Implementation report of network anomalies detection for discovering Botnet

Beuque Eric

June 3, 2009

Introduction

This document describe how I implemented different medium to detect anomalies on the network which can allow to discover Botnet on a local network.

First, we will see how to use the SCADE method presented in [2], to detect malicious port's scanners on the network. After, we will see how to detect inbound infection with SLADE by analysing payload of captured packets.

1 Using SCADE method

The SCADE method (Statistical sCan Anomaly Detection Engine) is divided in two part. The first one consist in inbound scan detection, to detect ports scanning on a victim. The second, for outbound scan detection, allow to detect if the victim is scanning computers ports in the network. Figure 1 show globally the architecture of the implementation.

1.1 Scan Detection

As a test case, we can use the famous Nmap to simulate port scanning on a host which works exactly like malware. NMap provides several method to do scanning with using TCP and UDP described in [3].

We can easily detect a TCP scan by analyzing packets transmitted between the source and the destination. When a host want to scan ports on the victim, he send a packet with the SYN message. If the port is open, the host receives the SYN/ACK response. Then, if the port is closed the host receives the RST message.

For a UDP scan, we have also to analyze packets transmitted between the source and the destination. When a host want to scan ports on the victim, he send a packet with an empty (no data) UDP header. If the port is open, the host receives an answer. Then, if the port is closed the host receives a ICMP port unreachable error (type 3).

Nevertheless, these techniques do not allow to detect all medium of port scanning, like the idle method which needs a very complex algorithm, to detect the scan.

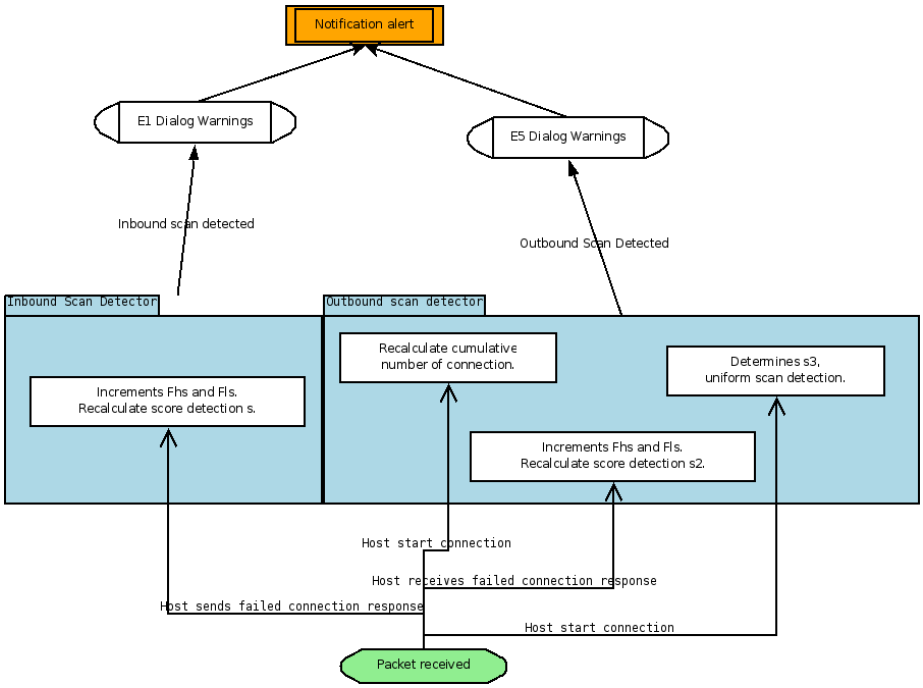

Figure 1: Implementation Scheme for SCADE

1.2 Ports classification

SCADE needs to classify ports between two categories. One for high-severity ports (HS) and one for low-severity ports (LS).

We put in HS ports most used which a scanner will try probably in first. It should be better to keep only approximately 30 ports. DShield [1] provides reports about most ports attacked per target and source, which should be used respectively for algorithms of the inbound scan and outbound scan detection. Table 1 is a non-exaustive list of HS ports. All others should be marked as LS ports.

