

Smart Penetration Testing System

Supervised by
Dr. Mahmoud Mostafa

By:

**Mohamed Hamed Fawaz,
Basem Mohamed Ali Ezzat,
Dina Ashraf Taha,
and Ahmed Ali Labib.**

Acknowledgments

A Few people have influenced how we think about Security and Software Engineering. Many amusing discussions have helped us shape and improve our views and perception, so thanks to Prof. Mostafa Sami and Dr. Mahmoud Mostafa

Contents

1. Introduction 1

1.1.	Motivation	2
1.2.	Research goal.....	3
1.3.	Approach.....	5
1.4.	Problem definition.....	6
1.5.	Purpose of the system.....	7
1.6.	Scope of the system.....	7
1.7.	Objectives.....	7
1.8.	Working Glossary.....	8

2. Tools and Techniques..... 9

•	Programming Language.....	9
•	OpenVAS.....	10
•	Metasploit framework.....	13
•	SQLite	16
•	SQLAlchemy.....	18
•	Nmap.....	19
•	The ElementTree XML API.....	22
•	BackTrack.....	23
•	Kali Linux.....	25
•	Vulnerability scanner.....	25
•	Pull Parsing.....	26
•	MessagePack.....	27
•	Remote procedure call.....	28
•	Object-relational mapping.....	28

3. System Requirements Specifications.....	30
• Introduction.....	30
• Functional Requirements.....	30
• Nonfunctional Requirements.....	30
• Target Environment.....	31
• System Models.....	31
▪ The Scenarios.....	31
▪ Use case model.....	34
▪ Dynamic model.....	41
○ Create User.....	41
○ Add Target.....	42
○ Setting Scanning Configuration.....	42
○ Start Scanning.....	43
○ Ending the scanner operation.....	44
○ Starting Exploitation.....	44
○ Exploiting.....	45
○ Saving Final Report.....	45
▪ Analysis Object model.....	47
4. System Design Specification.....	51
4.1. Introduction.....	51
4.2. Classes.....	52
4.3. Database Design Specification.....	61
4.4. User Interface Design.....	63
Bibliography.....	69

Chapter 1

Introduction

In these modern days, our daily lives rely on information technology; transactions, social communications, education, and what define a modern life are all so related to modern technology. Data flows through massive networks, processed and stored in lots of computing devices and different forms of data stores. To insure the security of data, lots of approaches have been used, like firewalls, proxies, anti-viruses, encryption methods, and more.

However, threats aren't always blocked by using those methods, and the perfect system can never exist. Sometimes an outsider or attacker can use an invisible flaw (vulnerability) that wasn't seen during building the system to be able to compromise it or steal data. Those vulnerabilities can't be detected easily unless by a well-studied method or set of methods. Once they are detected, they can be fixed so that no one can use them for his own advantage.

Many system owners care to hire security experts to perform what is called a penetration testing on their systems; it's a methodology of pre-defined steps to test any system for its ability to defend itself against any possible attack. The advantage of penetration testing comes in its final results, where the system administrator gets a full report with what was done and what are the vulnerabilities discovered in the system; so that they can be batched or protected by any available method.

1.1. Motivation

For a clearer picture of the risks to IT security, the term "penetration test" and the methods used for testing were established in 1995 when the first Unix-based vulnerability scanner "SATAN" was introduced. At that time the program was the first tool that was able to automatically scan computers to identify vulnerabilities.

Nowadays, there are a number of free-ware and commercial vulnerability scanners, most of which have an updatable database of known hardware and software vulnerabilities. These tools are a convenient way of identifying vulnerabilities in the systems being tested and therefore of determining the risks involved. Penetration testing is also referred as Pen Testing or White Hat Attack.

Penetration testing importance can be specified in some points as follows:

- Understand and reduce the impact, frequency, and sternness of security incidents.
- Meet compliance and regulatory requirements that require security assessments.
- Optimize and prioritize resources used to remediate vulnerabilities.
- Gain peace of mind about security safeguards, controls, and policies.
- Identifies vulnerabilities and risks in your networking infrastructure.
- Validates the effectiveness of current security safeguards.

- Quantifies the risk to internal systems and confidential information.
- Raises executive awareness of corporate liability.
- Provides detailed remediation steps to prevent network compromise.
- Validates the security of system upgrades.
- Protects the integrity of online assets.
- Helps to achieve and maintain compliance with federal and state regulations.
- Using an automated product allows you to consistently test your network and easily integrate the practice with your overall security program This means you'll have more confidence in the overall security of network.
- It will give information about the how much information is publicly available.

It's clear that it's important to apply penetration testing frequently on systems, but penetration tests are complex and most of it is done manually by the help of some tools, not to mention their complexity. Pen testers collect a different set of tools carefully to use them together in the process, where each step may require one or more tool. From information gathering and network mapping tools to exploitation and post exploitation tools. A pen tester should also choose a well-designed methodology to apply the test on the system to minimize his errors and maximize the effectiveness of the test; also try to minimize the risks of infecting the system with any malicious code.

Our aim is to decrease the effort done by pen testers and make pen testing easier and takes less time, by automating the whole process

from the first to the last step in penetration testing considering all the chances of testing either a complete unknown system or if there are some information already known by the tester. The tool we will present is supposed to give the pen tester full control over the process and at the same time make things a lot easier and faster, he has the option to make manual customization in the steps or just use them as they are.

1.2. Research goal:

We hope to enhance the penetration testing process by introducing a cheap and easy way for users to test their systems for security flaws without requiring any experience in information security. Our final product is a tool that automates the whole process from the start to the end. And offer security experts more aid in the penetration testing process to make it a lot easier and more accurate.

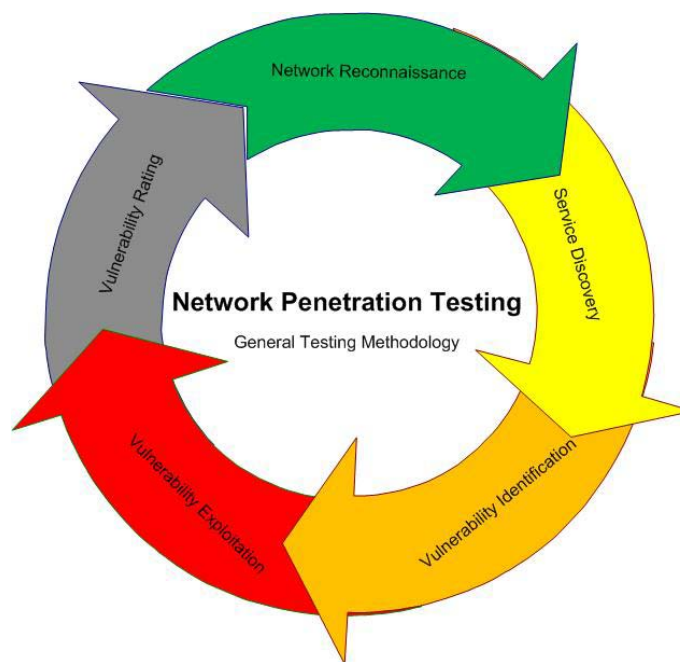


Figure 1--1 Penetration Testing General Methodolgy

1.3. Approach

We focus on studying the penetration testing process and identify the commonly faced problems and complexities to try to solve them in a well designed methodology and automate it to decrease effort, time, and human errors made in a usual manual process.

To identify the meaning of a methodology, a penetration testing methodology is a set of steps followed by a pen tester during his test, it's chosen according to the problem faced, the system being tested, and the effectiveness of that methodology and how suitable it is. A good pen tester should follow a methodology that suits the system he's testing; methodologies are good for making a clean test and for the final reports shown to the system administrator or the system owner at the end of the test to be aware of what was done during the test.

So we can say that our first step is learning about the penetration testing process and it's problems to identify them, then the second step is trying to make it better by our tool that uses the most useful known suitable tools used for pen testing process using Python programming language, and the last step is to create a new penetration testing methodology that's based on past, already existing methodologies with lots of improvement and customizations so that the new methodology would cover the gaps and fit our goal.

We plan to use Python as a programming language for implementing our tool as it's proved to be the most efficient language to be used and was recommended by most penetration testers, security experts, and

hackers. Python is an easy to use, cross platform, well designed, and well documented scripting language with lots of capabilities to offer to any security expert.

As for the tool itself, it will be like a core that connects and uses other tools used for pen testing by human pen testers like network mappers and vulnerability assessment tools and other, and adds some more functions like generating reports. It should perform the pen testing process automatically from the beginning to the exploitation phase, where it shows the tester a report with all results found. Results are IPs found, applications, available ports... etc. and a list of searched vulnerabilities that are probably in the system because of the found results and a list of verified vulnerabilities (They really exist in the system). The tester then chooses to either exploit manually or let the tool do the exploitation, the same goes for the post exploitation as well.

1.4. Problem definition

In Egypt, There is a lack of security in most of small and medium sized organizations, due to the high budget of security experts and from the most expensive tasks is the penetration testing.

Each Organization, have to hire a penetration tester (*at least*) or may be a team, they have to do a fully seven days work (*at least*), the servers, the network and the company don't operate at that time, also there is a threat presented as the penetration tester may be exposed to sensitive information about the servers, the buildings and the whole network.

1.5. Purpose of the system

Penetration testing is complicated, difficult, and exhausting job for both the testers and the corporation, and its automation (partially or totally) considered a great job.

Here we provide an automation technique for the penetration testing process, using a vulnerability scanner and an exploitation engine and our code to interact with them and save the results in our database.

1.6. Scope of the system

The system's scope is wide ranged; it is for the users that information security is important for them, from the novice system or network administrator to the penetration testers and security administrator

1.7. Objectives

- Black Swan is an easy-to-use automated penetration testing system, designed to be used by novice or inexperienced System administrators, Network administrators, or may be an amateur user.
- Provides a usable and easy Graphical User Interface.
- Provides an easy-to-install and update penetration testing system.
- Provides an automated penetration testing criteria.

1.8. Working Glossary

Table 1-1 Working Glossary

B.S	Black Swan
Scanner	[Scanning tool] an external program that perform the scanning task and it will be OpenVas.
Exploiter	[Exploitation tool] an external program that perform the exploitation task.
UUID	<p>Universally unique identifier</p> <p>A UUID is a 16-<u>octet</u> (128-<u>bit</u>) number. In its <u>canonical</u> form, a UUID is represented by 32 <u>hexadecimal</u> digits, displayed in five groups separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters</p> <p>(32 alphanumeric characters and four hyphens). For example:</p> <p>550e8400-e29b-41d4-a716-446655440000</p>

Chapter 2

Tools and Techniques

- **Programming Language**

Python 2.7.5

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts. Using third-party tools, Python code can be packaged into standalone executable programs. Python interpreters are available for many operating systems.

- **OpenVAS**

OpenVAS stands for Open Vulnerability Assessment System and represents comprehensive tool-chain for network security scanning including a graphical user front-end and incorporating various third-party security applications. The core is a server component with a set of Network Vulnerability Tests (NVTs) to detect security problems in remote systems and applications. All OpenVAS products are Free Software under the GNU General Public License (GNU GPL). OpenVAS derives from the Nessus project which turned into a proprietary product, but has progressed on its own since then.

The OpenVAS software consists of five distinct parts which are provided and maintained by the OpenVAS projects. The individual parts are:

OpenVAS-Server: This is the core component of OpenVAS. It contains the functionality used for scanning a large number of target servers at a high speed. Scans will always originate from the host where

OpenVAS-Server is running; therefore, this machine has to be able reach the intended targets.

The server requires three other modules:

OpenVAS-Libraries: This module contains functionality used by OpenVAS-Server.

OpenVAS-LibNASL: The NVTs are written in the “Nessus Attack Scripting Language” (NASL). This module contains the functionality needed by OpenVAS-Server to interface with NASL.

OpenVAS-Plugins: This module contains a base set of NVTs. Please be aware that the update cycle of this module is not intended to ensure the availability of the most recent NVTs. If you need up-to-date NVTs you should consider subscribing to a NVT feed.

OpenVAS-Client: OpenVAS-Client controls the OpenVAS server, processes the scan results and displays them to the user. OpenVAS-Client can run on any machine able to connect to the OpenVAS-Server and can control multiple servers.

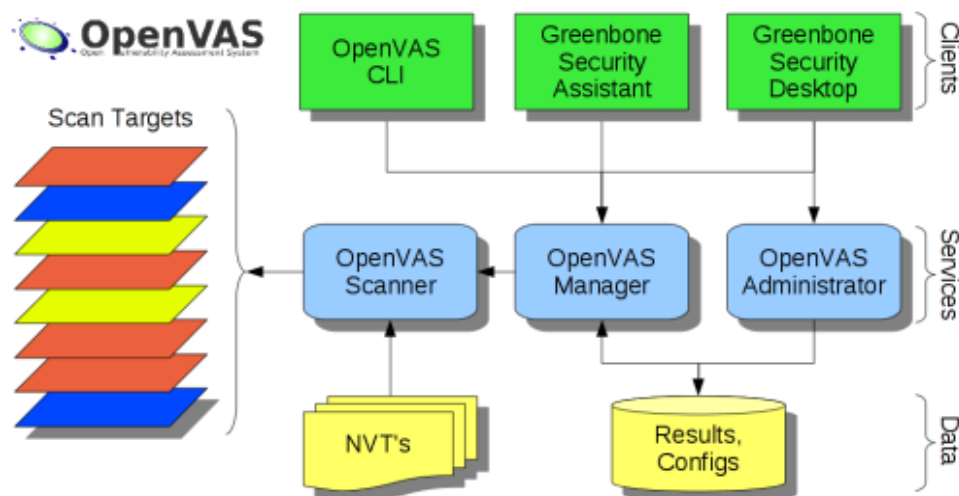


Figure 2-1 OpenVas Architecture

Scanning can be initiated at various levels. By default OpenVAS has four scanning options:

Full and fast - Exploits the majority of NVT's. And is optimized through the use of information previously collected.

