian A., Bailey M. and Halderman J.A. (2015). A Search Engine Backed by Internet-Wide Scanning. Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur. - CCS '15 , 542–553.

Durumeric Z., Bailey M. and Halderman J.A. (2014). An Internet-Wide View of Internet-Wide Scanning. 23rd USENIX Secur. Symp. (USENIX Secur. 14) , 65–78.

Durumeric Z., Wustrow E. and Halderman J.A. (2013). ZMap: Fast Internet-wide Scanning and Its Security Applications. Proc. 22nd USENIX Secur. Symp. (August), 605–619.

Exploit Database (2017). Exploit Database. Retrieved 16.03.2017 from https://www.exploit-db.com.

F-Secure (2017). Ready to explore the deep web? Retrieved 22.03.2017 from https://riddler.io/static/riddler_white_paper.pdf.

Google (2017). Google BigQuery Reference. Retrieved 22.03.2017 from https://cloud.google.com/bigquery/docs/reference/legacy-sql.

Graham R.D. (2014). MASSCAN: Mass IP port scanner. Retrieved 03.03.2017 from https://github.com/robertdavidgraham/masscan.

Hauke J. and Kossowski T. (2011). Comparison of Values of Pearson's and Spearman's Correlation Coefficients on the Same Sets of Data. Quaest. Geogr. 30(2).

Hevner A.R., March S.T., Park J. and Ram S. (2004). Design Science in Information Systems Research. MIS Q. 28(1), 75–105.

Identity Theft Resource Center (2017). Data Breaches Increase 40 Percent in 2016, Finds New Report from Identity Theft Resource Center and CyberScout. Retrieved 28.01.2017 from http://www.idtheftcenter.org/2016databreaches.html.

IETF (2017). Internet Protocol, Version 6 (IPv6) Specification. Retrieved 19.11.2017 from https://tools.ietf.org/html/rfc8200.

ISO, IEC and IEEE (2017). Systems and software engineering – Vocabulary , 1–522.

Kals S., Kirda E., Kruegel C. and Jovanovic N. (2006). SecuBat: A Web Vulnerability Scanner. In Proc. 15th Int. Conf. World Wide Web - WWW '06. 247.

Leavitt N. (2011). Internet security under attack: The undermining of digital certificates. Computer (Long. Beach. Calif). 44(12), 17–20.

Lichtblau E. (2016). Hackers Get Employee Records at Justice and Homeland Security Depts. Retrieved 28.01.2017 from https://www.nytimes.com/2016/02/09/us/hackers-access-employee-records-at-justice-and-homeland-security-depts.html.

Lyon G. (2011). Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning.

Martínez S., Garcia-Alfaro J., Cuppens F., Cuppens-Boulahia N. and Cabot J. (2013). Towards an Access-Control Metamodel for Web Content Management Systems. In Curr. Trends Web Eng., April. 148–155.

Mcgraw G. (2004). Software security. IEEE Secur. Priv. Mag. 2(2), 80–83.

Meike M., Sametinger J. and Wiesauer a. (2009). Security in Open Source Web Content Management Systems. IEEE Secur. Priv. Mag. 7(August).

Mell P., Scarfone K. and Romanosky S. (2006). Common Vulnerability Scoring System. IEEE Secur. Priv. Mag. 4(6), 85–89.

Ministry of Justice of Finland (2015). The Criminal Code of Finland .

MITRE Corporation (2013). CVE-2014-0160. Retrieved 19.11.2017 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-0160.

MITRE Corporation (2014). CVE-2014-6271. Retrieved 19.11.2017 from https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-6271.

MITRE Corporation (2017). Common Vulnerabilities and Exposures. Retrieved 17.03.2017 from https://cve.mitre.org/about/.

National Institute of Standards and Technology (2017). National Vulnerability Database. Retrieved 16.03.2017 from https://nvd.nist.gov/.

Okoli C. and Schabram K. (2010). Working Papers on Information Systems A Guide to Conducting a Systematic Literature Review of Information Systems Research. Work. Pap. Inf. Syst. 10(26), 1–51.

OWASP Foundation (2013a). OWASP - Top 10 2013. Retrieved 01.03.2015 from https://www.owasp.org/index.php/Top_10_2013.

