How to detect vulnerabilities using Vega web scanner

by Washington Umpierres de Almeida Junior

The choice of appropriate tools for analyzing vulnerabilities in the web environment by Cyber Security professionals takes into consideration several aspects. Particularly, I consider the tools that meet high standards of efficiency, such as those that match the Web Application Security Scanner Functional Specification developed by NIST, that stands for National Institute of Standards and Technology, a North American institution widely recognized due to the high level of its technical studies and researches, whose publications have accreditation by several institutes of quality around the world, such as INMETRO in case of Brazil, the National Institute of Metrology, Standardization and Industrial Quality, a Brazilian federal autarchy linked to the Ministry of Development, Industry and Foreign Commerce. According to the NIST SP 500-269 document, a web application security scanner is an automated program designed to examine web applications for security vulnerabilities.

In this article, I present the Vega web scanner, a sophisticated tool for web scanning, multiplatform, which has been developed by Subgraph and that I consider one of the best scanners in its category.

Legal note

Performing a web scanner does not constitute an attack. In the vast majority of instances, a web scanner does not cause any damage to its target system. Cyber Security experts use this technique to diagnose web problems and to detect vulnerabilities on their web environment.

However, experimenting with Vega on servers that do not belong to you and that you are not authorized to scan against it, does constitute illegal activity and it is subject to law enforcement that can vary from country to country.

About Subgraph Vega scanner

First of all, it is very important to mention that Subgraph has produced excellent documentation about its web scanner called Vega where part of this information has been extracted from. This article can be seen as a summary of what you will find in much more detail in Vega documentation that can be accessed at Subgraph web site in the URI https://subgraph.com/vega/documentation/index.en.html.

Vega is an open source and multiplatform web security scanner written in Java that can help the cyber security professional look for vulnerabilities such as SQL Injection, reflected XSS, stored XSS, remote file include, shell injection, disclosed sensitive information, and much more. It relies on a database that contains information required to check a web system for security holes in its services and potential paths to exploitable contents or scripts. Then the Vega web scanner tries to exploit each vulnerability that is discovered, which is sometimes called as ethical hacking.

Running its Graphical User Interface on Linux, OS X or Windows the professional cyber security analyst can also experience scenarios like probing for TLS/SSL security settings in order to identify opportunities for improving the security for TLS servers.

Vega is such a robust tool that equips the main pentest distros, like Kali Rolling, an advanced Linux Debian based penetration testing platform developed by Offensive Security and my preferred, and Parrot Security OS, which is another GNU/Linux distribution designed with cloud pentesting and IoT security developed by Lorenzo Faletra. Vega's features can be extended when using an API written in Javascript language.

Although we know that Vega can be found in the main advanced pentest distros, I am considering the need of installation on a Linux environment for the purpose of this article.

Installing Vega in a Linux environment

Vega packages 32 and 64bits for Linux, OS X or Windows can be downloaded at https://subgraph.com/vega/download/index.en.html or alternatively can be cloned from Github repository at https://github.com/subgraph/Vega/wiki/Vega-Scanner.

After downloading the zip package from Subgraph web site, just extract its content and Vega will be ready to be used.

If you prefer cloning Vega from the Github repository, open a terminal session on Linux shell, go to the folder you want to clone the Vega package and launch the following command:

```
git clone https://github.com/subgraph/Vega.wiki.git
```

After cloning the Vega package from Github repository, Vega web scanner will be ready to be used.

To launch Vega scanner GUI, just double click on Vega application inside the folder you have downloaded or cloned it.

As we have Vega installed, let us go ahead and get started working with Vega.

Working with Vega

If you have your computer equipped with Parrot Security OS 3.3 like me, you can find the Vega web scanner in the menu Parrot → Web Application Analysis → Vega.

When launching Vega for the first time you will see the Vega workspace under the scanner perspective. Vega has two perspectives to know: the scanner and the proxy. In this article I am going to concentrate our work in the scanner perspective since this situation covers most of the scenarios. For exploring the proxy workspace, I invite the Pentest Magazine reader to visit the Subgraph web site documentation located at URI https://subgraph.com/vega/documentation/Vega-Proxy/index.en.html.

As an exercise, you can click in the Proxy button above on the right to commute between the Scanner and Proxy workspaces. After having a look in the Proxy workspace, click in the Scanner button to go back to the original workspace.

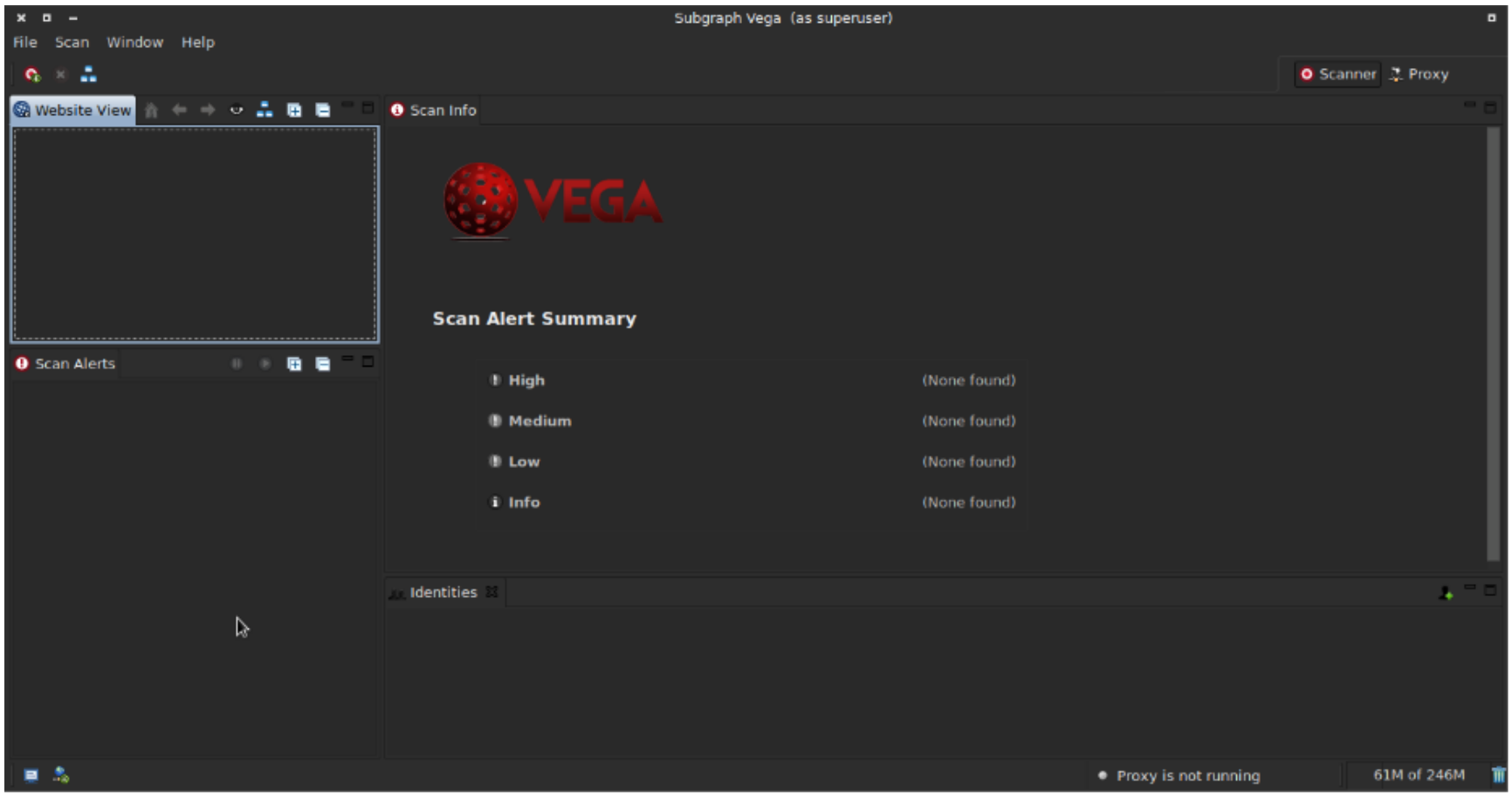

Figure 1: Vega Scanner workspace

Figure 1 above shows the original layout of Vega in its scanner workspace. You can see the objects organized in the layout as the panels Website View, Scan Alerts, Scan Info and Identities. These objects can be moved, changed and the workspace can be completely modified. At any time, the user can restore the original workspace layout by clicking on the Window menu and selecting the option “Reset Perspective...”.