Full and fast completed - Exploits the majority of NVT's, between them there are some that may cause a shutdown of the service/remote system. This profile is optimized by the use of information previously collected.

Full and very deep - Exploits most of NVT's but is slower because it doesn't use the information previously collected.

Full and very deep ultimate - Exploits the majority of NVT's, between them there are some that may cause a shutdown of the service/remote system. This profile is slower because it doesn't use the information previously collected.

- **Metasploit framework**

A tool for developing and executing exploit code against a remote target machine. The Metasploit Framework provides a truly impressive work environment. The MSF is far more than just a collection of exploits, it's an infrastructure that you can build upon and utilize for your custom needs. This allows you to concentrate on your unique environment, and not have to reinvent the wheel.

The MSF filesystem is laid out in an intuitive manner and is organized by directory.

- **data:** editable files used by Metasploit
- **documentation:** provides documentation for the framework
- **external:** source code and third-party libraries
- **lib:** the 'meat' of the framework code base
- **modules:** the actual MSF modules
- **plugins:** plugins that can be loaded at run-time
- **scripts:** Meterpreter and other scripts
- **tools:** various useful command-line utilities

Libraries

- **Rex**
 - The basic library for most tasks
 - Handles sockets, protocols, text transformations, and others
 - SSL, SMB, HTTP, XOR, Base64, Unicode
- **Msf::Core**
 - Provides the 'basic' API
 - Defines the Metasploit Framework
- **Msf::Base**
 - Provides the 'friendly' API
 - Provides simplified APIs for use in the Framework

Metasploit, as presented to the user, is composed of modules.

Exploits:

- Defined as modules that use payloads
- An exploit without a payload is an Auxiliary module

Payloads, Encoders, Nops:

- Payloads consist of code that runs remotely
- Encoders ensure that payloads make it to their destination
- Nops keep the payload sizes consistent.

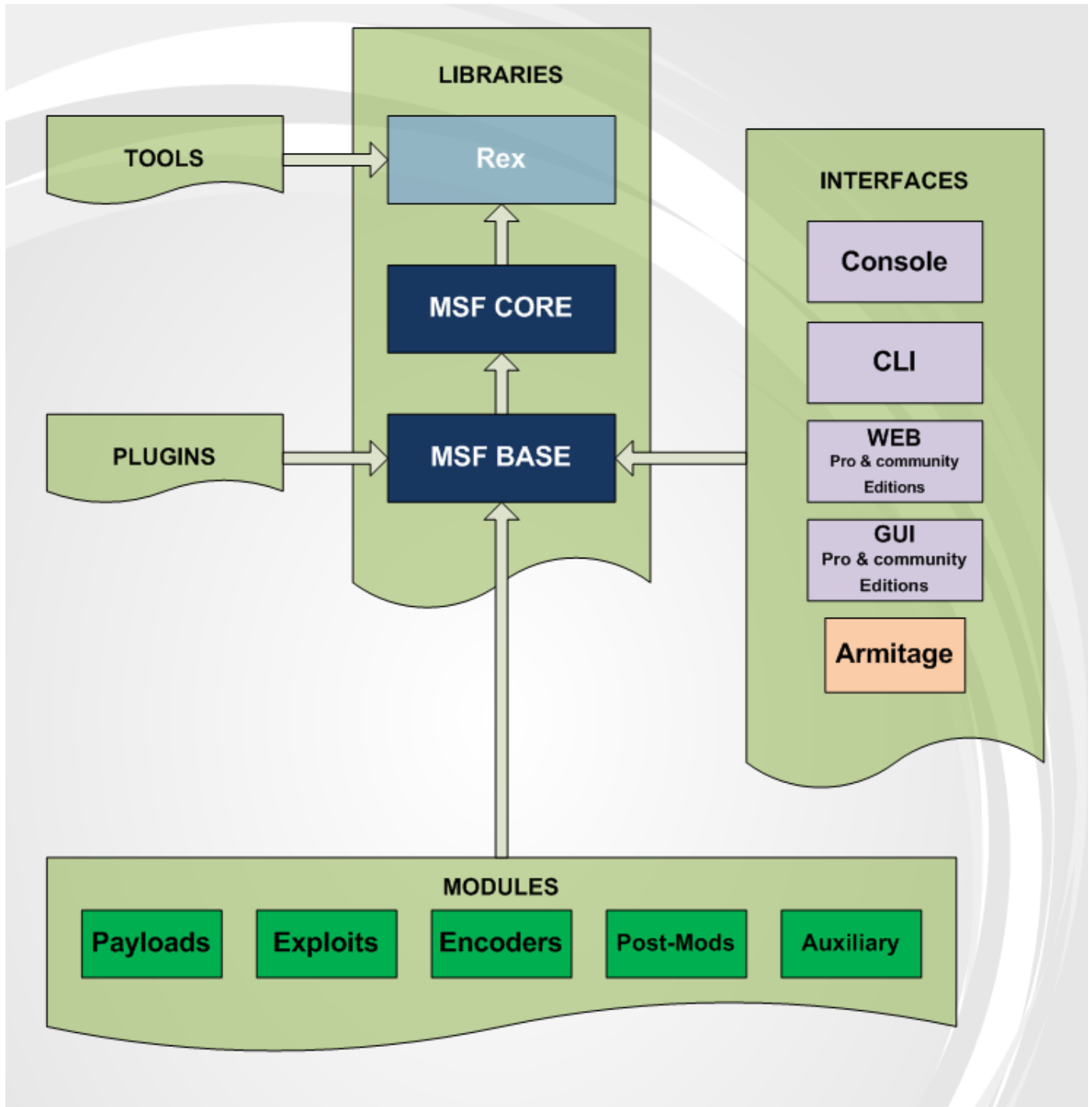


Figure2-2 MSF Architecture

- **SQLite:**

It is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is currently found in more applications than we can count, including several high-profile projects.

SQLite is an embedded SQL database engine. Unlike most other SQL databases, SQLite does not have a separate server process. SQLite reads and writes directly to ordinary disk files. A complete SQL database with multiple tables, indices, triggers, and views, is contained in a single disk file. The database file format is cross-platform - you can freely copy a database between 32-bit and 64-bit systems or between big-endian and little-endian architectures.

These features make SQLite a popular choice as an Application File Format. Think of SQLite not as a replacement for Oracle but as a replacement for fopen().

SQLite is a compact library. With all features enabled, the library size can be less than 500KiB, depending on the target platform and compiler optimization settings. (64-bit code is larger. And some compiler optimizations such as aggressive function inlining and loop unrolling can cause the object code to be much larger.) If optional features are omitted, the size of the SQLite library can be reduced

below 300KiB. SQLite can also be made to run in minimal stack space (4KiB) and very little heap (100KiB), making SQLite a popular database engine choice on memory constrained gadgets such as cellphones, PDAs, and MP3 players. There is a tradeoff between memory usage and speed. SQLite generally runs faster the more memory you give it. Nevertheless, performance is usually quite good even in low-memory environments.

SQLite is very carefully tested prior to every release and has a reputation for being very reliable. Most of the SQLite source code is devoted purely to testing and verification. An automated test suite runs millions and millions of test cases involving hundreds of millions of individual SQL statements and achieves 100% branch test coverage. SQLite responds gracefully to memory allocation failures and disk I/O errors. Transactions are ACID even if interrupted by system crashes or power failures. All of this is verified by the automated tests using special test harnesses which simulate system failures. Of course, even with all this testing, there are still bugs. But unlike some similar projects (especially commercial competitors) SQLite is open and honest about all bugs and provides bugs lists including lists of critical bugs and minute-by-minute chronologies of bug reports and code changes.

- **SQLAlchemy**

It is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

SQL databases behave less like object collections the more size and performance start to matter; object collections behave less like tables and rows the more abstraction starts to matter. SQLAlchemy aims to accommodate both of these principles.

SQLAlchemy considers the database to be a relational algebra engine, not just a collection of tables. Rows can be selected from not only tables but also joins and other select statements; any of these units can be composed into a larger structure.

SQLAlchemy's expression language builds on this concept from its core.

SQLAlchemy is most famous for its object-relational mapper (ORM), an optional component that provides the data mapper pattern, where classes can be mapped to the database in open ended, multiple ways - allowing the object model and database schema to develop in a cleanly decoupled way from the beginning.

- **Nmap**

Nmap ("Network Mapper") is a free and open source (license) utility for network discovery and security auditing. Many systems and network administrators also find it useful for tasks such as network inventory, managing service upgrade schedules, and monitoring host or service uptime. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. It was designed to rapidly scan large networks, but works fine against single hosts. Nmap runs on all major computer operating systems, and official binary packages are available for Linux, Windows, and Mac OS X. In addition to the classic command-line Nmap executable, the Nmap suite includes an advanced GUI and results viewer (Zenmap), a flexible data transfer, redirection, and debugging tool (Ncat), a utility for comparing scan results (Ndiff), and a packet generation and response analysis tool (Nping).

Nmap was named "Security Product of the Year" by Linux Journal, Info World, LinuxQuestions.Org, and Codetalker Digest.

It was even featured in twelve movies, including The Matrix Reloaded, Die Hard 4, Girl With the Dragon Tattoo, and The Bourne Ultimatum.

Nmap is ...

Flexible: Supports dozens of advanced techniques for mapping out networks filled with IP filters, firewalls, routers, and other obstacles. This includes many port scanning mechanisms (both TCP & UDP), OS detection, version detection, ping sweeps, and more. See the documentation page.

Powerful: Nmap has been used to scan huge networks of literally hundreds of thousands of machines.

Portable: Most operating systems are supported, including Linux, Microsoft Windows, FreeBSD, OpenBSD, Solaris, IRIX, Mac OS X, HP-UX, NetBSD, Sun OS, Amiga, and more.

Easy: While Nmap offers a rich set of advanced features for power users, you can start out as simply as "nmap -v -A targethost". Both traditional command line and graphical (GUI) versions are available to suit your preference. Binaries are available for those who do not wish to compile Nmap from source.

Free: The primary goals of the Nmap Project is to help make the Internet a little more secure and to provide administrators/auditors/hackers with an advanced tool for exploring their networks. Nmap is available for free download, and also comes with full source code that you may modify and redistribute under the terms of the license.

Well Documented: Significant effort has been put into comprehensive and up-to-date man pages, whitepapers, tutorials, and even a whole book! Find them in multiple languages here.

Supported: While Nmap comes with no warranty, it is well supported by a vibrant community of developers and users. Most of this interaction occurs on the Nmap mailing lists. Most bug reports and questions should be sent to the nmap-dev list, but only after you read the guidelines. We recommend that all users subscribe to the low-traffic nmap-hackers announcement list. You can also find Nmap on Facebook and Twitter. For real-time chat, join the #nmap channel on Freenode or EFNet.

Acclaimed: Nmap has won numerous awards, including "Information Security Product of the Year" by Linux Journal, Info World and Codetalker Digest. It has been featured in hundreds of magazine articles, several movies, dozens of books, and one comic book series. Visit the press page for further details.

Popular: Thousands of people download Nmap every day, and it is included with many operating systems (Redhat Linux, Debian Linux, Gentoo, FreeBSD, OpenBSD, etc). It is among the top ten (out of 30,000) programs at the Freshmeat.Net repository. This is important because it lends Nmap its vibrant development and user support communities.

- **The ElementTree XML API**

The Element type is a flexible container object, designed to store hierarchical data structures in memory. The type can be described as a cross between a list and a dictionary.

Each element has a number of properties associated with it:

- a tag which is a string identifying what kind of data this element represents (the element type, in other words).
- a number of attributes, stored in a Python dictionary.
- a text string.
- an optional tail string.
- a number of child elements, stored in a Python sequence

To create an element instance, use the Element constructor or the SubElement() factory function.

The ElementTree class can be used to wrap an element structure, and convert it from and to XML.

- **BackTrack**

BackTrack is a distribution based on the Debian GNU/Linux distribution aimed at digital forensics and penetration testing use.

BackTrack provides users with easy access to a comprehensive and large collection of security-related tools ranging from port scanners to Security Audit. Support for Live CD and Live USB functionality allows users to boot BackTrack directly from portable media without requiring installation, though permanent installation to hard disk and network is also an option.

BackTrack includes many well known security tools including:

- Metasploit for integration
- Wi-Fi drivers supporting monitor mode (rfmon mode) and packet injection
- Aircrack-ng
- Gerix Wifi Cracker
- Kismet
- Nmap
- Ophcrack
- Ettercap
- Wireshark (formerly known as Ethereal)
- BeEF (Browser Exploitation Framework)
- Hydra

- OWASP Mantra Security Framework, a collection of hacking tools, add-ons and scripts based on Firefox
- Cisco OCS Mass Scanner, a very reliable and fast scanner for Cisco routers with telnet and enabling of a default password.
- A large collection of exploits as well as more commonplace software such as browsers.