1.3 Inbound Scan Detection

The inbound scan detection consists to detect a port scan targeted to an internal host.

For this, we try to establish a score calculated as $s = w_{hs}F_{hs} + w_{ls}F_{ls}$. First, w_{hs} and w_{ls} are respectively, an affected weight for each ports type. Secondly, F_{hs} and F_{ls} are the cumulative number failed attempts, in other words when someone try to connect to a closed port (for example, when the victim send an RST packet) within a time window (few minutes).

At this time, I don't found what is the value we need for w_{hs} and w_{ls}. But we assume, that $w_{hs} > w_{ls}$.

When the score is upper than a reference value (unknown), we can notify that we have detect the E1 dialog warnings.

Table 1: High severity ports table

Port	TCP	UDP	Service
7	X	-	Echo
21	X	-	FTP
22	X	-	SSH
23	X	-	TelNet
53	-	X	DNS
67,68	-	X	DHCP
80	X	-	HTTP
135,1025	X	-	DCOM
161,162	-	X	SNMP
445	X	-	NetBIOS
445	X	-	NetBIOS
3127	X	-	My-Doom
5000	X	-	UPNP
5900	X	-	VNC

1.4 Outbound Scan Detection

The outbound scan detection is divided in three parts which consist to analyze traffic emitted by an internal host :

First, outbound scan rate (s1), we try to detects local hosts that conduct high-rate scans across large sets of external addresses. We have to estimate the cumulative number of connection per host, and detect a strange connection rate. However, we count all scan attempts, but not only the failed connection attempts.

Secondly, we want to detect outbound connection failure rate (s2). It works exactly like the inbound scan detection. We calculate the anomaly score $s2 = \frac{w_{hs}F_{hs}+w_{ls}F_{ls}}{C}$, where C is the total number of scans from the host within a time window. Others variables means the same thing that in the inbound scan.

Finally, we try to show if a host is scanning often the same target (Normalized entropy of scan target distribution (s3)). Based on a anomaly scoring technique calculating a Zipf distribution of outbound address connection patterns, we define : $s3 = \frac{H}{ln(m)}$, where the entropy of scan target distribution is $H = -\sum_{i=1}^{m} p_i ln(p_i)$, m is the total number of scan targets, and pi is the percentage of the scans at target i.

These tree module allow to detect suspicious outbound scan. A alert happens for each module, when we have $s_i > t_i$, where t_i is a threshold. Then we can use a voting scheme based on AND, OR, or MAJORITY to combine the alerts from the three modules. For example, the AND rule dictates that SCADE issues an alert when all three modules issue an alert.

1.5 UML model

Here a proposition for an UML model in figure 2. Basically, the global analyzer (ScadeAnalyzer) receives packets from the capture engine. It analyzes the packet and it records data with classes ScadeIPAnalyze and ScadeTargetIPAnalyze for internal hosts. ScadeAnalyzer extends AnalyzerTool which is part of Botnet Detector and provides, for each anomalies detection tool, a way based

