Practical Malware Analysis based on Sandboxing

Mihai Vasilescu, Laura Gheorghe, Nicolae Tapus
Computer Science Department
University Politehnica of Bucharest
Bucharest, Romania
mihai.vasilescu@cti.pub.ro, {laura.gheorghe, nicolae.tapus}@cs.pub.ro

***Abstract*—The past years have shown an increase in the both number and sophistication of cyber-attacks targeting Windows and Linux operating systems. Traditional network security solutions such as firewalls are incapable of detecting and stopping these attacks. In this paper, we describe our distributed firewall solution Distfw and its integration with a sandbox for malware analysis and detection. We demonstrate the effectiveness and shortcomings of such a solution. We use Cuckoo to perform automated analysis of malware samples and compare the results with the ones from manual analysis. We discover that Cuckoo provides similar results in a considerable amount of time.**

Keywords—malware, network security, sandbox, malware analysis

I. INTRODUCTION

A threat summary report by F-Secure informs us of an alarming low percentage of malware detection and mitigation worldwide: 15-20 malware blocked per 10000 users [1]. More and more criminal organizations change their profile moving to cyber-crime due to low risks involved in cyber-attacks and fast profits as a result. Due to this trend, many companies are subject to malware attacks, ranging from drive-by attacks to sophisticated targeted attacks.

In order to counter this trend, both security communities and companies have put effort in developing methods to protect their assets, using security products: breach detection appliances, web and email security platforms, etc.

The days when cyber criminals exploited computers and servers using only a couple of scripts that they would share amongst themselves are gone. Now cyber criminals are using special tailored tools designed to bypass our defenses and to avoid them. It would be a colossal task to analyze every suspicious piece of software that exists, therefore automated malware analysis can be very useful.

In this paper, we present our distributed firewall, called Distfw, implemented using iptables for filtering traffic and IPsec for securing the communication. We integrated Distfw with a sandbox for automatically analyzing malicious applications.

We integrated the Cuckoo sandbox solution into our distributed firewall and performed automated experimental evaluation of malware samples. In addition, we performed manual step-by-step malware analysis on the same samples and discovered similar information about the behavior of the executable. From our experimental evaluation, we can conclude that Cuckoo provides similar details regarding the behavior of the malware in a considerable smaller amount of time than manual analysis.

This paper is structured as follows: Section II presents the background and some of the related work, Section III describes the design and implementation of Distfw, Section IV includes details about the integration of Distfw with Cuckoo, Section V describes the experimental evaluation including manual analysis and automatic analysis using Cuckoo, and Section VI presents the conclusions and future work.

II. BACKGROUND AND RELATED WORK

In this section, we explore the types of malware analysis, define the concept of sandbox systems, explain the advantages and disadvantages of sandbox solutions and describe available sandbox systems.

A. Malware Analysis

By analyzing a malware, one can determine a lot of useful information: IPs of Command and Control (C&C) servers, indicators of compromise, file access, whether the malware was packed or not, if it has obfuscated code or not, whether it spreads on the network or not. All this information can help an investigator determine the impact of the attack: was it a targeted attack or just a dry-by malware attack; the sophistication of the attack can point out whether the attacker is an individual, an organized cyber-crime group, or even a national security entity.

In order to perform malware analysis, several methods are available [2][3][4]: static analysis, memory analysis, dynamic analysis and automatic analysis.

Static analysis – this method consists in obtaining information about the malware without executing it. We may obtain the strings, detect packers and observe certain operations using the disassembled version.

Memory analysis – this method allows investigating the memory of the infected system in order to reveal hidden information about the malware, such as DLLs, hidden network connections, etc.

Dynamic analysis or behavioral analysis – examining the malware's interaction with the host system at runtime. This includes analyzing the way the malware interacts with the file system, with the network, processes, etc. This method requires an isolated environment, in which the malware is launched and its behavior is monitored.

