MASTER PROJECT (Fall 2014)
Computer Science Section

# Detection of Vulnerable Web Applications Using Public Vulnerability Documentation Sources

Author:

Farzaneh MOGHADDAM

CERN Supervisor:

Sebastian LOPIENSKI

EPFL Supervisor:

Prof. Philippe JANSON

February 2015

# Acknowledgements

I would like to use this opportunity to express my special appreciation and thanks to Mr. Sebastian Lopienski for his inspiring guidance and enlightening views on a number of issues related to the project. I would also like to express my gratitude to Professor Philippe Janson for his invaluable advice on the progress of the project and the quality of this thesis, and Dr. Stefan Lueders for his continuous support and help with writing this thesis. In addition, a big thank you to my family and friends for their motivating words and their endless love. I would like to dedicate my work to my sister, Atefeh, without whose love and support, it would have been impossible for me to pursue a master's degree at EPFL.

# Abstract

It is crucial for big organizations like CERN with a very large web landscape to ensure the security of their web resources. One can use various open source or commercial web scanners to scan websites and look for common, well-known web vulnerabilities. However, these scanners often fail to find new vulnerabilities that are discovered every day in software products and commonly used libraries. The main objective of this project is to automate the procedure for detecting vulnerable web applications as soon as new vulnerabilities emerge. This thesis introduces a tool that monitors public vulnerability documentation sources to detect CERN resources that use third party products which may be affected by newly discovered vulnerabilities. In addition, another tool will be introduced to facilitate scanning CERN resources for critical vulnerabilities in emergency cases, specially when a vulnerability is not specific to a certain, detectable third party product.

# Contents

# Chapter 1

# Introduction

Over the past decade, web applications have been embraced by many companies to deliver their main services to customers. One can think of different reasons for this increase in popularity: ubiquity of web browsers makes it convenient to use them as a client; web applications can be updated and maintained locally with no need to distribute or install software on the client side and web applications are cross-platform compatible. Another reason for the increasing popularity of web applications is the simplicity of developing them. Nowadays there are many web application frameworks and content management systems that facilitate rapid application development. This simplicity comes with a cost: securing a web application is difficult. Web applications include code that resides on the web servers, application servers, databases, and back-end systems of an organization. The potential for a security breach exists in each of these layers. This opens the door to attackers trying to manipulate the application logic to perpetrate their misdeeds[1]. In this project, we are going to focus on approaches for maintaining the security of web applications at CERN[1]. CERN is a large particle physics research organization and it manages thousands of websites and web servers for different purposes. Ensuring the security of this large web landscape is one of the main priorities of its Computer Security Team.

Even if a web application is designed to be as secure as possible, vulnerabilities will invariably be discovered over time in its code or in the code of underlying packages it uses and relies on. Therefore, to continue ensuring that the application remains secure, it is important to constantly check for newly discovered vulnerabilities and make sure that the application is not

[1]Conseil Européen pour la Recherche Nucléaire (European Council for Nuclear Research)

affected. This task is easier for the application designers themselves, who are aware of all the technologies that are used in their application; however, in many organizations like CERN, any employee can set up a web server or launch a website and few employees bother to monitor their own web applications for security vulnerabilities. It is the job of the Computer Security Team to scan these various web servers and websites and detect all vulnerabilities remotely.

There are two main approaches for ensuring the security of web applications: The first approach is to use available automatic scanning tools, such as Acunetix or Skipfish[1], to detect vulnerabilities. Using this approach, we can find common vulnerabilities, but when a new vulnerability emerges we have to wait for the new version of the scanning tool to be released with tests for the new vulnerability. Also, due to the complexity of these tools, it is a time consuming task to configure them for a specific scanning purpose.

The second approach is to keep an eye on vulnerability sources, databases, security mailing lists, etc. to get informed about new vulnerabilities as soon as possible. This way we can keep abreast of any critical vulnerabilities, but then one needs to use the vulnerability information to identify all potentially affected resources in the organization.

## 1.1 Project Focus

The focus of this project is going to be on the second approach, meaning that we are going to propose two new methods for identifying vulnerable resources affected by newly disclosed vulnerabilities or vulnerabilities that common web scanners fail to identify.

The disclosed vulnerabilities are regularly published by different sources, such as mailing lists, databases, forums, etc. These vulnerabilities might be disclosed with a list of affected products; for example, a vulnerability that affects all websites using Drupal Content Management Systems. In Chapter 2, we take advantage of publicly available vulnerability sources and describe a tool we developed –The Vulnerability Notification Tool– that first reports all newly disclosed vulnerabilities and in a second step delivers lists of vulnerable components (from vulnerability information) and uses those lists

[1]Google open source security reconnaissance tool

to alert the CERN Computer Security Team about web applications that may be vulnerable as a result of using those components.

On the other hand, sometimes the vulnerabilities are not specific to a certain product, e.g. a vulnerability found in a network protocol or an encryption algorithm, or it is impossible with current tools to detect if a resource is using a certain product. Chapter 3 introduces a complementary new tool we developed –the Scanner– that facilitates scanning CERN web applications components with simple (home-grown) security tests to detect individual vulnerabilities on potentially affected components. Heartbleed[1] is a good example of a case when it was critical to detect vulnerable resources as soon as possible. The vulnerability was not specific to a single detectable product and most organizations had to use their own (or publicly available) scripts to detect the vulnerability and patch the resources.

## 1.2 Vulnerability Definition

According to the MITRE CVE initiative[2], a vulnerability is a mistake in software that can be directly used by a hacker to gain access to a system or network. A vulnerability would allow an attacker to:

- execute unauthorized commands
- access data against specified access restrictions for that data
- pose as another entity or user
- conduct a denial of service

Note that a vulnerability can be a result of a mistake in design or in configuration of a product. For example, not changing the default password is a configuration rather than a design vulnerability.

## 1.3 CERN Web Landscape

CERN provides different web services, such as web authoring and web publishing for its users. The purpose of the CERN Web-services[3] is to avoid website duplication and locally managed web servers, as well as proposing

[1]Critical OpenSSL vulnerability, discovered in April 2014
[2]`http://cve.mitre.org/about/terminology.html#Dist`
[3]`http://cern.ch/web`

standard web authoring technologies. From the security point of view, using CERN Web-services rather than setting up a locally managed web server is recommended. The people managing CERN Web-services are aware of the software used for different layers of the web stack they provide, and if a vulnerability in any of those layers emerges, it is easier to patch the central server, rather than expecting individuals to patch their local web server. However, there is no restriction on setting up a local web server and any employee can still host his websites on a local web server.

### 1.3.1 CERN Websites

There are more than 13,000 websites at CERN that are centrally hosted by the CERN Web-services. These websites have a URL in the format of `http://cern.ch/X` or `http://X.web.cern.ch`, where X is the name of the website. The CERN homepage at `http://home.web.cern.ch/` is an example of a such a centrally hosted website. When creating a new website, a user can choose the type of the website from the following list:

- **Centrally Hosted on DFS**[1]: This type is recommended for Windows users. The DFS site is linked to a dedicated DFS folder where the user can edit files as he pleases with any web authoring tool he may have.
- **AFS**[2] **Folder**: This type plays a similar role for Linux users. Each AFS site is related to an AFS folder.
- **Collaboration Workspace (SharePoint)**: This type is suitable to easily create a collaboration website to work within teams.
- **Social Community**: This type is used to create websites to allow communities to discuss any topics, find answers to questions and connect with others.
- **Drupal**: This type provides a Content Management System to publish, edit and organize content through a common web interface.
- **Java MiddleWare On Demand site**: This type is used as a Java solution for deploying servlets or JSP web applications.

[1]Distributed File System

[2]Andrew File System

The servers hosting these websites are monitored by the CERN Web-services team to keep them secure. For example, if a vulnerability in Drupal is discovered the Drupal team will make sure that all Drupal websites are patched immediately. However, each website owner can customize or configure his website to use additional technologies of which the security might then be uncertain. It is the job of the Computer Security Team to scan such individual websites and look for potential vulnerabilities.

### 1.3.2 Non-central Web Servers

The CERN web landscape is not limited to the servers and websites hosted centrally by the Web-services. There are more than 1000 instances of dedicated web servers at CERN in the format of `http://X.cern.ch` where X is the server name. Indico[1] at `http://indico.cern.ch/` is an example of such a dedicated web server. Most of the websites hosted on non-central web servers are only visible inside CERN, but some have firewall openings and are accessible from the Internet.

## 1.4 CERN Web Security

The vulnerabilities in web applications can result from three main causes:

1. A misconfiguration on the server side, such as using weak ciphers or expired certificates, that leaves a door open for attackers.

2. Wrong design or implementation choices, opening a door to cross-site scripting (XSS) or other attacks

3. Use of vulnerable third party software, e.g. an outdated Drupal instance, that needs to be updated or patched.

The CERN Computer Security Team uses various tools and procedures to prevent exposure of vulnerable web applications to the world outside the organization. Also, given how many outside contractors, guests, visitors, etc. work at CERN, it is important to detect any vulnerabilities in web applications inside CERN as soon as possible and take necessary actions to secure the whole web infrastructure from the inside as well.

[1]CERN tool for managing conferences, workshops and meetings

### 1.4.1 Prevention

In order to lower the probability of developing vulnerable software, users are encouraged to use CERN central Web-services to create websites. In addition, whenever a website is being published on the Internet and is accessible from outside CERN network; it has to go through the firewall opening procedure. In the firewall opening procedure, a member of the CERN Computer Security Team analyzes the case completely to make sure that the request is reasonable, e.g. there are enough reasons for not using the central Web-services. Additionally, tools like OpenVas[1] and Skipfish are used to scan the website and report the vulnerabilities and warnings back to the owner of the website.

### 1.4.2 Detection

Whenever a new vulnerability is published, it is a part of the CERN Computer Security Team job to find affected resources and notify their owners. Currently, the Computer Security Team members get informed about new vulnerabilities by subscribing to product mailing lists, security forums, Twitter accounts, etc. If they decide that a vulnerability is worth investigating the next step is to scan the whole web infrastructure for the vulnerable resources. For this purpose, small detection scripts are developed or downloaded (e.g. for detecting the Heartbleed vulnerability or ShellShock[2]). The detection scripts can also look for misconfigurations, such as empty landing pages or HTTP (instead of HTTPS) authentication.

## 1.5 Relevant Tools

Typically different tools are used by the CERN Computer Security Team to ensure the security of CERN resources. This section gives a short description of the tools that are relevant to this project and are mentioned in the following chapters.

### 1.5.1 Web Application Scanning

The Web Application Scanner (WAS) is a tool that can be used to generate a list of all web servers and websites at CERN, run the Skipfish web scanner on each of the detected URLs and report warnings and vulnerabilities, or run

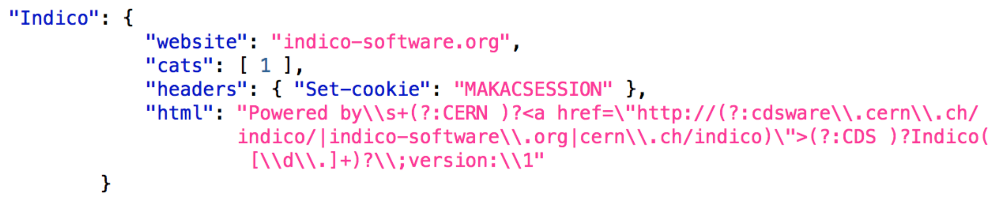
[1]Open Vulnerability Assesment System

[2]Critical Shell vulnerability discovered in September 2014

the Web Application Detection (WAD) tool to detect technologies used on a particular website or web server.