This action is useful when starting a new project where you know there is no link between the projects, so it makes sense to want to have on the screen only the elements of a certain job that normally has a completely different scope from another.

As an automated security testing tool, Vega web scanner starts its job crawling a website, analyzing each page content to find links and form parameters as injection points, referred to as path state nodes. Vega basic modules are run on path state notes. Then Vega runs its modules to analyze them and the responses are sent back from the server during the scan.

Just as a little example, the method "boolean pathstate.isSureDirectory()" returns true if the path state node is a known directory. In another scenario, the method "boolean pathstate.isRootPath()" returns true if the current path state node is the root path (e.g. /).

Also, Vega has been implemented with an instance of org.apache.http.HttpHost, which are objects built to store all information that describes an HTTP connection to a host.

Technically, httpHost properties are defined as httpHost.hostName, httpHost.port and httpHost.schemeName. The httpHost.hostName property carry an IP or DNS name string value, the httpHost.port carry integer value where "-1" value indicates the scheme for default port, and the httpHost.schemeName carry "http/https" string and the "null" value indicates the default scheme.

Thus the constructor detail is defined as following:

```
public HttpHost(String hostname,
int port,
String scheme)
```

So the defined constructor above will create a class HttpHost instance with the given hostname, port and scheme parameters.

It is possible to limit the scan setting in the scanner preferences, that include the parameters as number of path descendants, number of child paths for a single node, maximum path depth, maximum number of requests to send per second and others.

Limiting the scan scope is important because we can save time of scan processing if you have a tactical plan where the scan scope is well defined.

As we move forward in the exercise with Vega, we will comment on each aspect related to its resources in order to provide a better understanding as to its operation.

Starting a web scan with Vega

Our exercise in the Vega scanner workspace will be run against the web site OWASP Mutillidae II in my controlled environment running under Metasploitable 2 virtual machine which is hosting the web site.

OWASP Mutillidae is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, helping web developers to better understand the processes of securing web applications. The OWASP Mutillidae is available for download in its project page located at https://sourceforge.net/projects/mutillidae/.

The Metasploitable 2 virtual machine is running under Oracle VM VirtualBox and for security reasons, the virtual machine has been set with the network adapter in the Host-only mode. This means that the web site will not be exposed to the Internet since we know this web site has several vulnerabilities and it is not a good idea exposing it to the world.

But this will be enough to present the capabilities of the Vega. Figure 2 show us my Metasploitable 2 VM hosting lots of sites configured in my personal environment, which are up and running, including the OWASP Mutillidae (the seventh one) that is ready to be scanned.

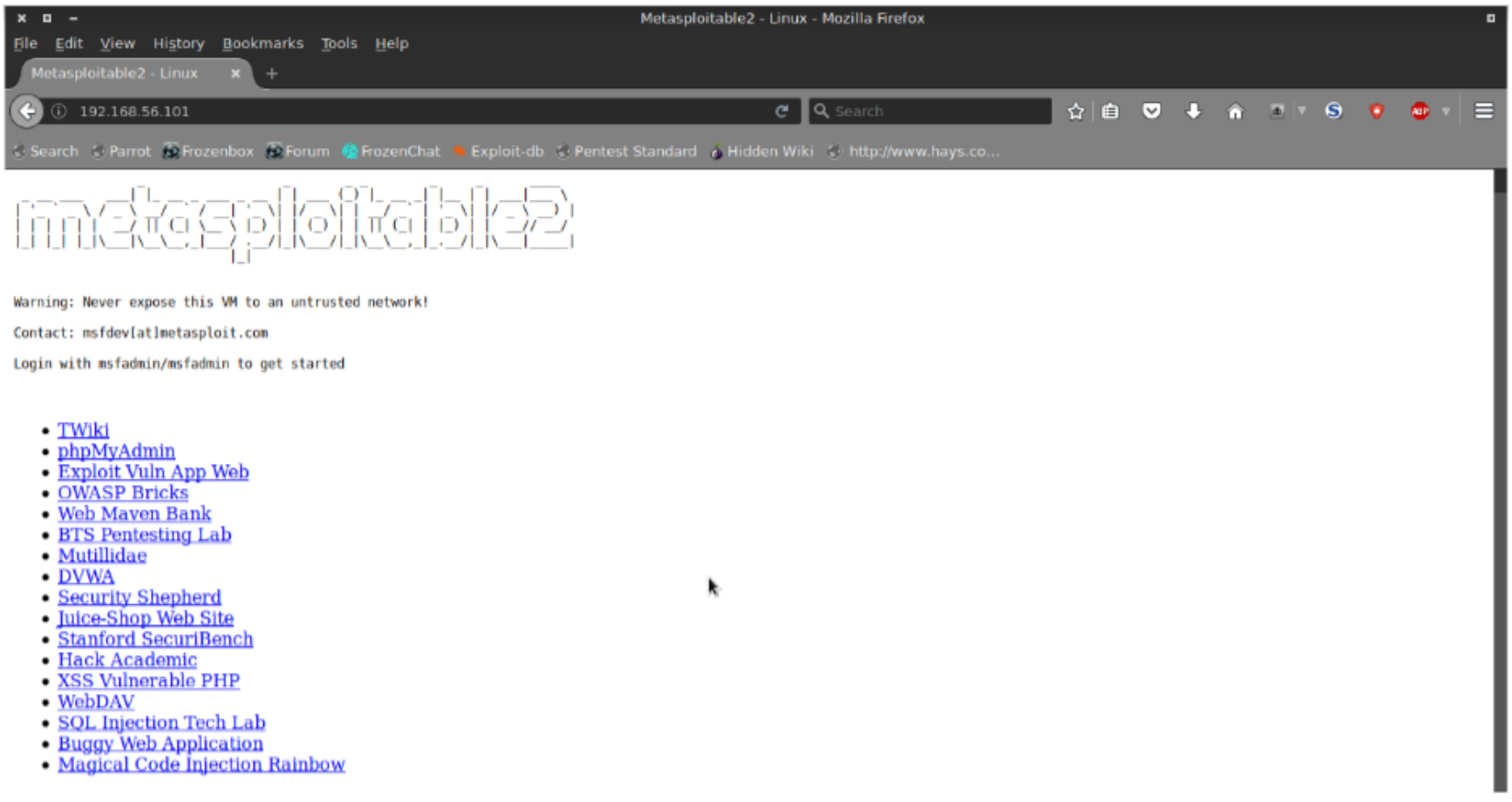

Figure 2: Metasploitable 2 VM hosting the site to be scanned.

Going back to the Vega environment, we can launch a new scan either pressing CTRL+N or accessing the option "Start New Scan" from the Scan menu. At this moment, we can scan a target by entering the URI directly in the Scan Target field or alternatively choosing a target scope for the scan as shown in the figure 3.

As the OWASP Mutillidae site is hosted under my controlled environment, I can type the URI http://192.168.56.101/mutillidae/ directly in the field previously mentioned.

At this point, let us assume this is all the information I have to perform the web scan. So we move forward clicking the "Next" button.