BackTrack arranges tools into 12 categories:

- Information gathering
- Vulnerability assessment
- Exploitation tools
- Privilege escalation
- Maintaining access
- Reverse engineering
- RFID tools
- Stress testing
- Forensics
- Reporting tools
- Services
- Miscellaneous

- **Kali Linux**

Kali is a complete re-build of BackTrack Linux, adhering completely to Debian development standards. All-new infrastructure has been put in place, all tools were reviewed and packaged, and we use Git for our VCS.

- **Vulnerability scanner**

A vulnerability scanner is a computer program designed to assess computers, computer systems, networks or applications for weaknesses. There are a number of types of vulnerability scanners available today, distinguished from one another by a focus on particular targets. While functionality varies between different types of vulnerability scanners, they share a common, core purpose of enumerating the vulnerabilities present in one or more targets. Vulnerability scanners are a core technology component of vulnerability management.

Types of Vulnerability Scanners:

- Port scanner [[ex: Nmap, Nessus]]
- Network enumerator
- Network vulnerability scanner [[ex: BoomScan]]
- Web application security scanner
- Database security scanner
- ERP security scanner

- Computer worm
- **Pull Parsing**

Pull parsing treats the document as a series of items which are read in sequence using the Iterator design pattern. This allows for writing of recursive-descent parsers in which the structure of the code performing the parsing mirrors the structure of the XML being parsed, and intermediate parsed results can be used and accessed as local variables within the methods performing the parsing, or passed down (as method parameters) into lower-level methods, or returned (as method return values) to higher-level methods. Examples of pull parsers include StAX in the Java programming language, XMLReader in PHP, ElementTree.iterparse in Python, System.Xml.XmlReader in the .NET Framework, and the DOM traversal API (NodeIterator and TreeWalker).

A pull parser creates an iterator that sequentially visits the various elements, attributes, and data in an XML document. Code which uses this iterator can test the current item (to tell, for example, whether it is a start or end element, or text), and inspect its attributes (local name, namespace, values of XML attributes, value of text, etc.), and can also move the iterator to the next item. The code can thus extract information from the document as it traverses it. The recursive-descent approach tends to lend itself to keeping data as typed local variables in the code doing the parsing, while SAX, for instance, typically requires a parser to manually maintain intermediate data within a stack of

elements which are parent elements of the element being parsed. Pull-parsing code can be more straightforward to understand and maintain than SAX parsing code.

- **MessagePack**

It is an efficient binary serialization format. It lets you exchange data among multiple languages like JSON but it's faster and smaller. For example, small integers (like flags or error code) are encoded into a single byte, and typical short strings only require an extra byte in addition to the strings themselves.

- The Features
 - Handling Primitive Types - integer, nil, boolean, float, double, raw
 - Handling Some Containers - array, map
 - Streaming Deserialization
 - Language Bindings
 - C, C++, Ruby, Python, Perl, PHP, Java, Haskell, D, Erlang, Lua
- Non-Features
 - User-defined record - simulate by array or map
 - Serialization with Cyclic Reference

- **Remote procedure call**

It is an inter-process communication that allows a computer program to cause a subroutine or procedure to execute in another address space (commonly on another computer on a shared network) without the programmer explicitly coding the details for this remote interaction. That is, the programmer writes essentially the same code whether the subroutine is local to the executing program, or remote. When the software in question uses object-oriented principles, RPC is called **remote invocation** or **remote method invocation**. Many different (often incompatible) technologies have been used to implement the concept.

- **Object-relational mapping**

Object-relational mapping (ORM, O/RM, and O/R mapping) in computer software is a programming technique for converting data between incompatible type systems in object-oriented programming languages. This creates, in effect, a "virtual object database" that can be used from within the programming language. There are both free and commercial packages available that perform object-relational mapping, although some programmers opt to create their own ORM tools.

Data management tasks in object-oriented (OO) programming are typically implemented by manipulating objects that are almost always non-scalar values. For example, consider an address book entry that represents a single person along with zero or more phone numbers and

zero or more addresses. This could be modeled in an object-oriented implementation by a "Person object" with attributes/fields to hold each data item that the entry comprises: the person's name, a list of phone numbers, and a list of addresses. The list of phone numbers would itself contain "PhoneNumber objects" and so on. The address book entry is treated as a single object by the programming language (it can be referenced by a single variable containing a pointer to the object, for instance). Various methods can be associated with the object, such as a method to return the preferred phone number, the home address, and so on.

However, many popular database products such as structured query language database management systems (SQL DBMS) can only store and manipulate scalar values such as integers and strings organized within tables. The programmer must either convert the object values into groups of simpler values for storage in the database (and convert them back upon retrieval), or only use simple scalar values within the program. Object-relational mapping is used to implement the first approach.

The heart of the problem is translating the logical representation of the objects into an atomized form that is capable of being stored on the database, while somehow preserving the properties of the objects and their relationships so that they can be reloaded as an object when needed. If this storage and retrieval functionality is implemented, the objects are then said to be persistent.

Chapter 3

System Requirements Specifications

- **Introduction**

This chapter describe the Black Swan system requirements specifications include scenarios, use case model, Object model and Dynamic model

- **Functional Requirements**

Black Swan supports one Type of Users:

- The Tester provides the system with the IP of the machine to be tested and then starts the the B.S and then the B.S should be able to provide the tester by the results after every stage and raise the conflicts if exists. The Tester manages the whole operation.

- **Nonfunctional Requirements**

- *Usability*. Novice or amateur users should know how to use the Black Swan, Users should be familiar with the CLI.
- *Extensibility*. B.S will be extended to support more than one person, extended to support anonymity.
- *Maintainability*. The maintenance support will be by the application developers.
- *Packaging*. The user is responsible for installing the system using python scripts.

- **Legality.** The program will be open source.

- **Target Environment**

Black Swan should run on any Linux operating system but preferable to run on

Kali Linux (Preferred) & ***BackTrack 5 R3***

- **System Models**

- **The Scenarios**

Table 3-1 PenetrationTesting Scenario

<i>Scenario name</i>	PenetrationTesting
<i>Participating actor instances</i>	Bob:User, Scanner, Exploiter
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. Bob, wants to scan for vulnerabilities for a specific target, He starts the B.S which in turn presents to Bob a log-in form. 2. If it was the first time for Bob to use the B.S, he should create a new account otherwise he enters his user name and password. 3. Bob enters the IP address of the device he wants to scan. 4. The Scanner is informed, it takes the target information and create a scanning task and

	<p>performs the scanning and then create a scanning report.</p> <ol style="list-style-type: none"> 5. Bob receives periodically the percentage of the progress done on scanning. 6. Bob receives a vulnerability report after the scanning is finished. 7. Bob proceed to the next step (Exploitation). 8. After processing the vulnerability to know which will be exploited, Bob proceed by pressing “start exploiting” button. 9. While the Exploiter is functioning Bob receive periodically the progress of the Exploiter until it is finished 10. Bob receives the result in the final report.
--	--

Table 3-2 ScanningTarget Scenario

<i>Scenario name</i>	ScanningTarget
<i>Participating actor instances</i>	Bob:User, Scanner.
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. Bob, wants to scan for vulnerabilities for a specific target, He starts the B.S which in turn presents to Bob a log-in form. 2. If it was the first time for Bob to use the B.S, he should create a new account otherwise he enters his user name and password. 3. Bob enters the IP address of the device he wants to scan. 4. The Scanner is informed, it takes the target information and create a scanning task and performs the scanning and then create a scanning report. 5. Bob receives periodically the percentage of the progress done on scanning. 6. Bob receives a vulnerability report after the scanning is finished.

• Use case model

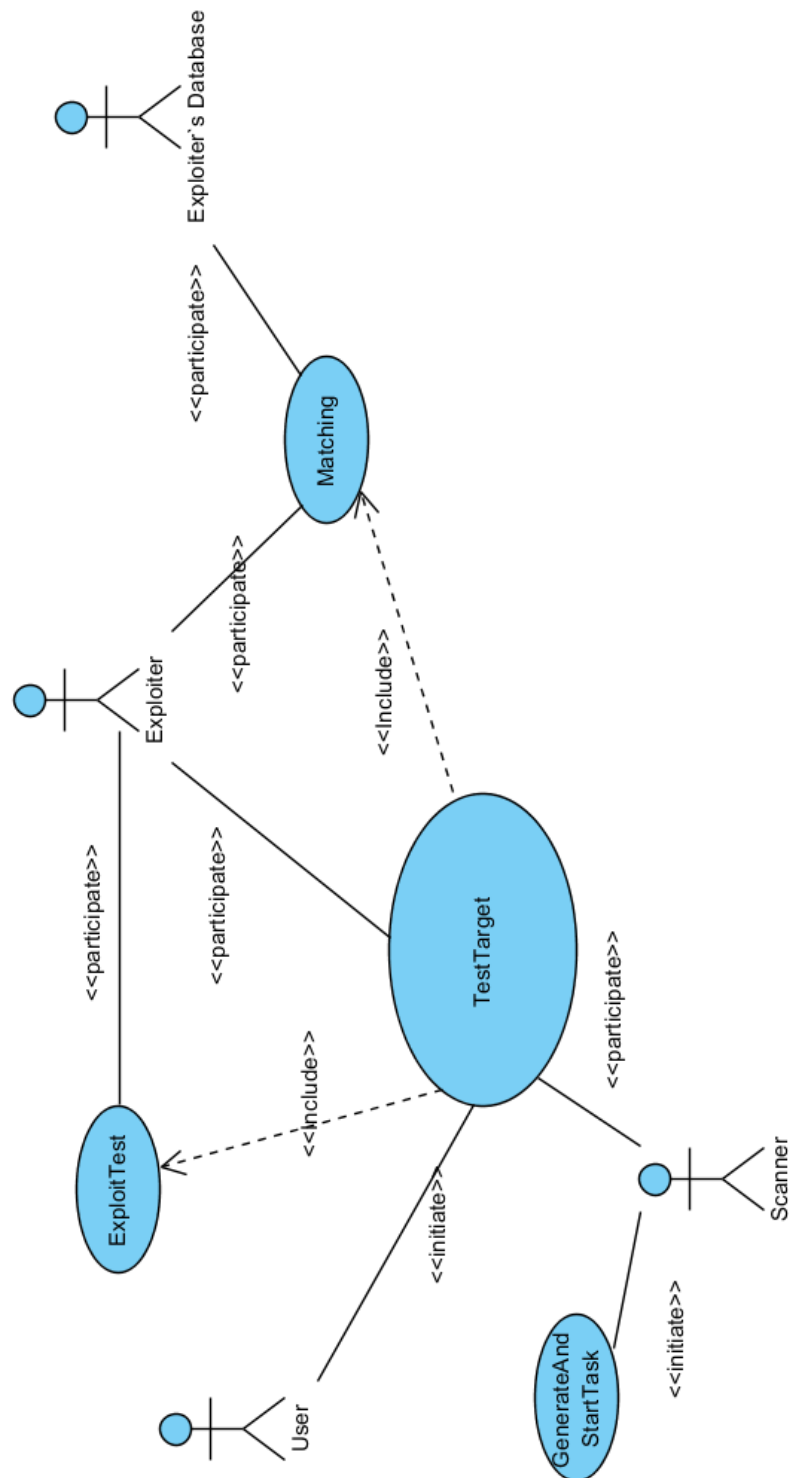


Figure 3-1 The Whole usecase model

Table 3-3 TestTarget Usecase

<i>Use case name</i>	TestTarget
<i>Participating actor</i>	Initiated by User Communicates with Scanner, Exploiter
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. The User starts the B.S. 2. B.S responds by presenting a log-in form to the User. The form includes two text fields one for the user name and the other for the password and a “sign up” button if it is the first time for the User to use B.S. 3. The User enters his user name and password. 4. If it is the first time for the User, B.S creates new user account by the given user name and password, if it is not the first time, B.S checks for the given password and present to the User a PC list in main interface of B.S, PC list contains previously scanned or added target and their information (OS, MAC, IP,) 5. The User is able to either enter a new target by entering its IP address or

	<p>chooses an existing one.</p> <p>6. B.S checks the IP address of the target to check if it is valid or not, if it is having a live connection or not and if it is existed in its database or not, and if not, it creates a new target by checking the configuration and with the IP address, B.S creates new target, if the IP address already exist, B.S proceed to the next step.</p> <p>7. The User then chooses the configuration which is default set to “Full and fast” and then starts the scan, invoke the GenerateAndStartTask which is initiated by the Scanner.</p> <p>8. After the Scanning is finished, B.S uses report UUID (acquired in the GenerateAndStartTask use case) and the file format UUID to send a request to Scanner to generate the report with the desired format.</p> <p>9. Scanner generates the report using the sent UUIDs</p>
--	--

	<p>10. B.S stores the report in its database and parse the results and present it (arranged by its threat level) to the User in a table that is in the Graphical User Interface, in this while B.S initiates the Matching use case which matches the vulnerabilities and the exploits.</p> <p>11. The User checks the results and proceed to the next step by pressing the glowing Exploitation tab, and then the User starts exploitation, invoke the exploit use case which is initiated by the Exploiter.</p> <p>12. After the exploit use case is finished, B.S takes the results from the Exploiter and shows the results to the User, save the final report in its database.</p> <p>13. The User can choose then to save it in any location.</p>
<i>Entry condition</i>	<ul style="list-style-type: none"> • The User runs the B.S.
<i>Exit conditions</i>	<ul style="list-style-type: none"> • B.S generates the final report.