Automatic malware analysis – this is usually done via sandbox systems. There are many reasons for using an automated malware analysis system, the most important being the ability to uncover artifacts about the malware in a fast manner. Usually, analyzing a malware requires a lot of effort and skill for the examiner. Even though it does not always produce the same level of details, it is a very good starting point in analyzing suspicious files.

B. *Sandbox Systems*

A sandbox is a security platform for running unknown executables in a dedicated environment without the risk of affecting the production systems. Basically, sandboxes are virtualized environments that simulate live systems to ensure that the tested executable runs in way that is almost the same, if not identical, to the real environment. Similarly, security sandboxes are used to execute suspicious files in a safe environment in order to analyze their behavior and to provide information regarding attacks to security officers.

Sandbox systems allow monitoring suspicious executable files in an isolated environment while eliminating the risk of compromising live systems. Another important aspect is that sandboxes eliminate a lot of human effort derived from complex and lengthy tasks such as disassembling the executable in order to understand its purpose. This method allows a security administrator without extensive training in malware analysis to perform a triage of suspicious files and only send confirmed malware for analysis.

Nowadays, most of the security products on the market use one or more types of sandboxing for behavior analysis, most of them are closed source (proprietary), but some notable solutions are provided as open source.

Some of the well known sandbox systems available are Cuckoo Sandbox and Zerowine. Cuckoo Sandbox [5] is an open source malicious code behavioral analysis system that consists of two components: 1) a Cuckoo Host system, which handles the execution and analysis, and 2) Analysis Guests, which are isolated virtual machines where the malware is executed and results are sent back to the Cuckoo Host. Analysis is done using packages - scripts that define automated tasks that the Cuckoo Host should perform during the analysis of a target application. Moreover, Cuckoo supports URL analysis in the guest machines, adding the possibility to determine whether the website that the user is accessing is malicious or not.

Zerowine [6] is an open source system that dynamically analyzes the behavior of target applications using Wine. The disadvantage of this solution is that it only analyzes Windows applications and it does so in an emulated environment (Wine).

It should be noted that there are also websites that allow users to submit files for analysis, eliminating the need for dedicated hardware for deployment and usage of dedicated sandbox systems. However, this method does not provide the best results as some malware target systems that have certain applications installed or specific registry keys in case of Microsoft Windows.

The most popular websites that provide such services are Anubis and Malwr. Anubis [7] is an online platform that allows a user to submit Windows executables or Android APKs for analysis.

Malwr [8] is an online platform developed by the same team that designed Cuckoo Sandbox. Users are allowed to submit files or URLs for analysis. Additionally, users can view reports of other submitted files if the original submitter configured the analysis report as public.

III. Distfw Design and Implementation

We implemented a distributed firewall, called Distfw [9][10]. In the design of this firewall, we have tried to meet the requirements of a distributed firewall, as stated by Steve Bellovin [11]:

- Policy language: the policy language includes the commands given to the scriptable firewall provided by the operating system. In our solution, this is accomplished using iptables commands.
- System management: This is provided by implementing a master/client framework.
- Safe distribution: The security policy is distributed securely to the clients using IPSec in order to secure the policy distribution.

The main components of the Distfw architecture are: the master node and the client nodes. The master node is responsible for the deployment and configuration of openswan on client nodes, log file integration from all clients. The master node is also responsible for the deployment of iptables rules based on the company policy but also according to sandbox malware analysis of URLs accessed by users or applications submitted for analysis.

The master/client framework is based on a series of scripts, implemented using bash and Expect, which reside on the management machine. The functionalities offered by these scripts are summarized in the following:

- Adding a client machine to the framework.
- Adding iptables rules to a client
- Listing iptables rules running on a client
- Capturing URLs accessed by users

A. *Adding a client machine to the framework*

Based on an IP address and initial credentials (user/password with elevated privileges) to remotely access the client machine via SSH, the script adds a user *distfw* on the client machine. After that, it modifies the /etc/sudoers file, in order to allow the *distfw* user to manipulate iptables.

For obvious security reasons, it is recommended to limit the use of the root user as much as possible, and delegate privileges instead. For this reason, a more elegant solution was to create a user, which will be used only to manipulate iptables on the client machine.