OWASP Foundation (2013b). OWASP Top Ten Project. Retrieved 01.03.2015 from https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

Özkan S. (2017). CVE Details. Retrieved 16.03.2017 from https://www.cvedetails.com.

Peffers K., Tuunanen T., Rothenberger M.a. and Chatterjee S. (2007). A Design Science Research Methodology for Information Systems Research. J. Manag. Inf. Syst. 24(3), 45–77.

Pfleeger C.P., Pfleeger S.L. and Margulies J. (2015). Security in Computing. Prentice Hall, 5th edition.

Potter B. and McGraw G. (2004). Software security testing. IEEE Secur. Priv. Mag. 2(5), 81–85.

Rapid7 (2017a). Metasploit. Retrieved 16.03.2017 from https://metasploit.com/.

Rapid7 (2017b). Project Sonar. Retrieved 22.03.2017 from https://sonar.labs.rapid7.com/.

Raymond E.S. (2003). The Art of Unix Programming. Addison-Wesley.

Seitz J. (2015). Black Hat Python - Programming for Hackers and Pentesters. No Starch Press, Inc.

Shahzad M., Shafiq M.Z. and Liu A.X. (2012). A large scale exploratory analysis of software vulnerability life cycles. In 2012 34th Int. Conf. Softw. Eng. IEEE, 771–781.

Shirey R.W. (2007). Internet Security Glossary, Version 2. In Req. Comments, volume 4949. The Internet Engineering Task Force, 1–365.

Sikorski M. and Honig A. (2012). Practical Malware Analysis, volume 53. No Starch Press, Inc.

Stuttard D. and Pinto M. (2011). The web application hacker's handbook: discovering and exploiting security flaws. Indianapolis: Wiley Publishing, 2 edition.

Supreme Court of Finland (2003). KKO:2003:36. Retrieved 22.03.2017 from http://www.finlex.fi/fi/oikeus/kko/kko/2003/20030036.

Thelwall M. and Stuart D. (2006). Web crawling ethics revisited: Cost, privacy, and denial of service. J. Am. Soc. Inf. Sci. Technol. 57(13), 1771–1779.

Thielman S. (2016). Yahoo hack: 1bn accounts compromised by biggest data breach in history. Retrieved 28.01.2017 from https://www.theguardian.com/technology/2016/dec/14/yahoo-hack-security-of-one-billion-accounts-breached.

Vaidyanathan G. and Mautone S. (2009). Security in dynamic web content management systems applications. Commun. ACM 52, 121.

W3C (2014). HTML5 A vocabulary and associated APIs for HTML and XHTML. Retrieved 30.01.2017 from https://www.w3.org/TR/html5/introduction.html.

W3Techs (2017a). Usage of IPv6 for websites. Retrieved 14.03.2017 from https://w3techs.com/technologies/details/ce-ipv6/all/all.

W3Techs (2017b). Usage Statistics and Market Share of Content Management Systems for Websites, April 2017. Retrieved 04.03.2017 from https://w3techs.com/technologies/overview/content_management/all.

WordPress Foundation (2013). WordPress 3.7 Changelog. Retrieved 01.04.2017 from https://codex.wordpress.org/Version_3.7.

WordPress Foundation (2016). WordPress Version 4.5.4. Retrieved 19.11.2017 from https://codex.wordpress.org/Version_4.5.4.

WordPress Foundation (2017a). WordPress GitHub. Retrieved 22.03.2017 from https://github.com/WordPress/WordPress/.

WordPress Foundation (2017b). Wordpress Release Archive. Retrieved 22.03.2017 from https://wordpress.org/download/release-archive/.

WPScan Team (2017a). WPScan. Retrieved 16.03.2017 from https://github.com/wpscanteam/wpscan.

WPScan Team (2017b). WPScan Vulnerability Database. Retrieved 16.03.2017 from https://wpvulndb.com/.

ZMap Team (2017). ZGrab. Retrieved 21.03.2017 from https://github.com/zmap.

FIRST APPENDIX

Scan results of running the demonstration program against Censys IPv4 dataset presented in chapter 5.