### Web Application Detection

The CERN web landscape –both independent web servers and centrally-hosted websites– are regularly scanned with the Web Application Detection (WAD) tool, in order to detect web applications and technologies they use, and possibly also their versions. This tool is developed using an open source project called “Wappalyzer”[1]. Wappalyzer is a cross-platform utility that uncovers the technologies, such as Content Management Systems, eCommerce platforms, web servers, etc. used on websites. Wappalyzer uses a set of predefined rules to detect technologies and these rules are stored in a JSON file. WAD uses this JSON file to detect technologies at CERN. Figure 1.1 illustrates a detection rule used to detect Indico instances. It uses regular expressions to search for the phrase “Powered by Indico” on the web page. Alternatively, it detects Indico instances by the name of the session variables a website uses.



```
"Indico": {
            "website": "indico-software.org",
            "cats": [ 1 ],
            "headers": { "Set-cookie": "MAKACSESSION" },
            "html": "Powered by\\s+(?:CERN )?<a href=\"http://(?:cdsware\\.cern\\.ch/
                    indico/|indico-software\\.org|cern\\.ch/indico)\">(?:CDS )?Indico(
                     [\\d\\.]+)?\\;version:\\1"
        }
```

Figure 1.1: Indico Detection Rule

## 1.5.2 Detection Scripts

Over the years, the CERN Computer Security Team has developed individual scripts. Most of these scripts are short security tests to check for a critical vulnerability (e.g. Heartbleed) on CERN resources. Additionally, some scripts detect misconfigurations (e.g. empty landing pages) on web resources that could threaten the security of CERN resources. Currently, CERN can detect the following vulnerabilities, using detection scripts:

[1] `https://github.com/ElbertF/Wappalyzer`

- **Heartbleed** - The device is vulnerable to OpenSSL Heartbleed vulnerability.
- **SSL2 Check**: - SSL version 2 is enabled.
- **SSL3 Check**: - SSL version 3 is enabled.
- **Weak Cipher** - The server is using an insecure cipher.
- **Homepage Check** - The landing page is empty or is using the default landing page.
- **CouchDB Check** - The CouchDB[1] database is protected.
- **Expired Certificate** - The server is using an expired certificate.
- **Self-signed Certificate** - The server is using a self-signed certificate.
- **Non-valid Certificate** - The server is using a certificate that is not valid for the domain.
- **Non-trusted Certificate** - The server is using a certificate that is not trusted.
- **Wildcard Certificate** - The server is using a wildcard certificate.
- **Basic HTTP Authentication** - The website or web server is using Basic HTTP Authentication.
- **OpenSSL CCS Injection** - The device is vulnerable to the OpenSSL Change Cipher Spec (CCS) injection vulnerability.

## 1.6 Project Objectives

The main objective of this project is to automate –as much as possible– the procedures for detecting vulnerable web applications. We would like to take advantage of publicly available vulnerability documentation sources to detect CERN resources that use potentially vulnerable third party products. Another objective of the project is to facilitate scanning CERN resources for critical vulnerabilities in emergency cases, specially when a vulnerability is not specific to a certain, detectable product. This project focuses on detection of vulnerable web applications remotely, more specifically when a new vulnerability emerges.

[1]Apache open source NoSQL database that can be accessed from web

# Chapter 2

# Vulnerability Notification Tool

## 2.1 Motivation

Every day new vulnerabilities and security updates are announced. Some of these vulnerabilities affect CERN websites or web servers and could be critical. Therefore, it is important to learn about them as soon as possible and notify the owners of the affected resources to take necessary actions, i.e. patch or update their resource. The current procedure in the CERN Computer Security Team for managing vulnerabilities is as follows:

1. Getting informed about vulnerabilities from various sources by monitoring public databases, project mailing lists, security mailing lists, Twitter accounts, blogs, etc.

2. Deciding if a vulnerability is worth investigating and is likely to affect CERN (human decision).

3. In case of vulnerabilities related to web applications, reviewing the output of WAD on all CERN websites and web servers, in order to get a list of resources that could be affected.

4. Sending notifications to the resource owners after making sure that the resource is in fact vulnerable, i.e. runs a vulnerable version or has a vulnerable configuration.

The main goal of the Vulnerability Notification Tool (VNT) we developed is to automate this procedure as much as possible and optimize each step of the process. Staying up-to-date with vulnerability sources and announcements is a time consuming task and it is hard to know if one is monitoring the best sources. With the current procedure, it is easy to overlook some vulnerabilities that are important for CERN, but do not create much noise in the public. On the other hand, so many vulnerabilities are published every day that it is impossible to investigate every single one of them even if we know about all of them.

Figure 2.1[1] shows the trend in the number of vulnerabilities published in the last 15 years. Apparently, 2014 with almost 8000 vulnerabilities has been the worst year so far for security. Due to the ever increasing volume of public vulnerability reports, the CVE MITRE vulnerability repository has even updated the CVE-ID syntax since January 2015. The new syntax, unlike the previous 4-digit one, allows identifying an unlimited number of vulnerabilities each year.[2]

VNT tries to save the CERN Computer Security Team's time by filtering out the non-critical vulnerabilities or the vulnerabilities that do not affect CERN, so that more time can be spent on remediation of the remaining ones.

## 2.2 Source Evaluation

There are plenty of public sources that announce vulnerabilities or maintain a database of all vulnerabilities over the years. The performance of VNT very much depends on the quality of the source it is using. Obviously, it is not possible to find a perfect source. For example, there is a trade-off between the speed of publishing the vulnerabilities and the accuracy of the information published; hence, it is crucial to know the needs of the organization and choose the most suitable source.

### 2.2.1 Approach

In order to evaluate vulnerability sources, the following factors have been considered:

1. **Completeness**: Does the source contain all the vulnerabilities that CERN would probably care about?

2. **Speed**: How long, since the disclosure, does it take for a vulnerability to be published?

[1]Picture taken from `http://web.nvd.nist.gov/view/vuln/statistics-results?adv_search=true&cves=on&pub_date_start_month=0&pub_date_start_year=2000&pub_date_end_month=11&pub_date_end_year=2014`

[2]`https://cve.mitre.org/cve/identifiers/syntaxchange.html`

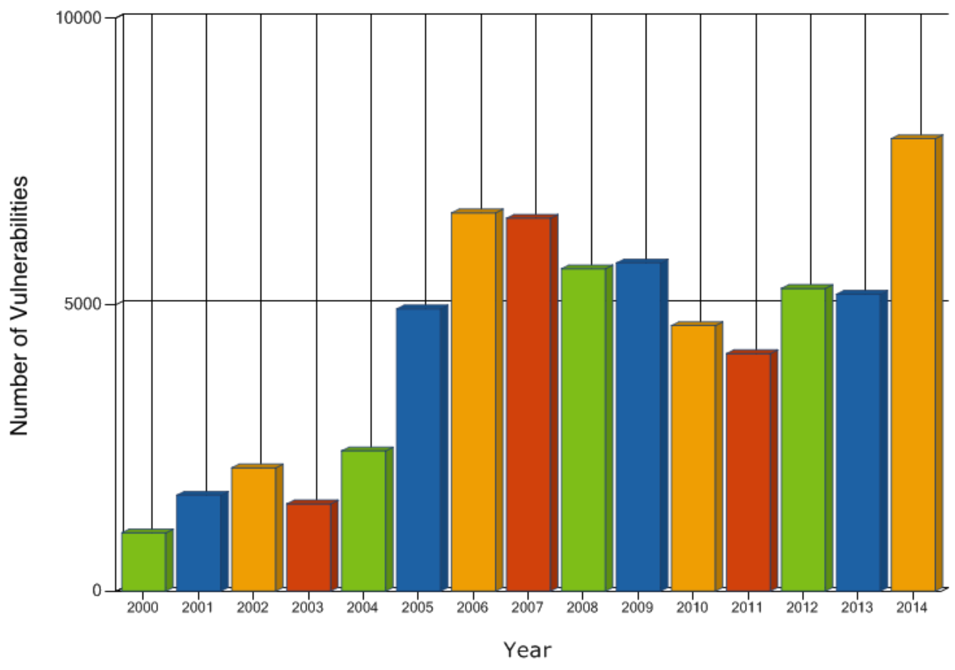


Figure 2.1: Number of Vulnerabilities Discovered in Each Year

3. **Information Quality**: What information (e.g. severity level, exploits, solutions, etc.) about a vulnerability is published?

4. **Parsable Feed**: Does the source provide a parsable feed that can be downloaded and updated automatically?

5. **Cost**: Is the source free for public use? If not, how much does it cost?

Table 2.1 contains 6 vulnerabilities published over the last few months. These vulnerabilities have been used as case studies to evaluate completeness and speed of different sources. These vulnerabilities were of great importance for CERN and can be a good indicator of how well a source fits CERN needs.

### 2.2.2 Vulnerability Sources

Plenty of mailing lists, databases, vulnerability sources, etc. are available online. Each of these sources provides different types of information for

| Vulnerability | CVE-ID | Disclosure Date |
|---|---|---|
| GitLab groups API | CVE-2014-8540 | 30.10.2015 |
| Wordpress 4.0.1 Security Release | CVE-2014-9032(*) | 20.11.2014 |
| Drupal SQL injection | CVE-2014-3704 | 15.10.2014 |
| Poodle | CVE-2014-3566 | 14.10.2014 |
| Twiki Remote Code Execution | CVE-2014-7236 | 09.10.2014 |
| ShellShock | CVE-2014-6271 | 24.09.2014 |

Table 2.1: Sample vulnerabilities
(*) Multiple vulnerabilities (CVE-2014-9032, CVE-2014-9033, CVE-2014-9034, CVE-2014-9035, CVE-2014-9036, CVE-2014-9037) were released at the same time.

different purposes. A part of this project was to do some research on these different sources and compare them with respect to CERN needs. In this section, we will go through a summary of the sources that were evaluated.

- **Natinal Vulnerability Database (NVD)**[1]: The U.S. government repository of standards-based vulnerability management data.
- **Open Sourced Vulnerability Database (OSVDB)**[2]: An independent and open sourced[3] web-based vulnerability database created for the security community.
- **CVE Details**[4]: A web interface providing CVE vulnerability data and statistics about vendors and products.
- **Secunia Vulnerability Intelligence Manager (VIM)**[5]: An online service providing a filtered feed of verified vulnerability intelligence in real-time.

**Completeness and Speed** Table 2.2 illustrates the publication date of sample vulnerabilities on individual sources. If a source is missing a particular vulnerability the corresponding cell is empty. Using this table, we can compare the completeness and speed of different sources. Note that in each row, the fastest source is marked in green, the slowest in red and the average ones are marked in yellow. Taking a glance at the table, we can realize that OSVDB has the highest publication speed and is the most complete source.

[1]`https://nvd.nist.gov/`
[2]`www.osvdb.or`
[3]The data is collected from publicly available sources
[4]`http://www.cvedetails.com/`
[5]`http://secunia.com/vulnerability_intelligence`

| Vulnerability(*) | **NVD** | **OSVDB** | **CVE Details** | **Secunia VIM** |
|---|---|---|---|---|
| **GitLab** | | 31.10.2014 | | |
| **Wordpress** | 25.10.2014 | 21.10.2014 | | 21.10.2014 |
| **Drupal** | 17.10.2014 | 17.10.2014 | 17.10.2014 | 16.10.2014 |
| **Poodle** | 16.10.2014 | 19.10.2014 | 16.10.2014 | 15.10.2014 |
| **Twiki** | | 09.10.2014 | 11.10.2014 | |
| **ShellShock** | 25.09.2014 | 25.09.2014 | 25.09.2014 | 25.09.2014 |

Table 2.2: Vulnerability Publication Dates
(*) For the complete name of the vulnerability refer to Table 2.1

| Data | **NVD** | **OSVDB** | **CVE Details** | **Secunia VIM** |
|---|---|---|---|---|
| **Description** | ✓ | ✓ | ✓ | ✓ |
| **Severity** | ✓ | ✓ | ✓ | ✓ |
| **Product** | ✓ | ✓ | ✓ | ✓ |
| **Exploits** | ✗ | ✗ | ✓ | ✗ |
| **Solution** | ✗ | ✓ | ✗ | ✓ |
| **Vulnerability Type** | ✓ | ✓ | ✓ | ✓ |

Table 2.3: Source Information Quality

**Information Quality** The comparison of the information quality in different sources is illustrated in Table 2.3. All sources have almost the same quality of the information. It is true that NVD provides less information than the other sources, but it is still providing most of the important fields that we are interested in.

**Parsable Feed and Cost** NVD provides free XML data feeds. OSVDB provides information via the web interface as a free resource for the community. Alternate methods of obtaining the data such as API or exports are no longer supported via OSVDB and are offered by Risk Based Security as a commercial product, called VulnDB API. CVE Details offers JSON feeds for the vulnerabilities, but unfortunately, the feeds are not complete and they do not contain vulnerable product information which is one of the most important fields for the purpose of VNT. Secunia VIM is also a commercial product[1] and it provides vulnerability information via a web portal. It is possible to download XML description of vulnerabilities limited to the last

[1]A trial account was used for evaluation.

72 hours; however, the main focus of VIM is on vulnerability management through the graphical web interface.

