

MetaNet

May 22, 2016

Powered by



Written by: Guy Gelber, Matan Ramrazker

Table of Contents

1.	Introduction	2
2.	Overall description	4
3.	System features.....	6
4.	External interface requirements.....	8
5.	Non-functional requirements.....	9
6.	Key milestones	10
7.	Key resource requirements	11
8.	Other requirements	12
9.	Glossary	13
10.	Project proposal	15
11.	Bot class diagram	18
12.	Flask server class diagram	23
13.	Sequence UML diagram	24
14.	GUI.....	26
15.	Software test plan	31
16.	Software test description and report	32

SRS – pages 2-14

Project proposal – pages 15

UML diagrams – pages 18-25

GUI presentation – pages 26-30

STP – page 31

STD&STR – pages 32-39

1. Introduction

1.1. Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of the functionalities of the MetaNet system. This document will cover each of the system's intended features; the document will also cover hardware, software, and various other technical dependencies.

1.2. Document conventions

This document features some terminology which readers may be unfamiliar with. See "Glossary" for a list of these terms and their definitions.

1.3. Intended audience and reading suggestions

This document is intended for all individuals participating in and/or the MetaNet Project. Readers interested in a brief overview of the product should focus on the rest of Part 1 (Introduction), as well as Part 2 of the document (Overall Description), which provide a brief overview of each aspect of the project as a whole. These readers may also be interested in Part 6 (Key Milestones) which lays out a concise timeline of the project.

Readers who wish to explore the features of MetaNet in more detail should read on to Part 3 (System Features), which expands upon the information laid out in the main overview. Part 4 (External Interface Requirements) offers further technical details, including information on the user interface as well as the hardware and software platforms on which the Botnet and server will run. Readers interested in the non-technical aspects of the project should read Part 5, which covers performance, safety, security, and various other attributes that will be important to users. Readers who have not found the information they are looking for should check Part 8 (Other Requirements), which includes any additional information which does not fit logically into the other sections.

1.4. Project scope

The MetaNet system is composed of three main components:

- Server-side application ("Server") – Its main purposes are to act as a "middleman" between the Bot and the Client-side application, its main features includes, storing and transferring relevant bot information, storing a list of all the bots that ever logged into the server, maintaining a list of bots that are connected in real time ("Online").
- Bot – A program that runs on a remote computer, its main purposes are carrying out tasks that are given from the Bot-master, sending updated information about the computer and operating system that it runs on, sending every configured specified time a "Sign of life" to the server.
- Client-side application ("Bot-master control panel") – A web application that run on the user browser, its main purposes are to provide Graphical User Interface to give the bot-master control of the bots features and actions, it will also list the online bots and the offline bots.

2. Overall description

2.1. Product perspective

MetaNet system in one hand is a remote computer management, in the other hand is a computer worm that replicates itself in order to spread to other computers.

While the Bot program is the main focus of the project, there are two more components, the server-side which will be responsible for database and communication services, and the client-side which will be used as Graphical User Interface.

The scope of the project encompasses Bot, server-side and client-side functionalities.

Below is a diagram of a Botnet system which illustrates the interactions between the bot the server and the client-side application (C&C).

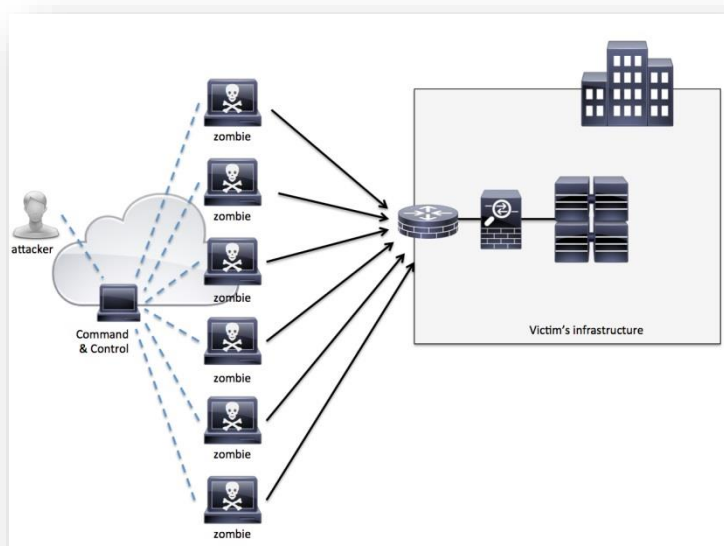


Figure 1. attacker sent to his Zombies(bot) to attack a victim.

2.2. Product features

The MetaNet system split into three main parts, client-side application, bot and server-side application. The bot is responsible for scanning port in the local net and to run exploits to distribute itself, furthermore it runs tasks that sent from the bot-master through the server. The C&C server is responsible for sending messages and receiving data from the bots and transferring them to the client-side application.

The client-side application's purpose is knowing in real time the status of the connected bots and it will give the bot-master an easy way to manage all of the bots.

2.3. User classes and characteristics

The MetaNet system has two user classes

- Bot-master – the controller of the all the slave bots
- Bot (slave) – A remote computer that is controlled by the bot-master

2.4. Operating environment

- Server-side application – Computer that runs a Web server, SQLite Database and Ruby.
- Bot – Any Linux distributions that runs on 32 or 64bit Computer and have Internet connection, the bot depends of number of system calls that are built-in typical 32 or 64bit Linux kernel.
- Client-side application – Browser that supports JavaScript and CSS.

2.5. Design and implementation constraints

- Bot is implemented specifically to the Linux platform, with possibility to add Windows support in the future.
- Bot installs Ruby interpreter that used in order to integrate with Metasploit.

2.6. User documentation

The primary goal of MetaNet is not targeted for the simple user even though we try to build a system that is easy to use for a typical user.

We will give a user documentation for the Bot-master admin interface with all the possible tasks it can be send to the bots.

The documentations will include:

- Software Requirement Specification.
- Bot-master admin interface documentation.

2.7. Assumptions and dependencies

- Server-side application – No assumptions and dependencies excluding operating environment.
- Client-side application - No assumptions and dependencies excluding operating environment.
- Bot - No assumptions and dependencies excluding operating environment, when the bot is being executed on a machine for the first time it installs in a hidden fashion all the dependencies it needs: Ruby and Metasploit.

3. System features

MetaNet system features are divided into two main categories: core features and additional features. The core features form the body of the application and include any features that are essential to the functionality of the MetaNet system. These features must be implemented in order to have a fully-functioning application. Additional features, however, are not required for the app to function. They include any features which, if needed to the software user, can be added to the application in order to provide extra functionality.