Version	Total Installs	No Cert	DV Cert	OV Cert	EV Cert	South America	Oceania	North America	Europe	Asia	Africa	None	Vuln. Count	Release Date
4.7.3	142555	54854	81656	5198	847	993	3046	86359	40127	11458	273	299	0	2017-03-06
4.6.4	27714	10161	16247	1159	147	252	695	16984	7131	2542	66	44	0	2017-03-06
4.5.7	17927	7075	9741	1003	108	181	336	10777	4621	1944	43	25	0	2017-03-06
4.4.8	15421	6454	8080	803	84	219	203	9418	4164	1375	29	13	0	2017-03-06
4.3.9	7738	3100	3832	776	30	88	116	3964	2342	1200	20	8	0	2017-03-06
4.2.13	6085	2309	3198	548	30	54	115	3367	1760	770	15	4	0	2017-03-06
4.1.16	3761	1330	1999	410	22	43	85	2024	990	605	9	5	0	2017-03-06
4.0.16	2412	829	1304	266	13	21	42	1266	696	384	3	0	0	2017-03-06
3.9.17	2496	819	1336	331	10	19	44	1388	554	474	13	4	0	2017-03-06
3.8.19	1650	556	930	154	10	8	28	953	418	239	4	0	0	2017-03-06

3.7.19	527	170	276	80	1	9	3	320	132	63	0	0	0	2017-03-06
4.7.2	32216	14535	16081	1371	229	309	687	18675	9187	3273	46	39	6	2017-01-26
4.6.3	4581	2079	2302	172	28	66	128	2725	1120	521	14	7	4	2017-01-26
4.5.6	2757	1239	1355	149	14	35	70	1654	654	336	6	2	4	2017-01-26
4.4.7	2370	1115	1135	111	9	34	52	1429	621	230	3	1	4	2017-01-26
4.3.8	1111	544	466	94	7	12	19	573	325	180	1	1	4	2017-01-26
4.2.12	937	385	456	93	3	8	15	523	231	155	5	0	4	2017-01-26
4.1.15	576	241	280	54	1	10	10	299	143	113	1	0	3	2017-01-26
4.0.15	320	134	146	36	4	5	5	169	87	52	1	1	3	2017-01-26
3.9.16	403	195	159	45	4	5	4	181	73	137	3	0	2	2017-01-26
3.8.18	266	111	136	17	2	6	3	163	53	41	0	0	2	2017-01-26
3.7.18	88	30	46	12	0	0	1	58	16	13	0	0	2	2017-01-26
4.7.1	10210	4889	4897	350	74	88	214	6103	2854	896	37	18	10	2017-01-11
4.6.2	4048	1901	1999	132	16	34	90	2440	1046	419	10	9	7	2017-01-11
4.5.5	2177	994	1057	114	12	17	32	1291	542	287	8	0	8	2017-01-11
4.4.6	1686	829	780	67	10	28	21	1027	451	153	5	1	8	2017-01-11
4.3.7	822	413	358	47	4	13	12	418	217	158	3	1	8	2017-01-11
4.2.11	629	282	294	52	1	8	9	358	162	91	1	0	6	2017-01-11
4.1.14	383	181	163	38	1	3	6	177	124	71	1	1	4	2017-01-11
4.0.14	262	113	119	27	3	3	6	146	58	48	1	0	4	2017-01-11
3.9.15	280	134	115	27	4	4	2	125	58	89	1	1	3	2017-01-11
3.8.17	171	81	83	6	1	1	4	88	35	42	1	0	3	2017-01-11
3.7.17	53	20	26	7	0	0	2	29	13	9	0	0	4	2017-01-11
4.7	15880	8287	7007	490	96	133	314	9439	4492	1418	33	51	18	2016-12-06
4.6.1	119238	72121	43224	3272	621	1338	3008	69194	33304	11553	740	101	13	2016-09-07
4.5.4	33417	21173	11316	799	129	304	852	18813	8788	4498	138	24	13	2016-09-07
4.4.5	23352	14934	7767	573	78	276	399	12577	7716	2237	131	16	13	2016-09-07