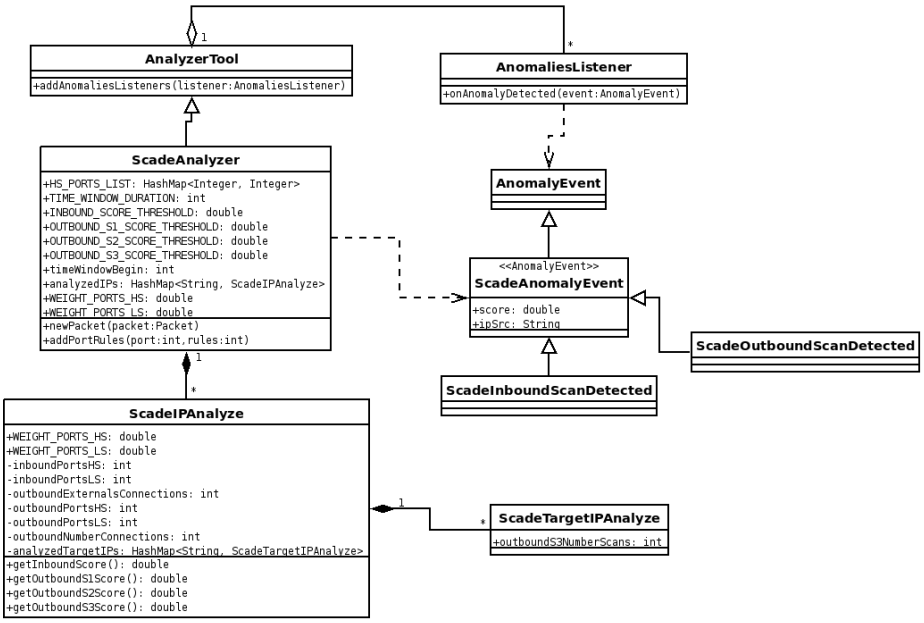

Figure 2: UML model for SCADE implementation

on listeners to notify the Botnet detector when an anomaly occurs. So when SCADE detect an anomaly, it notify listeners attached to it, informing the type of detection with data (IP, score...).

HS_PORTS_LIST is a HashMap which allow to store the profile of each port. For example, we can define that the port 22 should be considered as high severity port for inbound connection on TCP and UDP by using the method $addPortRules(port, rules)$ where $port = 22$ and rules is a bitwise param, $rules = TYPE\_TCP|TYPE\_UDP|SCAN\_INBOUND$.

Some variables and thresholds have an unknown value. So we need first to develop the software and run it on a safe network to determine what are reference values.

2 Using SLADE method

SLADE (Statistical payLoad Anomaly Detection Engine) has goal to detect an anormal payload on transmitted packets for a specified protocol. For example, on port 80 which is used for HTTP, the payload should contain a big part of ASCII characters. On port 22 for SSH, the payload is encrypted, so the occurrence frequency of each byte in the payload should be the same. SLADE is just an improvement of the PAYL method which give better results.

2.1 PAYL method

PAYL considers that the payload is just a byte vector. Each byte in the vector can have a value between 0 and 255. In a first time, we need to develop a tools to construct a profile of each port. For these each port, we have to determine

the mean and the standard deviation of each possible value for a byte in the packets of the specified protocol. The profile has to be established by analyzing a great number of TCP packets on a safe network, where all machine are trusted and without virus or bot installed.

Once we have profile for each port, we can run SLADE on the network to detect anomalies. It have to calculate deviation distance of a test payload from the normal profile using a simplified Mahalanobis distance : $d(x, y) = \sum_{i=0}^{255}(|x_i - y_i|)/(\sigma_i + \alpha)$ where x_i is the number of byte with value i in the test payload, y_i is the mean from the profile, σ_i is the standard deviation from the profile, and α is a smoothing factor.

Finally, if the calculated distance is upper than a threshold, we can emit an alert.

2.2 SLADE method

SLADE improves PAYL. It doesn't consider the payload as a vector bytes but as a string. So we analyse the frequency occurence of each substring of a fixed size n in the string. We know that for a normal payload of length = L, there is a total of substrings : $l = (L - n + 1)$.

Nevertheless, using the PAYL method to store a profile will need 256^n (e.g., even for a small $n = 3$, $256^3 = 2^{24} \approx 16M$). SLADE use a fixed vector counter (with size v) to store substring distribution of the payload. When processing a payload, we sequentially extract substring *str* and apply hash function $h()$ on it. After, we increment the counter at the vector space indexed by $h(str) mod v$. In JAVA, we can easily use the existant String.hashCode() methods to convert *str* in integer.

In fact, PAYL is just a SLADE profile where the vector size is $v = 256$ and the substring size is $n = 1$ (one character). SLADE uses less space than the PAYL mapping and is more efficient complexity in calculating distance is $O(v)$ instead of 256^n.