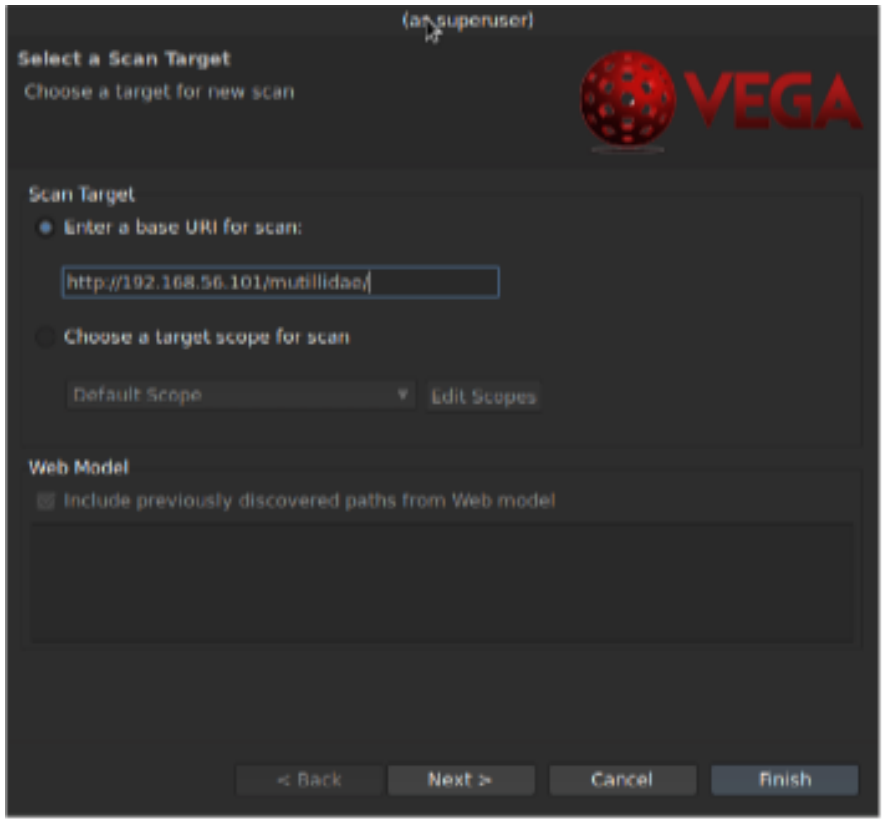

Figure 3: Selecting the target URI for scan

After clicking "Next", the user will have the option to select what Injection Modules and Response Processing Modules he wants to use to perform the scan. One pause to explain concepts. The Modules are units of extended functionality written in Javascript since the Vega engine is written in Java, but it also includes the Rhino JS interpreter. So Vega supports two kinds of modules: Basic Modules and Response Processing Modules. Basic Modules are those under Injection Modules options that run on path state nodes and perform active fuzzing, including URIs that are known to be files or directories and URIs with parameters, with each parameter being a distinct path state node. Response Processing Modules are those that run on all responses that are returned from the server. Both types of modules can store information in the shared knowledge base and generate alerts in XML-based format. We can explore the options for both modules expanding each one and selecting the options of our interest. For this exercise, I will select all options as shown in figure 4, but I suggest you to firstly have a good understanding of each one before selecting the available modules options since the scan can take longer.

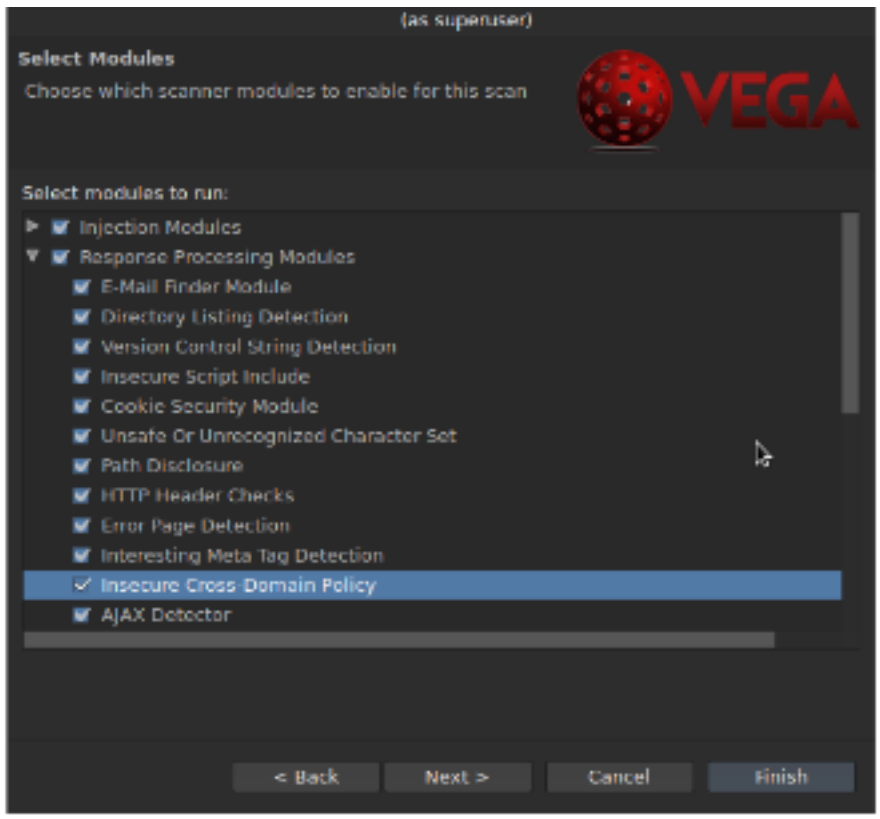

Figure 4: Selecting modules

The "Next" button will move us to the options to deal with authentication and cookies. Vega is sophisticated and supports the configuration of credentials for performing automated scans while authenticated to the application or server. These credentials include Basic HTTP, Digest HTTP, NTLM and Macro for form based authentication. Credentials must be configured using Identities with POST method since the information of credentials must be passed to the remote system in order for authentication to happen. But depending on the exercise, this option can be dispensed. Let us suggest an example: suppose that a company hires you to perform a web assessment and they don't want to give you this kind of information in order to see if you are able to find any vulnerability on their web resources without giving you any credentials. This is a good point to have in mind so let us consider this in our exercise and do not fill in any information related to credentials. This will bring us the potential of Vega.

Figure 5 just illustrates the Authentication Options commented before and moving to the "Next" we will have the option to work with the parameters shown in figure 6.

Parameters can be added or removed depending on the interest on these resources. For our tactical inspection, we will leave these options with the default selections.

By clicking the Finish button, the web scan will start and the Vega log starts populating the information, the time about when the crawling phase has started, and each resource discovered in the web site.

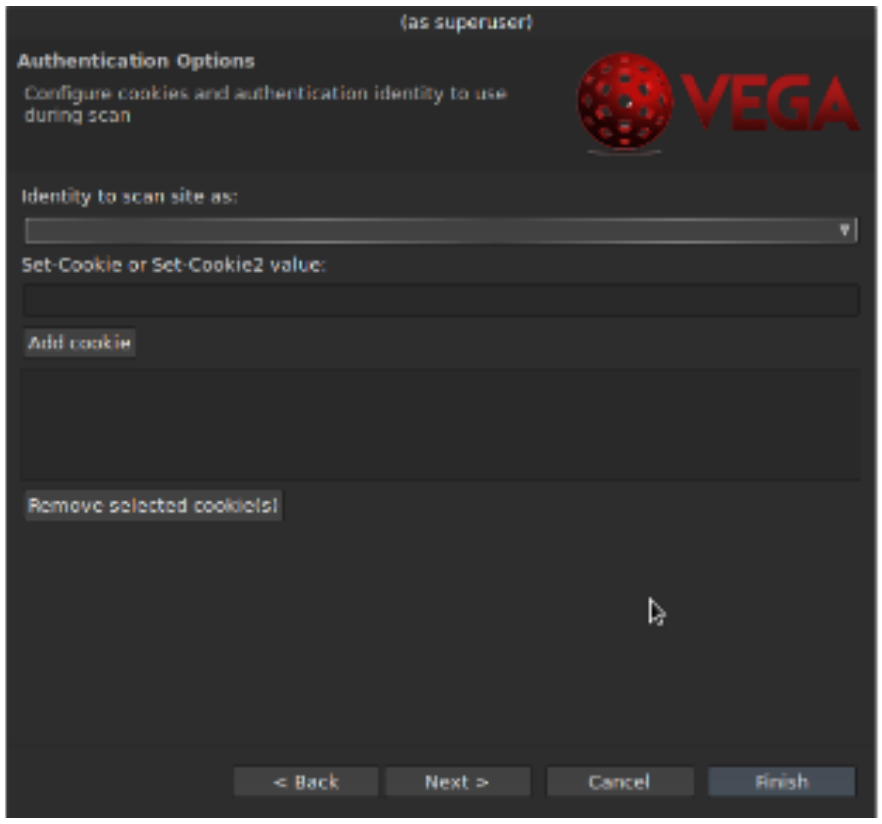

Figure 5: Authentication Options.