Next, the master node checks whether openswan is installed on the client node, and if not, it automatically installs the package.

Following this step, the master node generates a configuration file for the communication via IPSec with the client node and deploys it on the client node.

The last step of this process is to perform an initial configuration of iptables on the client machine. This is done via a pre-defined list of rules, which are meant to perform a lockdown to the system. This was done using the 3 pre-defined iptables chains:

- INPUT – all traffic destined to the client machine is dropped, with one exception: SSH connections generated from the management machine.
- FORWARD – all traffic that is supposed to pass through the client machine is dropped. Considering the fact that most clients in this framework are intended to be either end-user machines, or servers (web, email, etc), we consider that there is no real need to allow traffic to be forwarded.
- OUTPUT – only traffic marked as related, or established is allowed to pass, everything else is dropped.

This initial lockdown is performed in order to prevent any other traffic to or from the client, until it is secured with the iptables rules provided, dictated by the security policy.

The last step of this script is related to creating a new chain of rules, called *distfw*. From this point on, this is the chain that is used to process traffic related to the client machine.

One of the reasons behind the creation of this chain is to protect the communication channel provided by the INPUT chain, allowing the client to communicate with the management machine. In this case, all further rules are added in the *distfw* chain of rules, while the INPUT, OUTPUT and FORWARD are not modified from now on.

An important consequence of this is that, even if by mistake we send the client a rule that would block SSH connection with the management machine, this rule will never trigger, and communication with the management machine will not be lost.

While the issue of getting locked out might not seem a big deal when it comes to client machines that are in your campus LAN, it can be a serious problem when it comes to client machines, which are in a different geographic location.

B. *Adding iptables rules to a client.*

This script is used for adding a firewall rule for a remote client. The script will prompt the administrator to type in the iptables rule that we want to add and the IP address of the client machine, where it will be applied. This is useful when only one or two rules are necessary to be applied.

However, if there are numerous IP's that have to be blocked on several clients, this solution does not scale. To prevent this sort of situations, the script allows the administrator to load a file containing iptables rules and send the rules to the client to be applied.

Finally, the configuration is saved on the client using *iptables-save*. This is done in order to provide a fallback in case the client machine powers off.

C. *Listing iptables rules running on a client*

This script is used for displaying the rules configured on a certain client. The script prompts the administrator to type the IP address of the client, and after that, it prints out the active configuration of iptables on that client.

Similar to the methods described above, the *distfw* user created earlier is used to connect via SSH connection over the IPSec VPN tunnel to the client node. Using expect, the client's iptables rules are printed out on the management node.

D. *Capturing URLs accessed by users*

The script launches *httpry*, which is an application designed to monitor HTTP traffic, in the background, by recording the URLs accessed by users and periodically sending them to the master node. In our case, they are introduced in the sandbox system for analysis.

This method is implemented in order to analyze potential malicious URLs used by hackers for drive-by downloads. The script records HTTP requests (GET, POST, etc.) and then filters the results until only the fully qualified domain name is left, which is sent for analysis.

IV. CUCKOO INTEGRATION

In our implementation, we chose to integrate Cuckoo sandbox to our Distfw distributed firewall solution. The main reason for this choice was the fact that Cuckoo allows guest machines using Virtual Box, KVM or VMware, permitting the analysis of files and applications on most operating systems. Moreover, Cuckoo facilitates the analysis of URLs, thus enabling the administrator to determine whether the websites accessed by the users are legitimate or not. All analysis results are stored in a database, and can be later used for reporting or retrospective analysis.

We installed the Cuckoo sandbox on the same machine that is responsible for managing the distributed firewall (Figure 1). The idea behind this was to integrate the benefits of automated analysis and to use those results in the distributed firewall. Cuckoo relies heavily on Python and there are some Python applications necessary to properly run Cuckoo (e.g. Magic, Pydeep, Yara, Pefile, etc.).