2.3 UML model

Figure 3 presents the UML diagram of the SLADE implementation. The main class SladeAnalyzer receives packets from the capture engine and extends AnalyzerTool like ScadeAnalyzer.

This class has a calibrator which allow to construct the SLADE profile by analyzing packets. The module can be in two modes. In the calibration mode, packets are analyzed and data are extracted in the SladeCalibrator which contains one vector for each packet on each port. Once the calibrator has collected enough packets, we can calculate the Slade profile from these data with the getProfile() method. It calculates the mean and the standard deviation for each possible value of the vector on each port and returns them in a SladeProfile object. This object implements Serializable and can be saved and loaded for future usage.

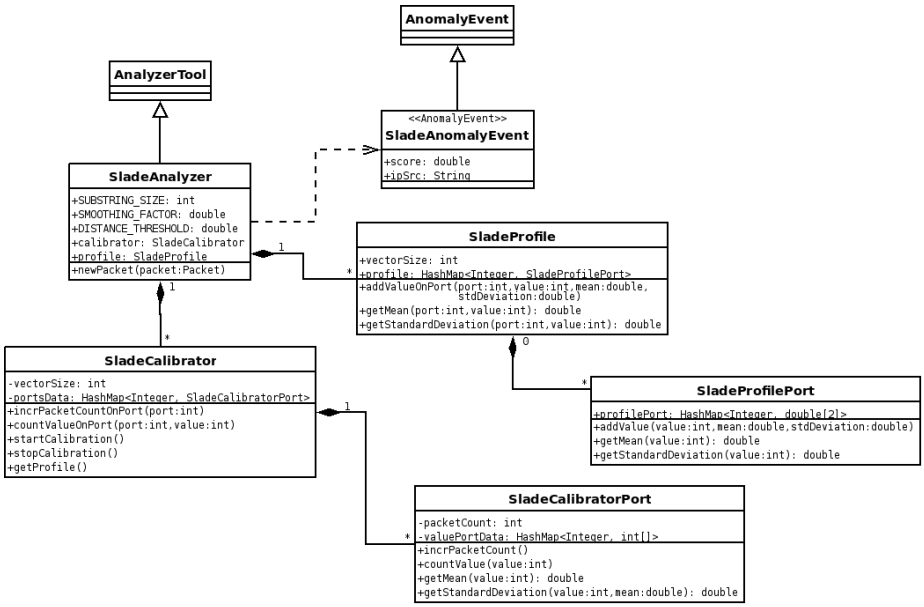

Figure 3: UML model for SLADE implementation

3 Botnet Detector

You can see the Botnet Detector interface on figure 4. When an anomaly is detected, it displayed in the table showing its type, its date and its description with the score and the source IP.

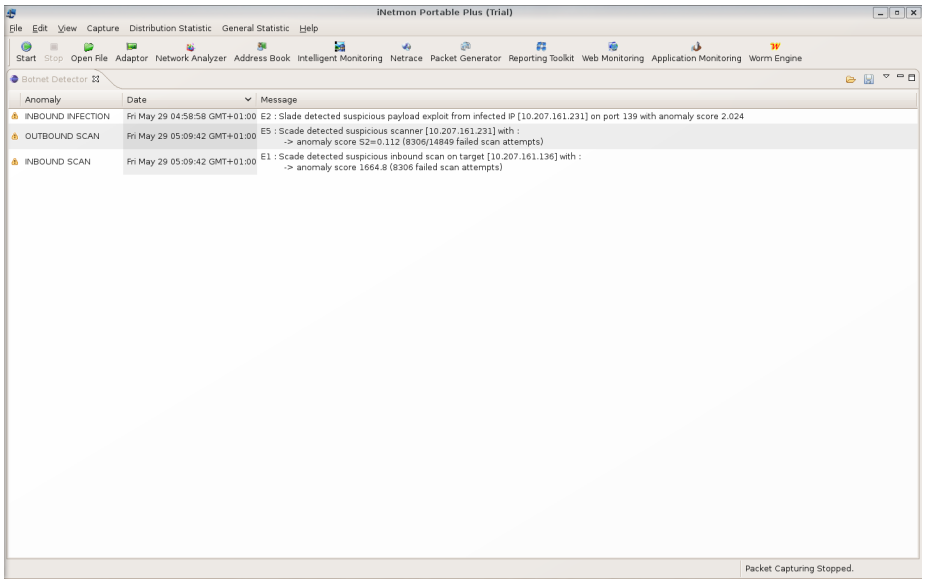

Figure 4: Botnet Detector GUI

The interface has a menu to configure Botnet Detector and manage profile.