Core features

- Bot installation

Upon executing the bot main executable file on a new Linux system, the bot will copy itself to hidden directory in the filesystem and will install Ruby and Metasploit in the same directory, the bot will add itself to the program start-up list.

- Bot uninstallation

Upon receiving uninstallation message from the server, the bot will remove itself completely from the operating system.

- Bot update

Upon receiving update message from the server, the bot will stop itself and replace the executable file of the bot with a new executable that is received from the server and will run it.

- Bot system profiling

The bot will gather information about the operating system and the hardware it is installed on, the bot will create "unique" identification string that will help the bot-master to identify the bot and the machine it is running on.

- Bot "Sign of life"

The bot will send every configured specified time a message to the server, the purpose of the message is to notify the server that the bot is live, connected to the internet and can receive tasks to carry out.

- Bot port scanner

The bot will scan the local network for ports that can be exploited.

- Bot Metasploit exploiting

The bot tries to exploit specific configured services on machines in the network that has open ports of that service, if the exploiting of the service was successful the bot will transfer the executable file and execute it in the exploited machine.

- Carrying out a task

Upon receiving a task message from the server, the bot will perform specific task from list of supported tasks with parameters that are sent from the bot-master through the server.

- On connect task

A Task that will be sent from the server to the bot immediately when bot connected to the server.

Additional features (Tasks)

- Bandwidth test

Measure the maximum internet data throughput, download and upload speed of the machine the bot runs on the result will be sent to the server.

- SOCKS5 Server

SOCKS5 is A protocol that routes network packets between a client and server through a proxy server, the bot will create SOCKS5 server on a specified port.

- FTP Server

FTP is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet, the bot will create FTP server on a specified port and will serve the home directory of the executable's owner user.

NOTE: Some of these features are not part of our core design and will only be added if time permits, Additional tasks can be added, the code will be programmed in a manner that adding additional tasks (features) will be easy.

4. External interface requirements

4.1. Software Interfaces

The bot is to be developed in C++ with Boost libraries to provide easy implementation for Windows if wanted in the future, the program will run in low-level programming language in order to perform well and to eliminate the need of any other software dependencies that can increase the chances of the bot being detected.

4.2. Communications Interfaces

The server is web-based and created using Python(Flask) language. As mentioned before The server's main purpose is to act as a "middleman" between the client-side application (which used by a bot-master) and the bot.

The bot message the server on "Sign of life" message, the server sends to the bot right after this message a list of tasks to carry if available, if there is a task the bot will carry out the task with the parameters specified by the bot-master and once it finished the task it will send the result back to the server.

The client-side application is connected to the server using WebSocket which will accomplish real-time messaging and updates, when bot connects or when receiving a task result.

The client-side application will maintain a list of online bots by checking if an online bot did not send "Sign of life" message in configured period of time.

5. Non-functional requirements

5.1. Performance requirements

The Bot will take as little resources as possible in order to stay hidden from the user that is using the compromised computer; the bot will consume above average network usage when it is scanning the network for vulnerable machines, the bot will consume high CPU usage only if it is asked by the bot-master (ex. by sending to the bot a task that is CPU intensive).

The server performance should not be an issue because its purpose is to store bots information and connect between the bot and the bot-master, In case of large amount of bots, the server and client will be programmed to transfer between them small amount of bot objects on each requests in order to prevent high network usage, prevent low database performance on the server-side, and prevent high memory usage on the client-side.

5.2. Safety requirements

MetaNet will not affect any data, software, server that is not related to the application unless `asked to do so` using a task applied by the Bot-master.

5.3. Security requirements

- The password will be stored in the database and will be hashed using salted-MD5 hash.
- The application giving the user that compiled the application option to configure up to three domain names in the bot configuration that will operate in a failover manner, which means if one of the domain are resolved to bad IP or to a server that is not functional, the bot will try to resolve the server IP address from the next configured domain name, the bot will loop forever until it successfully connected to a server.
- The application assumes that only the bot-masters will have control over the three domain names that configured in the configuration file.
- The application assumes that only authorized bot-master have a valid username and password in order to login to the GUI command and control panel.

5.4. Software quality attributes

- Stability - The bot has to be stable, it must not crash unless hardware error occurred, if software exception occurred the bot will send debugging information to the server.
- Maintenance - The software has to be written in a manner that it will be easy to maintain and modify in the future, the main parts of the code will be commented for that purpose.
- Testing – The software has to be testing friendly.
- Usability – GUI interface will be easy to handle and will navigate in the most expected way with no delays. In that case the system program reacts accordingly and transverses quickly between its states.

6. Key milestones

Milestone	Deadline	Comments
Server setup	15/11/15	
Finalized interface design	20/12/15	
SRS document	3/1/16	We started designing and writing the program before SRS submission
SDD document	20/1/16	
implementation	1/2/16	
Completion of the project	30/5/16	

7. Key resource requirements

Major project activities	Skill/Expertise Required	Internal resource	External resource	Issues or Constraints
Design bot architecture	System design experience, Object oriented programming	Matan and Guy have general knowledge in this field	Internet	
Implement bot	Object oriented programming, design patterns, networking, operating system knowledge, Metasploit experience	Matan and Guy have general knowledge in this field	Boost libraries documentation, Internet, Linux man pages	Potential schedule conflicts
Design the Interface	Design and usability experience; knowledge in AngularJS and jQuery and CSS development	Matan and Guy have general knowledge in this field	Internet, Twitter Bootstrap tutorial	
Implement the interface	knowledge in AngularJS and jQuery development, Twitter bootstrap knowledge, networking knowledge	Matan and Guy have general knowledge in this field	Internet, AngularJS documentation	
Create a Server application	Database systems & servers experience, Python Flask knowledge	Matan and Guy have general knowledge in this field	Internet, Flask documentation	No physical server
Sync the Server to the bot	Networking experience and knowledge	Matan and Guy have general knowledge in this field	Internet	
Sync the Server to the Interface	Knowledge in Real-time networking with socketIO	Matan and Guy have general knowledge in this field	Internet	

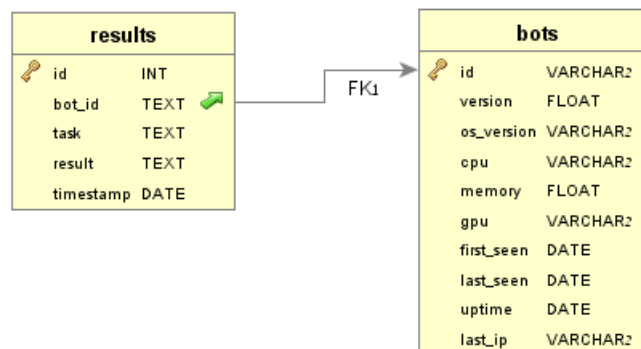
8. Other requirements

A database on the server that holds information of the bot that ever connected to the server. The database will be using SQLite. The following provides an example of information that may be stored in the database:

Bots table: ID, Bot app version, Operating system version, CPU model, memory, first seen, last seen, uptime, last connected IP.