Table 3-4 GenerateAndStartTask Usecase

<i>Use case name</i>	GenerateAndStartTask
<i>Participating actor</i>	Initiated by Scanner Communicates with B.S
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. B.S sends the target IP address that is set. 2. The Scanner takes the IP address and sends back the target UUID. 3. B.S sends the target UUID along with the configuration UUID (selected before by the USER) in a request to the Scanner to create a task. 4. The Scanner receives the target UUID and the configuration UUID and creates a task and returns the task's UUID. 5. B.S issues a command to the Scanner to start the task. 6. The Scanner starts the task, generate report UUID and sends the state of the task "Requested". 7. B.S monitors the task until it is finished.
<i>Entry condition</i>	<ul style="list-style-type: none"> • B.S sends the IP address to the Scanner.
<i>Exit conditions</i>	<ul style="list-style-type: none"> • The scanning task is created and its UUID is sent.

Table 3-5 Matching Usecase

<i>Name</i>	Matching
<i>Participating actor</i>	Initiated by B.S Communicates with Exploiter & Exploiter's database
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. B.S takes the vulnerabilities` CVE, BID and XREF numbers retrieved from the scanning report and sort them by rank (CVSS) and sends them to the Exploiter's database. 2. Exploiter's Database selects the matched exploits (by CVE, BID and XREF numbers) and returns them to B.S ordered by their ranks. 3. B.S takes the ordered exploits and sends them to the ExploitTest uses case.
<i>Entry condition</i>	<ul style="list-style-type: none"> • B.S requests the exploits.
<i>Exit conditions</i>	<ul style="list-style-type: none"> • B.S takes the ordered exploits

Table 3-6 ExploitTest Usecase

<i>Use case name</i>	ExploitTest
<i>Participating actor</i>	Initiated by B.S Communicates with Exploiter
<i>Flow of events</i>	<ol style="list-style-type: none"> 1. B.S sends only one exploit each time from the exploits list (chosen before) to the Exploiter. (B.S set the options and setting for each exploit), B.S should wait after sending the exploit to get its response from the Exploiter. 2. Exploiter receives commands for exploiting and then runs these commands. 3. B.S continues with step 1 until all exploits are sent to the Exploiter and their response are got each time B.S sends an exploit and then B.S restore all the responses for all the exploits in its database in order to be written in the final report. (The responses are divided to three types: <i>“Nothing”: There is a server that is down</i> <i>“Exploited”: Successful exploitation</i> <i>“Error”: Failed to exploit)</i>
<i>Entry condition</i>	<ul style="list-style-type: none"> • B.S sends an exploit to the Exploiter
<i>Exit conditions</i>	<ul style="list-style-type: none"> • B.S gets responses for all the exploits

• Dynamic Model

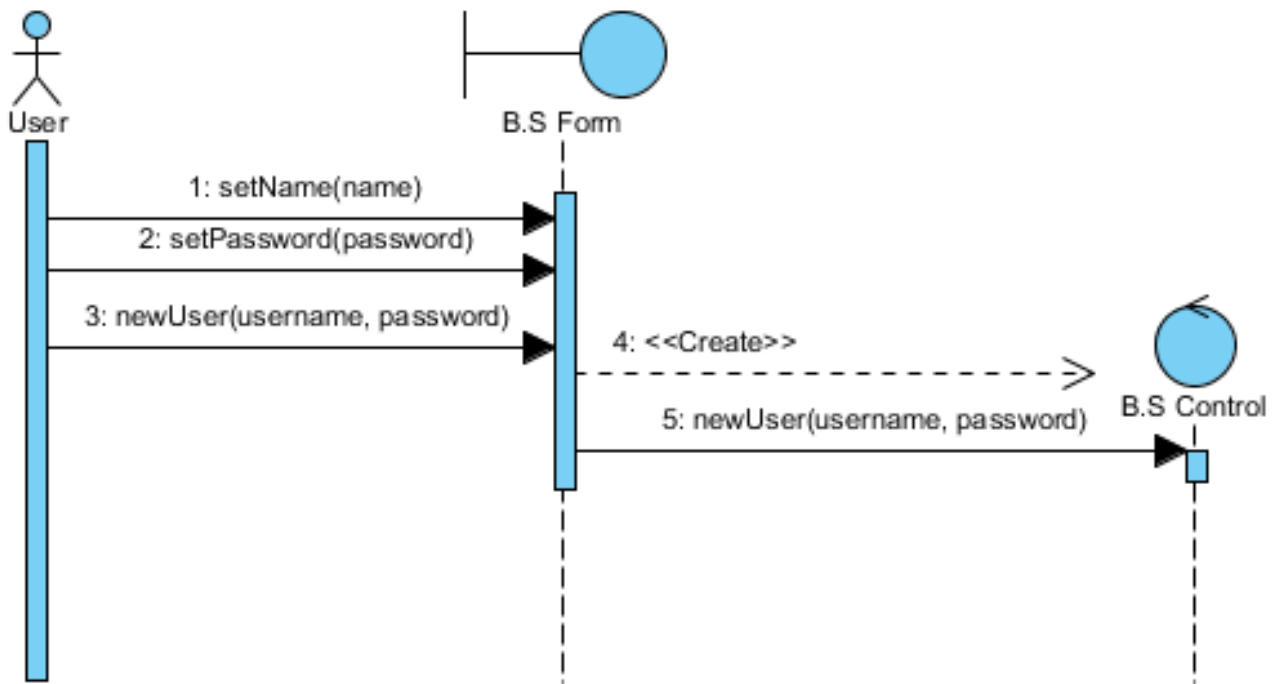


Figure 3-2 Creat User

○ Create User

The User start the Black Swan and set the name and password and username and then the black Swan creates the account using the information provided by the User

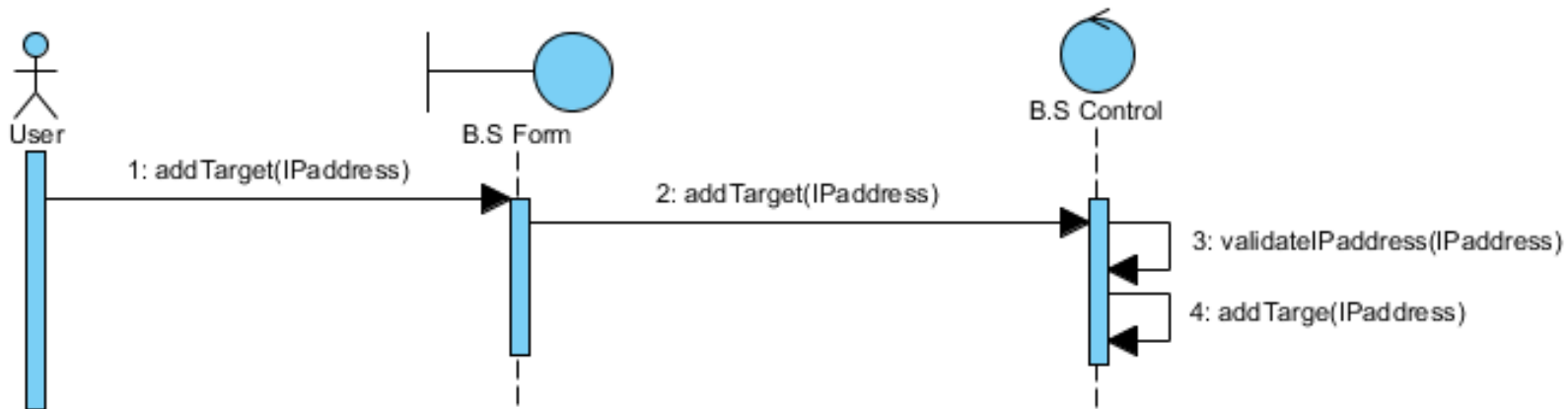


Figure 3-3 Add Target

○ Add Target

The User add target by inserting the IP address of the that target and then the Black swan will validate it if the IP is wrong or if the if the host have a live connection.

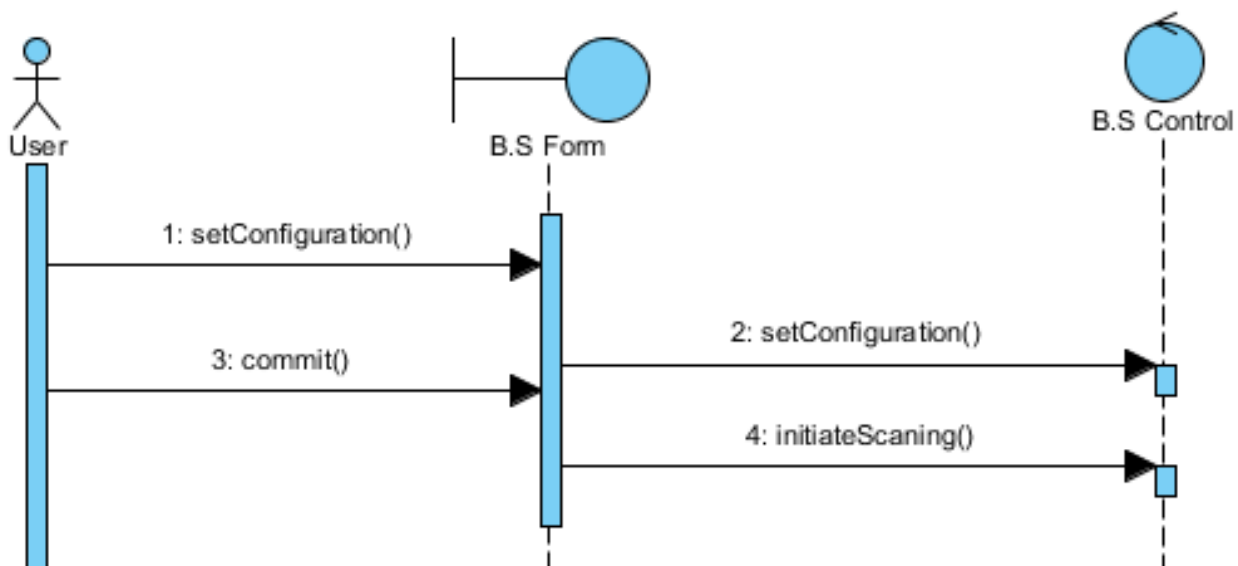


Figure 3-4 Setting Scanning Configuration

○ Setting Scanning Configuration

The User then set the configuration of the scanning and then proceed to scan

○ Start Scanning

The Black Swan sends the target IP address and then the Scanner responds by storing the the target IP and send its UUID to the B.S and then B.S send the target UUID and the Configuration UUID to add target in the scanner the Scanner responds by sending the Task UUID and the report UUID after the scanning is finished the B.S request the scanning report by sending the report UUID and the file formate UUID to the Scanner which respond by sending the report and then B.S parse the results from the report and sort the vulnerabilities.