### 2.2.3 Conclusion

Table 2.4 illustrates a summary of the source evaluation results. Like the two previous tables, green cells show a high evaluated score, yellow represents an average score and red is marking an unacceptable score. Downloading the latest vulnerability information automatically is one of the requirements of VNT. In addition, VNT is supposed to analyze the vulnerability data to filter out the non-critical vulnerabilities and those unrelated to CERN. Therefore, it is important to have the vulnerability information in a structured and parsable format. The only sources that provide a useful parsable feed are NVD, OSVDB and Secunia VIM. Secunia VIM focuses on vulnerability management, assuming that there is a human using its portal and taking necessary actions. There is no easy way of connecting it to another tool, i.e. WAD, and it is missing an API access. Although an XML description of vulnerabilities from last 72 hours can be downloaded in a semi-automatic way, by using tools like wget, we decided not to go for Secunia VIM, because buying the whole product and using only a non-main functionality of it was not beneficial.

Table 2.4 shows clearly that OSVDB is faster, more complete and more informative than NVD. In spite of that, NVD was the source we finally used as the input to VNT because OSVDB is offering the API access and its feed commercially (under the name VulnDB API) at a price that was deemed prohibitive.

Although it has not been a decision factor, it is worth mentioning that vulnerability databases use different methods of abstraction. Some, like NVD, keep one entry per vulnerability, while some others, like Secunia VIM, create multiple entries for the same vulnerability, because there are different patches available (for different operating systems, for example).

## 2.3 National Vulnerability Database

NVD is the U.S. government repository of standards-based vulnerability management data and contains 68054 CVE vulnerabilities. It is recommended by the MITRE Corp. hosting CVE[1] to obtain information about a vulnerability

[1]https://cve.mitre.org/

| Data | **NVD** | **OSVDB** | **CVE Details** | **Secunia VIM** |
|---|---|---|---|---|
| **Completeness** | ✓ | ✓ | ✓ | ✓ |
| **Speed** | ✓ | ✓ | ✓ | ✓ |
| **Information Quality** | ✓ | ✓ | ✓ | ✓ |
| **Feed Completeness** | ✓ | VulnDB | ✗ | ✗ |
| **Feed Cost** | ✓ | VulnDB | ✓ | * |

Table 2.4: Source Evaluation
(*) Price was not even considered because of the incomplete feed

severity rating, fix information, vulnerable software versions, etc.

### 2.3.1 Data Feeds

The entire NVD can be downloaded from its web page for free and for public use. The XML vulnerability feed contains security related software flaws. Each vulnerability in the file includes a description and associated reference links from the CVE dictionary feed, as well as a CVSS base score[1], vulnerable product configuration, and weakness categorization. The feed provides vulnerability information since 1999 (one file for each year, except that the file from 2002 includes all vulnerabilities published in 2002 and before). In addition, there is a "recent" feed, listing recently published vulnerabilities and a "modified" feed, which includes the "recent" feed plus all recently modified vulnerabilities, where "recently" means the previous eight days. The feeds are updated approximately every two hours.

### 2.3.2 Challenges

#### Downloading

The "modified" feed from NVD contains the latest vulnerabilities published or modified in the last eight days. Obviously, if the tool is not used for more than eight days, it will definitely miss some updates. This issue can be fixed by looking at the yearly feeds and detecting if a vulnerability information has changed in those feeds since the last execution. Anyhow, it was decided that the NVD feed will be consulted at least once a day and therefore, there is no need for consulting the yearly feeds.

[1]Common Vulnerability Scoring System is a standard measurement system for rating the severity of IT vulnerabilities

Another point worth mentioning is that NVD keeps no record of the changes that happen in vulnerability data. It is impossible to see how a vulnerability has changed over time, because new data are always overwriting the old ones. VNT does not care about any intermediate changes in a vulnerability information between two subsequent executions, but for a complete evaluation of the tool and in order to be able to simulate the tool over a period of time, we kept a local copy of the “modified” feed, downloaded daily since October 31th, 2014.

### Modification Date

Since NVD provides the last modified date as one of the fields in the vulnerability information, the easiest way to find the vulnerabilities that have changed since the last execution of the tool, would be to compare their last modified time with the time of the last execution. However, this method will not work because the last modified time reported by NVD is not always correct. Imagine a vulnerability X published at time $t_0$. If we look at the NVD feed at time $t_2$ we see that the last modified time for the vulnerability is $t_0$, which means that the vulnerability has not changed since its publication. Now we may look at the vulnerability at time $t_3$ and to our surprise, we may see that the last modified time is $t_1$ ($t_0<t_1<t_2$). This means that although the vulnerability information was updated at time $t_1$, the update was not visible for sometime and we missed it at time $t_2$. If we compare the last modified time ($t_1$) to the last execution time ($t_2$) we might decide to ignore the vulnerability and assume it has not changed.

A script was written to check for existence of similar cases in NVD feeds and many instances of this controversy were found. For example, on 05.11.2014 the last modified time of CVE-2012-5500 was reported as 03.11.2014 and if we check for the same vulnerability on 06.11.2014 we get the last modified time as 04.11.2014. In addition, one can observe that sometimes the details of a vulnerability, such as its CVSS score change without changing the last modified field. Due to these problems, it was decided to ignore the last modified time field completely and use the following alternative solution to find updated vulnerabilities:

The tool stores a local copy of vulnerability information (one JSON file per vulnerability). For each vulnerability in the downloaded feed we look at our local version of that vulnerability. If there is no local version, the vulnerability is reported as newly published and its data is stored locally, otherwise we compare the local version with the new data, report the changes and update the stored data.

This solution solves the modification date problem and makes it possible to report what exactly has changed in a vulnerability.

### Vulnerable Configuration

In addition to the list of vulnerable software packages, NVD provides a list of vulnerable configurations. For example, if Safari 5.0.5 is vulnerable only on Mac OS X 10.5.8 the vulnerable-software-list would contain Safari 5.0.5 but no information about the operating system. In this case, it would be useful to extract the configuration data from NVD and use it to make sure when a resource is vulnerable. Given the scope of this project, unavailability of configuration specifications on CERN resources and the fact that only a few vulnerabilities are bound to a specific configuration, we decided to ignore this field and only consider the products from the vulnerable-software-list field.

## 2.4 Product Name Matching

The next step after extracting new and modified vulnerabilities is to find the CERN resources that might be affected by these vulnerabilities. As an intermediate step, it is important to find which WAD names[1] are affected by each vulnerability. If we know about the affected WAD names, finding the affected resources is an easy task that just requires going through WAD output on all websites and web servers at CERN and reporting the URLs of all those including the affected WAD names.

### Common Platform Enumerations

NVD is using the Common Platform Enumeration syntax to list the vulnerable softwares for each vulnerability. The Common Platform Enumeration (CPE) is a structured naming schema for IT systems, platforms and packages. It aims at providing a formal, consistent and uniform naming schema, so that the community members are able to generate names for new IT platforms in a consistent and predictable way.[2] This will facilitate automation in security practices. CPE is based on the generic syntax for

[1]Names that WAD uses to report detected technologies, such as Apache, Microsoft SharePoint, PHP-Fusion, etc.

[2]`https://nvd.nist.gov/cpe.cfm`

| CPE | Description |
|---|---|
| `cpe:/h:samsung:galaxy_note_2:-` | Samsung Galaxy Note 2 (Hardware) |
| `cpe:/o:microsoft:windows_xp:home` | Microsoft Windows XP Home Edition Operating System |
| `cpe:/a:wordpress:wordpress:1.0` | Wordpress 1.0 |

Table 2.5: CPE Examples

Uniform Resource Identifiers. A standard CPE name is in the format of `cpe:/{type}:{vendor}:{name}:{version}`. The type of a platform can be either hardware (h), operating system (o), or application environment (a). Some CPEs can contain more components to provide additional information. Table 2.5 contains some examples of CPE names.

In this project we only care about vulnerabilities that affect operating systems or applications, therefore, we can ignore the hardware related vulnerabilities when mapping CPE to WAD names. As you can see in the examples, the first three components of the CPE (after the type indicator) are the ones that describe the vulnerable software in the format of `{vendor}:{name}:{version}`.

**WAD Product Names**

As already described in section 1.5, WAD uses detection rules coming from Wappalyzer which is an open source browser plugin with a community of over 100 contributors. Wappalyzer relies on its contributors' common sense to choose a meaningful name for each technology it detects and therefore there is no standard syntax for Wappalyzer names. As of the time of writing this thesis, Wappalyzer detects 707 technologies from 50 different categories.

### 2.4.1 Approach

There are two approaches that we can follow to do the mapping from CPE names to WAD names:

1. **Static Mapping**: In static mapping we would need to create a dictionary that lists all WAD names for each CPE name. Generating this

dictionary for the first time would be a time consuming manual work, but would guarantee a high level of accuracy. This dictionary needs to be updated whenever there is a new WAD name or CPE name.

2. **Dynamic Mapping**: In the dynamic mapping approach we would need to design an algorithm that would find WAD names for a CPE name on the fly. By using this approach, there would be no need for any maintenance, except optionally improving the matching algorithm over time. But the price we pay with dynamic mapping is lower accuracy, i.e more false positives (wrongly matching a CPE name to a WAD name) and false negatives (missing a match between a CPE name and a misleading but actually corresponding WAD name).

**Challenges**

The lack of a standard naming approach in Wappalyzer and consequently in WAD, makes it difficult to match CPE names to WAD names. On the other hand, CPE itself is not 100% consistent and sometimes for the same application, one can find multiple CPE names,e.g. `cpe:/a:django_piston_project:django_piston:0.2.2.0`, `cpe:/a:djangoproject:django_piston:0.2.2.0` and `cpe:/a:djangoproject:piston:0.2.2.0` are available CPE names that refer to the same application.

There is an official CPE dictionary available online that is supposed to provide an agreed upon list of official CPE names[1]. Unfortunately, one can notice that many of the CPE names that appear on NVD vulnerability feeds are not present in this dictionary and therefore there is no way of knowing which CPEs we are going to find in vulnerability feeds, in advance. Considering the CPE dictionary as well as all the vulnerabilities published on NVD, currently there are 33,615 unique CPE names (vendor and product names, ignoring other components such as version). Table 2.6 tries to show how challenging it might be to find equivalent WAD names of a CPE name.

### 2.4.2 Algorithm

Considering the challenges we discussed in the previous section and the likelihood of new CPE names appearing, as well as the overload of maintaining a dictionary of the mappings, it was decided to go for a dynamic mapping approach.

[1]`https://nvd.nist.gov/cpe.cfm`

| CPE | WAD Name |
|---|---|
| `cpe:/a:adobe:coldfusion:8.0` | Adobe ColdFusion |
| `cpe:/a:yandex.metrics_project:yandex_metrics:1.0` | Yandex.Metrika |
| `cpe:/a:woothemes:woocommerce_plugin:2.1.0` | WooCommerce |
| `cpe:/a:djangoproject:django:1.6` | Django CMS |
| `cpe:/a:cagintranetworks:getsimple_cms:1.0` | GetSimple CMS |
| `cpe:/a:drupal:drupal:4.6.2` | Drupal |

Table 2.6: CPE to WAD Name Mapping

Coming up with a matching algorithm that is complete and accurate at the same time is not an easy task. The more complete the matching is (higher recall), the less accurate it is going to be (lower precision). After analyzing the list of CPE names and WAD names and trying different matching algorithms, two algorithms were designed. Algorithm 1 is an algorithm that matches a CPE name to a WAD name with a high level of accuracy. A match is found only if the WAD name is the combination of CPE product and vendor names or it equals the CPE product name. Algorithm 2 adds more matches, but has a lower level of accuracy. The user can choose to use only algorithm 1 for a higher precision or both algorithms together to get a higher recall rate. Note that both algorithms receive the CPE and WAD names in lower case with leading and trailing white spaces removed. In the first algorithm these names are being normalized before they are used, which means that all punctuation marks and white spaces inside the names are removed.

**Algorithm 1** Name Matching Algorithm

```
$matches \leftarrow$ array()
$cpe\_name \leftarrow$ normalize($cpe\_vendor$)+normalize($cpe\_product$)
for all $wad\_name \in wad\_names$ do
    if normalize($wad\_name$)= $cpe\_name$ then
        $matches += wad\_name$
    else if normalize($wad\_name$)=normalize($cpe\_product$) then
        $matches += wad\_name$
    end if
end for
return $matches$
```

**Algorithm 2** Name Matching Algorithm

```
matches ← array()
for all $wad\_name \in wad\_names$ do
  if normalize($wad\_name$)=normalize($cpe\_vendor$) then
    $matches += wad\_name$
  else
    $cpe\_name \leftarrow cpe\_vendor + cpe\_product$
    $found \leftarrow$ True
    for all $wad\_word \in wad\_name$ do
      if $wad\_word$ is not a word in $cpe\_name$ then
        $found \leftarrow$ False
      end if
    end for
    if $found = True$ then
      $matches += wad\_name$
    end if
  end if
end for
```

| CPE | WAD Name |
|---|---|
| `cpe:/a:20_20_applications:20_20_auto_gallery` | Gallery |
| `cpe:/a:altiris:dell_client_manager_solution` | Dell |
| `cpe:/a:redhat:cygwin` | Red Hat |

Table 2.7: False Matches

### 2.4.3 Evaluation

#### False Positives

Very generic WAD names result into false positives. Table 2.7 shows some examples of the incorrect matches due to the generic WAD names.