At this moment, the Botnet Detetor profile just contains data of the SladeProfile but it can be improve to support others tools.

- Open a profile : Use this option to open a profile from a BDP file (Botnet Detector Profile). The opened profile will replace the current profile and will be used directly by SladeAnalyzer for processing packets. This function loads from the file the serializable object SladeProfile.
- Save the current profile : Use this option to save the current profile generated by the calibration tool into a BDP file. This function stores in the file the serializable object SladeProfile.
- View current profile : This opens a window which display the values stored in the current profile.
- Merge two profile : This opens the merge tool to combine data from two differents profiles. Support only SladeProfile.
- Start calibration tools : This starts the calibration tool to generate a profile from the current captured packets.
- Preferences : Open the Botnet Detector preferences pages.

3.1 Merge two profile

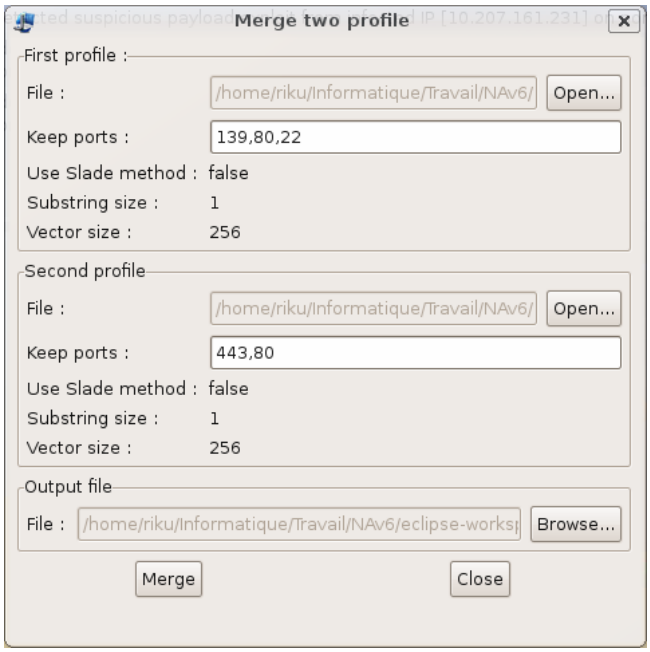

Figure 5: Botnet Detector merge tool

The merge tool (figure 5) asks the user to choose two input file and one output file. These input files must have the same profile (vector size, substring size...). After, he has to choose which ports he want to keep from each profile

for the output file. If the the two profiles have both a same port, the user needs to choose from which profile he want to keep the port.

3.2 Calibration tool

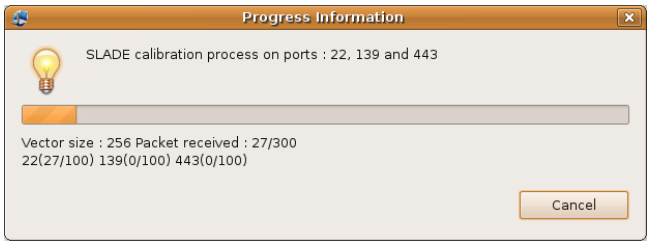

Figure 6: Botnet Detector calibration tool

The calibration tool captures packets and collects data to construct a Slade-Profile at the end. The calibration tool works with preferences in the Slade preferences page. So before running a calibration, you need to configure the port numbers you want to analyse, the packet count per port to capture, if you want to use the Slade or the PAYL method. If you wan to use Slade, you have also to configure the substring size in the analyzed payload and the vector size.