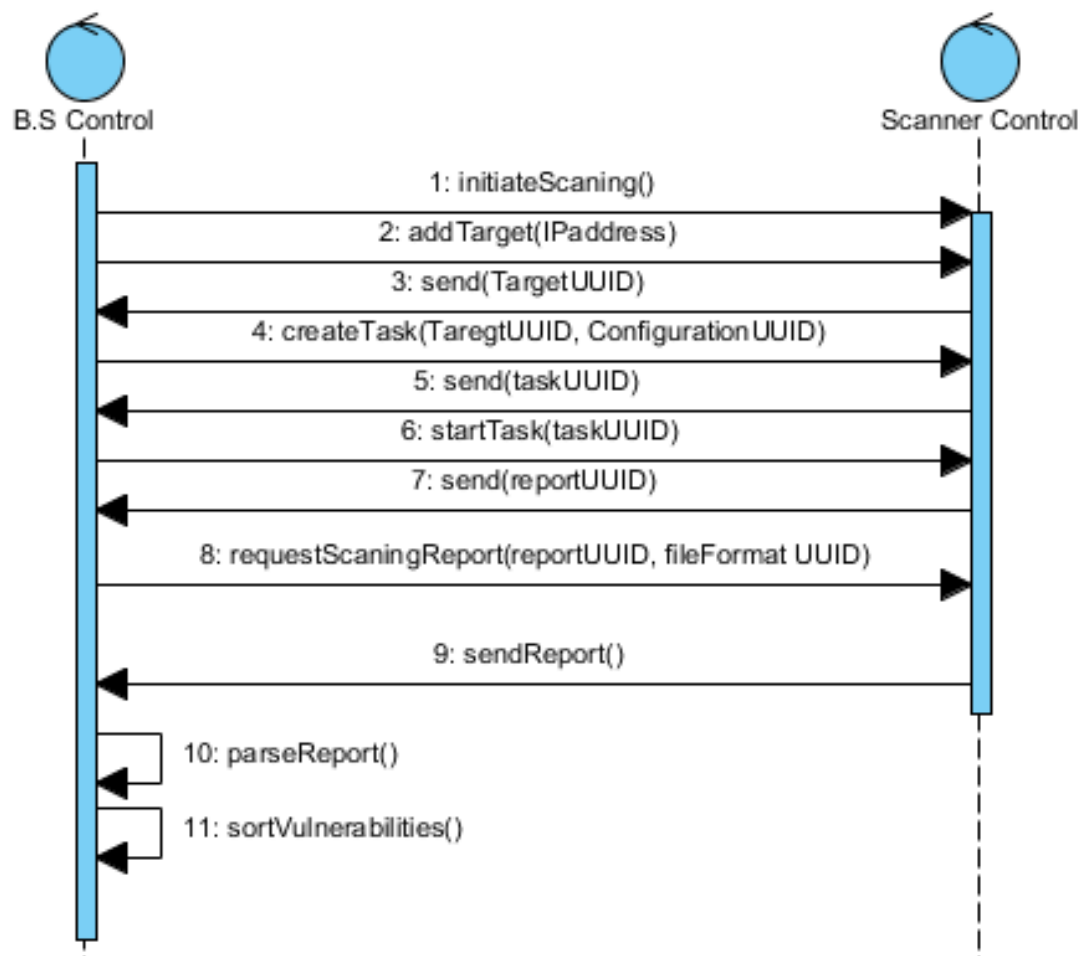


Figure 3 -5 Start Scanning

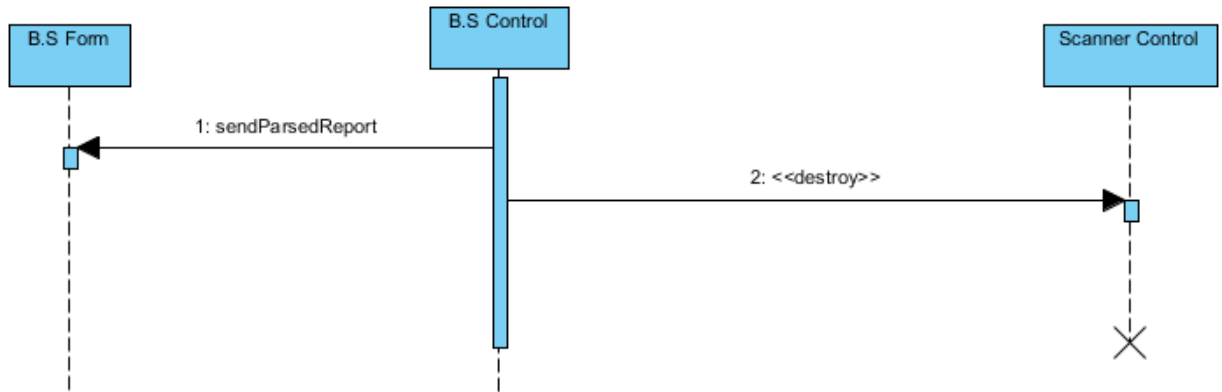


Figure 3--6 ending the Scanner operation

○ Ending the scanner operation

After the Black Swan shows the scanning report in its Interface it sends a destroying signal to the scanning operation

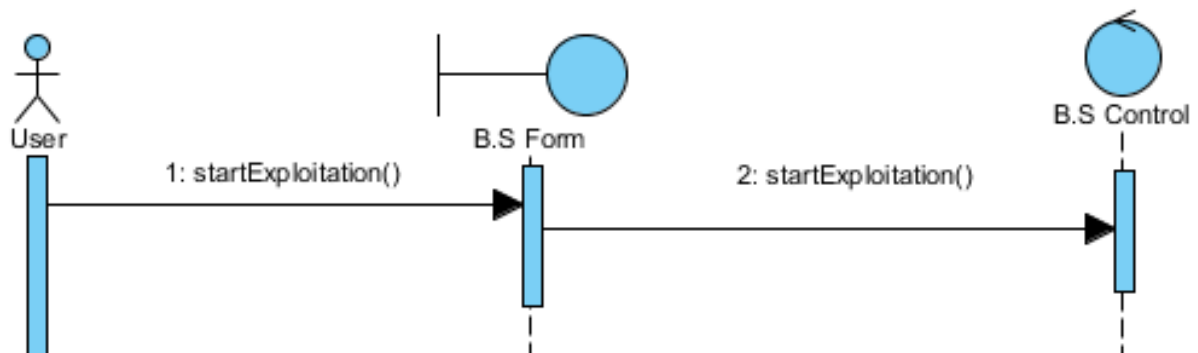


Figure 3--7 Starting Exploitation

○ Starting Exploitation

When the B.S shows the User the scanning report the User then Starts the Exploitation.

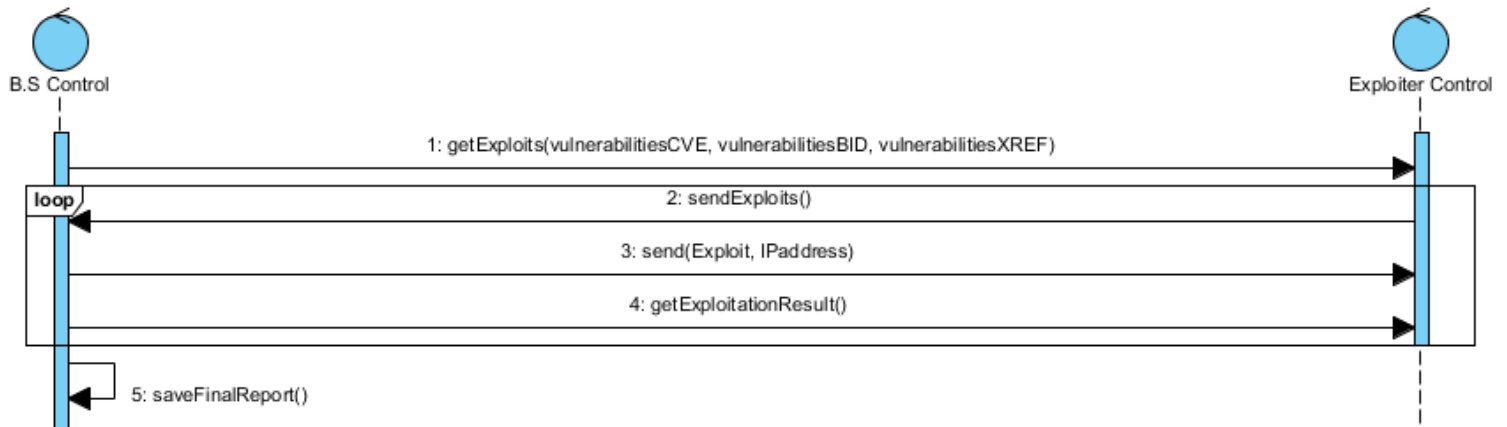


Figure 3-8 Exploiting

○ Exploiting

Exploiting the vulnerabilities is done through a loop sending only one exploits per time and taking the response and so on.

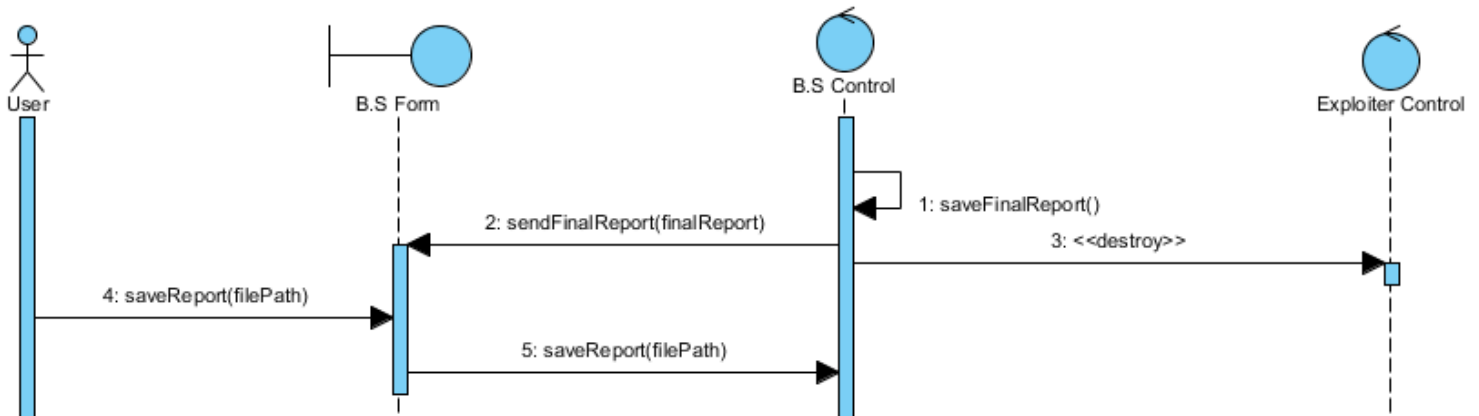


Figure 3-9 Saving Final Report

○ Saving Final Report

The Black Swan saves the final report and then shows it to the user in the interface and then the user save it in the location he wants in his device