4.3.6	9650	5847	3324	442	37	119	227	4891	2777	1536	87	13	13	2016-09-07
4.2.10	6790	3965	2474	318	33	109	165	3421	1972	1073	46	4	11	2016-09-07
4.1.13	4400	2488	1639	256	17	68	119	2148	1252	798	11	4	9	2016-09-07
4.0.13	2599	1444	951	188	16	38	75	1285	721	472	8	0	9	2016-09-07
3.9.14	3105	1936	957	200	12	32	55	1407	603	914	91	3	8	2016-09-07
3.8.16	1722	915	706	90	11	20	23	893	419	345	19	3	8	2016-09-07
3.7.16	521	267	214	39	1	4	14	294	116	93	0	0	8	2016-09-07
4.6	7572	4711	2595	227	39	114	158	4415	2086	773	21	5	15	2016-08-16
4.5.3	26358	20271	5335	653	99	421	446	9798	13271	2356	58	8	16	2016-06-21
4.4.4	1934	1318	563	44	9	45	60	1066	498	260	4	1	15	2016-06-21
4.3.5	925	603	280	40	2	11	28	489	236	157	4	0	15	2016-06-21
4.2.9	775	499	232	41	3	21	21	362	214	156	1	0	13	2016-06-21
4.1.12	334	208	104	22	0	5	4	200	85	39	1	0	11	2016-06-21
4.0.12	291	137	88	65	1	0	7	191	70	21	2	0	11	2016-06-21
3.9.13	319	210	97	10	2	7	10	134	58	110	0	0	10	2016-06-21
3.8.15	188	122	61	5	0	0	1	110	30	21	21	5	10	2016-06-21
3.7.15	46	22	17	7	0	0	3	26	14	3	0	0	10	2016-06-21
4.5.2	8569	5327	2816	356	70	185	187	4617	2535	1001	32	12	21	2016-05-06
4.4.3	1183	696	429	51	7	16	24	704	297	137	5	0	18	2016-05-06
4.3.4	444	251	167	24	2	8	5	258	99	73	1	0	18	2016-05-06
4.2.8	976	750	193	31	2	5	6	205	91	666	3	0	16	2016-05-06
4.1.11	176	94	67	15	0	4	6	101	48	17	0	0	13	2016-05-06
4.0.11	105	65	35	4	1	1	1	50	34	16	3	0	13	2016-05-06
3.9.12	138	89	37	10	2	1	0	74	20	41	2	0	12	2016-05-06
3.8.14	50	23	25	2	0	1	0	31	8	10	0	0	12	2016-05-06
3.7.14	23	10	9	4	0	1	0	18	3	1	0	0	12	2016-05-06
4.5.1	1528	922	462	132	12	21	31	873	438	163	2	0	20	2016-04-26

4.5	2177	1443	633	87	14	33	39	1347	546	200	11	1	20	2016-04-12
4.4.2	14755	8662	5422	579	92	242	211	8596	4102	1553	46	5	23	2016-02-02
4.3.3	1006	593	362	44	7	20	25	579	238	139	4	1	23	2016-02-02
4.2.7	736	443	245	46	2	11	15	386	163	158	3	0	21	2016-02-02
4.1.10	416	217	163	33	3	5	6	252	92	57	4	0	17	2016-02-02
4.0.10	198	113	63	21	1	8	5	107	56	22	0	0	17	2016-02-02
3.9.11	281	131	133	17	0	2	9	137	81	49	3	0	16	2016-02-02
3.8.13	157	85	64	8	0	2	2	84	53	16	0	0	16	2016-02-02
3.7.13	45	23	18	4	0	0	1	29	11	4	0	0	16	2016-02-02
4.4.1	3182	1930	1079	156	17	64	83	1670	1023	333	8	1	25	2016-01-06
4.3.2	508	293	190	20	5	5	6	233	181	79	3	1	25	2016-01-06
4.2.6	201	120	67	7	7	2	2	124	46	27	0	0	23	2016-01-06
4.1.9	141	69	66	6	0	2	1	92	26	20	0	0	19	2016-01-06
4.0.9	83	50	29	4	0	1	1	40	25	16	0	0	19	2016-01-06
3.9.10	54	27	24	2	1	0	0	35	13	6	0	0	18	2016-01-06
3.8.12	43	22	20	1	0	0	1	22	13	7	0	0	18	2016-01-06
3.7.12	6	3	3	0	0	0	1	5	0	0	0	0	18	2016-01-06
4.4	2726	1766	826	114	20	56	46	1412	928	282	2	0	26	2015-12-08
4.3.1	9727	5702	3408	525	92	175	150	5312	2927	1114	45	4	27	2015-09-15
4.2.5	1933	1520	357	54	2	28	11	474	1210	200	8	2	25	2015-09-15
4.1.8	435	239	178	14	4	8	4	263	104	53	3	0	21	2015-09-15
4.0.8	288	159	103	25	1	4	2	157	62	63	0	0	21	2015-09-15
3.9.9	303	114	163	20	6	4	16	128	119	35	1	0	20	2015-09-15
3.8.11	150	65	78	7	0	1	7	98	32	11	1	0	19	2015-09-15
3.7.11	56	24	24	8	0	1	2	35	12	6	0	0	19	2015-09-15
4.3	2495	1519	849	111	16	57	46	1430	680	274	6	2	29	2015-08-18
4.2.4	2205	1025	1058	110	12	44	31	1426	455	240	8	1	27	2015-08-04