3.3 Preferences

Botnet Detector integrates a preference dialog to configure many variables (figure 7). For now, the variables are divided in three categories : Botnet Detector, SCADE and SLADE.

- Botnet Detector
  - Net prefix : the net prefix of the current analyzed network
  - Subnet mask : the subnet mask of the current analyzed network
  - Max anomalies to display : the maximum anomalies to display in the table
- SCADE
  - Time window (milliseconds) : the duration in milliseconds of the time window to analyse the data. At the end, SCADE processes data to determine if an anomalies happens.
  - Weight ports HS : the double value which correspond to the weight of the ports HS
  - Weight ports LS : the double value which correspond to the weight of the ports LS
  - Inbound scan threshold : the double value which correspond to the threshold of the inbound scan
  - Outbound S1 score threshold : the double value which correspond to the threshold of the outbound S1 score

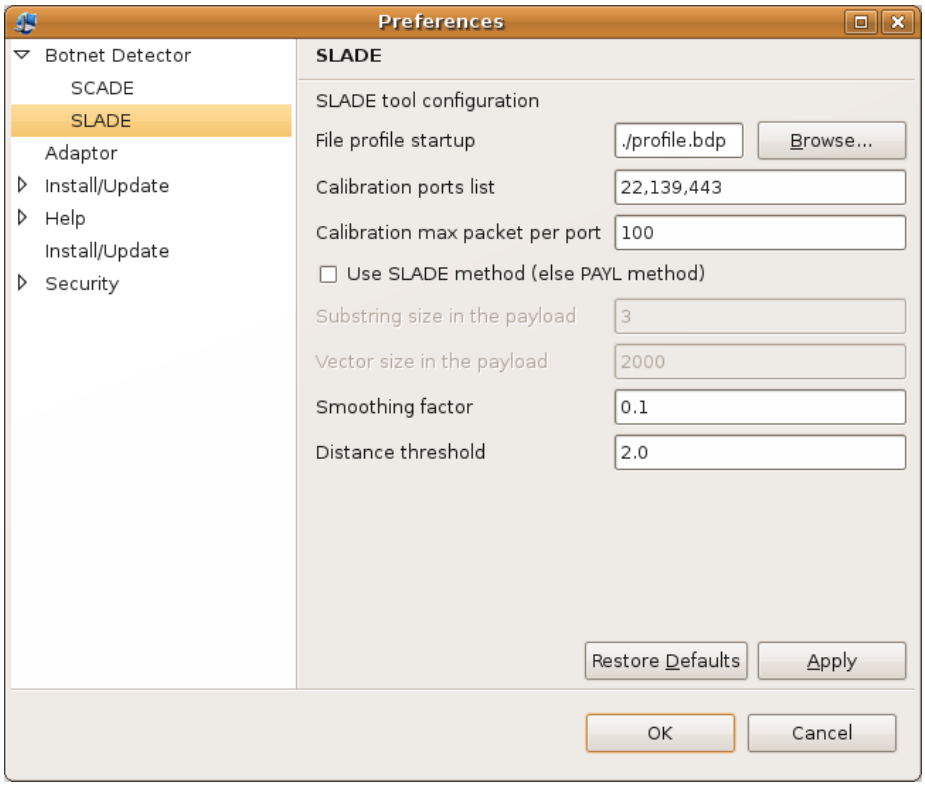

Figure 7: Botnet Detector preferences

  - Outbound S2 score threshold : the double value which correspond to the threshold of the outbound S2 score
  - Outbound S3 score threshold : the double value which correspond to the threshold of the outbound S3 score

- SLADE
  - File profile startup : the file that will be loaded at the Botnet Detector startup and that contains a profile
  - Calibration ports list : the list of port that the calibration tool must analyse. Each port are separated by ','
  - Calibration max packet per port : the packet count per port that the calibration tool have to capture before
  - Use SLADE method (else PAYL method) : if checked, the collected data by the calibration tool will provide a profile to be used with the SLADE method, else for the PAYL method
  - Substring size in the payload : if you use the SLADE method, you have to specify the substring size for the analyzed payload
  - Vector size in the payload : if you use the SLADE method, you have to specify the vector size for the profile of each port

- Smoothing factor : the double value which correspond to the smoothing factor needed to calculate the distance
- Distance threshold : the double value which correspond to threshold of the calculated distance

4 Testing

I didn't have enough time to make complete test of my applications. So I just give here a plan for testing SCADE and SLADE.