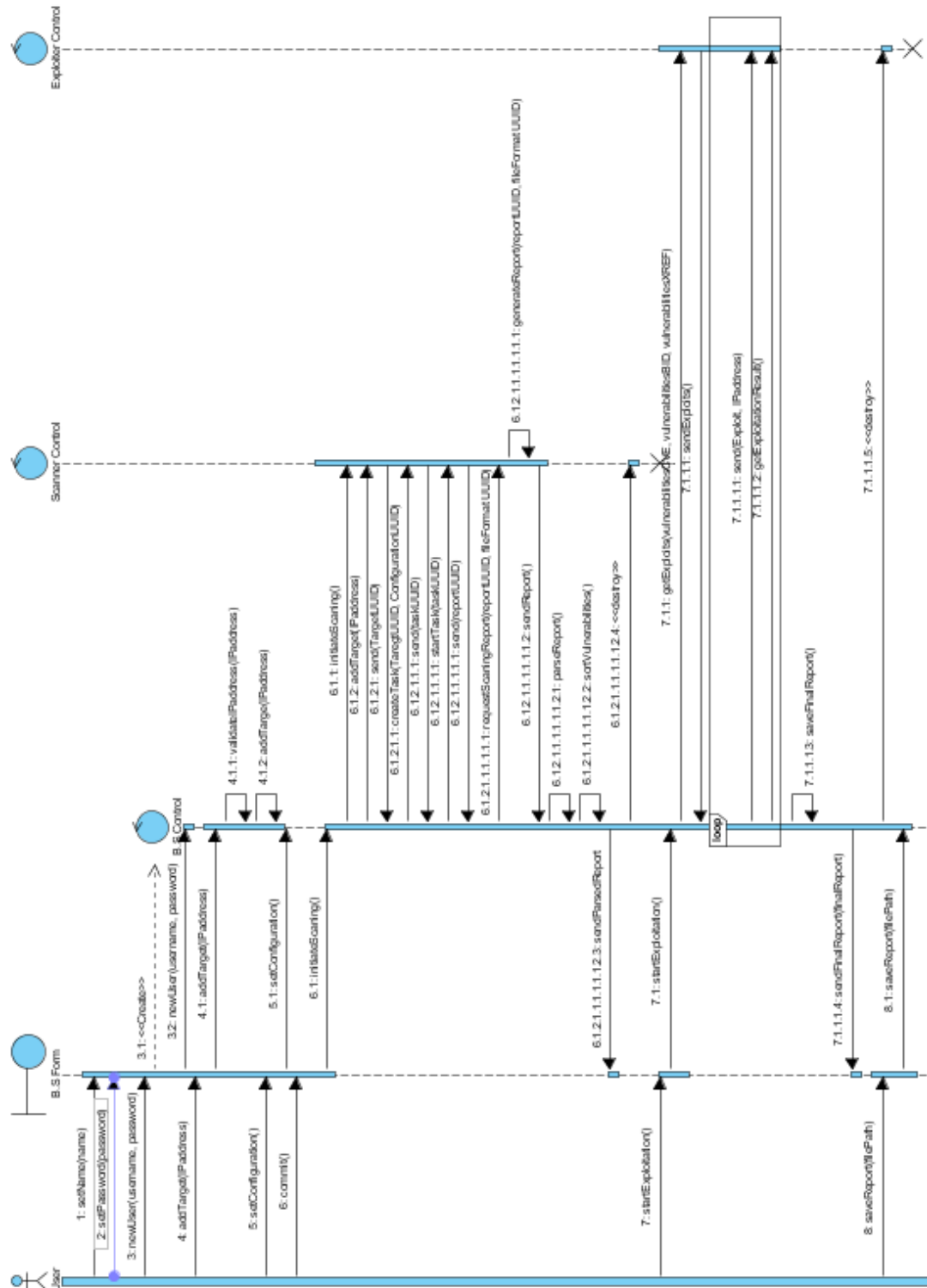


Figure 3 -10 The Whole Sequence model

• Analysis Object Model

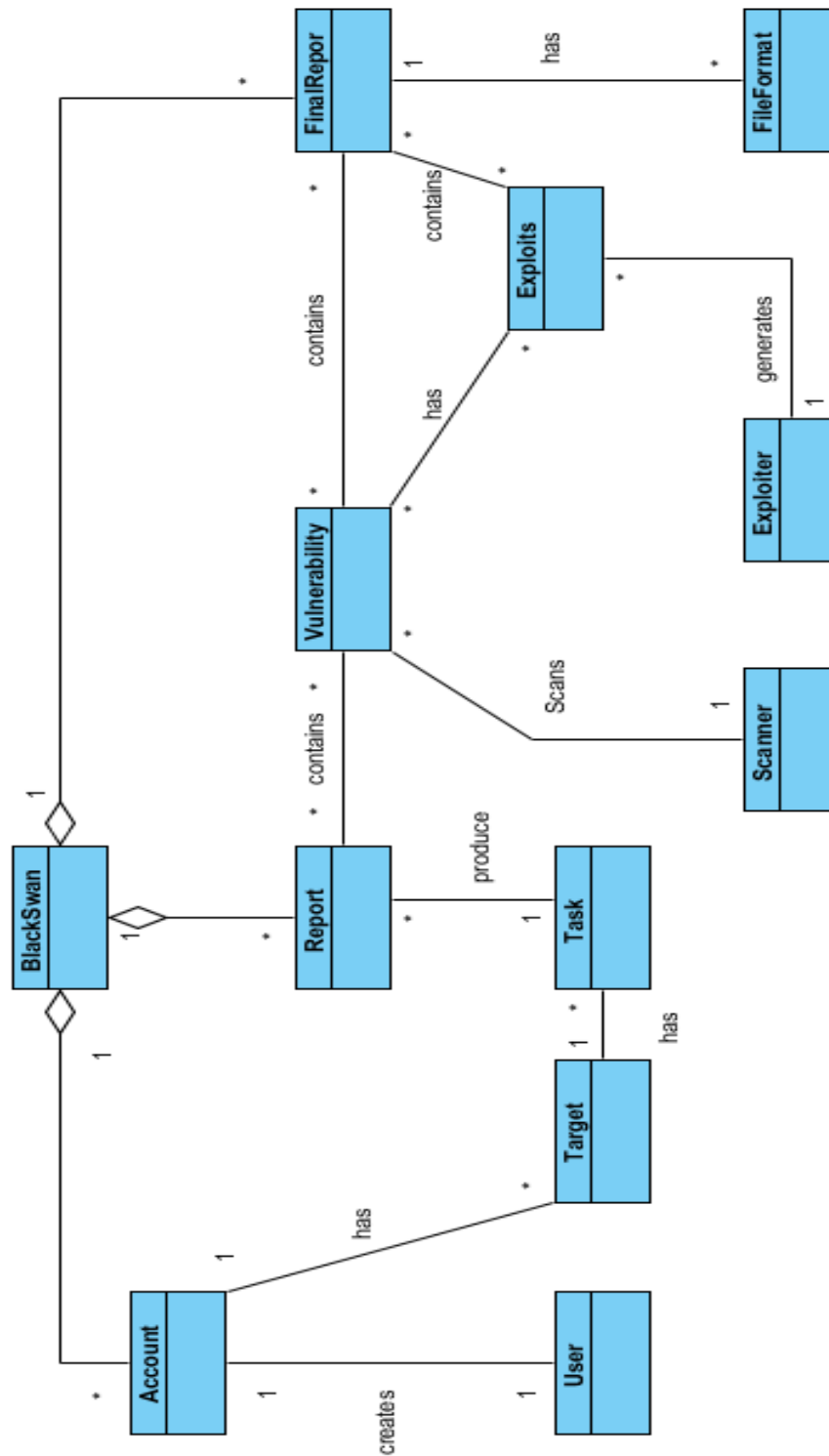


Figure 3-11 Analysis Object model

Table 3-7 Entity Objects

Entity Object	Attributes	Definition
User	<ul style="list-style-type: none"> • name • username • password • pcs • logs • report 	A User represents the person that creates Account and interacts with the BlackSwan interface.
BlackSwan		An instantiation of the Black Swan system.
Account	<ul style="list-style-type: none"> • username • password • accountId • name 	It is an entity that allows a User to authenticate to BlackSwan's services and be granted authorization to access them.
Target	<ul style="list-style-type: none"> • ipAddress • macAddress • targetUUID • dateCreated • pcId • state 	It is a Vulnerable system to be scanned by the User using BlackSwan.
Scanner	<ul style="list-style-type: none"> • scannerUsername • scannerPassword • serverIpAddress 	It is a computer program designed to assess computers, computer systems, networks or

		applications for weaknesses.
Task	<ul style="list-style-type: none"> • configurationUUID • targetUUID • taskUUID 	A Task is an activity that needs to be accomplished within a defined period of time or by a deadline.
Report	<ul style="list-style-type: none"> • reportUUID • reportFileFormat 	It is a file that has that contains the results of the Task done.
Vulnerability	<ul style="list-style-type: none"> • vulnerabilityCVE • vulnerabilityBID • vulnerabilityXREF • platform • portNumber • protocols • cvssRank 	A vulnerability is a weakness which allows an attacker to reduce a system's information assurance.
Exploit	<ul style="list-style-type: none"> • portNumber • platforms • rank • kindofApplication • exploitBID • exploitCVE • exploitEDB • exploitXREF • name 	An Exploit is a piece of software, a chunk of data, or sequence of commands that takes advantage of a bug, glitch or vulnerability in order to cause unintended or unanticipated behavior to occur on computer

		software.
Exploiter		It is a computer program designed to initiate the Exploit.
FinalReport	<ul style="list-style-type: none"> • locationOnPc 	It is the Report that the user save in any location on his computer to use it any time he wants.
FileFormat	<ul style="list-style-type: none"> • formatType • formatUUID 	A FileFormat is a standard way that information is encoded for storage in a computer file. A FileFormat specifies how bits are used to encode information in a digital storage medium.

Chapter 4

System Design Specification

4.1. Introduction

Black Swan aims to make the User to make an automated penetration testing because penetration testing is complicated, difficult, and exhausting job for both the testers and the corporation, and its automation (partially or totally) considered a great job.

Here we provide an automation technique for the penetration testing process, using a vulnerability scanner and an exploitation engine and our code to interact with them and save the results in our database.

4.2. Classes

Interaction with the OpenVas server

This Class (Figure 4-1) interacts with the openvas server, as it create/get target, create/get scanning configuration, create task, start scan, wait until the scanning task is finished, and return the report using a format.

The methods' names describe its functionality.

execom() method is used to execute various commands and return the output to us, that's why it is used by almost every other functional, and it is shortened for execute command.

BsVas
<pre> -init(self, server = "127.0.0.1", usr = "talta", Password = "toor") -rFormat(self, f_id = 3, f_name = 1) -getconf(self, conf_num = 0, conf_name = 1) -execom(self, rawcmd, rawcmd2="", out=subprocess.PIPE, inn=subprocess.PIPE) -checkTarget(self, tar_ip) -crTarget(self, tar_ip) -CheckTask(self, tar_ip) -crTask(self, tar_ip, conf_id = 1) -startTask(self, task_ID) -stopTask(self, task_ID) -delTask(self, task_ID) -wait(self, T_ID) -getreport(self, report_id, dir, fomtype = 4) -main()</pre>

Figure 4-1 BsVas Class

Parsing Results

This class (Figure 4-2) performs the following:

- Parses the vulnerability report produced by openvas (in xml format).
- Returns a variety of things from the report, NVTs(vulnerabilities), cves , bids, xrefs, number of vulnerabilities found in the report,...

BsXml
-file(self, f) -xmlprint(self) -cvefinder(self) -bidfinder(self) -xreffinder(self) -getNVTs(self) -main()

Figure 4-2 BsXml Class

Implementing the connections to msf3

This class (Figure 4-3) performs the following:

- Implements the connections to msf3 (metasploit framework postgresql database name) using the sqlalchemy.
- Queries the exploits only, its functionality to return a list of exploits.

Exp_Detail
-init(self) -queryf(self, cve) -fullexp(self, resultL) -expList(self, cves)

Figure 4-3 Exp_Detail Class

Maintains a connection to msf3 and returning specific information

This class (Figure 4-4) is very similar to Exp_Detail, as it maintains a connection to msf3 database also, but instead of returning a list of exploits only, it returns a list of exploits and its details as fname, rank and description.

PgSelect
-init(self) -queryf(self, cve) -print1(self, resultL) -print2(self, resultL) -fnamefinder(self, resultL) -expList(self, cves) -main()

Figure 4-4 PgSelect Class

Connecting with the Metasploit framework console

This class (Figure 4-5) does the following:

The connection with the metasploit framework console,as it executes the command and reads the output for the command,it is used to exploit the vulnerability , and determine the status (success/failure).

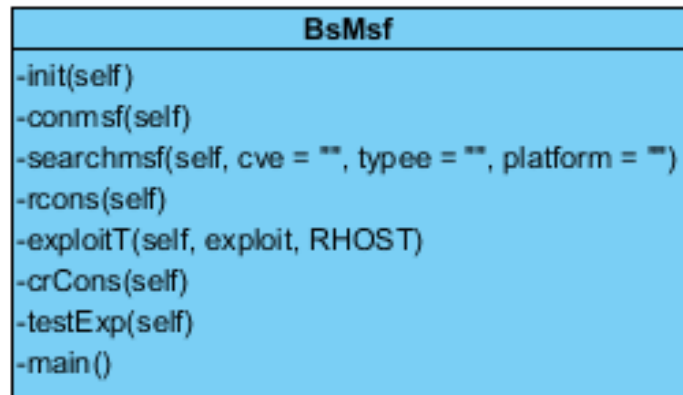


Figure 4-5 BsMsf Class

Interacting directly with the console

This Class (Figure 4-6) contains an interactive shell for the metasploit framerok console to allow the user to interact directly with the console, within our console shell (in the case of CLI[command line interface] and not GUI)

MsfConsole
<pre> -init(self) -testExp(self, RHOST, exploit) -wait(self) -main() </pre>

Figure 4-6 MsfConsole Class

Connecting with the Nmap

This class (Figure 4-7) does the following:

Connects with the Nmap port scanner to fingerprint (determine the host state (up/down), and the host's OS).

BsNmap
<pre> -init(self, IP) -hostScan(self) -host_status(self) -host_name(self) -host_mac(self) -os_match(self) -os_family(self) -os_vendor(self) -printInfo(self) -main() </pre>

Figure 4-7 BsNmap Class

Main Class

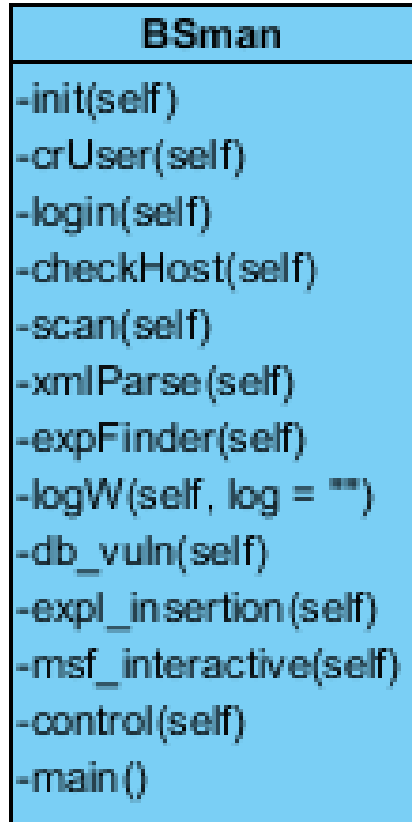


Figure 4-8 BSman Class

This Class (Figure 4-8) is the controller of the Black swan, as it performs the following:

- Handling database (our database) operations (select,update,delete,insert), as it controls these operations (when it applied)
- Uses BsVas class to interact with openvas vulnerability scanner server
- Uses BsXml class to parse the report of the scan

- Uses PgSelect class module, to get the exploits of the matched vulnerabilities from msf3 (the metasploit database).
- Uses BsNmap class to scan and check target status, OS and ip.
- Uses BsMsf class to issue the exploit command to the metasploit framework console, and check the exploit state (succeed or failed to exploit).
- Uses the MsfConsole class to provide a live msf console to the user in case of CLI interface.

You can consider this as a controller that controls all the operations of BlackSwan, in almost all user interactions with the system.

DataBase Classes

These classes (Figures 4-9 to 13) have two uses:

- -every class has a representation in the database as it represents a table
- using in selecting, updating, deleting and inserting from/to/into the database as every object mapped to a table entry in a database using sqlalchemy (which handles the manipulation of all this!)

User
-tablename = users
-user_id : int
-username : string
-full_name : string
-password : string
-user_dir : string
-init()

Figure 4-9 User Class

PC
-tablename : string = pc
-pc_id : int
-mac_address : string
-osmatch : string
-family : string
-pc_uuid : string
-status : int
-user : User
-init()

Figure 4-10 PC Class

Report
-tablename : string = reports
-report_id : int
-rate : int
-date : TIMESTAMP
-report_uuid : string
-config_id : string
-pc : PC
-init()

Figure 4-11 Report Class

Vuln
-tablename : string = vulns
-vul_id : int
-vul_oid : string
-vul_name : string
-vul_cve : string
-vul_bid : string
-vul_xref : string
-vul_desc : string
-risk_factor : string
-report_id : int
-report : Report
-init()

Figure 4-12 Vuln Class

Exploit
-tablename : string = exploits
-exploit_id : int
-exploit_fname : string
-vul_id : int
-vul : Vuln
-msf_id : string
-init()

Figure 4-13 Exploit Class

4.3. Database Design Specification

Black Swan system has a simple database schema (Figure 4-14), it consists of some tables. First table represents the users and the table name is *Users* = {user_id, username, password, full_name, user_dir}, so each user has a unique ID which represents the primary key and also has username, password, full name and finally user directory.

Second table is PC table, which represents the basic information about the PC like his ID, Mac address, OS, and has a primary key which is PC ID and also has foreign key which is the User ID.