4.1.7	199	116	71	11	1	2	7	106	50	34	0	0	23	2015-08-04
4.0.7	129	85	38	6	0	3	2	61	27	36	0	0	23	2015-08-04
3.9.8	123	48	63	11	1	9	0	63	29	22	0	0	22	2015-08-04
3.8.10	57	21	31	3	2	0	1	35	13	8	0	0	22	2015-08-04
3.7.10	12	5	4	3	0	0	1	6	3	2	0	0	24	2015-08-04
4.2.3	907	548	305	49	5	22	23	491	256	112	2	1	32	2015-07-23
4.1.6	74	35	34	4	1	1	0	51	16	6	0	0	28	2015-07-23
4.0.6	32	19	12	1	0	0	1	15	11	5	0	0	28	2015-07-23
3.9.7	29	12	13	4	0	0	0	14	9	6	0	0	27	2015-07-23
3.8.9	13	2	9	2	0	1	1	5	5	1	0	0	27	2015-07-23
3.7.9	1	0	0	1	0	0	0	1	0	0	0	0	27	2015-07-23
4.2.2	5773	3318	2071	340	44	134	137	3028	1628	800	44	2	33	2015-05-07
4.1.5	515	253	216	43	3	11	18	258	140	87	1	0	29	2015-05-07
4.0.5	220	103	89	27	1	3	2	126	48	40	1	0	29	2015-05-07
3.9.6	250	127	108	14	1	7	4	106	58	75	0	0	28	2015-05-07
3.8.8	115	57	52	4	2	1	0	68	28	18	0	0	28	2015-05-07
3.7.8	26	8	8	10	0	1	1	16	5	3	0	0	28	2015-05-07
4.2.1	836	500	291	44	1	7	21	488	234	84	1	1	34	2015-04-27
4.1.4	495	147	327	13	8	5	2	411	57	19	1	0	30	2015-04-27
4.0.4	40	16	20	4	0	2	0	25	10	3	0	0	29	2015-04-27
4.2	302	180	99	20	3	14	14	142	90	40	1	1	35	2015-04-23
4.1.3	49	19	27	2	1	1	0	16	19	13	0	0	30	2015-04-23
4.0.3	23	9	12	2	0	0	0	16	3	4	0	0	29	2015-04-23
3.9.5	57	32	20	5	0	2	0	22	17	16	0	0	28	2015-04-23
3.8.7	13	6	6	1	0	0	0	8	4	1	0	0	28	2015-04-23
3.7.7	6	3	3	0	0	1	0	3	0	2	0	0	28	2015-04-23
4.1.2	201	117	74	9	1	4	6	101	53	36	1	0	31	2015-04-21