Results table: Bot id, task name, result string, timestamp.

Processes to be done on the server include: pushing/pulling data, updating data.



9. Glossary

Bot-master

A bot-master is a person who operates the command and control of botnets for remote process execution. The bot-master will often hide his/her identify via proxies, TOR and or shells to disguise their IP Address from detection of investigators and law enforcement.

"Sign of life"

A message that is send from a bot that its purpose of the message is to notify the server that the bot is online.

MD5

The MD5 message-digest algorithm is a widely used cryptographic hash function producing a 128-bit (16-byte) hash value, typically expressed in text format as a 32-digit hexadecimal number. MD5 has been utilized in a wide variety of cryptographic applications, and is also commonly used to verify data integrity.

Salt

random data that is used as an additional input to a one-way function that "hashes" a password or passphrase, the primary function of salts is to defend against dictionary attacks versus a list of password hashes and against pre-computed rainbow table attacks.

Exploit

A piece of software, a chunk of data, or a sequence of commands that takes advantage of a bug or vulnerability in order to cause unintended or unanticipated behavior to occur on computer software, hardware, or something electronic (usually computerized). Such behavior frequently includes things like gaining control of a computer system, allowing privilege escalation, or a denial-of-service attack.

Boost (C++ libraries)

A set of libraries for the C++ programming language that provide support for tasks and structures such as linear algebra, pseudorandom number generation, multithreading image processing, regular expressions, and unit testing. It contains over eighty individual libraries. The libraries are aimed at a wide range of C++ users and application domains. They range from general-purpose libraries like the smart pointer library, to operating system abstractions like Boost FileSystem, to libraries primarily aimed at other library developers and advanced C++ users, like the template metaprogramming (MPL) and domain-specific language (DSL) creation (Proto).

Metasploit

The Metasploit Project is a computer security project that provides information about security vulnerabilities and aids in penetration testing and IDS signature development.

Its best-known sub-project is the open source Metasploit Framework, a tool for developing and executing exploit code against a remote target machine. Other important sub-projects include the Opcode Database, shellcode archive and related research.

10. Project proposal

MOTIVATION

We are designing an application that provides a platform to manage a number of Internet-connected computers ("Bots") that are connected to a remote Command and Control ("C&C") system that gathers all the information and data needed to identify and control each one of those computers remotely.

PROBLEM STATEMENT

The problems we aim to alleviate with this app include the following:

- Remote controlling a computer system without being detected easily
- Spreading the bot within local networks
- Creating a friendly graphical user interface for bot-master C&C panel

The app we design solve these problems and several more. Consequently, we will include the following functionalities each bot will have:

- Metasploit framework integration
- Automatic local network port scanner
- Carrying out a particular task specified by C&C server.

We will also add the following features (task/objective) the bot will able to perform, which extend beyond the core functionalities:

- Bandwidth test
- SOCKS5 server
- FTP server

NOTE: Some of these features are not part of our core design and will only be added if time permits.

OBJECTIVES

Beyond implementing the above features, there are a number of additional objectives we must accomplish as a team. First and foremost, as a group we must familiarize ourselves with the Linux Platform to make the best design, as this will lay the groundwork for future.

Additional objectives include the following:

- Develop a timeline detailing each stage of development
- Create a fully-functional, bug-free application
- Ensure each member is up to speed and completing work on time
- Make sure that the will code be efficient as possible
- Create detailed software specifications

METHODOLOGY

A team of two people that strive to split their part of the project equally, but with overlapping issues with mixing tasks that causes everyone on your team to understand the smallest thing in the project.

Developing process:

1. Research
2. Design and architecture
3. Server-side, Client-side, Bot implementation.
4. Metasploit integration
5. Testing

In addition,

In first we dedicated to get acclimated to the new development environment. We will do this by finding and sharing resources (documentation, sample code, etc.).

Before beginning development, we will establish architecture designing and guidelines in order to get main idea of the project.

Next, we will work up specifications, establish deadlines, define roles, and allocate tasks to each. We plan on dividing up the development process into two central phases:

- Core development
- Additional features development

HISTORY

There are many applications on the cyber-crime black market which are similar to our proposed app (examples include Zeus, Andromeda, Citadel, SpyEye). A majority of these applications targets Windows operating system whereas our application runs on Linux operating system. In addition, as far as we know none of them includes Metasploit integration within them. Our application supports Metasploit framework, which brings foundation to support newly (public) released or private exploits in the future.

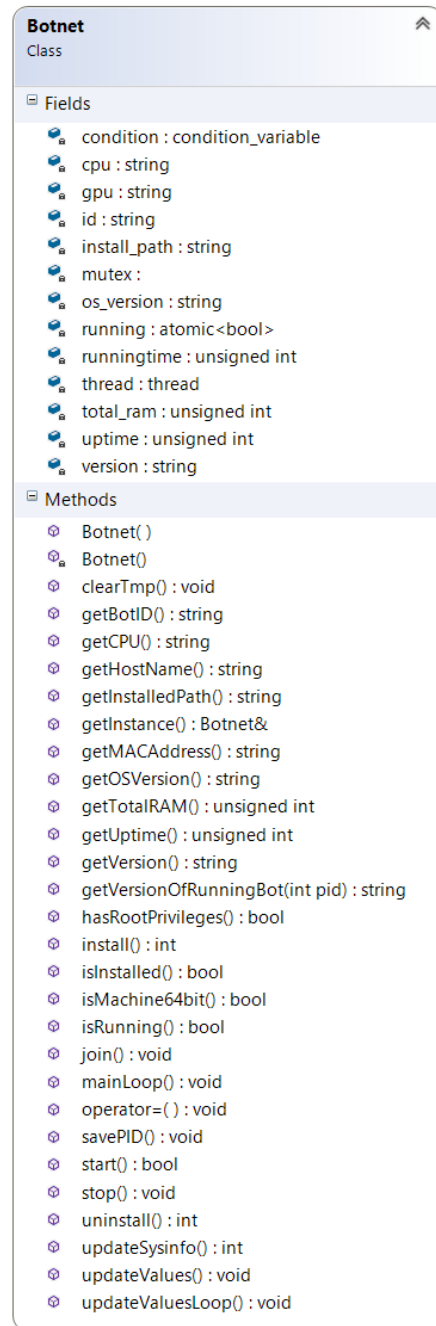
Famous botnets that are being sold on the black market

Zeus - Zeus, ZeuS, or Zbot is a Trojan horse malware package that runs on versions of Microsoft Windows. While it can be used to carry out many malicious and criminal tasks, it is often used to steal banking information by man-in-the-browser keystroke logging and form grabbing.