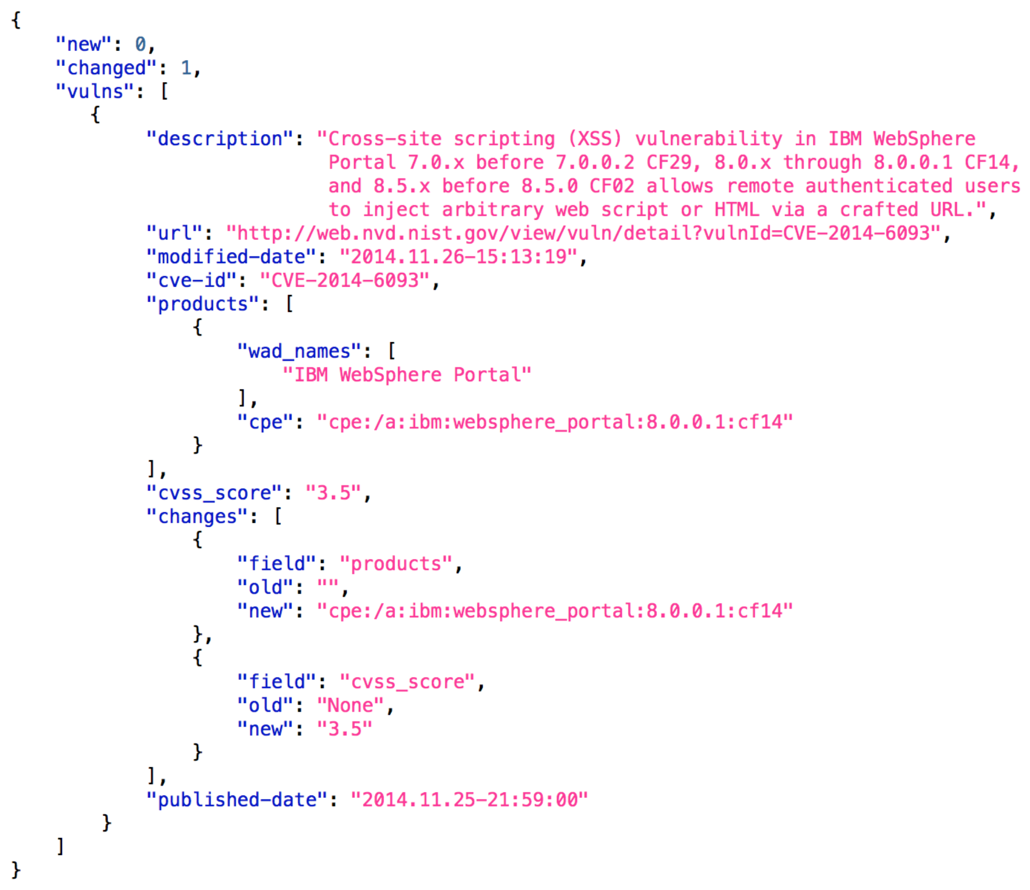
#### False Negatives

Finding false negatives, or in other words, missing matches, is more difficult than finding false positives because it involves going through both lists of WAD names and CPE names and finding matches manually to compare them with the algorithm output. Our algorithm might miss some mappings because of the slight differences in product names, for example `cpe:/a:yandex.metrics_project:yandex_metrics:1.0` will not be matched with *Yandex.Metrika* or `cpe:/a:microsoft:outlook_web_access` will not match with *Outlook Web App*. A possible improvement would be to use string distance algorithms to find similar names; however, that would make the algorithm much more complex.

Another cause of false negatives is inconsistency in word boundaries. Imagine a CPE name in the format of `cpe:/a:adobe:adobe_coldfusion`; it will not match with *Cold Fusion*, because Algorithm 2 looks for the WAD name in CPE product name preserving the word boundaries. In this case, the algorithm could check for WAD names that are substrings of CPE names, but that approach would lead to many more false positives.

## 2.5 Vulnerable Resources

Now that we have a reasonably good idea of which WAD names are affected by a vulnerability, obtaining the list of vulnerable resources is quite an easy task, because the data is already available in WAD output. The Vulnerability Notification Tool only needs to load the output of WAD, group resources by the technologies they use and for each vulnerability report the resources that use the affected WAD name.

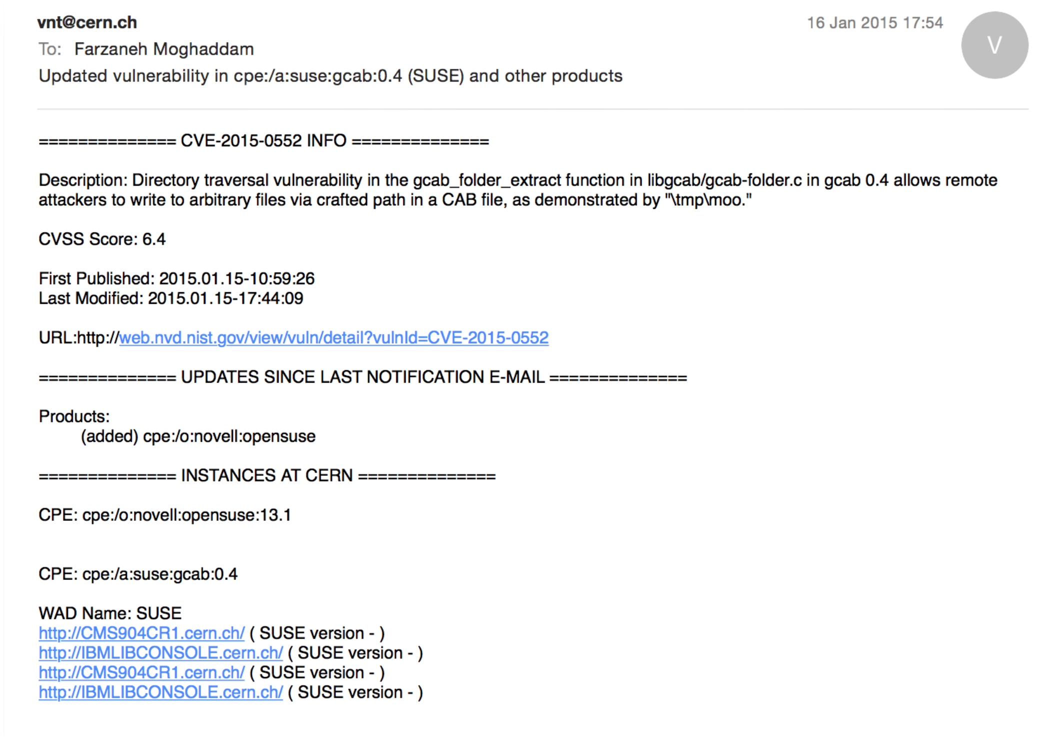


```
{
    "new": 0,
    "changed": 1,
    "vulns": [
       {
            "description": "Cross-site scripting (XSS) vulnerability in IBM WebSphere
                            Portal 7.0.x before 7.0.0.2 CF29, 8.0.x through 8.0.0.1 CF14,
                            and 8.5.x before 8.5.0 CF02 allows remote authenticated users
                            to inject arbitrary web script or HTML via a crafted URL.",
            "url": "http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6093",
            "modified-date": "2014.11.26-15:13:19",
            "cve-id": "CVE-2014-6093",
            "products": [
                {
                    "wad_names": [
                        "IBM WebSphere Portal"
                    ],
                    "cpe": "cpe:/a:ibm:websphere_portal:8.0.0.1:cf14"
                }
            ],
            "cvss_score": "3.5",
            "changes": [
                {
                    "field": "products",
                    "old": "",
                    "new": "cpe:/a:ibm:websphere_portal:8.0.0.1:cf14"
                },
                {
                    "field": "cvss_score",
                    "old": "None",
                    "new": "3.5"
                }
            ],
            "published-date": "2014.11.25-21:59:00"
        }
    ]
}
```

Figure 2.2: Sample JSON output

## 2.6 Output and Notifications

VNT stores the results of its findings in JSON format. Figure 2.2 illustrates a sample of the stored JSON file. These files contain vulnerability information, updates and affected WAD names. They are easy to parse and can be used later for more analysis about vulnerabilities. In addition, the tool sends email notifications to the CERN Computer Security Team members whenever there is a vulnerability that affects CERN and has a higher CVSS score than 6.0. Figure 2.3 shows one of the emails sent by the tool.



**vnt@cern.ch**
To: Farzaneh Moghaddam
Updated vulnerability in cpe:/a:suse:gcab:0.4 (SUSE) and other products

16 Jan 2015 17:54

```
============== CVE-2015-0552 INFO ==============

Description: Directory traversal vulnerability in the gcab_folder_extract function in libgcab/gcab-folder.c in gcab 0.4 allows remote
attackers to write to arbitrary files via crafted path in a CAB file, as demonstrated by "\tmp\moo."

CVSS Score: 6.4

First Published: 2015.01.15-10:59:26
Last Modified: 2015.01.15-17:44:09

URL:http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-0552

============== UPDATES SINCE LAST NOTIFICATION E-MAIL ==============

Products:
        (added) cpe:/o:novell:opensuse

============== INSTANCES AT CERN ==============

CPE: cpe:/o:novell:opensuse:13.1


CPE: cpe:/a:suse:gcab:0.4

WAD Name: SUSE
http://CMS904CR1.cern.ch/ ( SUSE version - )
http://IBMLIBCONSOLE.cern.ch/ ( SUSE version - )
http://CMS904CR1.cern.ch/ ( SUSE version - )
http://IBMLIBCONSOLE.cern.ch/ ( SUSE version - )
```

Figure 2.3: Sample Notification Email

## 2.7 Results

The Vulnerability Notification Tool is a tool that downloads the latest “modified” feed from NVD, finds the vulnerabilities that are newly published or have been changed since its last execution and for each of these vulnerabilities reports CVE-ID, description, CVSS score, published date and time, last modified date and time, and a list of affected software (vulnerable software in CPE format). In addition, the tool lists the URLs of the CERN websites and web servers that are likely to suffer from this vulnerability. Figure 2.4 summarizes all functionalities of VNT and illustrates its architecture.