4.0.2	17	8	7	2	0	0	0	10	4	3	0	0	29	2015-04-21
3.9.4	48	10	37	1	0	0	0	11	2	35	0	0	28	2015-04-21
3.8.6	9	4	4	1	0	0	0	5	3	1	0	0	28	2015-04-21
4.1.1	3437	1937	1273	204	23	72	63	1753	1031	511	7	0	34	2015-02-18
4.1	2918	1656	1033	199	30	85	57	1395	979	396	6	0	33	2014-12-17
4.0.1	2159	1128	882	139	10	54	30	1189	588	288	10	0	31	2014-11-20
3.9.3	531	270	232	28	1	12	5	293	136	84	0	1	29	2014-11-20
3.8.5	457	244	193	19	1	7	2	262	135	51	0	0	28	2014-11-20
3.7.5	92	38	52	2	0	2	1	58	21	10	0	0	28	2014-11-20
4.0	3316	1941	1110	245	20	87	47	1733	964	474	9	2	35	2014-09-04
3.9.2	2076	1182	729	153	12	43	27	971	419	587	29	0	35	2014-08-06
3.8.4	297	175	110	10	2	4	3	187	68	35	0	0	29	2014-08-06
3.7.4	94	24	67	3	0	0	1	74	13	6	0	0	29	2014-08-06
3.9.1	3081	1850	1027	199	5	83	36	1657	773	470	57	5	38	2014-05-08
3.9	703	391	249	61	2	18	11	355	186	133	0	0	39	2014-04-16
3.8.3	532	270	230	31	1	5	8	300	144	75	0	0	36	2014-04-14
3.7.3	77	41	29	7	0	0	4	48	13	12	0	0	27	2014-04-14
3.8.2	312	187	100	24	1	12	7	147	76	70	0	0	36	2014-04-08
3.7.2	26	13	12	1	0	0	0	13	4	9	0	0	26	2014-04-08
3.8.1	2220	1305	714	196	5	35	38	1173	593	358	21	2	41	2014-01-23
3.8	1042	629	325	84	4	25	26	570	326	95	0	0	38	2013-12-12
3.7.1	1087	635	351	98	3	17	11	599	311	146	3	0	41	2013-10-29
3.7	89	66	15	6	2	2	0	43	26	18	0	0	36	2013-10-24
3.6.1	3422	2010	1039	363	10	64	55	1473	950	836	37	7	23	2013-09-11
3.6	2499	1465	792	237	5	43	44	1647	440	310	15	0	32	2013-08-01
3.5.2	3123	1489	1436	188	10	25	33	1513	1112	325	111	4	24	2013-06-21
3.5.1	7036	3708	2690	609	29	90	80	4035	1741	1023	63	4	29	2013-01-24

3.5	1573	791	639	141	2	35	18	888	416	207	1	8	30	2012-12-11
3.4.2	3944	2258	1272	397	17	44	32	1990	1148	571	147	12	27	2012-09-06
3.4.1	1836	896	746	186	8	46	28	1065	418	275	3	1	25	2012-06-27
3.3.3	6	1	5	0	0	0	0	1	3	2	0	0	20	2012-06-27
3.4	482	250	166	64	2	13	5	276	101	85	1	1	25	2012-06-13
3.3.2	1291	639	523	123	6	23	7	775	285	198	3	0	27	2012-04-20
3.3.1	2540	1326	964	244	6	23	31	1550	599	334	2	1	26	2012-01-03
3.3	336	156	132	48	0	3	2	212	75	44	0	0	25	2011-12-12
3.2.1	2386	1258	853	265	10	36	43	1302	658	322	24	1	24	2011-07-12
3.2	219	79	131	9	0	4	0	84	110	21	0	0	24	2011-07-04
3.1.4	443	241	162	39	1	6	3	97	56	281	0	0	24	2011-06-29
3.1.3	507	249	191	64	3	8	2	298	99	99	0	1	25	2011-05-25
3.1.2	488	225	191	67	5	10	0	268	129	81	0	0	25	2011-04-26
3.0.6	5	3	1	1	0	0	0	1	1	3	0	0	22	2011-04-26
3.1.1	282	132	106	44	0	4	2	156	55	64	0	1	25	2011-04-05
3.1	1038	644	269	123	2	3	4	449	442	138	2	0	25	2011-02-23
3.0.5	250	134	94	22	0	2	2	122	77	47	0	0	25	2011-02-07
3.0.4	456	251	149	55	1	4	3	289	90	70	0	0	25	2010-12-29
3.0.3	156	73	64	19	0	1	2	106	23	24	0	0	27	2010-12-08
3.0.2	80	38	25	17	0	1	0	60	9	10	0	0	26	2010-11-30
3.0.1	1125	544	441	134	6	20	17	651	258	175	2	2	30	2010-07-29
3.0	407	213	163	28	3	4	5	290	64	44	0	0	30	2010-06-17
2.9.2	1142	647	380	115	0	5	10	683	294	149	0	1	19	2010-02-15
2.9.1	256	136	91	29	0	2	1	159	60	34	0	0	19	2010-01-04
2.9	89	51	27	11	0	6	1	47	16	19	0	0	21	2009-12-18
2.8.6	262	121	115	26	0	3	1	143	50	65	0	0	18	2009-11-12
2.8.5	100	49	38	12	1	2	1	70	17	9	0	1	19	2009-10-20