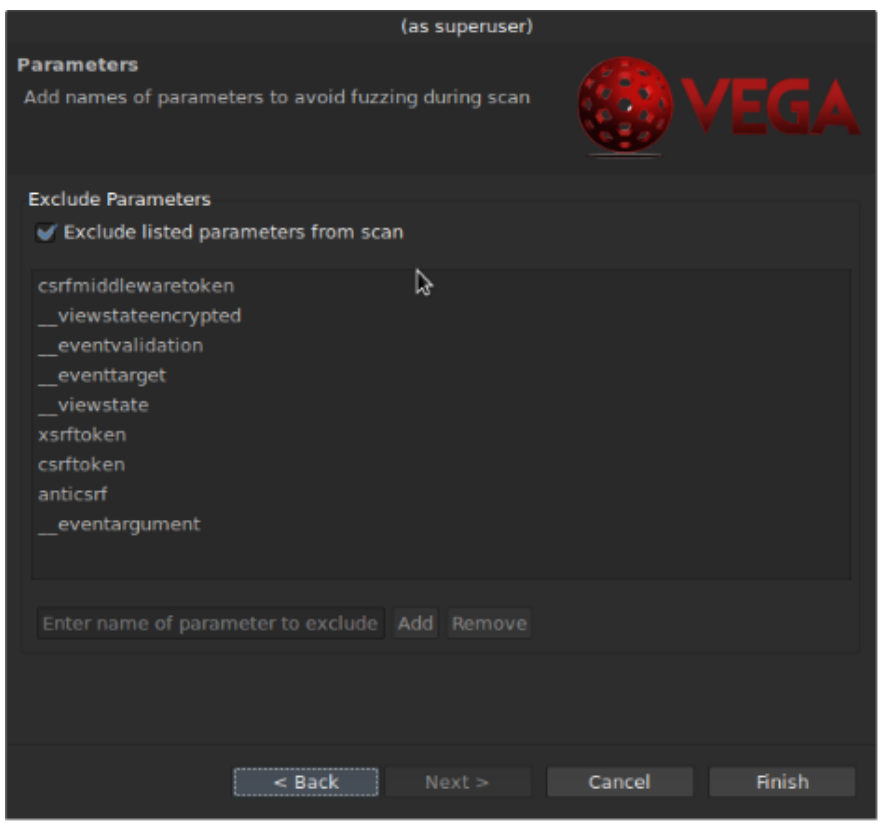

Figure 6: Parameters.

When Vega starts performing the scan, its progress is indicated in a progress bar exhibited inside the "Scan Info" panel object. Observing the scan being executed by Vega, we can see that the total number of links to crawl increases as Vega discovers new ones and generates variations of them to perform more tests as shown in figure 7.

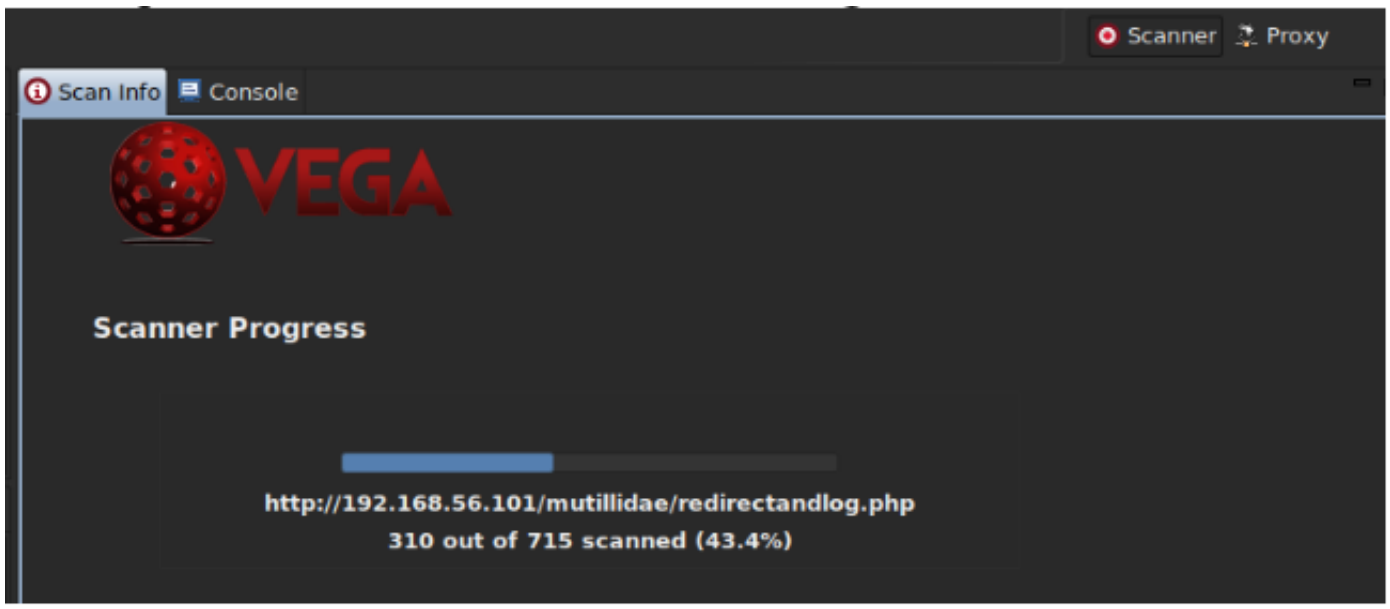

Figure 7: Vega web scan progress.

We can see the alerts being reported on the Scan Alerts panel as shown in figure 8. The user can expand each severity class on the panel to analyze its content in detail.

The Scan alerts are classified in severities as High, Medium, Low and Info. The High class indicates the web site has a critical vulnerability that can be exploited. The severity scale can be seen as the minor issues represented in the Info class and going up to the critical ones represented by the High class. The Low class are minor issues that although not offering a high risk deserves attention from the security and developer teams. And the Medium class includes those issues that are not as critical as the High class but security team has to deal with them the same way. All alerts are shown in the Scan Alerts panel where we can expand the content and analyze one by one. The alert incorporates both dynamic content from the module and static content from a corresponding XML file.

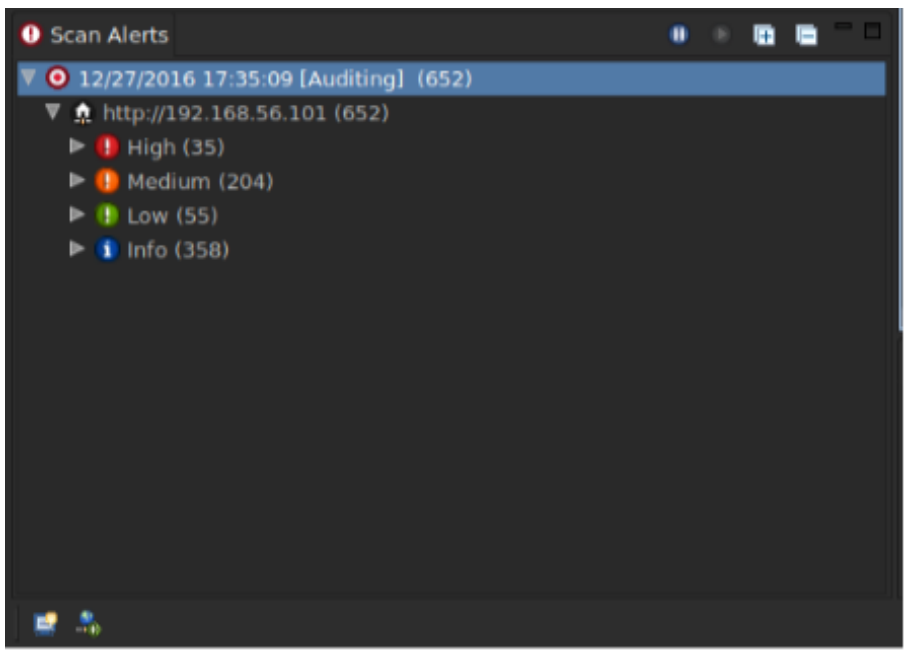

Figure 8: Scan Alerts panel

At any time during the scan, we can see what is happening if we open the console, clicking on the icon located in the lower left corner of the Vega workspace. The console will be opened and the entire content that is being published into the Vega log can be seen.