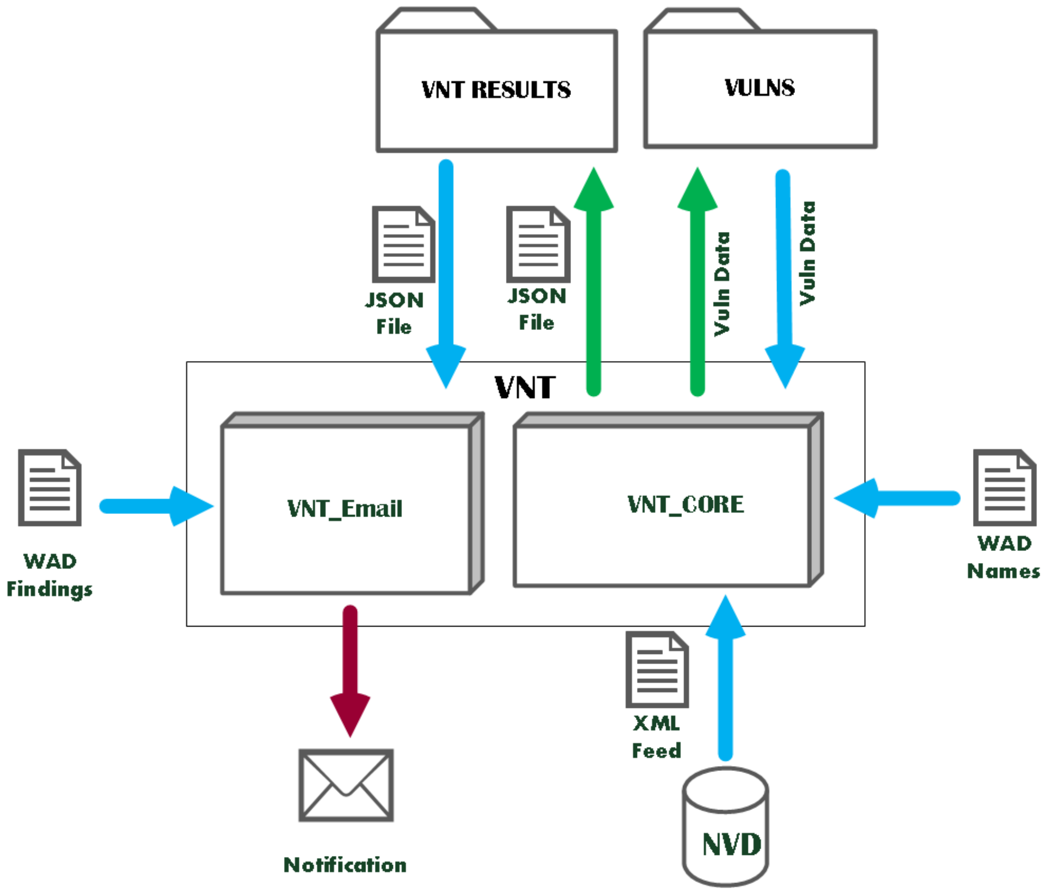


Figure 2.4: VNT Architecture

As mentioned before, there is no archive of NVD modified feeds online; however, we had downloaded and stored NVD modified feeds since 31.10.2014 so we could use this data to reconstruct by simulation how the tool would have performed over the period of time from 01.11.2014 to 31.12.2014 (two months). Table 2.8 shows the results of this simulation. The feed from 31.10.2014 has been used for the first execution of the tool and initializing the local copy of vulnerabilities. Appendix A describes the methods used to get these numbers from the VNT output.

| | New Vulnerabilities | | Changes in Vulnerabilities | |
|---|---|---|---|---|
| | All | At CERN | All | At CERN |
| All | 1098 | 31 | 2131 | 238 |
| Critical* | 190 | 9 | 622 | 95 |

Table 2.8: VNT Results from 01.11.2014 to 31.12.2014

(*) Vulnerabilities with a CVSS score equal to or greater than 6.0 are considered critical

**Number of Notification Emails** Knowing about all the published vulnerabilities regardless of whether that they affect CERN could be important for the CERN Computer Security Team and that is why the JSON output of the tool stores all the vulnerabilities it finds regardless of their severity level or impact on CERN. On the other hand, if we sent an email for every new vulnerability or a change in vulnerabilities we would be flooding users with an average of (1098+2131)/60 = 54 emails per day. Based on the findings illustrated in Table 2.8, we decided to send emails for vulnerabilities that affect CERN **and** have a CVSS score equal to or greater than 6.0. In other words, the CERN Computer Security Team will receive an email whenever a new critical vulnerability is published or there is a change in a critical vulnerability that affects CERN. Using this strategy, we would be expecting an average of (9+95)/60= 2 emails per day.

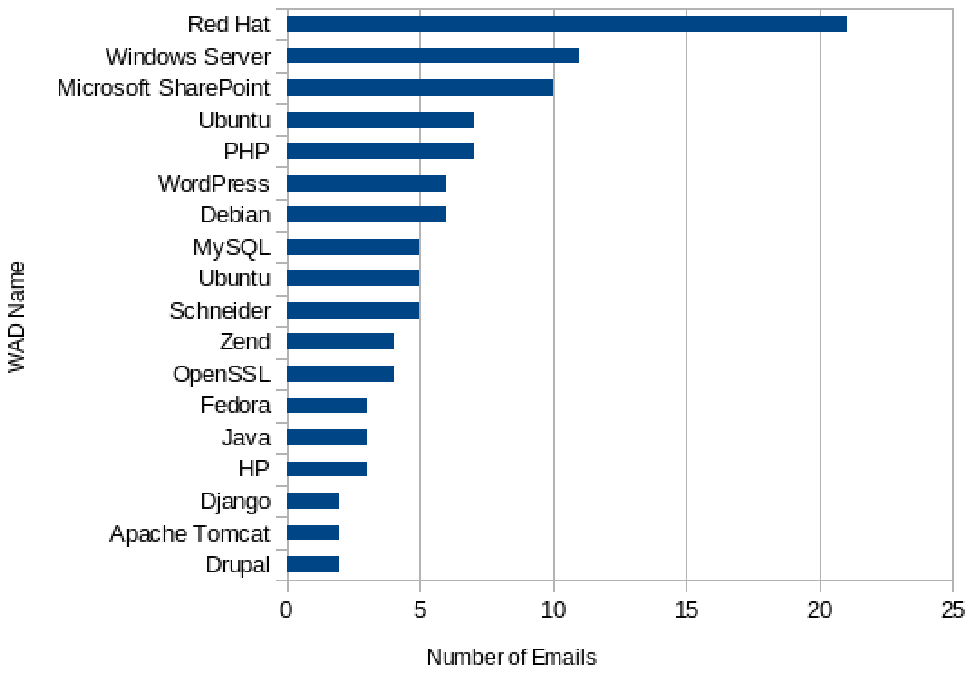


Figure 2.5: Number of Email Alerts Sent for Each WAD Name in a Two-month Simulation

**Accuracy** Now we are going to have a closer look at the critical vulnerabilities that affected CERN (the green cells from table 2.8) to evaluate the tool and we will go over some statistical data obtained from the tool. Figure 2.5 shows the number of emails that would be sent per each WAD name. This chart contains only the WAD names that occur more than once. From this figure one can get an idea of the most vulnerable technologies used at CERN. But we should keep in mind that the accuracy of our name matching algorithm has a direct impact on the number of emails we send for each WAD name. For example, if we look at all the vulnerabilities that have been matched to the WAD name 'Red Hat', we realize that among the 21 emails that are sent, only 12 of them are in fact referring to the Red Hat operating system and the other 9 are a consequence of false positives in our matching algorithm. Figure 2.6 shows the number of true positive and false positive emails per WAD name and can be a better indicator of the most vulnerable products at CERN. For example, although more emails were sent about Debian, it does not mean that Debian has been more vulnerable than OpenSSL.

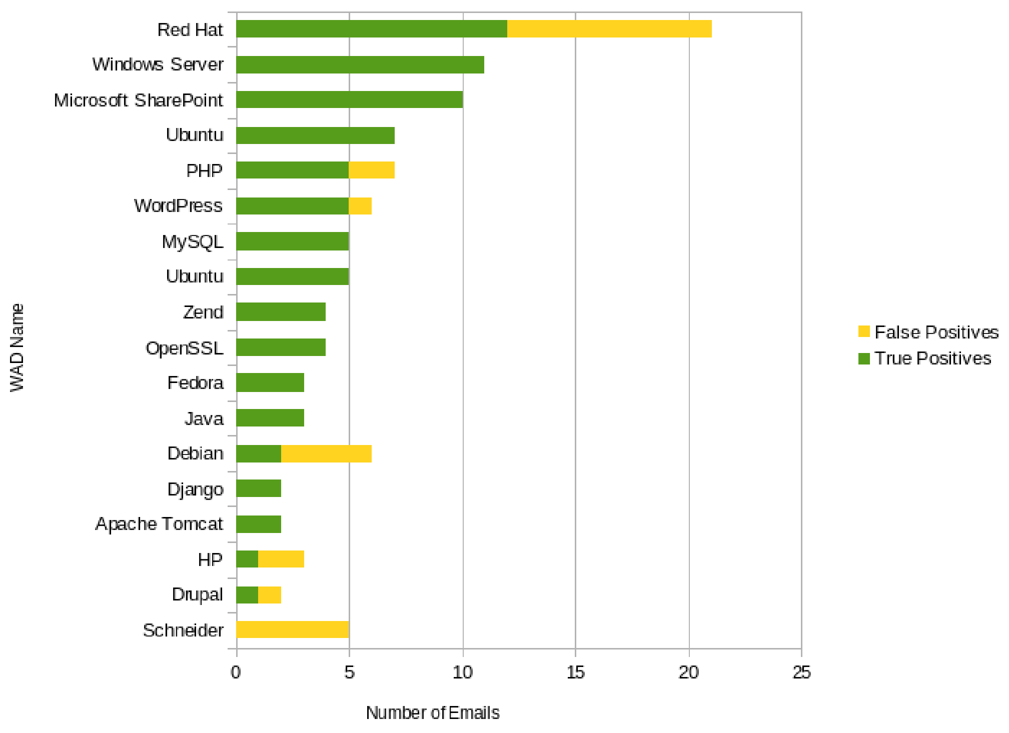


Figure 2.6: Number of Email Alerts Sent for each WAD Name in a Two-month Simulation

**Name Matching Accuracy** Figure 2.7 shows that in 23% of the cases the matching from the CPE to a WAD name has been wrong (too generic WAD names like 'Red Hat'). The accuracy of the matches has been checked manually to get the data for this chart.

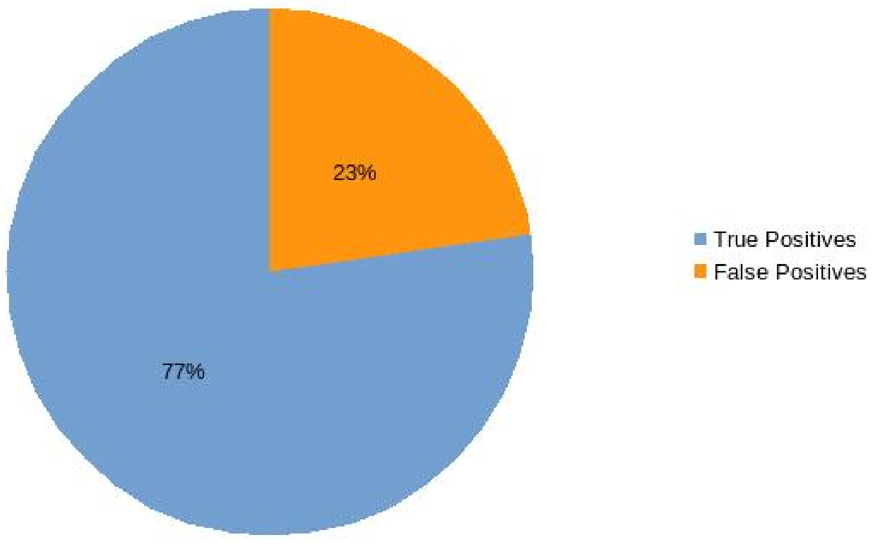


Figure 2.7: False Positive Rate in Name Matching

Table 2.8 shows that in 90% of the cases the email contains updated vulnerability information. Looking at the updated fields, we can see that most of the times the affected products (CPEs) are the fields that change. Figure 2.8 shows the frequency of changes in different fields. Further investigation shows that the CVSS Score field has always changed from “None” (no information available) to a value. This is expected as it usually takes some time (one or two days) until a CVSS score is available and once it is available it is highly reliable and it does not change. This does not hold for CPE name changes and in 20% of the cases the CPE field changes from a non empty value.

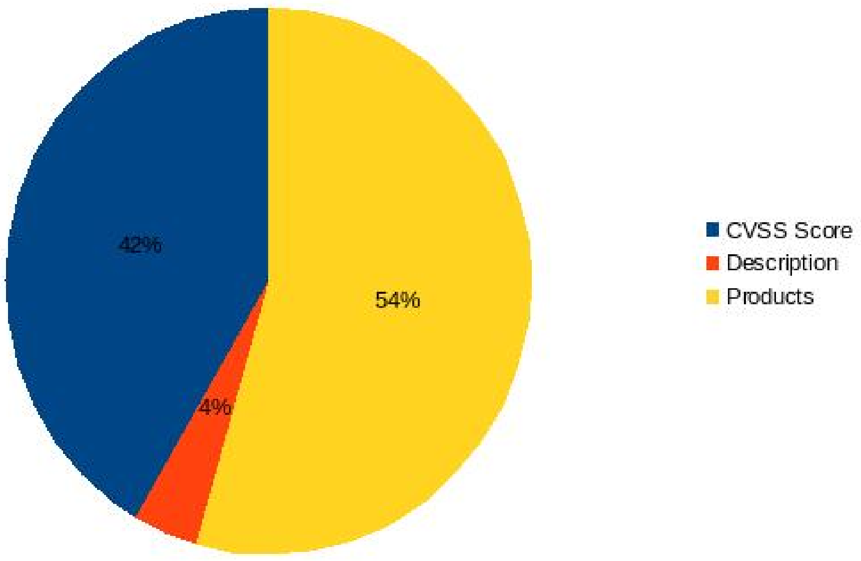


Figure 2.8: Updated Fields in Vulnerabilities

# Chapter 3

# Scanner

## 3.1 Motivation

Chapter 2 introduced the Vulnerability Notification Tool that is designed to identify CERN resources that might be affected by vulnerabilities associated with specific software components, however, sometimes we are not able to accurately detect software components on CERN resources (due to WAD limitations) or some vulnerabilities are not specific to any component and spotting them takes targeted scanning, which is the topic of this chapter.