Citadel - Citadel Trojan is malware created by a malicious code generating program. Citadel was designed to steal personal information, including banking and financial information, from its victims. The Citadel Trojan, based on the Zeus source code, constructs a botnet consisting of a large number of infected computers. The attacker can execute malicious code on an infected computer, including ransomware and scareware.

11. Bot class diagram

Main botnet class



Classes that are responsible for communication with the server

Connection
Class

Fields

- last_working_server_index : unsigned int
- running : atomic<bool>
- servers : vector<SERVER>
- thread : thread

Methods

- Connection()
- getWorkingServer() : SERVER
- join() : void
- mainLoop() : void
- sendSignOfLife(const string& , SERVER) : string
- signOfLife() : string
- start() : void
- stop() : void

HTTPClient
Class

Methods

- sendRequest(const HTTPRequest req) : string

HTTPRequest
Class

Fields

- contenttype : string
- data : string
- domain : string
- port : string
- url : string

HTTPStatusCodeException
Class
→ exception

Fields

- ec : unsigned int

Methods

- HTTPStatusCodeException(unsigned int _ec)
- what() : const char*

ResolveException
Class
→ system_error

Methods

- ResolveException(error_code ec)

HTTPProtocolException
Class
→ runtime_error

Methods

- HTTPProtocolException()

ConnectException
Class
→ system_error

Methods

- ConnectException(error_code ec)

WriteException
Class
→ system_error

Methods

- WriteException(error_code ec)

ReadException
Class
→ system_error

Methods

- ReadException(error_code ec)

Classes that are responsible for port scanning

Scanner

Class

Fields

- h : HostTracker
- mutex : mutex
- parentThread : thread
- ports : vector<unsigned int>
- printMutex : mutex
- producer_count : atomic_int
- producer_threads : thread_group
- root : bool
- running : bool
- threads : thread*[THREADS_LIMIT]

Methods

- mainThread() : void
- portScan() : void
- printerExec(string) : void
- Scanner(vector<unsigned int>& p)
- setPorts(vector<unsigned int>&) : void
- start(unsigned int time) : void
- stopParentThread() : void

HostTracker

Class

Fields

- fin : bool
- myip : address_v4
- octA : unsigned char
- octB : unsigned char
- octC : unsigned char
- octD : unsigned char
- subnet : int

Methods

- bulidAddress() : void
- getOctA() : unsigned char
- getOctB() : unsigned char
- getOctC() : unsigned char
- getOctD() : unsigned char
- getSubnet() : int
- hasNext() : bool
- HostTracker()
- incAddress() : void
- initTracker() : void
- next() : string
- setMyip() : void
- setOctA(unsigned char) : void
- setOctB(unsigned char) : void
- setOctC(unsigned char) : void
- setOctD(unsigned char) : void

PortCheck

Class

Fields

- isStop : bool
- mutex : mutex
- printMutex : mutex
- tcpSocket : socket
- timer : deadline_timer

Methods

- checkConnection(const error_code&, endpoint) : void
- checkDeadline() : void
- PortCheck(io_service&)
- printerExec(string) : void
- start(endpoint&, io_service&) : void
- startConnect(endpoint&) : void
- stop() : void