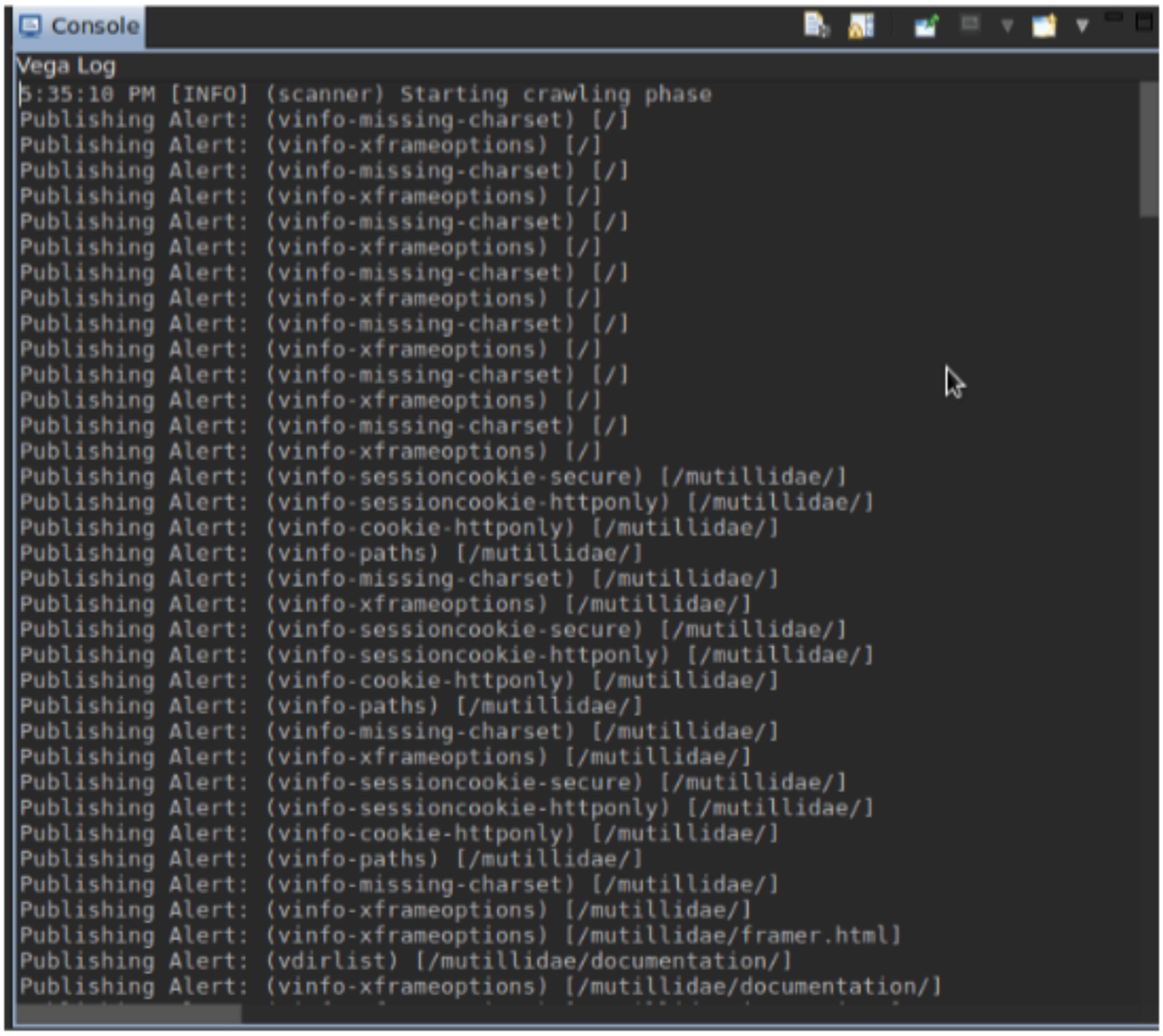

Figure 9: Vega console

One great feature about alerts is the link to the saved requests and responses. To slide open a fast view with the message editor, click on the request link towards the bottom of the alert. As an example, the quantity issues in the High severity has drown my attention and I decided to have a look at it. Then I expand the High severity and select the shell injection attack class to see its content. At this point, we can see an excellent job done by Subgraph. The report is complete and the Scan Info panel brings the following information to the user: AT A GLANCE, REQUEST, RESOURCE CONTENT, DISCUSSION, IMPACT, REMEDIATION and REFERENCES.

AT A GLANCE shown in figure 10 brings an overview of the vulnerability identified. Here the user can see where the resource is located and which parameter with the corresponding method has been used in order to identify the vulnerability, as well as the detection type used. Note that I have informed that I was using a Metasploitable 2 VM that runs under a Linux environment. So it makes sense that the detection type has brought the information about Linux/Unix checks.

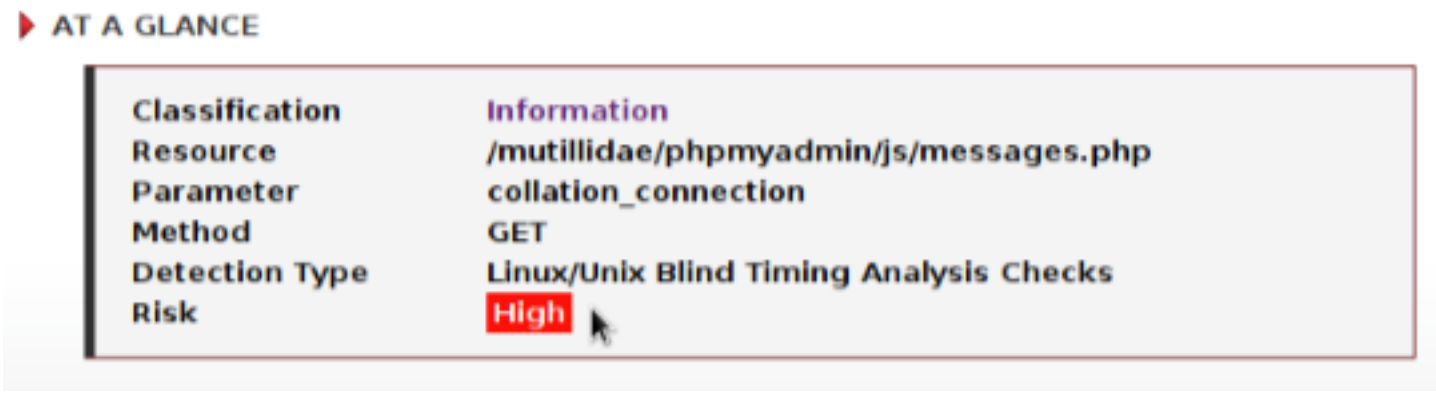

Figure 10: AT A GLANCE

The REQUEST information brings the details of the request done by Vega and response of the web site. Note that the method informed in the AT A GLANCE box above can be viewed here in more detail. It is possible to right click in the request link and open it or copy the link location.

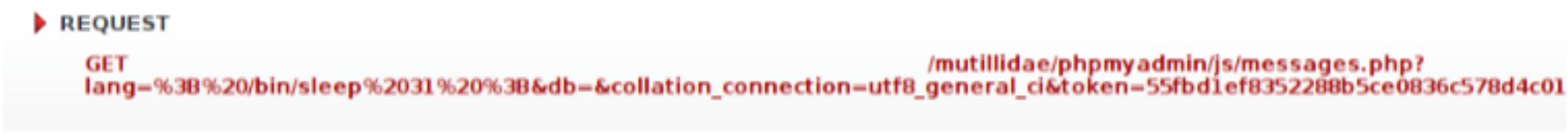

Figure 11: REQUEST

The RESOURCE CONTENT shows the information about the resource taken by Vega where the security specialist can analyze the source code.

Figure 12: RESOURCE CONTENT

Below in the DISCUSSION section, the symptoms about the vulnerability discovered by Vega are explained and the consequences of exploiting it.

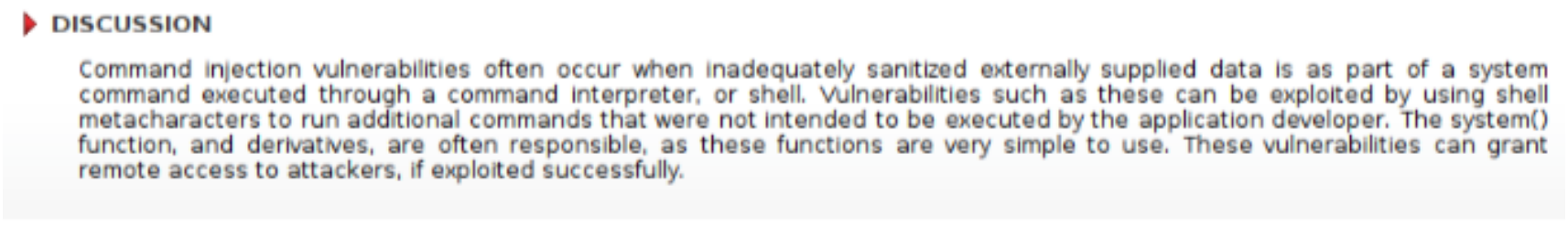

Figure 13: DISCUSSION

The section IMPACT, as the name suggests, explains which potential vulnerability has been discovered by the Vega engine and what can be done by an attacker in the case of working around the exploit reported.