The CERN Computer Security Team performs scans of various types of resources, such as devices, web servers, web sites, etc. It is impossible to scan these resources manually and there is a need for a scanning engine to facilitate scheduling and running these scans, share the load across multiple scanning hosts in a fault-tolerant way, and combine the results.

“GNU Parallel” is a command-line utility for Unix-like operating systems that allows executing jobs concurrently locally or on remote computers. A job is typically a single command or a small script that has to be run for each of the lines in the input[1]. It is trivial to provide GNU Parallel with a list of CERN resources and have the scans run on these resources concurrently.

On the other hand, as discussed in Section 1.5, CERN has a collection of scripts for detecting misconfigurations, such as expired certificates and basic HTTP authentication, or vulnerabilities, such as Heartbleed.

[1]`http://savannah.gnu.org/projects/parallel`

There is a need for a tool that fills the gap between detection scripts and GNU Parallel to enable the continuous and automatic scanning of resources. This tool (the Scanner) is meant to act like as a wrapper around detection scripts, standardizing the input and output format of the scripts, so that parallel runs of it can be triggered by GNU Parallel and their results can be analyzed automatically. The Scanner should also make it possible to run a subset of available detection scripts on targets. Another objective of the Scanner is to make it as simple and fast as possible to add a new detection

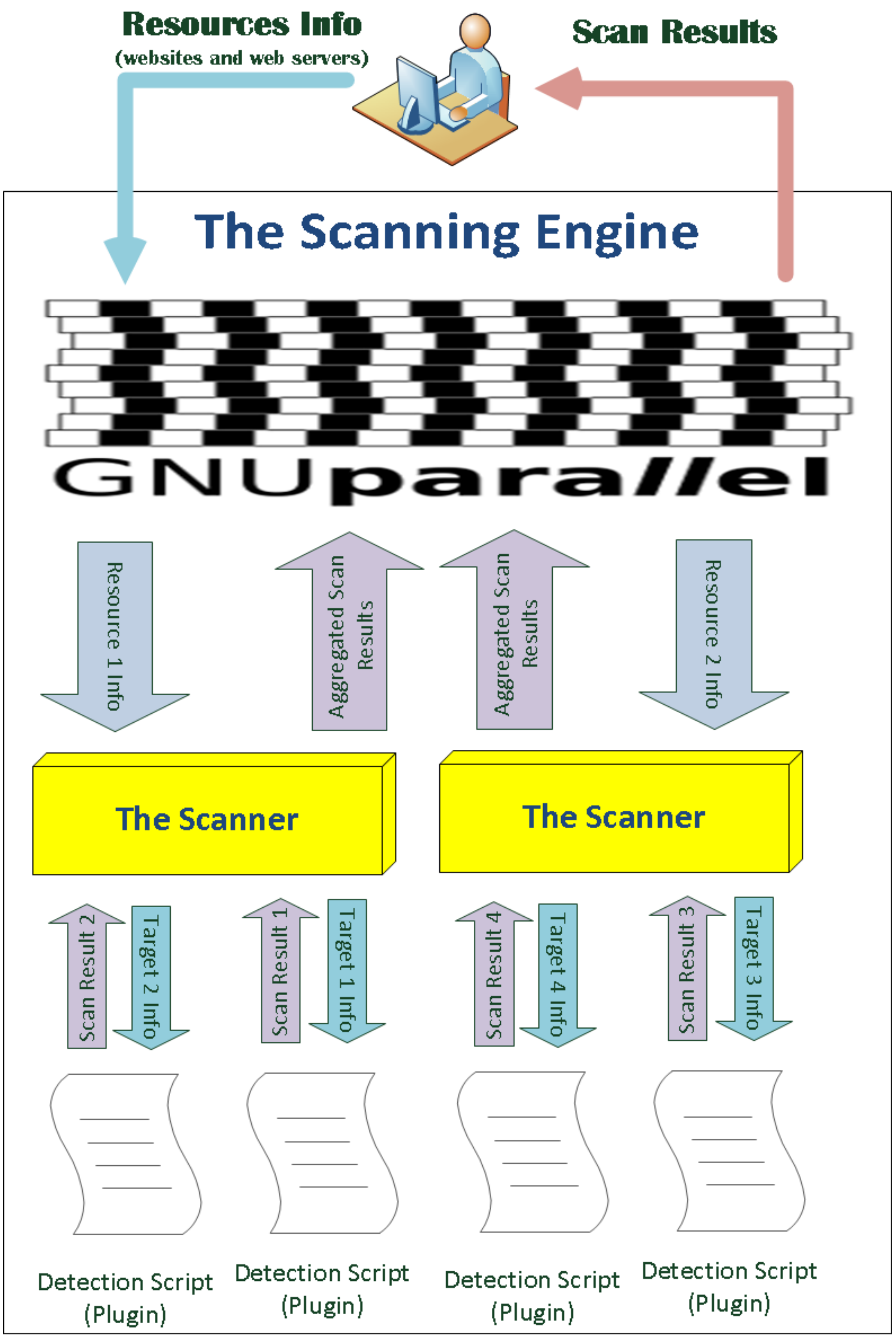


Figure 3.1: Scanning Engine Architecture

script, and run it on all CERN resources to ensure an acceptable detection/response speed when new vulnerabilities emerge. Figure 3.1 illustrates the relation between the detection scripts, the Scanner and GNU Parallel.

### 3.1.1 Use Cases

There are two major use cases for the Scanner:

- **Manual execution**: When we need to run an existing set of detection scripts on a new target (or existing targets, to get fresh results), or when we need to create a new script (e.g. Heartbleed test, when OpenSSL vulnerability surfaced) on the usual targets.
- **Automatic execution**: For continuous scanning of stable target lists (e.g. all official web sites, all web servers exposed on the firewall, etc.) with relevant test sets.

## 3.2 Scanner Specifications

The Scanner runs a set of security tests (called plugins) on a single resource, and collects, combines and delivers plugins scan results. When calling the Scanner, we ask it to scan a given resource type with a unique name. The Scanner ensures that only plugins for that resource type will be executed.

### 3.2.1 Plugin Specifications

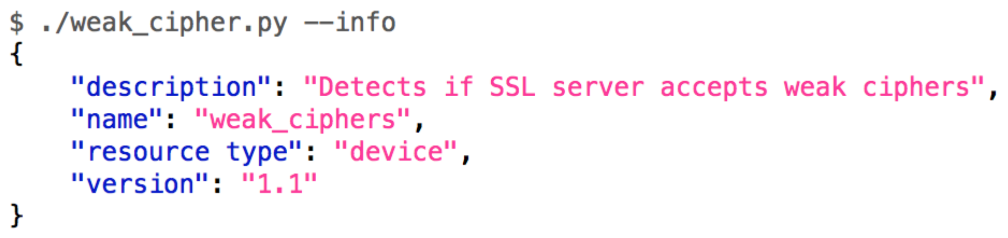
A plugin is a single security test for a given type of resource. After scanning a target, it says if a security problem was found or not (it can output more details when relevant). Detection scripts described in section 1.5 are examples of plugins.

#### Resource Types

There are various types of resources at CERN. In most cases the plugins need to scan websites or web servers (or other devices), but our implementation of the Scanner is independent of the type of the resource that is being scanned. If a plugin scans more than one resource type, depending on the type of the resource, it will require different inputs and consequently it needs to be executed differently. The Scanner, on the other hand, is the tool that executes the plugins. Therefore, it needs to be configured to execute the same plugin in different ways depending on the resource type. To avoid complexity, a given plugin should deal with only one type of resource. Occasionally, the same test will need to be developed for different types of resources (e.g. the default landing page test is needed for both web sites and web servers). In that case, the logic of the test should be shared, but still separate scripts should be developed for each resource type. For example, imagine that the script `landing_page.py` contains the logic for testing for a default landing page on websites and web servers. A possible implementation is to develop one script for websites (`landing_page_site.sh`) that runs '`./landing_page.py -site $1`' and one script for web servers (`landing_page_server.sh`) that runs '`./landing_page.py -server $1`'

### Input

The Scanner provides the plugin with inputs it should process. Depending on the nature of the security test a plugin is performing, it needs different input data. It is important that all plugins follow a convention about the way they receive inputs, so that the Scanner can run all the plugins in the same way. A plugin can assume that it is going to receive the input in the following order: {`IP/HostName PORT ALIAS URL`} for web servers/devices and {`URL`} for websites. The plugin can decide to ignore any of these input arguments, if irrelevant. Also, the Scanner might pass an empty string if the user does not specify a field. Additionally, plugins can be called with the following arguments instead of the target resource: `--info` - returns plugin details in JSON format (see Figure 3.2)



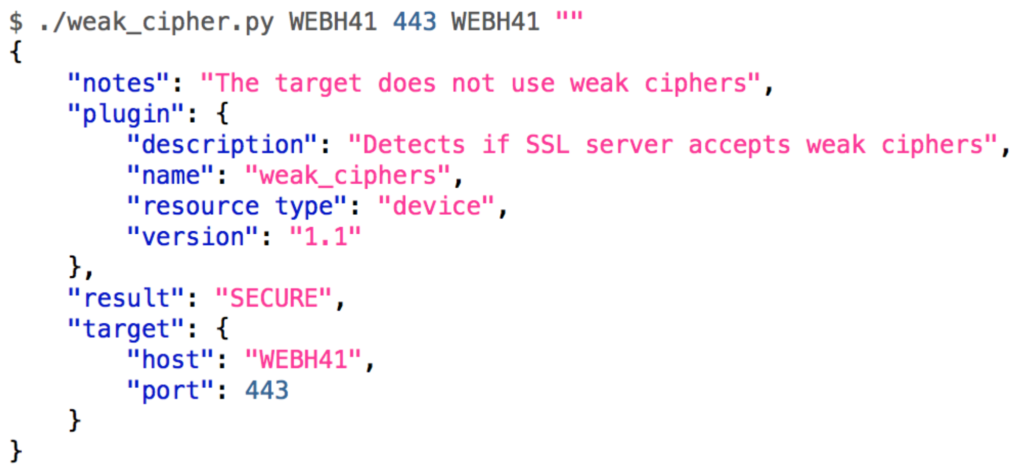
```
$ ./weak_cipher.py --info
{
    "description": "Detects if SSL server accepts weak ciphers",
    "name": "weak_ciphers",
    "resource type": "device",
    "version": "1.1"
}
```

Figure 3.2: Plugin Information

### Output

It is important that the plugins follow a convention in their output format, so that the Scanner can analyze the results and group them, if necessary. Figure 3.3 illustrates a sample plugin output. After scanning a given target, each plugin should return the following information in JSON format (mandatory fields in bold):

- **result** (mandatory): severity of the findings; one of these values (any other value in the result will be accepted by the Scanner, but a warning will be logged):
  - *SECURE* - no security problem was found.
  - *VULNERABLE* - security problem detected.
  - *WARNING* - there is something sub-optimal (e.g. a certificate expiring soon) but not a real security problem, yet.
  - *TIMEOUT* - results not available, test timed out (the plugins do not have to recover from timeouts as this can be done in the Scanner level).
  - *UNKNOWN* - results not available, test couldn't conclude (e.g. host unreachable).
- **notes** (mandatory): human-readable details of the scan result (e.g. which weak ciphers were detected, which URLs are affected).
- target (recommended): what was scanned: If the target information is provided by the plugin it can be used to show more precise results, for example if a default port has been used, etc. The plugin can choose to report all or a subset of following:
  - host
  - port
  - alias
  - url
- plugin info (recommended):
  - name
  - version
  - resource type
  - description - a one-line description of the plugin.



```
$ ./weak_cipher.py WEBH41 443 WEBH41 ""
{
    "notes": "The target does not use weak ciphers",
    "plugin": {
        "description": "Detects if SSL server accepts weak ciphers",
        "name": "weak_ciphers",
        "resource type": "device",
        "version": "1.1"
    },
    "result": "SECURE",
    "target": {
        "host": "WEBH41",
        "port": 443
    }
}
```

Figure 3.3: Plugin Output

### 3.2.2 Scanner Options

The Scanner will be directly used by the CERN Security Team members; therefore, it is important to design a robust user interface for it. The user can use the following options to customize Scanner parameters:

- `-l|--list TYPE` - list all plugins available in the plugin directory (for the given resource type(or all)).
- `-q|--quiet` - be quiet (only output the scan result and no other information regarding the progress of the scan).
- `-v|--verbose LEVEL` - be more verbose. (to the given verbosity level in the range of 1 to 5, with 5 being the most verbose).
- `-t|--timeout SECONDS` - set timeout per plugin (same for all plugins).
- `-plugin PLUGIN1.sh,PLUGIN2.py,PLUGIN3` - run selected plugins.
- `--plugin-list PLUGINS-FILE` - run plugins listed in the file (one line per plugin executable).
- `--plugin-dir DIRECTORY` - look for plugins in this directory.
- `--log FILE` - log output (but not results) to a file instead on printing on standard output.