Third table is the Reports table which represents the information about the report like report ID and UUID, configuration ID and it has a primary key which is the report ID and also has a foreign key which is the PC ID.

Forth table is the Vulns table which represents the information about each vulnerability like vulnerability ID, OID, Name ,Cve and etc and it has also a primary key which is the Vulnerability ID and also has a foreign key which is the report ID.

Fifth table is the Exploits table which represents the information about the Exploits like exploit ID,MSF ID, exploit rank, etc and it has a primary key which is the Exploit ID and also has a foreign key which is the Vulnerability ID.

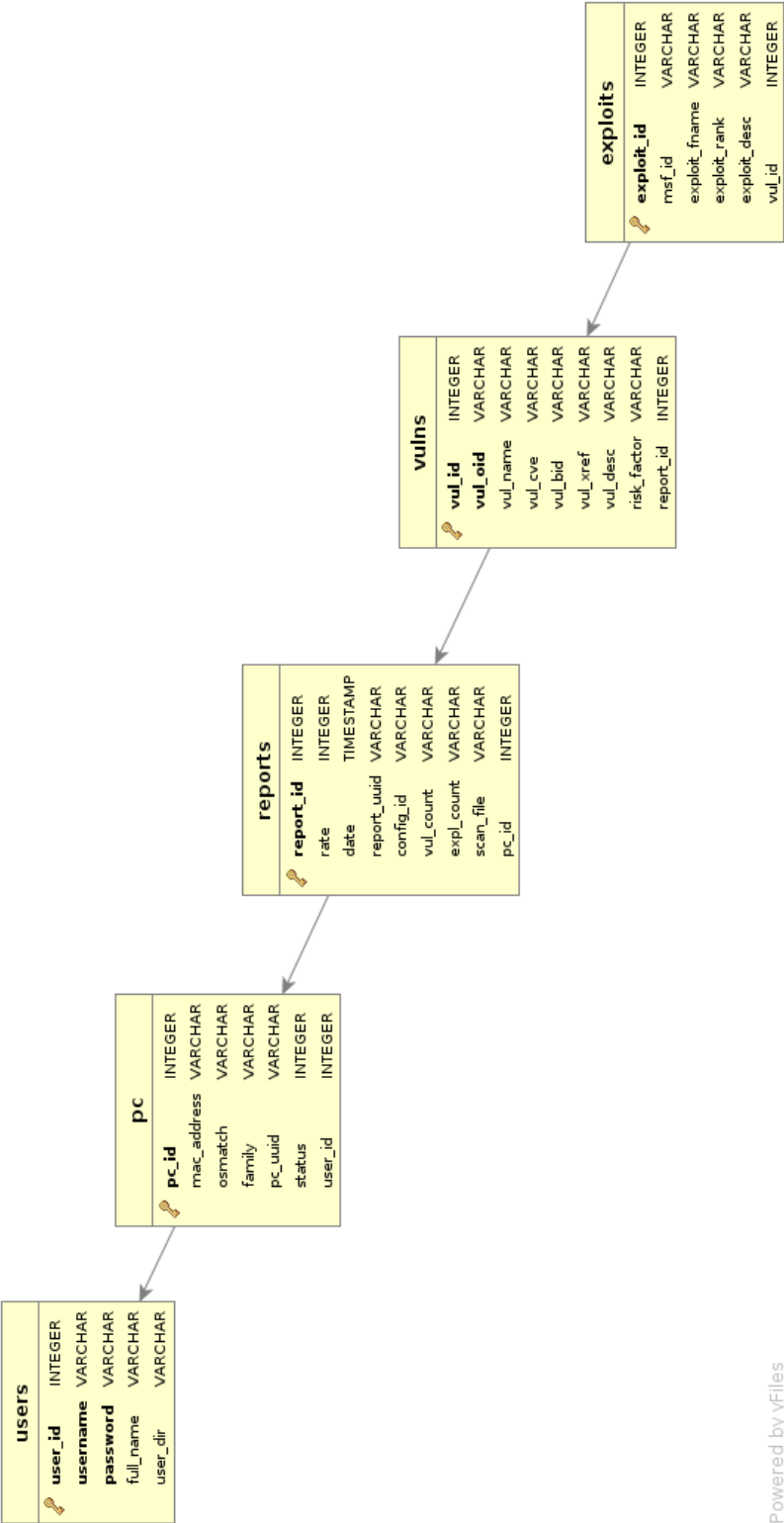
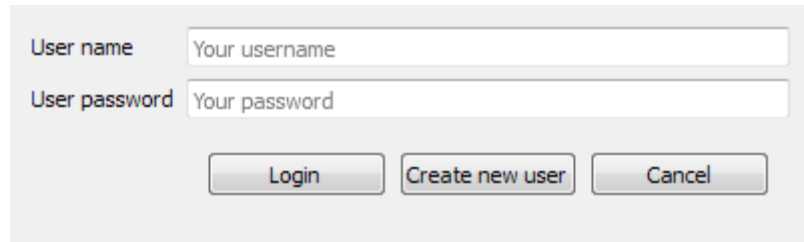


Figure 4-14 Database Schema

4.4. User Interface Design

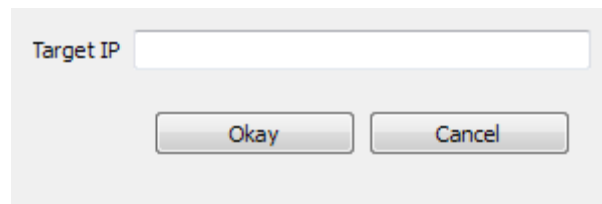
Login Form



A login form with a light gray background. It contains two input fields: 'User name' with placeholder text 'Your username' and 'User password' with placeholder text 'Your password'. Below the fields are three buttons: 'Login', 'Create new user', and 'Cancel'.

Figure 4-15 Login Form

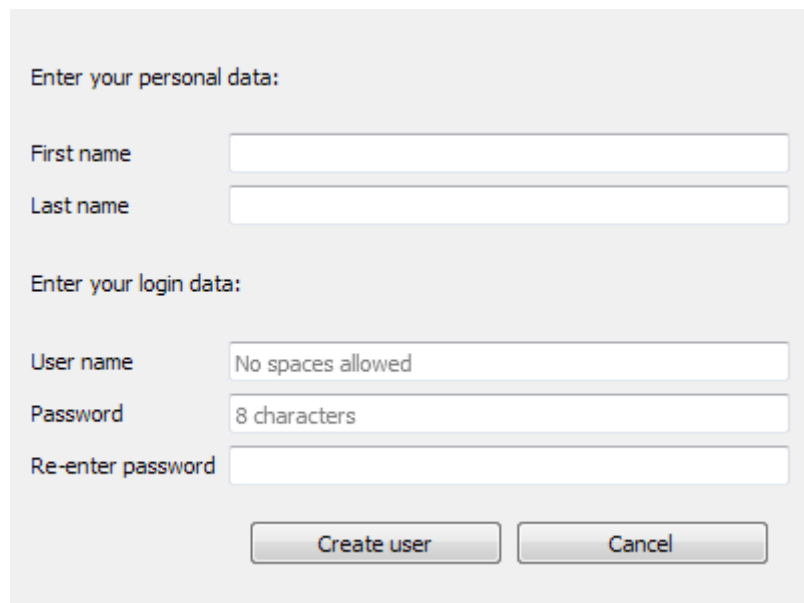
Add Target



An 'Add Target' dialog box with a light gray background. It features a single input field labeled 'Target IP'. Below the field are two buttons: 'Okay' and 'Cancel'.

Figure 4-16 Add Target Form

Create User



A 'Create User' form with a light gray background. It is divided into two sections. The first section, 'Enter your personal data:', contains two input fields for 'First name' and 'Last name'. The second section, 'Enter your login data:', contains three input fields: 'User name' with placeholder text 'No spaces allowed', 'Password' with placeholder text '8 characters', and 'Re-enter password'. At the bottom are two buttons: 'Create user' and 'Cancel'.