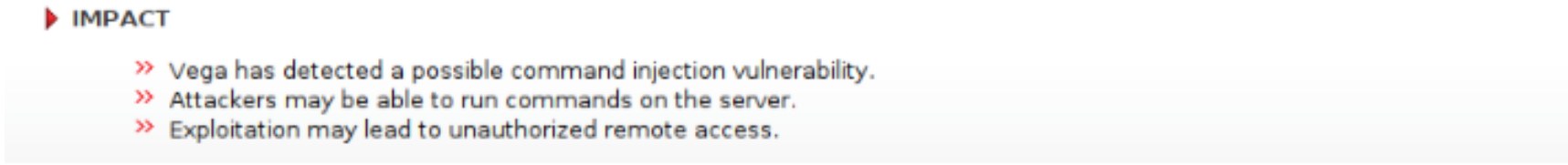

Figure 14: IMPACT

Finally, the REMEDIATION section brings us useful information in regarding to what can be done in order to minimize or completely eliminate the issue.

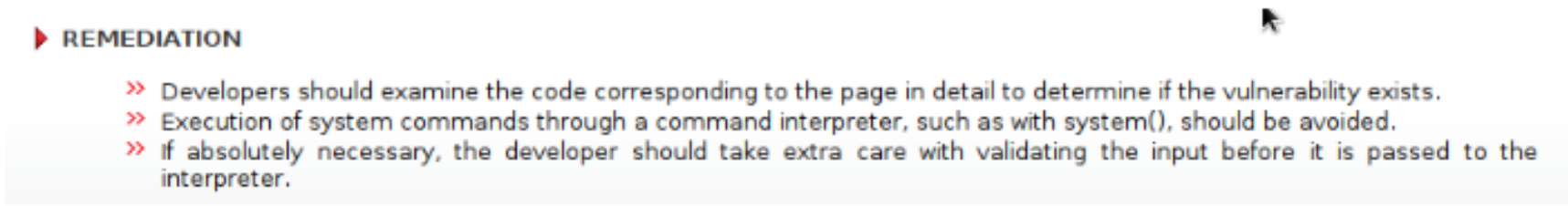

Figure 15: REMEDIATION

It is also possible to consult all the requests done by Vega during the web scan (or after it) if the analyst wants to research some specific topic, such as which method has been used in a particular operation, the time it has taken, the response from the web server, etc., as shown in figure 16.

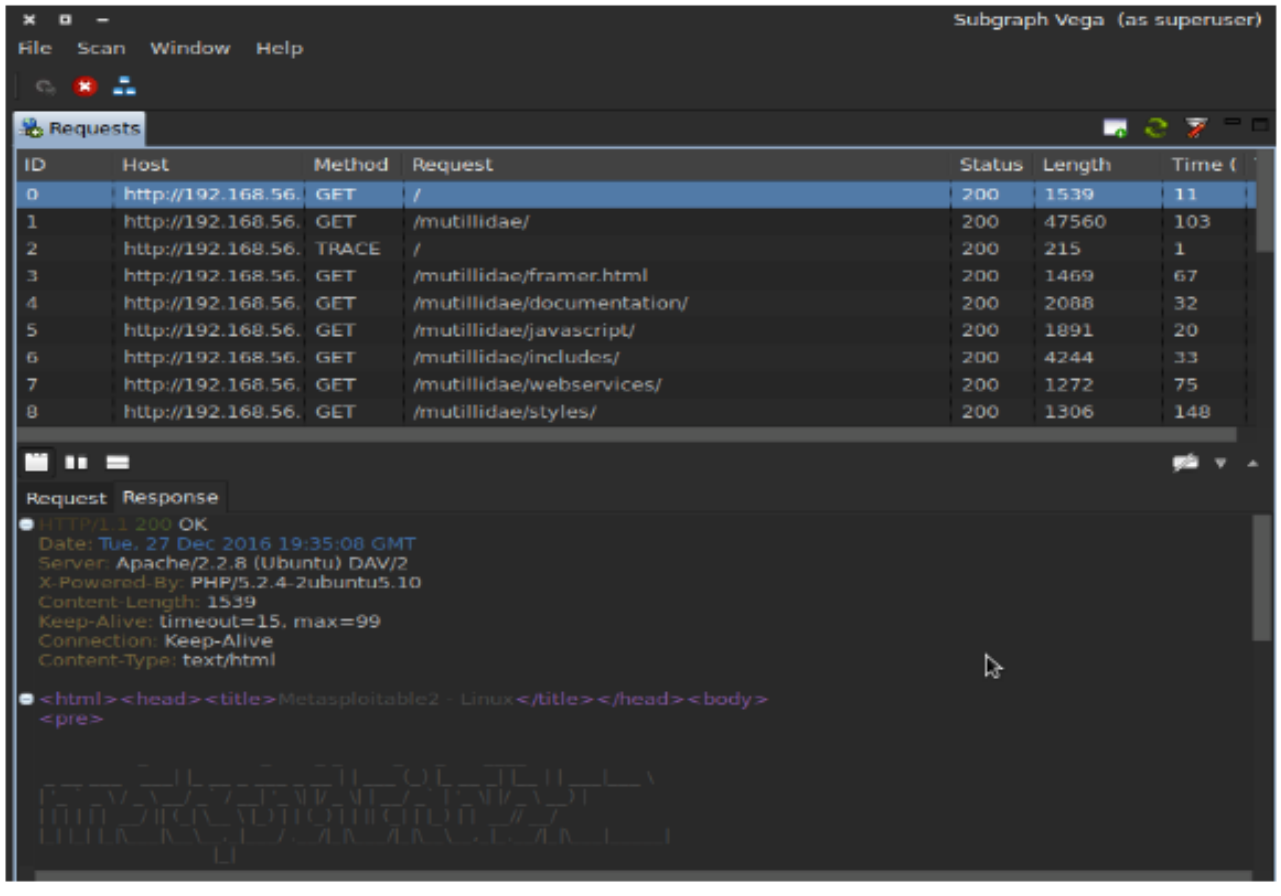

Figure 16: Exploring the requests

Inspecting the Vega results

Independent of the tool you are working with, it is a good practice to check the results for any vulnerability reported. Sometimes, depending on how the environment has been deployed and/or configured, it can bring a type of information that security experts are used to calling a false positive.

Also, it is important that the cyber security specialist have the experience and knowledge to analyze the information that Vega is bringing up in order to perform the appropriate tests and certify in a positive way the alert reported by Vega. False positives can be included in the report as it can be useful to the customer if they're thinking they can perform a fine tune setting in their web environment.

We have to remember that it is responsibility of the cyber security specialist, not of the tool, for certifying the threats reported. So keep in mind, always check any alert and be aware that this will take longer on your tasks.

So let us carefully analyze the Vega report in figure 17.

Vega has tested the resource /mutillidae/index.php using GET method and generating the request / mutillidae/index.php?page=document.viwer.php&PathToDocument=http://vega.invalid/%3B%3F which has been classified with the severity Medium.

Let us divide the request generated by Vega into three parts and understand each part of it.

- Part one: /mutillidae/index.php?page=
- Part two: document.viwer.php&PathToDocument
- Part three: =http://vega.invalid/%3B%3F

Parts one and two are those that we will be using to inspect the Vega report results.

Part one we could call the fixed part represented by the element of the origin of the resource.

Part two we could call the variable part of the request done by Vega.

Part three represents the Vega segment element that drives the report to deal with it. For the purpose of the post analysis, we can remove part three "=http://vega.invalid/%3B%3F" because it is not necessary.

This way of understanding the tool leads us to suggest changing the variable part with information that we can use in substitution of it.

Now we need to identify a PHP document and its path to analyze it.