And the following options can be used to define the target resource:

- `--device NAME` - scan the device (mandatory for devices).
- `--website NAME` - scan the centrally hosted website (mandatory for websites).
- `--ip IPs` - set device target IP addresses (comma separated values).
- `--port PORTs` - set device target port numbers (comma separated values).
- `--alias ALIASes` - set device target aliases (comma separated values).
- `--url URLs` - set device/website target URLs (comma separated values). For devices a url means more like a directory on the device.

### 3.2.3 Targets

A CERN resource can be composed of multiple targets. For example, a device can have multiple IP addresses, aliases, open ports or URLs and each combination of these fields can specify a scanning target. A plugin deals with only one target (a single IP, alias, port, etc.) and it is the job of the Scanner to loop over all combinations of IP addresses, aliases, ports and URLs when calling plugins .

#### Example

Consider the following command:

```
$ ./scanner.py  --plugin basic_auth.py
                --device PCITDI86
                --ip 137.138.43.51,2001:1458::112:33
                --port 80,443
                --alias CSC,PHOTOCLUB
                --url /admin,/tmp
```

With this command the Scanner will run the Basic Authentication plugin (which checks if the target is doing the authentication over HTTP instead of HTTPS) on a resource of type ‘device’ and with the name ‘PCITDI86’. In addition, there are multiple IP addresses, aliases, ports and URLs that we

| **IP/Hostname** | **Port** | **Alias** | **URL** |
|---|---|---|---|
| 137.138.43.51 | 80 | PCITDI86 | /admin |
| | | | /tmp |
| | | CSC | /admin |
| | | | /tmp |
| | | PHOTOCLUB | /admin |
| | | | /tmp |
| | 443 | PCITDI86 | /admin |
| | | | /tmp |
| | | CSC | /admin |
| | | | /tmp |
| | | PHOTOCLUB | /admin |
| | | | /tmp |
| 2001:1458::112:33 | 80 | PCITDI86 | /admin |
| | | | /tmp |
| | | CSC | /admin |
| | | | /tmp |
| | | PHOTOCLUB | /admin |
| | | | /tmp |
| | 443 | PCITDI86 | /admin |
| | | | /tmp |
| | | CSC | /admin |
| | | | /tmp |
| | | PHOTOCLUB | /admin |
| | | | /tmp |

Table 3.1: Targets

would like to scan. The Scanner is going to run the plugin 2*3*2*2=24 times on PCITDI86. Table 3.1 shows the different targets that will be scanned.

For many plugins (e.g. Heartbleed) the URLs are irrelevant, while for others (e.g. basic_authentication) the possibility to scan several URLs/subdirectories for a given host name is very useful. Obviously, we want to avoid that the Scanner loops over multiple URLs when scanning for Heartbleed. This means that some plugins (e.g. Heartbleed and basic-authentication) should not be used together for the same target details, but rather separately:

```
$ ./scanner.py  --plugin-list https-plugins.txt
                --device PCITDI86
                --alias CSC,PHOTOCLUB
                --port 443,8123

$ ./scanner.py  --plugin basic-auth.py
                --device PCITDI86
                --alias CSC,PHOTOCLUB
                --port 80,443,8123
                --url /admin,/tmp
```

If, incidentally or pressed by an emergency, we call the Scanner and provide "too many" details for plugins that do not need them, things should still work. We just risk waiting for the results a bit longer, and getting duplicated results.

### 3.2.4 Output

The Scanner prints exactly one line per resource and vulnerability, in the form of semicolon-separated values:

```
resource_type;resource_name;plugin_name;result;notes;targets
```

Targets from the same resource with the same scanning results (including the notes) will be grouped into one line. This gives the plugin designer the opportunity to decide on grouping. For example, if the plugin designer includes the port number in the notes, the targets with the same results but different port numbers will be reported in two different lines. This is useful, because the grouping very much depends on the nature of the test the plugin is running. For example, in the case of Heartbleed, the plugin designer would prefer to include the IP address and port number in the notes, but ignore the URLs. On the other hand, the landing_page plugin needs to report the complete target fields, including URLs, in the notes.

**Examples**

The following command checks if the device uses SSL3[1]:

```
 $ ./scanner.py  --plugin detect_ssl3.py
                 --device lcg-argus
                 --port 443,8443
```

The output of the scan is:

```
#resource_type;resource_name;test_name;result;notes;target

device;lcg-argus;detect_ssl3;VULNERABLE;SSLv3 is enabled on
lcg-argus:443;[lcg-argus:443]

device;lcg-argus;detect_ssl3;VULNERABLE;SSLv3 is enabled on
lcg-argus:8443;[lcg-argus:8443]
```

As one can see, for each scan a different result line is printed, because the plugin reports the device name and port in the notes. This other example checks for a default landing page on a web server:

```
 $ ./scanner.py  --plugin landing_page.py
                 --device pcitdi86
                 --alias pcitdi86-alias
                 --url /index.html
```

The output will be:

```
#resource_type;resource_name;test_name;result;notes;targets

device;pcitdi86;landing_page;SECURE;The
target does not use an empty or default
page;[pcitdi86-alias(pcitdi86):80/index.html,pcitdi86:80/index.html]
```

In this case, the scanning results of the two targets (with different aliases) are grouped into one line. If the Scanner faces a problem while running the plugin (plugin crash, output not parsable, etc) it will report a line with the result FAIL. Also, if the user defines a timeout (via --timeout) and a timeout occurs while running the plugin, it will be reported in the

[1]SSL version 3 is vulnerable to the POODLE vulnerability discovered in October 2014

final results.

```
device;pcitdi87;landing_page;TIMEOUT;plugin timeout after 10
ms;[pcitdi86:80/index.html]

device;pcitdi88;landing_page;FAIL;There was a problem parsing
plugin output(error code 127);[pcitdi86:80/index.html]
```

## 3.3 Conclusion

The Scanner fills the gap between a scanning engine and individual security tests. It makes it easier to develop detection scripts that would test for a misconfiguration or a vulnerability on a single target. This can improve the procedure of scanning CERN resources, whenever a new vulnerability is discovered. Using the Scanner we can develop short detection scripts that deal with only one target and use the Scanner to take care of looping over different targets on the same resource. One level higher, we would like to scan multiple resources at CERN (maybe the whole CERN infrastructure) to find vulnerable resources, in a concurrent and fault tolerant manner. GNU Parallel as described in Section 3.1 can feed the resource data (from prepared files, obtained by running `was list-servers` and `was list-sites`) to the Scanner and have it scan multiple resources concurrently.

In addition, the Scanner will group the findings of individual tests to remove duplicate results and generate a parsable output that can be used for further investigations. For example, the Scanner output can be inserted into the CERN System Security DataBase (SSDB), used to keep track of vulnerabilities. Integration to SSDB is not implemented, yet, because SSDB will soon be replaced by an improved solution.

# Chapter 4

# Conclusion and Future Work

This project introduced two new tools for detection of vulnerable web applications deployed in a large IT installation. The approaches used in these tools are different from common web scanners, such as Skipfish. In this project we focused more on detecting vulnerable applications as soon as possible by monitoring publicly available vulnerability information. Once we are aware of a vulnerability, we can take two approaches to detect applications that may be a subject to it. If the vulnerability is specific to a software or product, VNT described in Chapter 2 will report the resources (applications) that use those products. In other cases when the vulnerability is not specific to one product but to its configurations, e.g. an expired certificate, or when detecting the vulnerable technology on resources or applications is not possible or efficient, e.g detecting technologies not supported by WAD, the Scanner (Chapter 3) can be used to run other detection scripts. In addition, the Scanner can be used to run automated regular scans of the whole web infrastructure at CERN and analyze the scan results automatically. Both of these tools are currently being used at CERN to enhance the vulnerability management process of web applications, but there is a potential for improvements in both of them.

## 4.1 Vulnerability Notification Tool

VNT can be used to get informed about all vulnerabilities, regardless of the product they affect or whether they are web-related. At the moment, it stores all these findings in parsable files. Therefore, it can become a part of the overall CERN vulnerability management mechanism to help keep track of vulnerabilities anywhere. It is possible to extend VNT and evolve it into a vulnerability management framework, where users can decide to ignore some vulnerabilities or get different notifications based on the type of the changes that happen in a vulnerability.

The performance of VNT is highly dependent on the vulnerability source it is using. At the time of this project, it was decided to use NVD, but it is important to keep an eye on the market and use other sources instead of –or along with– NVD, if they fit the needs described in Section 2.2.

One of the main challenges while using NVD as the vulnerability source, was the randomness of CPE names. Although CPE is introduced as a standard way of enumerating platforms, there are still many problems with it: Its dictionary is incomplete and contains redundant entries, such as multiple CPE names for the same product. On the other hand, it seems that the available vulnerability sources have not agreed on a standard way of listing vulnerable softwares and each has come up with its own format. The CPE name mapping algorithm described in Section 2.4 has been tested on NVD CPE data, but would probably be efficient enough on other naming formats with some minor changes. In any case, there seems to be a need for a standard product naming format that is complete and robust enough to meet the requirements of all interested parties.

The product name matching algorithm that is currently being used to find vulnerable products can also be improved. In addition to application names, WAD provides the category (type) of each application. The current algorithm described in Section 2.4 ignores the category of WAD names. A possible improvement to the algorithm would be to report matches only if the CPE type (first field of the CPE name) and WAD category match (e.g. both are operating systems). Another potential improvement is to use available algorithms for computing the string distance between CPE and WAD names, reporting only the pairs that have a similarity higher than a decided threshold.

Alternatively, WAD can be extended to include the equivalent CPE name(s) for each WAD name. This approach can ensure a higher accuracy level, because there will be no need for a name matching algorithm. However, maintaining this dictionary of WAD names to CPE names will be necessary whenever new WAD names are added to WAD. Ideally, this extension can be done to Wappalyzer to benefit from the community of Wappalyzer to maintain the dictionary. But Wappalyzer does not have a security purpose and it would be a challenge to convince its community to care for finding equivalent CPE names whenever they add a new detection rule for a new technology.

NVD reports a list of affected product versions, but this data changes very often. One can notice that previously announced vulnerable versions sometimes get removed from NVD updates or versions that were not reported as vulnerable, enter the list. There are many cases where vulnerability information gets updated on NVD with a small change in the sub-sub-version of an affected product. If VNT was sending update notifications for every change in affected product versions, it would flood its users with updates, encouraging them to ignore such notifications completely. Therefore, in the current tool, we decided to ignore any changes in product versions. Apparently, knowing about these changes can be useful in some cases though. Imagine that there is a vulnerability that is affecting Drupal version 6. We might decide to ignore this vulnerability at CERN, because all Drupal instances are using version 7 or higher. If a couple of days later, NVD updates the same vulnerability and lists Drupal 7 as a vulnerable product, we would definitely like to receive an update notification. As one can see, there is a trade-off here and it can be an open question to the CERN Computer Security Team to decide if they would rather receive more notifications (among which many are useless), for the sake of not missing important product version changes in vulnerability information.

For the time being, VNT is sending email notifications to the CERN Computer Security Team members rather than the resource owners. The reason is that, in most cases, further investigation of a vulnerability is needed to make sure that the reported resources are indeed affected. One step towards fully automating the tool is to consider product versions when reporting the resources.

The product version reported by WAD, however, is not always reliable. For instance, Apache servers might be reported to have obsolete versions while the new updates have been back-ported[1] to them. In addition, WAD is unable to detect the version of many products, e.g. CakePHP framework. Due to these limitations, it was decided to ignore product versions when reporting vulnerable resources and leave it to the investigator of the vulnerability to decide if a certain resource is indeed affected. Once WAD supports more reli-

[1]Parts of a newer version of a software have been added to an older version of the same software, without upgrading.

able version detection, VNT can be slightly changed to only report resources that use a certain version of a product.