Figure 4-17 Create User Form

Main Frame Pages

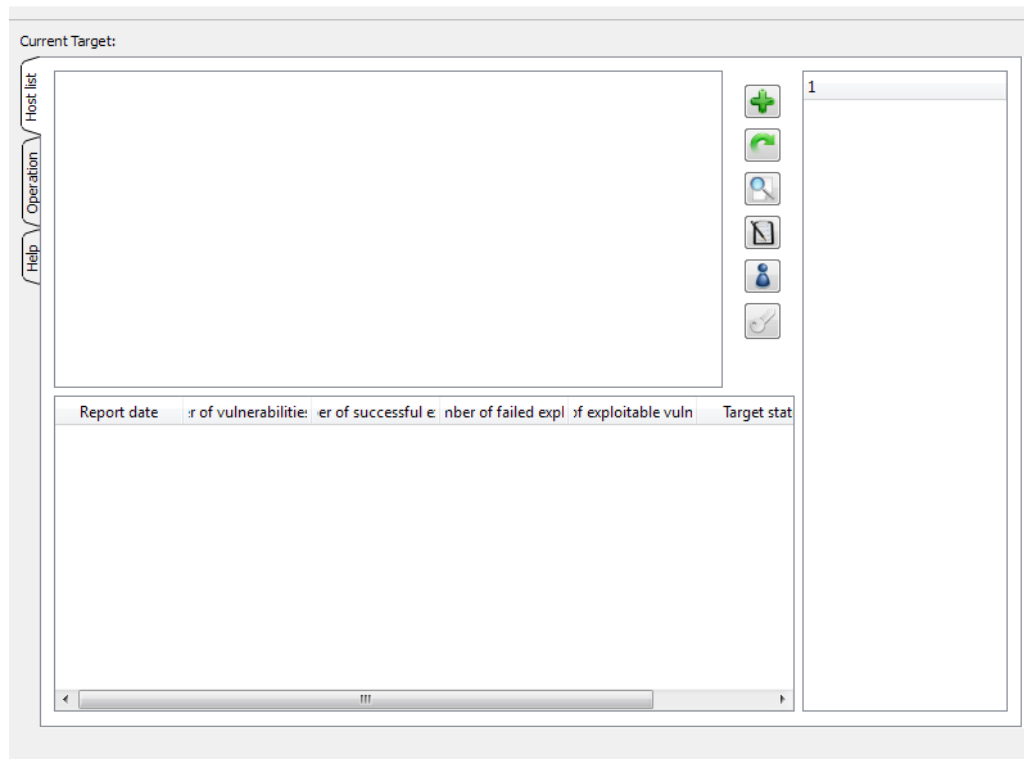


Figure 4-18 Host List panel

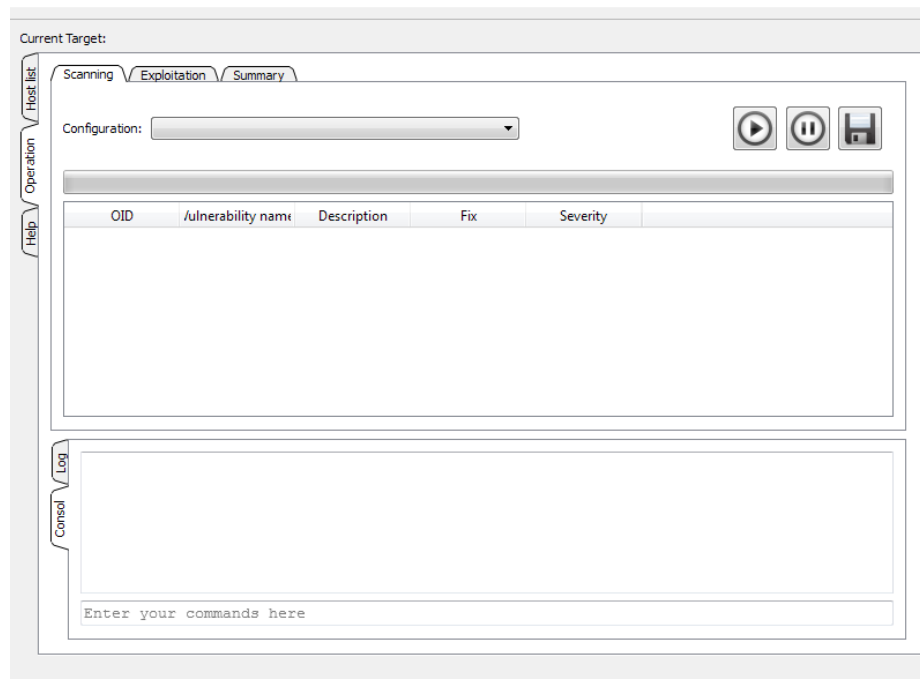


Figure 4-19 Scanning panel

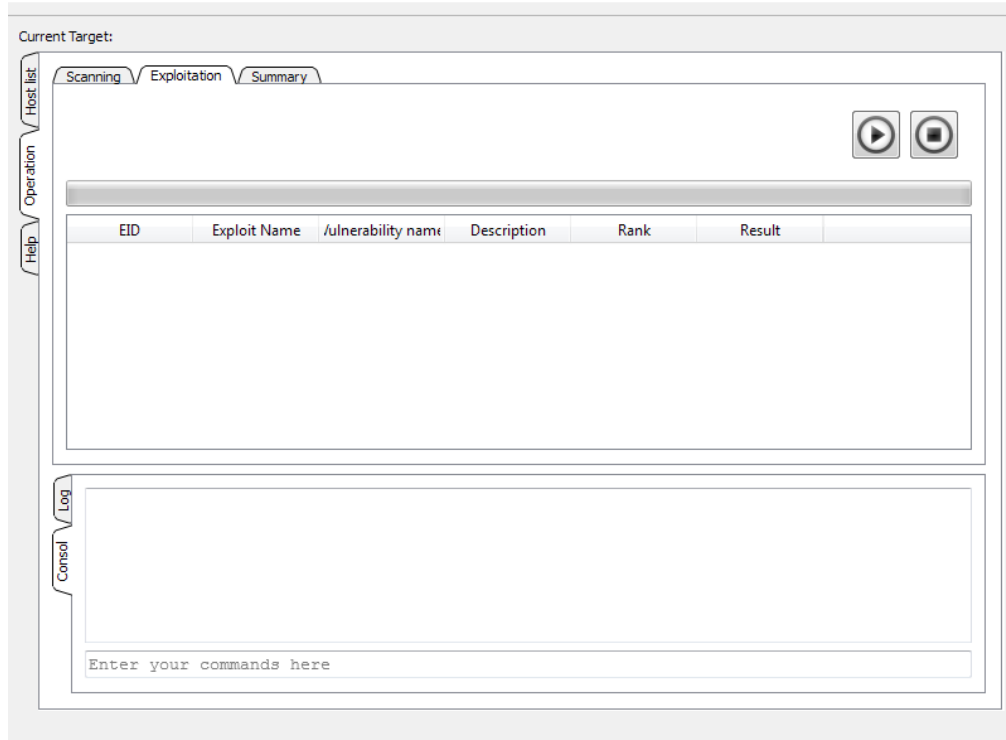


Figure 4-20 Exploitation panel

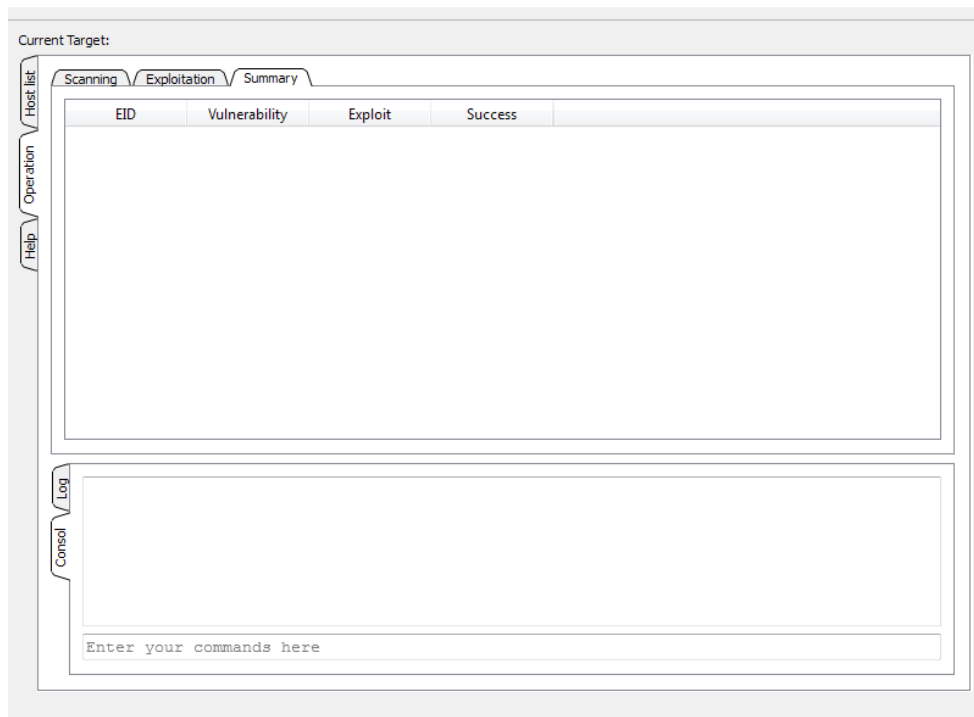


Figure 4-21 Summary panel

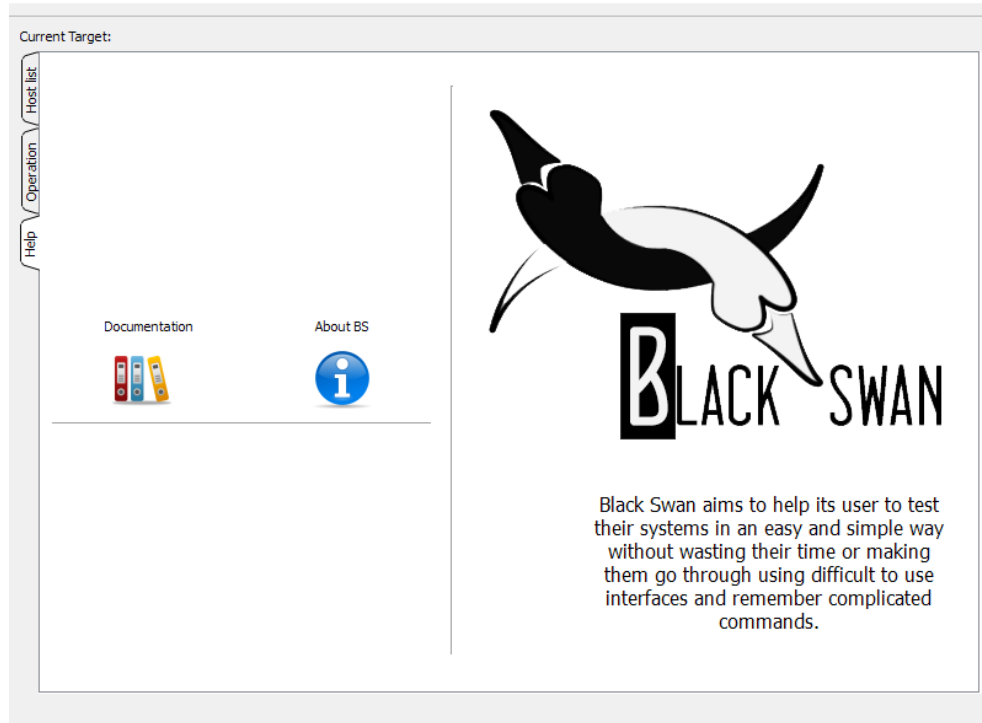


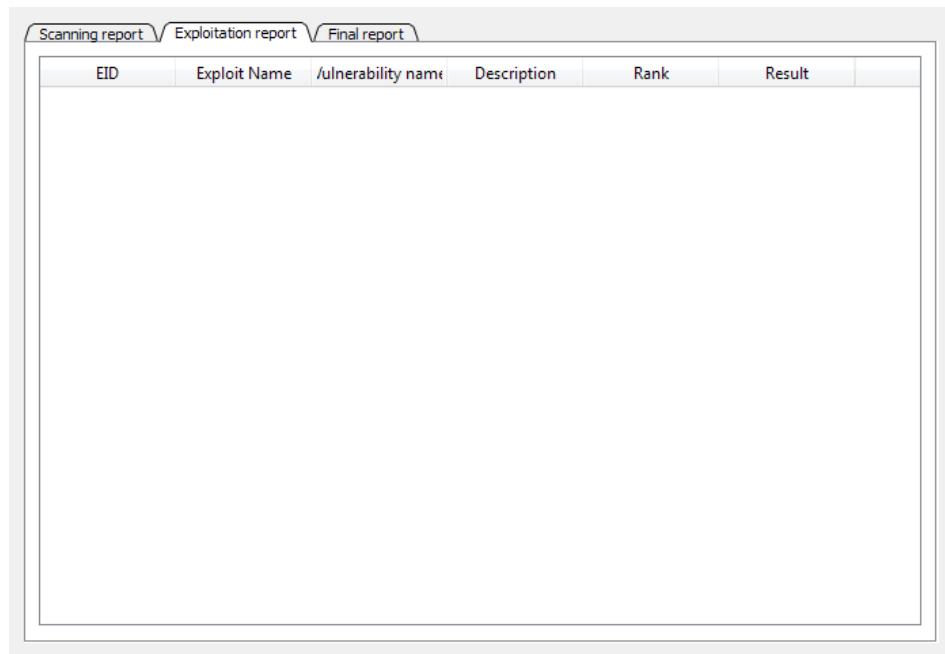
Figure 4-22 Help panel

Reports Form

Scanning report | Exploitation report | Final report

OID	Vulnerability name	Description	Fix	Severity
-----	--------------------	-------------	-----	----------

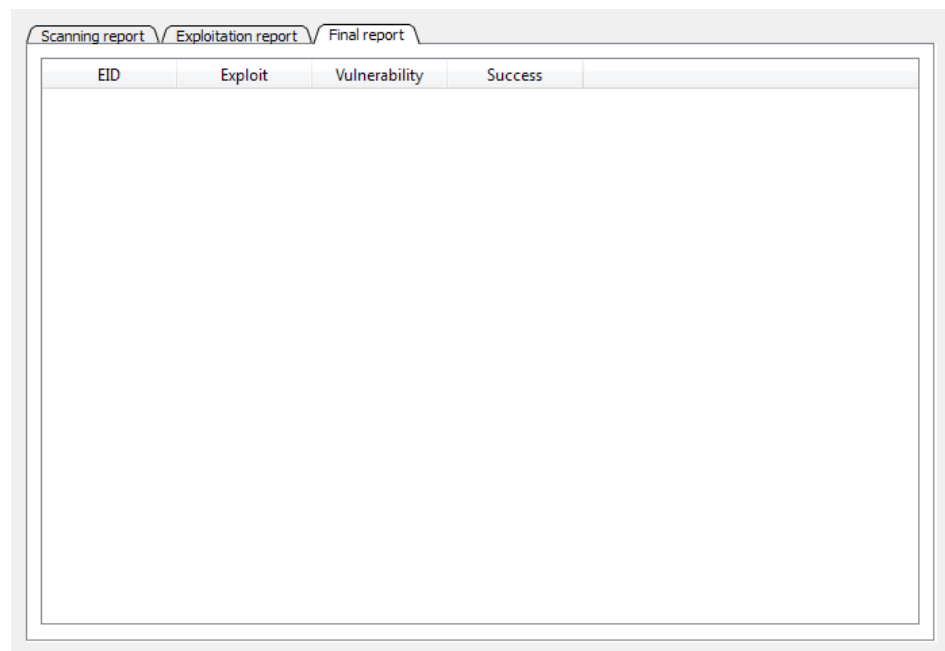
Figure 4-23 Scanning report panel



The image shows a software interface for an exploitation report. It features a tabbed header with three tabs: 'Scanning report', 'Exploitation report' (which is selected), and 'Final report'. Below the tabs is a table with seven columns: 'EID', 'Exploit Name', 'Vulnerability name', 'Description', 'Rank', and 'Result'. The table is currently empty, showing only the header row.

EID	Exploit Name	Vulnerability name	Description	Rank	Result
-----	--------------	--------------------	-------------	------	--------

Figure 4-24 Exploitation report panel



The image shows a software interface for a final report. It features a tabbed header with three tabs: 'Scanning report', 'Exploitation report', and 'Final report' (which is selected). Below the tabs is a table with four columns: 'EID', 'Exploit', 'Vulnerability', and 'Success'. The table is currently empty, showing only the header row.

EID	Exploit	Vulnerability	Success
-----	---------	---------------	---------

Figure 4-25 Final report panel

Log Form



The Log panel is a rectangular window with a tabbed interface on the left side. The tabs are labeled 'Log' and 'Consol'. The 'Log' tab is currently selected, indicated by a checkmark. The main area of the panel is a large, empty rectangular box, likely intended for displaying log messages or system output.

Figure 4-26 Log panel

Console Form



The Console panel is a rectangular window with a tabbed interface on the left side. The tabs are labeled 'Log' and 'Consol'. The 'Consol' tab is currently selected, indicated by a checkmark. The main area of the panel is a large, empty rectangular box, likely intended for displaying console output. At the bottom of the panel, there is a text input field with the placeholder text 'Enter your commands here'.

Figure 4-27 Console panel

Bibliography

<http://docs.python.org/release/2.7/>

<http://docs.python.org/2/library/xml.etree.elementtree.html>

http://docs.sqlalchemy.org/en/rel_0_7/

<http://www.openvas.org/src-doc/openvas-scanner-3.4.0/index.html>

<http://www.openvas.org/src-doc/openvas-manager-4.0.1/index.html>

<http://www.openvas.org/src-doc/openvas-administrator-1.3.0/index.html>

<http://blog.spiderlabs.com/2012/01/scripting-metasploit-using-msgrpc-.html>

<https://github.com/SpiderLabs/msfrpc>

<https://github.com/rapid7/metasploit-frameworkdev.metasploit.com/documents/api/>

http://www.offensive-security.com/metasploit-unleashed/Main_Page

<http://www.securitytube-training.com/online-courses/securitytube-metasploit-framework-expert/index.html>

<http://srinikom.github.io/pyside-docs/>

<http://www.qtrac.eu/pyqtbook.html>

<http://resources.infosecinstitute.com/metasploitable-2-exploitation/>

Object-Oriented Software Engineering Using UML, Patterns, and Java™ Third Edition by Bernd Bruegge & Allen H. Dutoit (2010)