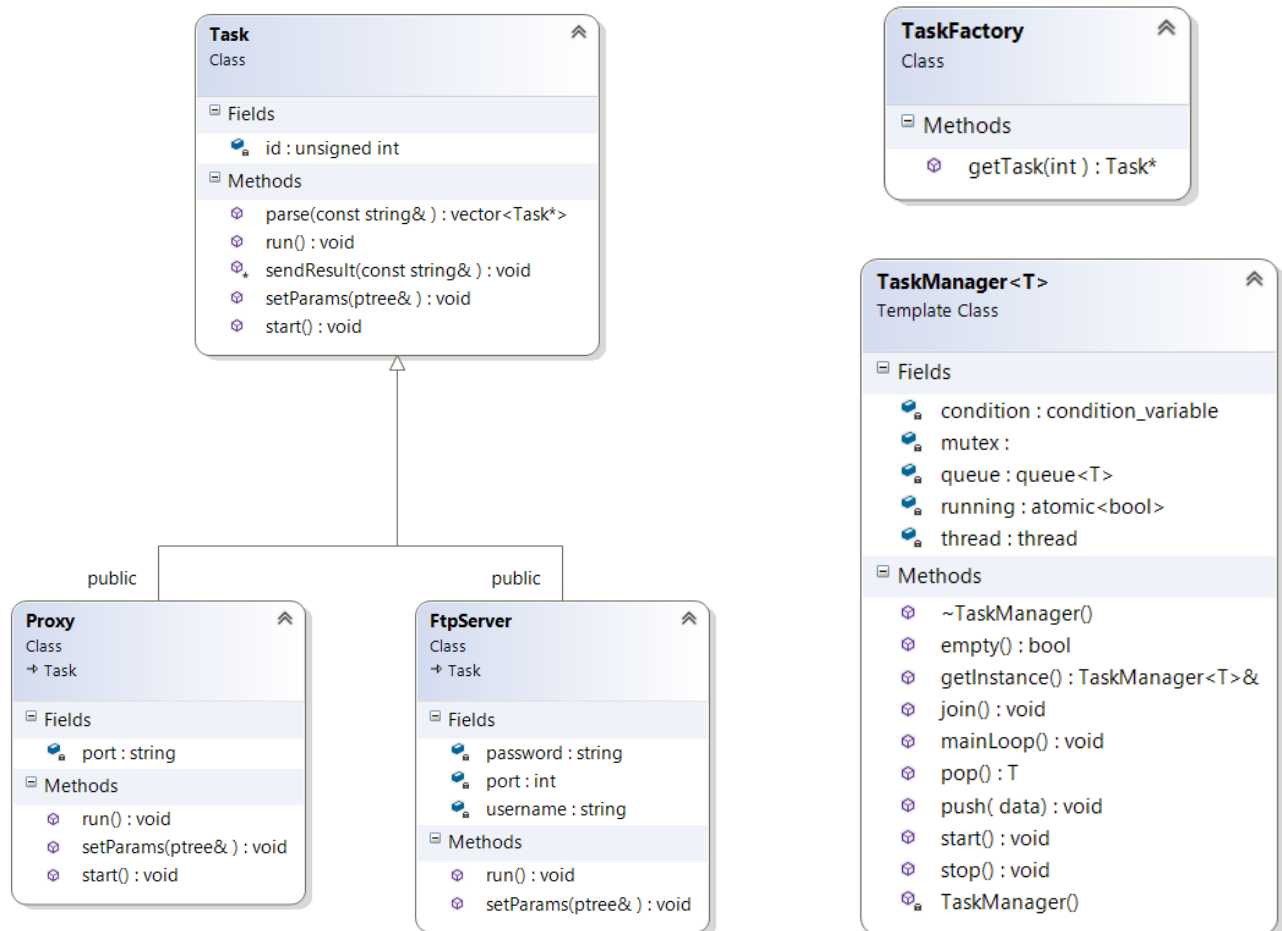
IP

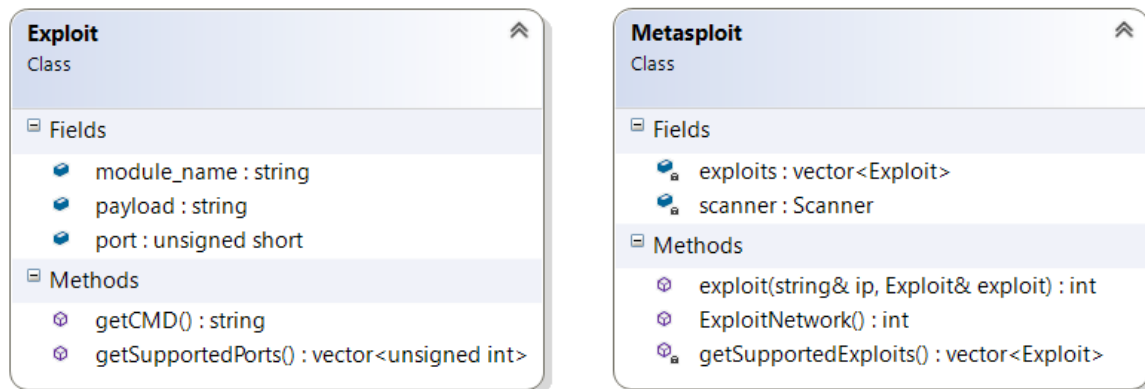
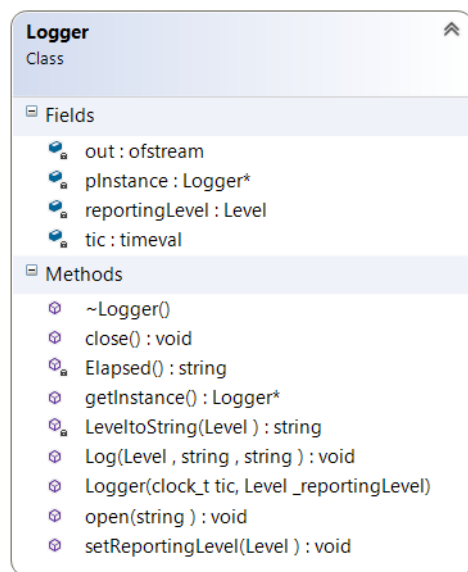
Struct

Fields

- oc2 : unsigned char
- oc3 : unsigned char
- oct1 : unsigned char
- oct4 : unsigned char

Classes that are responsible for task management and execution



Classes that are responsible for Metasploit integration and exploit execution**Logger class: responsible for logging debug information**

12. Flask server class diagram

Bot.Bot

```

m __init__(self, json="", remote_addr="", db=None, gi=None)
m hasTasks(self)
m addTask(self, task)
m getTasks(self)
m getTasksJSON(self)
m emptyTasks(self)
m updateLastSeen(self)
m isOnline(self)
m toJSON(self)
m newBot(self)
m addTaskResult(self, task, result)

```

```

f id
f first_seen
f db
f last_ip
f json
f gi
f uptime
f cpu
f tasks
f last_seen
f memory
f gpu
f os_version
f version

```

Task.Task

```

m __init__(self, id, params)
m toJSON(self)