Used in a slightly different way, VNT can be used to make sure that new resources at CERN are not vulnerable. Whenever there is a new resource detected by WAD, we can use VNT data to ensure that this new resource is not vulnerable to recently found vulnerabilities.

## 4.2 Scanner

The Scanner facilitates running individual detection scripts on CERN resources. Currently, it supports detection scripts that run on websites and web servers (devices). It can be easily extended to support other types of resources at CERN. For example, CERN is managing user accounts and each of these accounts can be considered as a resource. Each account is associated with an AFS space and one can think of scanning all accounts at CERN to find out if a user is storing his private keys in an AFS public folder. This would be a security risk for the user, because everyone in the organization would have access to his private key. The Scanner could be used to run this script on all accounts at CERN if provided with relevant inputs to pass into the detection script.

Currently VNT and the Scanner are two separate tools for different purposes. In cases that a vulnerability detected by VNT needs further investigations, it would be valuable to connect VNT to the Scanner. This could help the CERN Computer Security Team test for themselves whether a given server or site is actually subject to a VNT-reported vulnerability by writing a corresponding plugin and launching the Scanner to check the VNT-reported targets, then automatically trigger a mail notification if the vulnerability is confirmed.

# Bibliography

[1] I. Bar-Gad and A. Klein. Developing secure web applications. Technical report, Sanctum Inc., June 2002.

[2] T. Wittbold A. Buttner and N. Ziring. Common platform enumeration (cpe)-name format and description. Technical report, MITRE Corporation, 2007.

[3] Y. Younan. 25 years of vulnerabilities: 1988-2012. Technical report, Sourcefire, 2013.

[4] P. C. Meunier and E. H. Spafford. Running the free vulnerability notification system cassandra. Technical report, Purdue University, 2002.

[5] J. T. Chambers and J. W. Thompson. Vulnerability disclosure framework: Final report and recommendations by the council. Technical report, National Infrastructure Advisory Council, 2004.

# Appendix A

# VNT Evaluation Method

Once we have the output of the tool in JSON format, it is easy to convert it to other formats to do an analysis of the results or obtain some statistics.

## A.1 Calculating the Number of Vulnerabilities

The python script `vnt_format.py` receives the JSON description of VNT results as an input and generates an output in text format describing each vulnerability in one line. This text output can be used by various command line tools, such as grep, cut, etc., to obtain statistics. Each line is in the following format:

```
c|n;cvss_score;wad_names
```

The first component describes if the vulnerability is new (n) or has changed (c). If there are multiple entries in one component, i.e multiple WAD names, entries are separated by comma.
For Example:

```
c;6.8;Red Hat,Fedora,JBoss Web,OpenSSL
```

describes a vulnerability that has changed, has a CVSS score of 6.8 and matches with WAD names ‘Red Hat’, ‘Fedora’, ‘JBoss Web’ and ‘OpenSSL’. The following command will output the number of new vulnerabilities and changes in vulnerabilities using the TEXT description of all the results.

```
$ cat *.txt| cut ‘-d;’ -f1| sort| uniq -c
```

In order to get the critical vulnerabilities one can use the following command:

```
$ cat *.txt| grep ‘;\([6-9]\|10\)\.’| cut ‘-d;’ -f1| sort|\
  uniq -c
```

It is also important to know the number of vulnerabilities that affect CERN. For this purpose WAD can be used to generate a file with all WAD names used at CERN (one name per line) and then using the following command can give us the same statistics as before, but only considering the vulnerabilities that affect CERN.

```
$ cat *.txt| grep -f wad_names_at_cern.txt| cut ‘-d;’ -f1|\
  sort| uniq -c
```

```
$ cat *.txt| grep -f wad_names_at_cern.txt|\
  grep ‘;\([6-9]\|10\)\.’| cut ‘-d;’ -f1| sort|\
  uniq -c
```

# Appendix B

# Implementation Details

## B.1 Vulnerability Notification Tool

Figure B.1 illustrates the internal architecture of VNT. There are 2 classes and four scripts in the tool. Class Vulnerability represents vulnerabilities with all their NVD reported fields. Class Product takes care of everything that is related to CPE names. The main method in this class is the `guess_wad_names()` method which implements the mapping algorithm. Different mapping algorithms were tested and all of them are implemented in this class, however the best one was chosen to be called when mapping was necessary. `vnt-init.sh` is a shell script that can be used to initialize the tool when it is being deployed on a new machine. This script downloads yearly feeds from NVD and stores all the vulnerability data, so that in future executions, the tool can discover new or updated vulnerabilities. `send-notification.sh` is the main script of the tool, in the sense that it uses both main components of VNT (VNT_CORE and VNT_EMAIL) to download the feeds and report vulnerabilities.

VNT needs to have WAD names (apps.txt) to match CPE names to WAD names and WAD findings at CERN (wad_finidings_cern.txt) to filter out the irrelevant vulnerabilities. These two text files were generated using WAD.

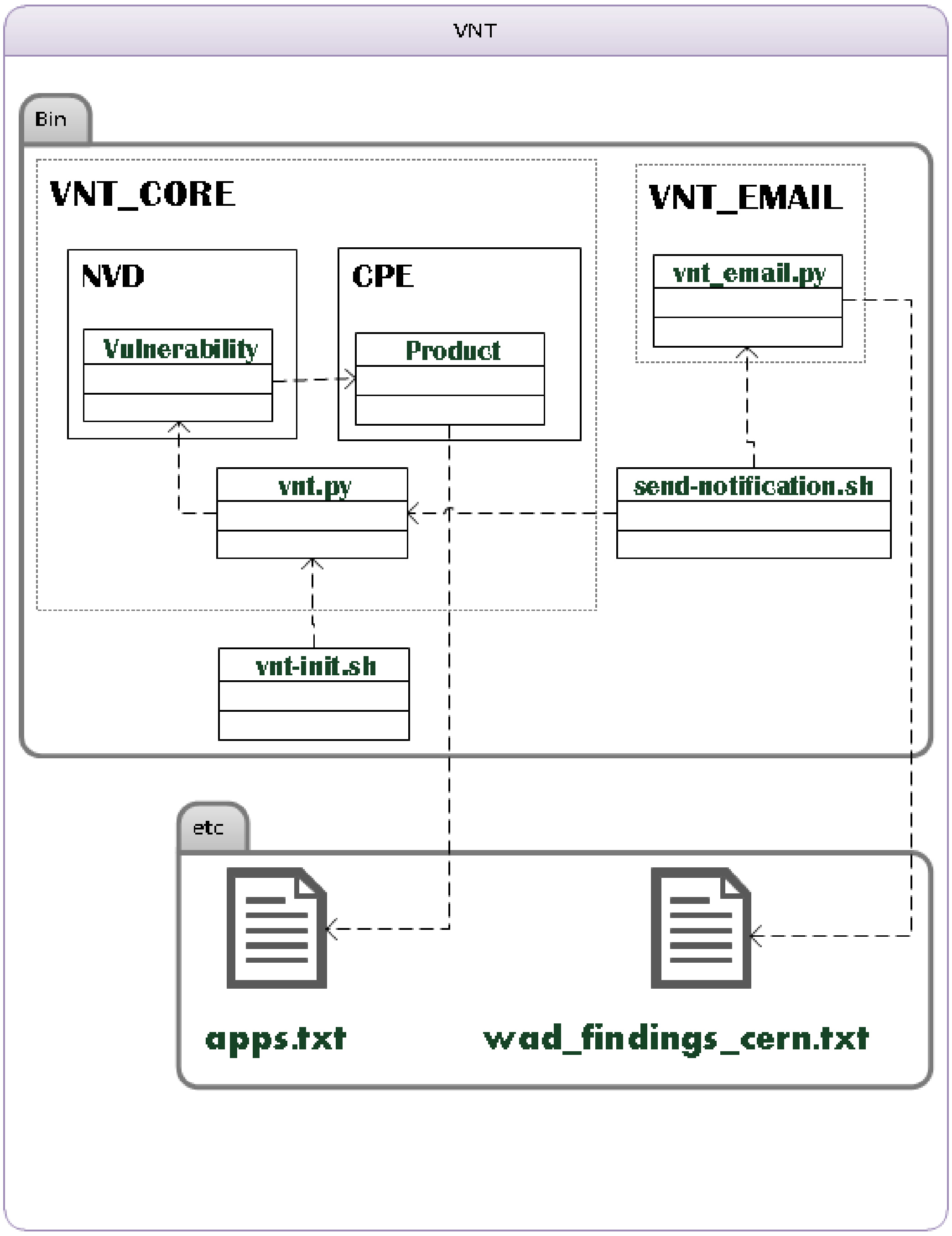


Figure B.1: VNT Internal Architecture

Python 2.6 was used as the main programming language for this project (except for the shell scripts) and a total of 1,026 lines of code were developed for this tool.

## B.2 Scanner

Figure B.2 illustrates the internal architecture of the Scanner. The Scanner is using 4 main Python classes to represent targets, plugins, plugins output and scan results. `scanner.py` is the main script of the tool to execute when scanning targets. The Scanner is dependent on the available plugins. In order to be able to test the Scanner, it was necessary to develop some plugins or modify the existing ones to follow the Scanner conventions. For this purpose, the plugin `landing_page.py` for both websites and servers and `weak_cipher.py` for web servers were modified. Also, two other plugins (`detect_ssl2.py` and `detect_ssl3.py`) were developed by other members of the CERN Computer Security Team to detect if a server is using SSL version 2 or 3 and these scripts were used to test the Scanner.

In total, the Scanner contains 977 lines of Python code (without considering the plugins).

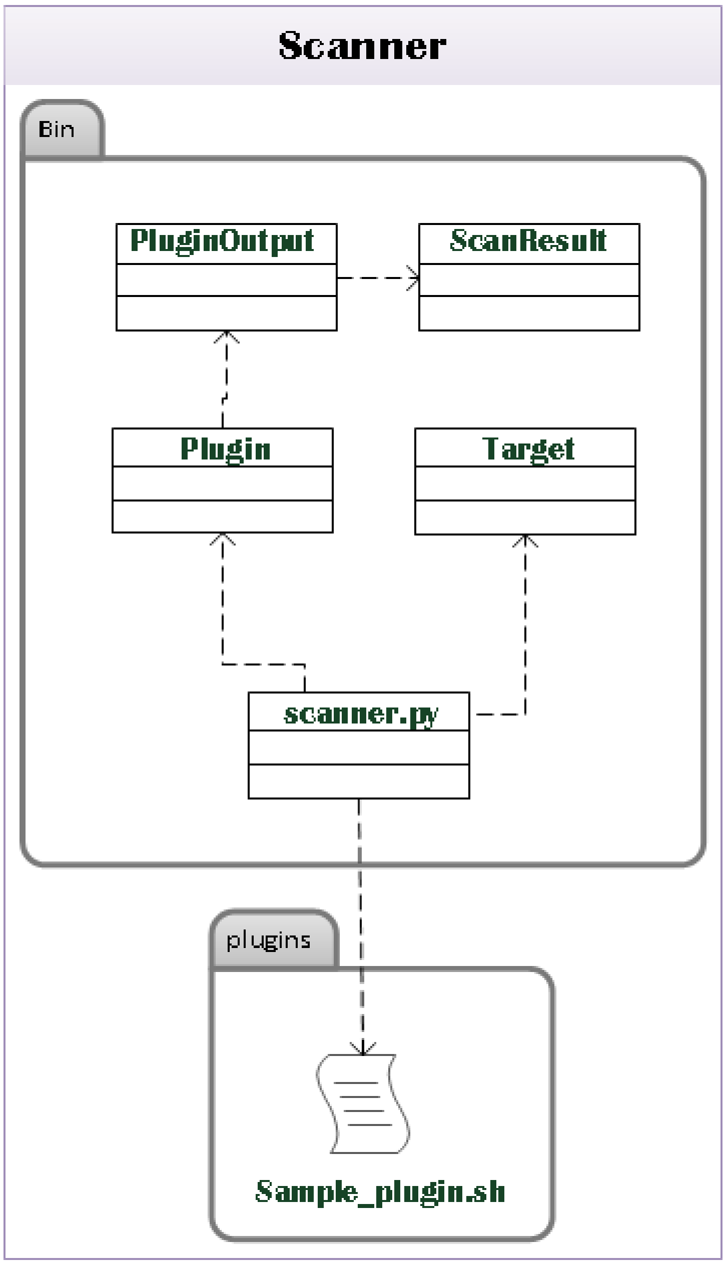


Figure B.2: Scanner Internal Architecture