Considering that 93% of malicious programs involved in web attacks are executed via malicious URLs [12], we chose to integrate the sandbox in the distributed firewall implementation and automatically analyze URL requests. In order to achieve this, we created a script that listens for URL requests, saves them to a file and sends them over the existing

IPSec VPN channel to the distributed firewall manager machine. Then each URL is submitted to analysis via Cuckoo.

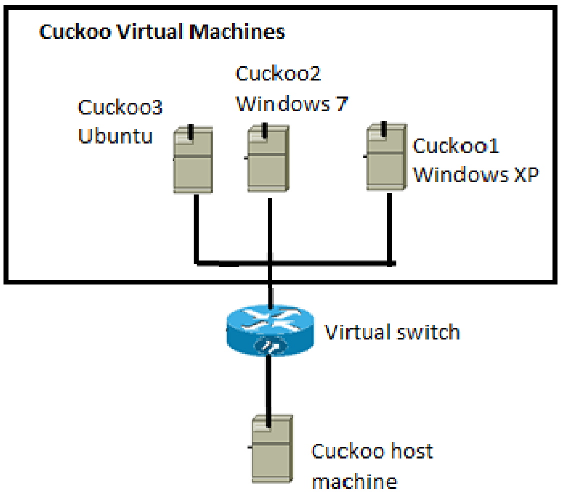


Fig. 1. Cuckoo system architecture

V. Tests and Results

In order to test the implemented solution and to prove the effectiveness of the sandbox, we chose to perform a manual analysis of the malware and an automated analysis using Cuckoo.

In this section we present the analysis of a botnet malware, which infects the computer and then connects to the C&C server waiting for commands.

A. Manual analysis

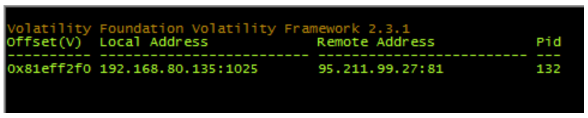
In order to perform the manual analysis, we created a Windows XP virtual machine, which includes Wireshark, DumpIt [13], Volatility [14] and Ida [15] for disassembly.

The first step is to launch the malware and monitor its network activity. However, by monitoring its activity in Wireshark, and by analyzing the output of the netstat command, we do not obtain any information regarding network traffic. There is no network activity reported by Wireshark or netstat command. This can mean two things. Either the malware is in an idle state because it detected that it is executed in a virtual environment or its activity is hidden from the winpcap driver. In order to determine which is the case, we launch Wireshark in the host environment. Now, we can see that the guest machine is actually making connections to an IP address: 95.211.99.27 with the destination port set to 81.

With this information, we can conclude that the application is connecting to its C&C server. However, there are still a lot of questions to be answered: What does the malware do? How does the malware hide its connections? What type of malware is it?

In order to answer these questions, we follow up with a memory analysis of the malware. We begin by dumping all information in the RAM memory with DumpIt and then load the dump in Volatility (an open source multi-platform framework that enables the memory analysis). We choose to run the following jobs: *connections, pstree, and dllist, dlldump*.

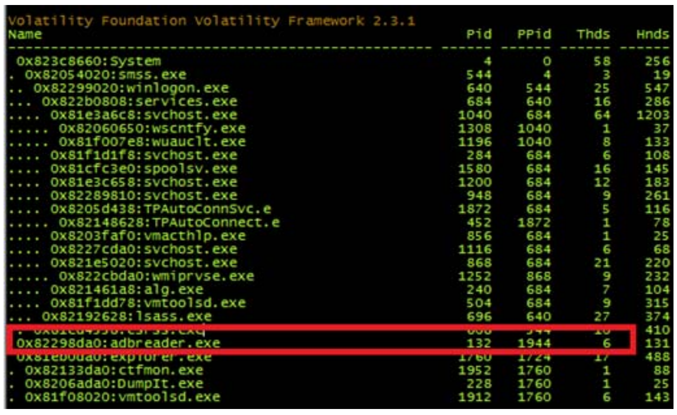
As we look at the output of the *connections* job in Figure 2, it confirms what Wireshark has already pointed out to us on the host machine: that the malware has network activity, even though it is not visible on the guest machine.



```
Volatility Foundation Volatility Framework 2.3.1
Offset(V)  Local Address             Remote Address            Pid
---------- ------------------------- ------------------------- ---
0x81eff2f0 192.168.80.135:1025       95.211.99.27:81           132
```