```

```

f id
f params

```

DB.DB

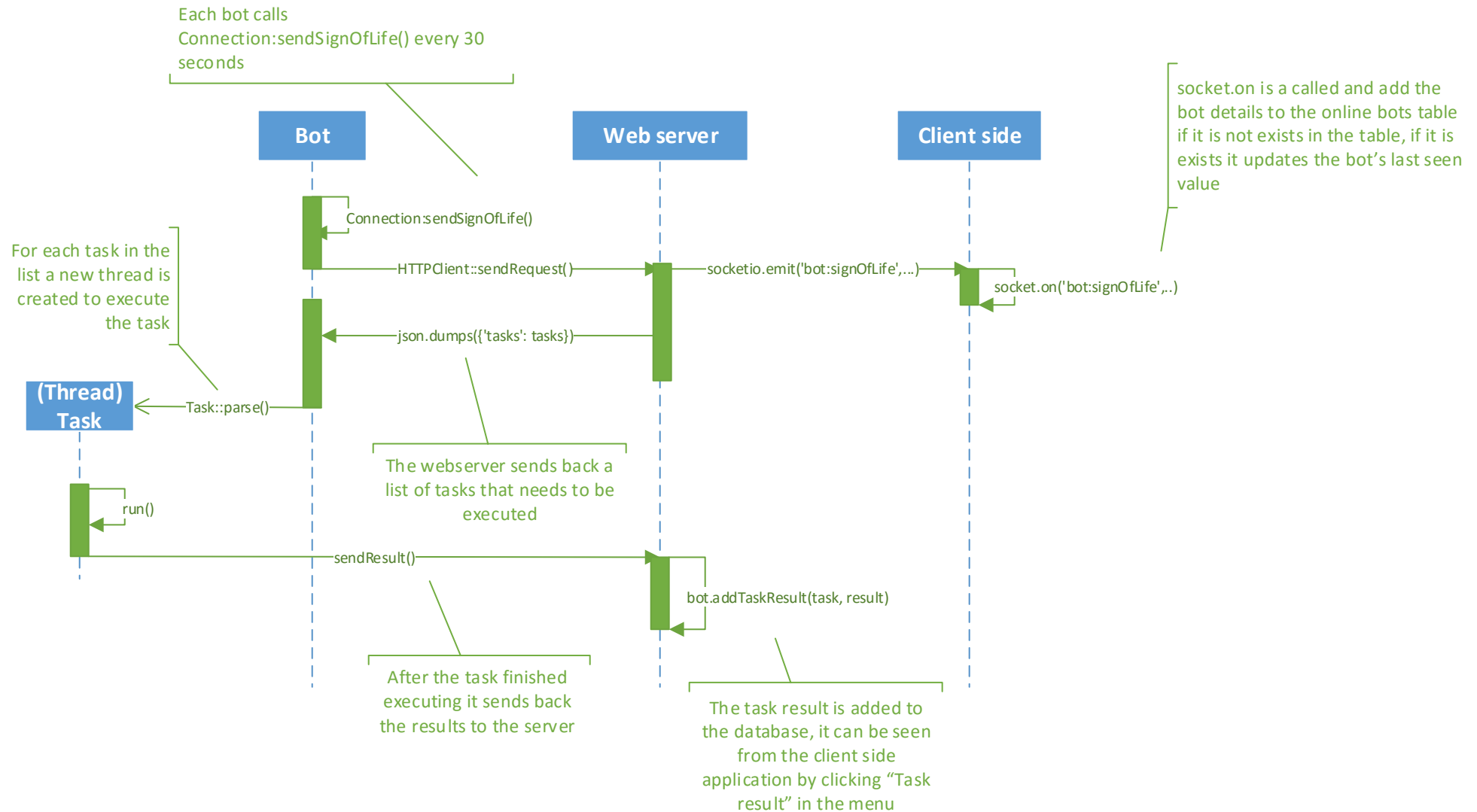
```

m __init__(self, dbfile)
m addBot(self, bot)
m isNewBot(self, bot)
m updateLastSeen(self, bot)
m getFirstSeen(self, bot)
m getBots(self, limit)
m addTaskResult(self, bot, task, result)
m getResults(self, offset)
m dict_factory(self, cursor, row)
m connect_db(self, dbpath)
m init_db(self)
m initdb_command(self)
m get_db(self)
m close_db(self, error)
m getOfflineBots(self)
f dbfile

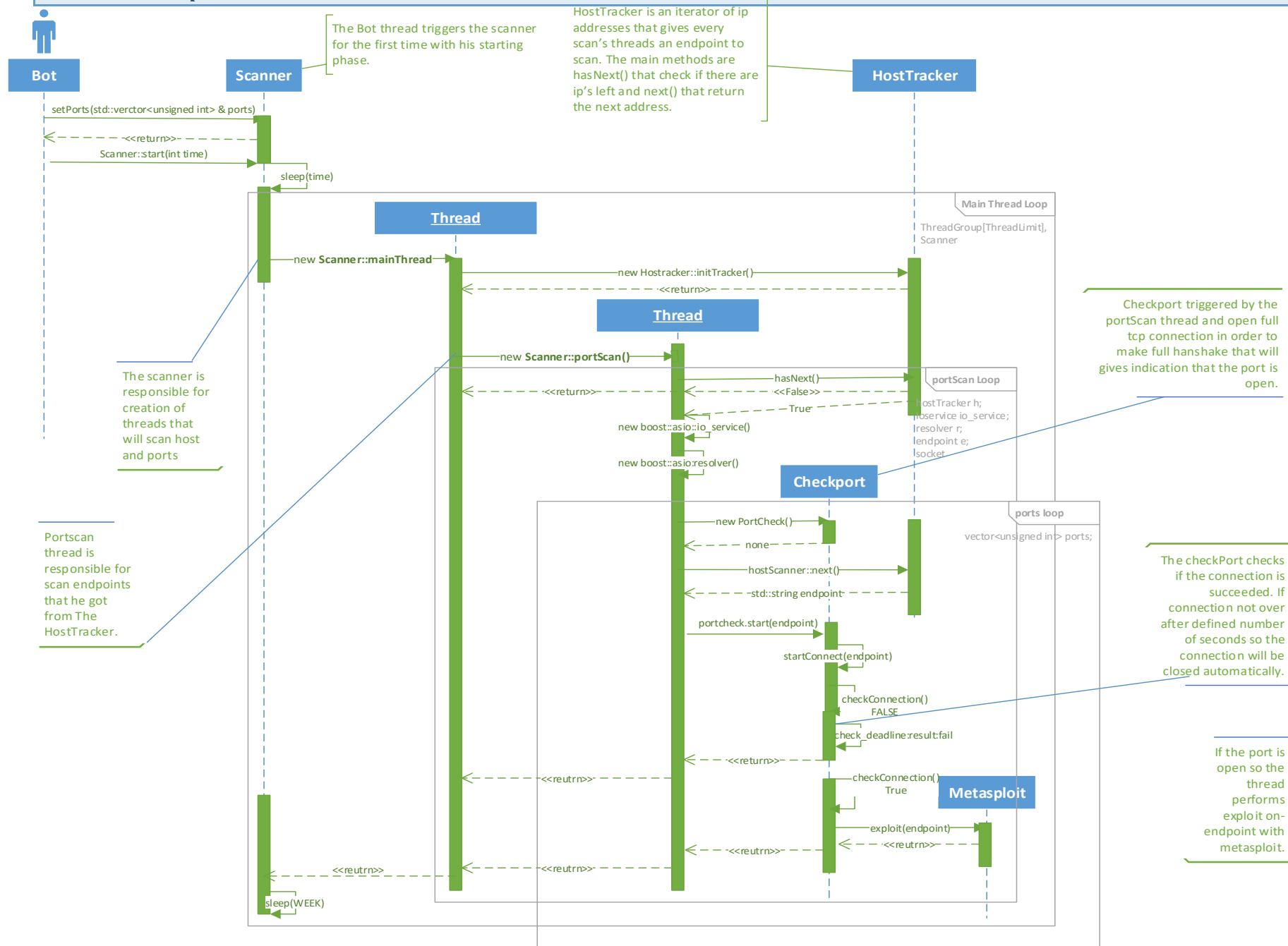
```