2.8.4	330	191	117	22	0	5	1	184	91	47	1	1	18	2009-08-12
2.8.3	26	11	10	5	0	1	1	9	6	9	0	0	19	2009-08-03
2.8.2	60	26	27	7	0	0	0	50	7	3	0	0	18	2009-07-20
2.8.1	26	10	15	1	0	2	0	13	7	4	0	0	19	2009-07-09
2.8	182	129	41	12	0	1	4	98	57	21	0	1	17	2009-06-10
2.7.1	389	248	121	20	0	9	1	219	96	63	1	0	16	2009-02-10
2.7	242	185	45	12	0	8	0	148	43	42	1	0	16	2008-12-10
2.6.5	22	9	11	2	0	0	0	11	7	4	0	0	14	2008-11-25
2.6.3	69	40	26	3	0	1	0	29	30	9	0	0	14	2008-10-23
2.6.2	59	39	19	1	0	0	0	36	13	10	0	0	15	2008-09-08
2.6.1	38	21	14	3	0	3	0	16	12	7	0	0	16	2008-08-15
2.6	44	25	16	3	0	1	2	27	8	6	0	0	14	2008-07-15
2.5.1	84	48	28	8	0	0	1	52	15	16	0	0	14	2008-04-25
2.5	5630	5492	124	14	0	71	12	3960	1426	160	1	0	14	2008-03-29
2.3.3	25	17	6	2	0	0	0	10	15	0	0	0	11	2008-02-05
2.3.2	11	7	3	1	0	0	0	8	3	0	0	0	12	2007-12-29
2.3.1	33	21	10	2	0	1	0	19	10	3	0	0	13	2007-10-26
2.3	15	5	9	1	0	0	0	12	2	1	0	0	11	2007-09-24
2.2.3	13	11	2	0	0	0	0	10	3	0	0	0	12	2007-09-08
2.2.2	21	15	4	2	0	0	0	16	4	1	0	0	12	2007-08-05
2.0.11	6	3	3	0	0	0	0	2	3	1	0	0	12	2007-08-05
2.2.1	27	19	5	3	0	0	0	20	6	1	0	0	12	2007-06-21
2.2	20	11	9	0	0	0	0	13	6	1	0	0	14	2007-05-16
2.1.3	40	19	20	1	0	0	0	7	32	1	0	0	13	2007-04-03
2.0.10	1	0	1	0	0	0	0	1	0	0	0	0	12	2007-04-03
2.1.2	8	4	3	1	0	0	0	4	4	0	0	0	14	2007-03-02
2.1.1	3	2	1	0	0	0	0	2	0	1	0	0	14	2007-02-21

2.1	47	42	5	0	0	0	0	26	20	1	0	0	11	2007-01-22
2.0.7	3	3	0	0	0	0	0	3	0	0	0	0	12	2007-01-15
2.0.6	2	0	1	1	0	0	0	1	1	0	0	0	13	2007-01-05
2.0.5	18	14	4	0	0	0	0	11	6	1	0	0	13	2006-10-27
2.0.4	22	17	4	1	0	0	0	18	3	1	0	0	13	2006-07-29
2.0.3	8	4	4	0	0	0	0	3	4	1	0	0	13	2006-06-01
2.0.2	25	12	12	1	0	0	0	25	0	0	0	0	13	2006-03-10
2.0.1	3	2	1	0	0	0	0	2	1	0	0	0	12	2006-01-31
2.0	12	6	5	1	0	0	0	10	0	2	0	0	12	2005-12-26
1.5.2	15	9	6	0	0	0	0	14	1	0	0	0	6	2005-08-14
1.5.1.2	1	0	1	0	0	0	0	1	0	0	0	0	10	2005-05-27
1.5.1.1	1	0	1	0	0	0	0	1	0	0	0	0	12	2005-05-09
1.5	12	10	2	0	0	0	0	11	0	1	0	0	10	2005-02-17
1.2.1	2	0	2	0	0	0	0	1	1	0	0	0	6	2004-10-06
1.2	1	1	0	0	0	0	0	1	0	0	0	0	7	2004-05-22