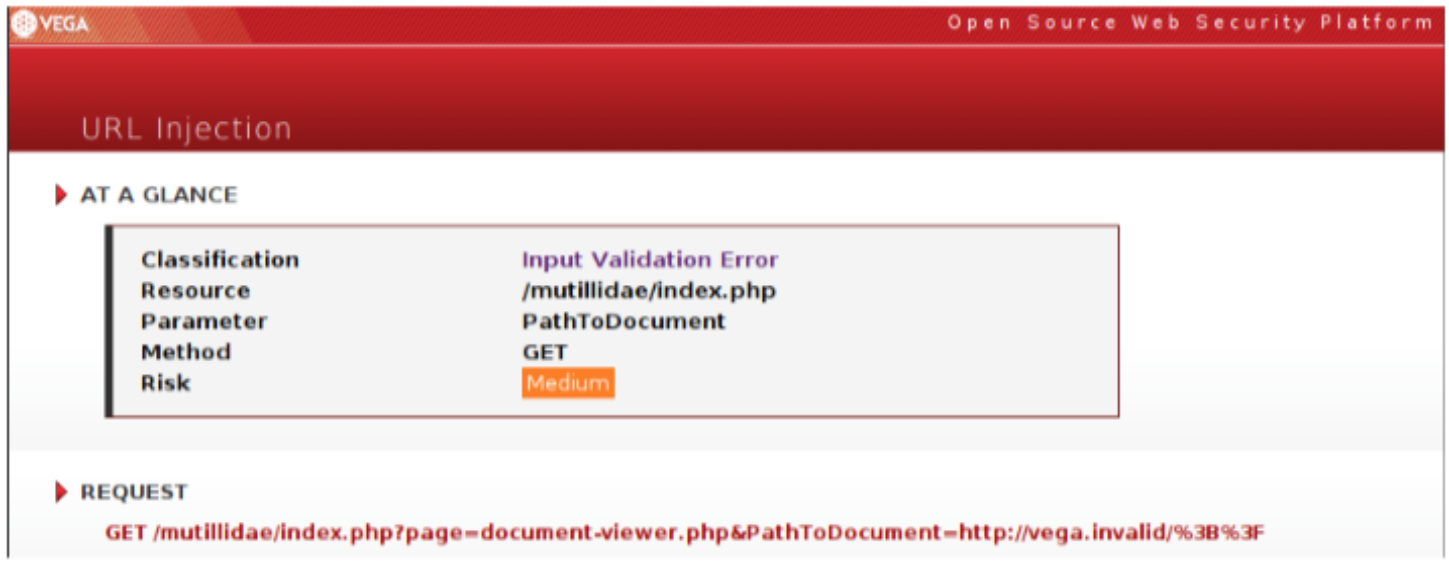

Figure 17: Inspecting Vega report

Looking at the Mutillidae directory structure in my private lab, I can see the document named dns-lookup.php in the root folder. Then I just have to replace the variable part: "document.viwer.php&PathToDocument" with "dns-lookup.php". Thus, our investigative analysis will be done under the resource "mutillidae/index.php?page=dns-lookup.php".

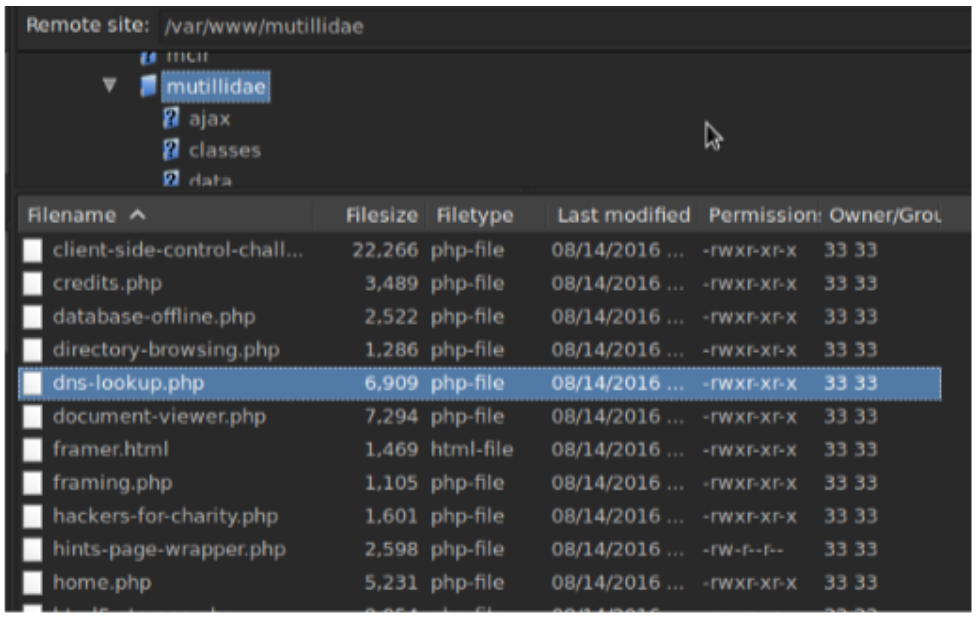

Figure 18: dns-lookup.php file location

Only for additional information and better understanding to the reader, let us suppose the file "dns-lookup.php" would be located inside a folder named "dns". In this case, the investigative analysis would be done under the resource "mutillidae/index.php?page=/dns/dns-lookup.php".

When accessing the resource "mutillidae/index.php?page=dns-lookup.php" we are redirected to a page where it is possible to perform a DNS lookup against an IP address or a hostname. On this interaction it is expected from us to fill in the field with a hostname or an IP address of a host in order to perform the DNS lookup consult.

In fact, when filling the field with the internal IP address of my modem, I can see the DNS lookup results in the page dns-lookup.php as shown in figure 19.

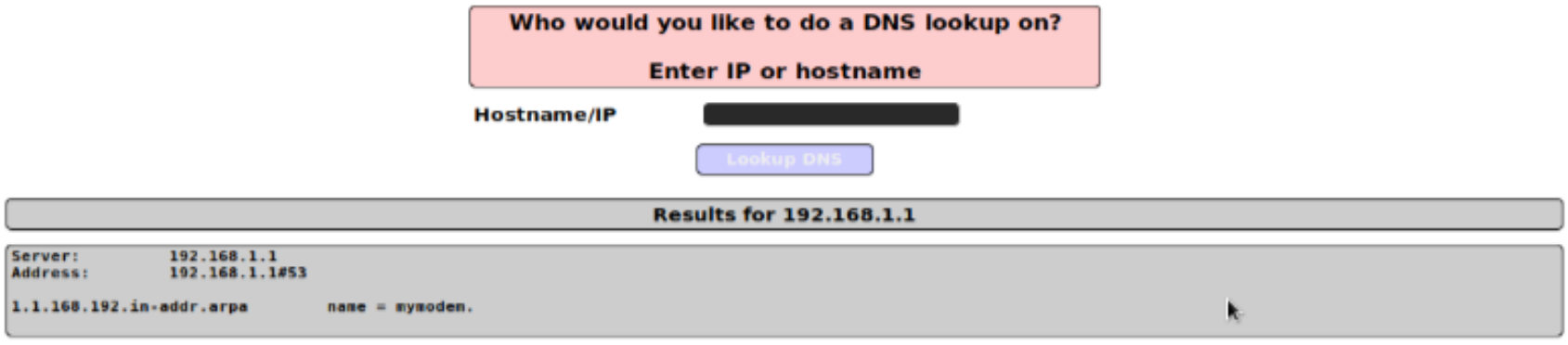

Figure 19: Performing DNS lookup against my modem

However, according to Vega IMPACT report, this issue can include automatic client fetching of remote malicious content.

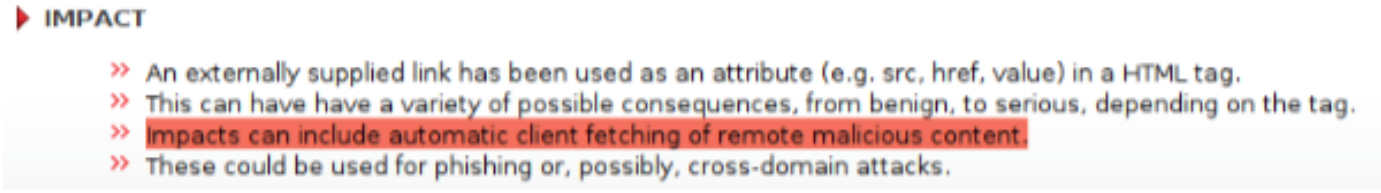

Figure 20: Checking IMPACT information

So, instead of entering a hostname or IP address, I will test the resource to check if it is vulnerable to shell injection class attack, as reported by Vega, because believing that the Vega report is correct, my intention is to fetch some system information remotely.