4.1 SCADE

SCADE detects port scanning. First, we need to determine the base value of the thresholds on the network depending of the weight of ports HS and LS and the time window. To do this, we need to run the analyzer on a safe network where we are sure that there is no machine infected by a bot. This should be run for a long time (few days) with minimal thresholds on a network where there is a normal traffic. This is normal that often some connection attemps fail, but this must be with a small rate (e.g if an user try to connect to an unreachable service). So this test allow to detect these rates and determine the good thresholds for the network. I think these thresholds can be different for each network depending of the size of them. We can accept that thresholds can detect a wrong anomaly but this shouldn't have to happen often. After, this is the correlation engine which determine the importance of the anomaly with it score.

To easily generate anomaly on the network for scan detection we can use the famous software NMap. We can try the detection of scan TCP and UDP. We can run scan on each port with commands :

```
# for a TCP scan :
$> nmap -p- theTargetIP
# for a UDP scan (in root mode)
$> nmap -sU -p- theTargetIP
```

The number of detected connection attempts can depend with the power of the machine which run the scan, and the one which listen the network. After, we can try to detect the inbound and outbound scan by creating a network and trying to infect it with a botnet.

4.2 SLADE

For SLADE, we need to construct a profile of each sensitive port. So we have to create a small safe network where we can install all services assosiated to a port. Figure 8 shows the network to implement to do this. We see that this network must contain few machines on Windows and Linux where we install services on theirs default port : Apache on port 80, SSH on 22, FTP on 21, MySQL, DNS, Telnet, SQLServer, MSN, Bittorent... We have to be the most exaustive than possible, but lot of port aren't used by famous services, so, theirs profiles don't need to be etablished. JPortable must capture all packets from external to internal network, it's why we install it on a machine which also has a