Fig. 2. Output of Volatility connections job

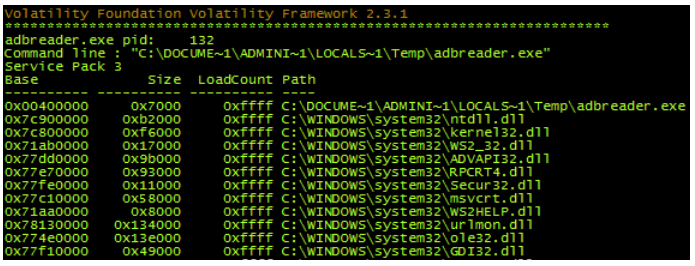
Knowing that the process responsible for the connection on the guest machine has PID 132, we issue a *pstree* job to list the active processes. The output is shown in Figure 3, which points out that the malware runs as *adbreader.exe*.



```
Volatility Foundation Volatility Framework 2.3.1
Name                                                  Pid   PPid   Thds   Hnds
-------------------------------------------------- ------ ------ ------ ------
 0x823c8660:System                                      4      0     58    256
. 0x82054020:smss.exe                                 544      4      3     19
.. 0x82299020:winlogon.exe                            640    544     25    547
... 0x822b0808:services.exe                           684    640     16    286
.... 0x81e3a6c8:svchost.exe                          1040    684     64   1203
..... 0x82060650:wscntfy.exe                         1308   1040      1     37
..... 0x81f007e8:wuauclt.exe                         1196   1040      8    133
.... 0x81f1d1f8:svchost.exe                           284    684      6    108
.... 0x81cfc3e0:spoolsv.exe                          1580    684     16    145
.... 0x81e3c658:svchost.exe                          1200    684     12    183
.... 0x82289810:svchost.exe                           948    684      9    261
.... 0x8205d438:TPAutoConnSvc.e                      1872    684      5    116
..... 0x82148628:TPAutoConnect.e                      452   1872      1     78
.... 0x8203fafo:vmacthlp.exe                          856    684      1     25
.... 0x8227cda0:svchost.exe                          1116    684      6     68
.... 0x821e5020:svchost.exe                           868    684     21    220
..... 0x822cbda0:wmiprvse.exe                        1252    868      9    232
.... 0x821461a8:alg.exe                               240    684      7    104
.... 0x81f1dd78:vmtoolsd.exe                          504    684      9    315
... 0x82192628:lsass.exe                              696    640     27    374
. 0x81cd4550:csrss.exe                                608    544     10    410
 0x82298da0:adbreader.exe                             132   1944      6    131
 0x81eb0da0:explorer.exe                             1760   1724     17    488
. 0x82133da0:ctfmon.exe                              1952   1760      1     88
. 0x8206ada0:DumpIt.exe                               228   1760      1     25
. 0x81f08020:vmtoolsd.exe                            1912   1760      6    143
```

Fig. 3. Output of pstree job in Volatility

The next step is to list the DLLs used by the *adbreader.exe* application. This is done by issuing the *dlllist* job. The result can be seen in Figure 4. However, with this information we cannot tell which DLL is part of the malware. However, we can submit the DLLs to an antivirus check to determine which one is part of the malware, and we find that *module.132.2498da0.40000.dll* is actually the malware itself, while the rest of the DLLs are actually harmless.