13. Sequence UML diagram

Server<-->Bot communication and task execution



Port scan and exploit execution



14. GUI

Online bots page: Sending task to multiple bots that have common value is possible using the filter textbox

Tasks panel

Bots connected: 5

On bot connect

Add task

Filter
Task (to filtered)

Send task

Proxy

SOCKS5


Bandwidth test


ID	Version	OS	CPU	Memory	GPU	First seen	Last seen	Uptime	Last IP	Country
ubuntu_2000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 14m ago	~1463829656s ago	1d 21h 16m	127.0.0.1	
ubuntu_3213000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 12m ago	~1463829641s ago	1d 21h 17m	127.0.0.1	
ubuntu_1111000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 10m ago	~1463829628s ago	1d 21h 19m	127.0.0.1	
cent_os_00co3311cc2f	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 5m ago	~1463829634s ago	1d 21h 24m	127.0.0.1	
fedora_00co3311a328	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 4m ago	~1463829642s ago	1d 21h 25m	127.0.0.1	


Tasks menu: opens a list of possible tasks that can be sent to a bot , the menu will open when right clicking on a bot from the table.

ID	Version	OS	CPU	Memory	GPU	First seen	Last seen	Uptime	Last IP	Country
ubuntu_2000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @	1.94		0d 0h 0m ago	~1463829476s ago	1d 21h 16m	127.0.0.1	
ubuntu_3213000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @	1.94		0d 0h 0m ago	~1463829461s ago	1d 21h 17m	127.0.0.1	
ubuntu_1111000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @	1.94		0d 0h 0m ago	~1463829478s ago	1d 21h 19m	127.0.0.1	
cent_os_00co3311cc2f	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 0m ago	~1463829484s ago	1d 21h 24m	127.0.0.1	
fedora_00co3311a328	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 0m ago	~1463829462s ago	1d 21h 25m	127.0.0.1	

Tasks

 Proxy

 SOCKS5

 Bandwidth test

Task sending dialog: user is prompted for a list of parameters that are needed in order to send the chosen task

The screenshot displays the MetaNet interface. A 'Tasks panel' is visible on the left, showing 'Bots connected: 5', 'On bot connect' (with an input field), and 'Filter' (with an input field). A 'Proxy parameters' dialog box is overlaid in the center, prompting for a 'Port:' (with an input field) and featuring 'Cancel' and 'Send' buttons. In the background, a table lists tasks with columns: ID, Version, OS, CPU, Memory, GPU, First seen, Last seen, Uptime, Last IP, and Country. Two tasks are listed, both with ID 'ubuntu_2000c2983113b' and 'ubuntu_3213000c2983113b'.

ID	Version	OS	CPU	Memory	GPU	First seen	Last seen	Uptime	Last IP	Country
ubuntu_2000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 0m ago	~1463829506s ago	1d 21h 16m	127.0.0.1	
ubuntu_3213000c2983113b	v0.1	4.2.0-36-generic x86_64	Intel(R) Core(TM) i5-5300U CPU @ 2.30GHz	1.94		0d 0h 0m ago	~1463829515s ago	1d 21h 17m	127.0.0.1	

Task result table: contains the result/data of the task that were sent back to the server by the bots after executing a task

id	bot_id	task	date
	ubuntu_000c29f24e5e	Proxy	2016-04-14 06:09:49
	ubuntu_000c29f24e5e	Proxy	2016-04-14 07:29:51
	ubuntu_000c29f24e5e	Proxy	2016-04-14 07:39:51
	ubuntu_000c29f24e5e	Proxy	2016-05-20 21:15:00
	ubuntu_000c29f24e5e	Proxy	2016-05-20 21:39:39
	ubuntu_000c2983113b		2016-05-20 21:45:04
	ubuntu_2000c2983113b		2016-05-21 11:18:54
	ubuntu_1111000c2983113b		2016-05-21 11:18:58
	cent_os_00co3311cc2f		2016-05-21 11:19:04
	ubuntu_3213000c2983113b		2016-05-21 11:19:05

Botnet debug window: Display debug information when running the bot, this window will be displayed only in debugging environment.

```

00:00:00 DEBUG      : main => Started.
sh: 1: /usr/lib/.bot/rbenv/shims/ruby: not found
00:00:00 DEBUG      : Botnet => Starting...
00:00:00 DEBUG      : Botnet => Getting installed path...
00:00:00 DEBUG      : Botnet => Got Installed Path
00:00:00 DEBUG      : Botnet => Checking if the bot is installed.
00:00:00 DEBUG      : Botnet => Checking if the bot is already running.
00:00:00 DEBUG      : Botnet => Saving bot process id.
00:00:00 DEBUG      : Botnet => Started.
00:00:00 DEBUG      : TaskManager => Starting...
00:00:00 DEBUG      : Connection => Starting...
00:00:00 DEBUG      : TaskManager => Started.
00:00:00 DEBUG      : Connection::mainLoop => Started.

{"tasks": []}
00:00:00 DEBUG      : Connection::mainLoop => Sent sign of life.
00:00:30 DEBUG      : Connection::mainLoop => Sent sign of life.
OK
{"tasks": []}
OK
{"tasks": []}
00:01:00 DEBUG      : Connection::mainLoop => Sent sign of life.

00:01:30 DEBUG      : Connection::mainLoop => Sent sign of life. {"tasks": []}

{"tasks": [{"params": {"Port": "id"}, "id": 1}]}
00:02:00 DEBUG      : Connection::mainLoop => Sent sign of life.
00:02:01 DEBUG      : Proxy => Opened

00:02:30 DEBUG      : Connection::mainLoop => Sent sign of life.
{"tasks": [{"params": {"Port": "ls"}, "id": 1}]}

00:02:30 DEBUG      : Proxy => Opened

{"tasks": [{"params": {"Port": "ip addr"}, "id": 1}]}
00:03:00 DEBUG      : Connection::mainLoop => Sent sign of life.
00:03:00 DEBUG      : Proxy => Opened

00:03:30 DEBUG      : Connection::mainLoop => Sent sign of life.
OK
{"tasks": []}
00:03:36 DEBUG      : main => Stopping threads.
00:03:36 DEBUG      : Botnet => Stopping...
00:03:36 DEBUG      : Botnet => Clearing running process id...
00:03:36 DEBUG      : main => Joining threads.
00:03:36 DEBUG      : Connection => Stopping...
00:03:36 DEBUG      : TaskManager => Stopping...
00:03:36 DEBUG      : TaskManager => Stopped.