There are so many ways to do that, but I just want to inject a shell command to see if it will return some remote content result as part of Vega's impact report. In other words, I will launch a shell command line inside the field and try to get some information about the system hosting the web as if I was launching the shell command line in a terminal session in front of the server. Ethical hackers and cyber security experts know that we need to change the syntax in order to have positive results. In the case of injecting shell commands in a system running PHP and SQL variations, we have to put the character "&&" before the command we want to launch and add the character ";" in the end of the shell command.

So the basic objective in this activity is to get the information about the remote system with the command line "uname -a". The command line "uname" is used to get certain system information and the clause "-a" instructs the command to print all information about the system. Additionally, I would like to read the content of the file /etc/issue, which is the file where personalized messages are exhibited before the users login into the Unix/Linux system.

Thus, in a Unix/Linux terminal session, we would use the following shell command line:

```
uname -a && cat /etc/issue
```

The same command line in a web session for shell command injection purposes, I must fill in the field with the instruction as follows:

```
&& uname -a && cat/etc/issue;
```

Note that I inserted the "&&" in the beginning of the command and also added the character ";" in the end as previously explained.

The results are shown in figure 21 where we can see the shell injection being applied with success and the information about the system is exhibited right below and I certify that it was correctly reported by Vega.

The output "Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686 GNU/Linux" was generated by the command "uname -a" and the rest of the information that ends with the line "Login with msfadmin/msfadmin to get started" was generated with the command line "cat /etc/issue".

Note that the "cat /etc/issue" shows the sensitive information about the remote system, and although I have left this information purposely for the Pentest Magazine readers, it is scary to see the amount of sensitive information that is frequently exposed on the Internet because of a lack of consistent compliance in the security policies deployed by companies.

As a matter of fact, we could verify on this exercise that the resource in the web site reported by Vega is vulnerable to the shell injection attack class and the impact could be confirmed in practice.

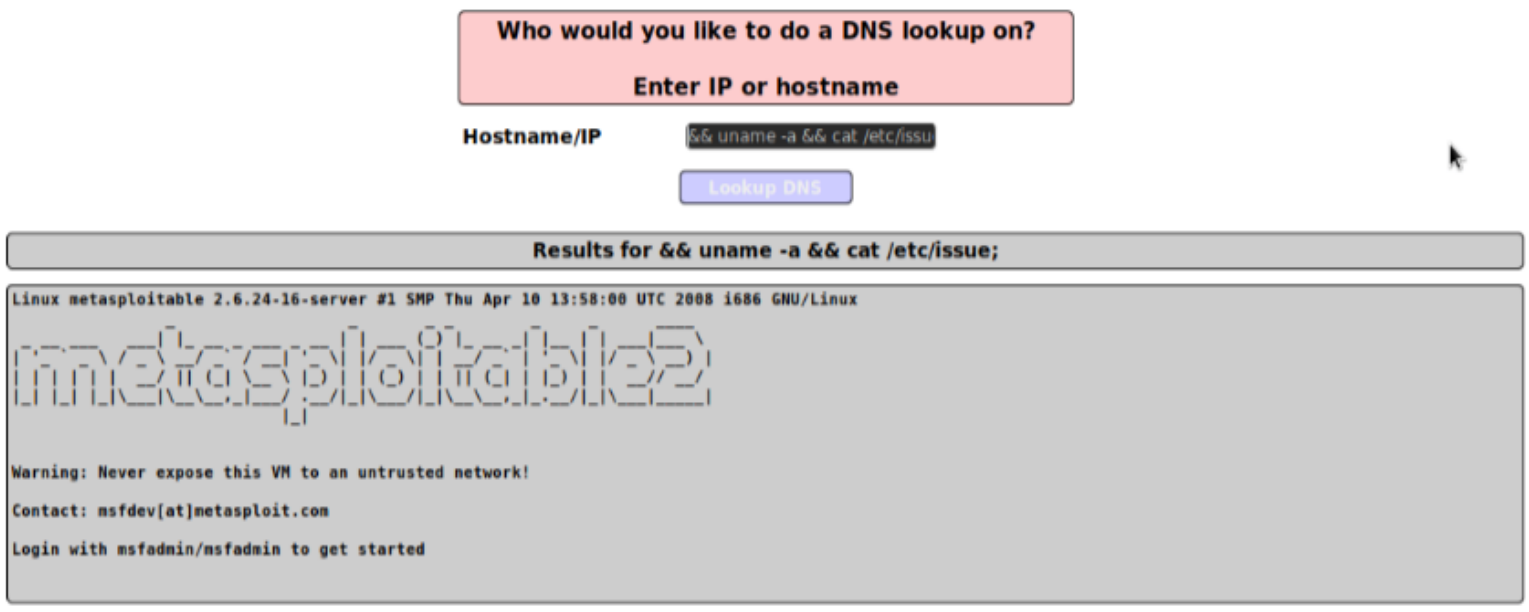

Figure 21: Shell command injection in action.

This is just a little example of what can be explored if opting to detect vulnerabilities using Vega web scanner. This is a great tool.

Important things about working with any tool

Vega is an extraordinary tool for discovering vulnerabilities in a web environment. However, very rarely it is possible to verify some exceptions as shown in figure 22 below. As if working in a great team, I recommend you contact the developer whenever you face an exception. I think this is the least contribution we can make to a team or company that worked hard to make a tool available to us on a daily basis. In this case, the contact is represented by the Subgraph team's e-mail info@subgraph.com to which the user should send a message with very well documented information and let them know about any minor issues if it comes to happen. Very often the problem can be realized in the system environment hosting Vega application that is dependent on modules working on your system.

This is the reason why it is mandatory to consult Vega's original documentation in order to correctly deploy this excellent tool.

Advantages and disadvantages comparison with commercial tool

The Pentest Magazine reader may be wondering: Why would I opt for an open source product rather than a commercial one? Well, to better answer this question, it is a good idea to put in a comparative table of advantages and disadvantages of both types of products. Let's do this exercise with the same topic for both cases.

Commercial tools Advantages:

- Ease of installation and upgrade;
- Bug Fixes;
- Good documentation.

Commercial tools Disadvantages:

- Expensive;
- Licensing restrictions;
- Proprietary;
- Development supported by demand/commercial interest;
- Sometimes mono platform;
- Support paid;
- End Of Life.

Open Source tools Advantages:

- Ease of installation and upgrade;
- Bug Fixes;
- Good documentation;
- Not expensive;
- Not proprietary;
- Development supported by large community;
- Support free.

Open Source tools Disadvantages:

- Sometimes mono platform;
- End Of Life.

I have done this exercise with a few topics to demonstrate that nowadays the open source tools can be as good as the commercial ones or better.

Summary

Exploring all Vega features can make the article a book, but with this little exercise, we can see that Subgraph Vega showed itself an extraordinary web scanner tool that can be useful for detecting and analyzing vulnerabilities in a web environment.

It requires some experience of the cyber security specialist to deal with the results by interpreting them correctly.

We know there is a wide range of open source and commercial applications that can assist us in vulnerability analysis inside web environments beyond Vega, but if you are looking for one good web scanner to start analyzing a web environment, I suggest you to also consider Vega.

References

https://samate.nist.gov/docs/webapp_scanner_spec_sp500-269.pdf Access in December 30, 2016.

https://subgraph.com/vega/documentation/index.en.html Access in December 30, 2016.

https://hc.apache.org/httpcomponents-core-ga/httpcore/apidocs/org/apache/http/HttpHost Access in December 30, 2016.

Author: Washington Umpierres de Almeida Junior

Washington Almeida is an Electronic Engineer specialized in Cyber Security with more than 25 years of experience in the Information Technology and Engineering areas, working for large companies in the sectors as Engineering, Information Technology, Consulting, Chemical and Mining. Microsoft Enginner and Cisco Certified acts as Digital Forensic with in-depth knowledge of computer hardware, network technologies, telephony, programming, data communication protocols and a vast amount of information security knowledge with a set of skills known by ethical hackers, where this knowledge base is fundamental to assist the Justice.

JOB OFFER

ER: Caendra.

Job: VP of Business Development in Santa Clara, CA.

Responsible for sales & business dvlpmnt of IT security SW in B2B services & prod setting. Supervise business dvlpment & sales roles.
Reqs: Bach in Bus Admin, Comm, Mktg, Ind Eng, CS, or sim + 5 yrs exp.
Dom & int'l travel up to 5%.

Apply to: jobs@elearnsecurity.com and ref. Job Code BD-300.