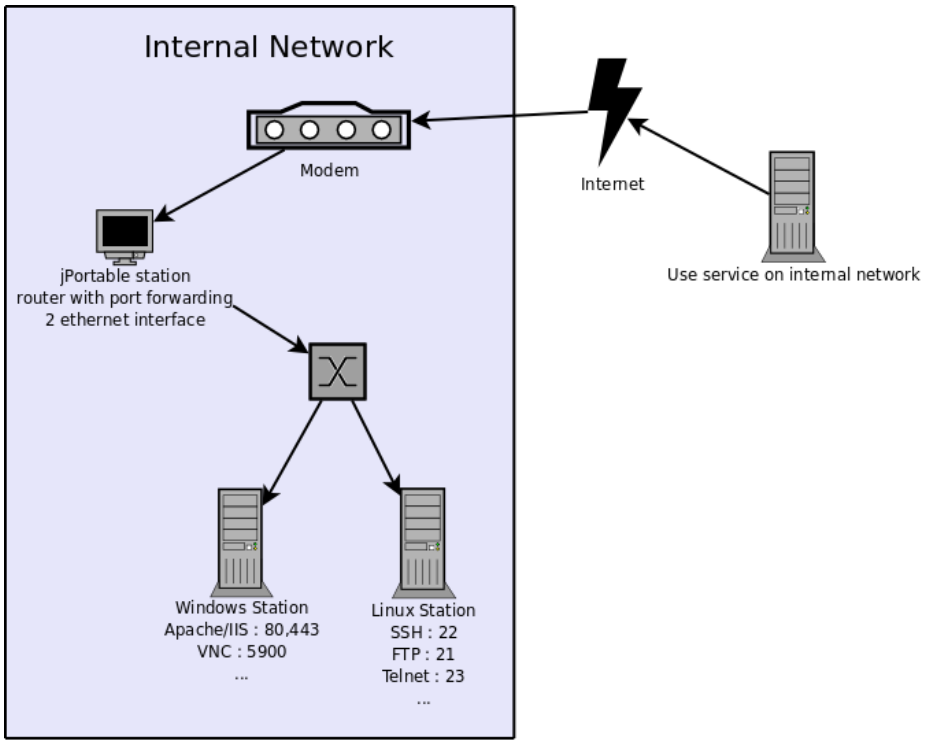

Figure 8: Network scheme to use to generate profile

router function. It must have two Ethernet interfaces. One interface to connect the network to internet and the other to connect the machines plugged over a switch. We also need to enable port forwarding on the router to allow an external user to access on the service. Once the network is correctly installed, we can run the calibration tool on JPortable after configuring it. The external user has to generate traffic on each protocole to provide data for the calibrator.

When profiles are etablished, we need to configure others networks where we can deploy a botnet for testing our profiles. It's why it's important to create separate networks as you can see on figure 9. We need a Botnet, so I think the best way is to find the source code of an existant one and to modify it to specify the targeted network for infection and to disable its dangerous features. Then we deploy a network with a machine where we install the bot. And in the other side, we create a small unsecure network without security tool like antivirus, firewall, antispyware... This will improve the infection step. Moreover all ports which we have a profile must be opened and forwarded to the unsecure machines. Once the infection happened you should be able to detect the anormal payload in the transmitted packets. The infection step can take lot of time but you should be able to detect the inbound scan first, with SLADE analyzer.

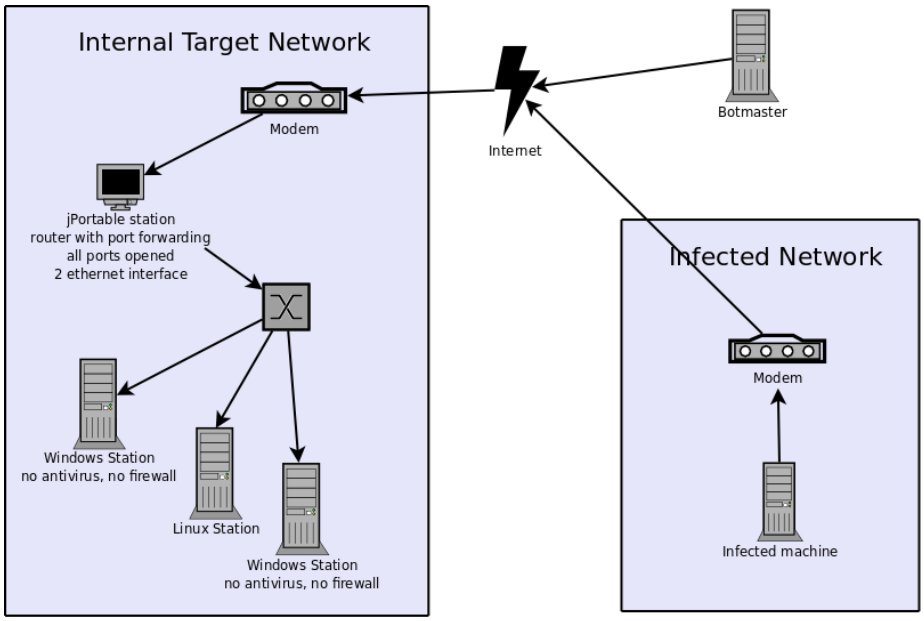

Figure 9: Network scheme to detect Botnet infection

References

[1] dshield.org. Most attacked ports reports. `http://www.dshield.org/portreport.html`, March 2009.

[2] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. Technical report, College of Computing (Georgia Institute of Technology) and Computer Science Laboratory (SRI International), 2007.

[3] nmap.org. Nmap reference guide : Port scanning techniques. `http://nmap.org/man/en/man-port-scanning-techniques.html`, March 2009.