```

15. Software test plan

The test plan outlines the scope, approach, resources, and schedule of all testing activities, It identifies the items and features to be tested and the types of testing that are needed.

Testing strategy

Functional testing – Conducting tests to the functional requirements that were defined in the SRS.

Graphical User interface testing – Conducting tests that will ensure the GUI meets its specification.

Automation testing – Conducting tests that will ensure that the operations that the program executes by itself (without user interaction) are working correctly.

Boundary testing – Those tests using the extremes of the input domain, e.g. maximum, minimum, just inside/outside boundaries, typical values, and error values.

Performance testing – Tests that check how the system performs in terms of responsiveness and stability under a particular workload.

Tests environmental requirements

The goal is to create a testing environment as close to the development environment as possible and to provide an automated black box test suite that can be run when changes are made to the software. The test environment is the same windows environment used for software development. Below are the hardware and software requirements for the test environment.

Hardware

- Intel Core 2 Duo, Quad Core i3, i5, i7, or higher
- AMD Athlon II, Phenom X4, FX or higher
- at least 4GB of RAM.

Software

- VMWare or VirtualBox installed.
- The server side application will be installed in a virtual machine that has Ubuntu 14.04 LTS or above installed in it.
- The bot will be installed in a virtual machine that has Ubuntu 14.04 LTS or above installed in it.
- The PC that contains the VMs will act as the client (The bot master) and will need a browser that has JavaScript and CSS support.

16. Software test description and report

Test No.	1			
Test purpose	Test if a connection is made between the bot and the server and the bot is being displayed in the online table when it connects.			
Preconditions	Server is running			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Execute the bot on a machine			
3	Wait 10 seconds			
4	Look if the bot is added to the online bot list	The bot is added	Passed	The bot is added

Test No.	2			
Test purpose	Test if bot search works			
Preconditions	Server is running, Two bots with different ID number are connected to the server			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Choose one online bot			
3	Write the chosen bot ID to "Filter" field.	Only the chosen bot is displayed in the table.	Passed	Only the chosen bot is displayed in the table.

Test No.	3			
Test purpose	Test if Proxy task that needs to be sent to filtered bots opens the correct corresponding dialog			
Preconditions	Server is running, A bot connected to the server, The bot is filtered so it is the only one that is displayed in the table.			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Select 'Proxy' from Task (to filtered) Dropdown			
3	Click Send task	'Proxy parameters' string is displayed on the title of the dialog.	Passed	'Proxy parameters' string is displayed on the title of the dialog.

Test No.	4			
Test purpose	Test if bot menu is working			
Preconditions	Server is running, A bot connected to the server			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Right click on a bot in the table	A task menu opens	Passed	A task menu opened

Test No.	5			
Test purpose	Bot received and executed a task			
Preconditions	Server is running, A bot is running and connected to the server			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Right click on a bot in the table	A task menu opens		
3	Choose 'Proxy'	A parameter dialog opens		
4	Enter '8080'			
5	Click 'Send Task' button	The dialog disappears		
6	Open the bot debug logging window			

7	Wait 0-30 seconds	`DEBUG: Proxy => Opened` is displayed	Passed	`DEBUG: Proxy => Opened` is displayed
---	-------------------	---------------------------------------	--------	---------------------------------------

Test No.	6			
Test purpose	Test if the server is not allowing to send a task with empty parameters.			
Preconditions	Server is running, A bot is running and connected to the server			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Right click on a bot in the table	A task menu opens		
3	Choose 'Proxy'	A parameter dialog opens		
4	Click 'Send Task' button	'No enter' sign is displayed in the mouse cursor and the dialog stays open.	Passed	'No enter' sign was displayed in the mouse cursor and the dialog stayed open.

Test No.	7			
Test purpose	Test if Proxy task from menu that needs to be sent a specific bot opens the correct corresponding dialog			
Preconditions	Server is running, A bot is running and connected to the server			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel			
2	Right click on a bot in the table	A task menu opens		
3	Choose 'Proxy'	A parameter dialog opens and 'Proxy parameters' string is displayed on the title of the dialog and a textbox labeled 'Port' is displayed in the body of the dialog.	Passed	A parameter dialog opens and 'Proxy parameters' string is displayed on the title of the dialog and a textbox labeled 'Port' is displayed in the body of the dialog.

Test No.	8			
Test purpose	Check Upper bound of port scanner – check what the last IP address that scanned.			
Preconditions	Bot is installed, running and ready to start a scan			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Start scan			
2	Finish scan ports			
3	Check Last IP octet	Last IP octet "XXX.XXX.XXX.254"	Passed	Last IP octet "XXX.XXX.XXX.254"

Test No.	9			
Test purpose	Checks if Port Scanner is will not scan when computer disconnected form network.			
Preconditions	Bot is running;			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Bot starts scan.			
2	Disconnect bot host from network.			
3	Wait for bot to finish scan.	Error no adapter	Passed	Error expected: no adapters.

Test No.	10			
Test purpose	Check CPU usage with thread limit of 10 less than 30%			
Preconditions	*THREAD_LIMIT=10, hardware: CPU i3 or above, 4 GB RAM or above.			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Start bot			
2	Start scan			
3	Check process Bot CPU usage	Less than 30% CPU usage	Passed	CPU less than 30%

Test No.	11			
Test purpose	Test if bots disappear from offline bots table after bot connected.			
Preconditions	Server is running			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Connect to the server command and control panel and choose offline bots			
2	Execute the bot on a machine			
3	Wait 60 seconds			
4	Look if the bot is disappearing from offline bots table	The bot is disappearing.	Passed	The bot is disappearing

Test No.	12			
Test purpose	Check asynchronous connection to endpoint that is not exist.			
Preconditions	Server is running			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Start scan endpoint that no exits			
2	Send asynchronous TCP connection			
	Wait for handle connections			
3	Wait 3-5 seconds	Check deadline and close socket.		Check deadline and close socket.
4	Get result of connection	Unreachable host or no route to host	Passed	No route to host.

Test No.	13			
Test purpose	Check deadlocks with thread limit=10			
Preconditions	Configure IP address with subnet class C, bot is running.			
Steps to follow:	Step description	Expected result	Passed/Failed	Actual result
1	Change Thread limit to 10			
2	Set 3 ports 22,8080,5000			
3	Set deadline time to 3			
4	Start scan ports			
5	Wait (3*255*3 +15) sec	Finished scan properly with no deadlocks	Passed	Finished scan properly with no deadlocks