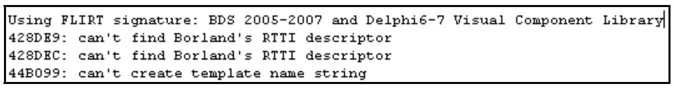


```
Volatility Foundation Volatility Framework 2.3.1
************************************************************************
adbreader.exe pid:    132
Command line : "C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\adbreader.exe"
Service Pack 3
Base             Size  LoadCount Path
---------- ---------- ---------- ----
0x00400000     0x7000     0xffff C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\adbreader.exe
0x7c900000    0xb2000     0xffff C:\WINDOWS\system32\ntdll.dll
0x7c800000    0xf6000     0xffff C:\WINDOWS\system32\kernel32.dll
0x71ab0000    0x17000     0xffff C:\WINDOWS\system32\WS2_32.dll
0x77dd0000    0x9b000     0xffff C:\WINDOWS\system32\ADVAPI32.dll
0x77e70000    0x93000     0xffff C:\WINDOWS\system32\RPCRT4.dll
0x77fe0000    0x11000     0xffff C:\WINDOWS\system32\Secur32.dll
0x77c10000    0x58000     0xffff C:\WINDOWS\system32\msvcrt.dll
0x71aa0000     0x8000     0xffff C:\WINDOWS\system32\WS2HELP.dll
0x78130000   0x134000     0xffff C:\WINDOWS\system32\urlmon.dll
0x774e0000   0x13e000     0xffff C:\WINDOWS\system32\ole32.dll
0x77f10000    0x49000     0xffff C:\WINDOWS\system32\GDI32.dll
```

Fig. 4. Output of dlllist in Volatility

At this point, we have managed to identify the file that is responsible for the infection, and the IP address of the C&C server, but we still do not know what the malware actually does. To answer this question we proceed to load the identified file in a disassembler and analyze the code.

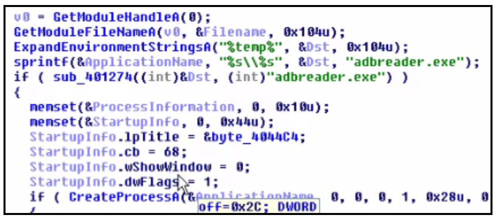
First, we need to know which programming language was used to write the malicious code. Loading adbreader.exe in Ida, we discover that the application was written in Delphi, as we can see from Figure 5. This information is useful for the analysis of the target DLL.



```
Using FLIRT signature: BDS 2005-2007 and Delphi6-7 Visual Component Library
428DE9: can't find Borland's RTTI descriptor
428DEC: can't find Borland's RTTI descriptor
44B099: can't create template name string
```

Fig. 5. Compiler information extracted from Ida

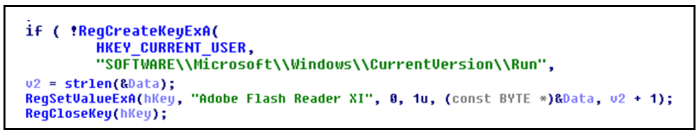
We begin to analyze the DLL function by function in order to understand the malware's behavior.



```
v0 = GetModuleHandleA(0);
GetModuleFileNameA(v0, &Filename, 0x104u);
ExpandEnvironmentStringsA("%temp%", &Dst, 0x104u);
sprintf(&ApplicationName, "%s\\%s", &Dst, "adbreader.exe");
if ( sub_401274((int)&Dst, (int)"adbreader.exe") )
{
  memset(&ProcessInformation, 0, 0x10u);
  memset(&StartupInfo, 0, 0x44u);
  StartupInfo.lpTitle = &byte_4044C4;
  StartupInfo.cb = 68;
  StartupInfo.wShowWindow = 0;
  StartupInfo.dwFlags = 1;
  if ( CreateProcessA(&ApplicationName, 0, 0, 0, 1, 0x28u, 0
                      off=0x2C; DWORD
```

Fig. 6. CreateProcess function

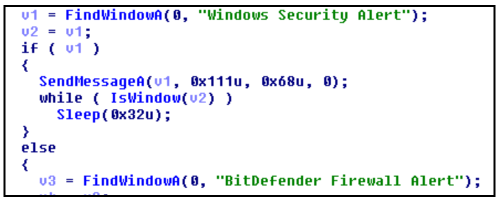
By looking at the code in Figure 6, we understand that the malware creates the *adbreader.exe* file and launches a process on the guest machine. It then creates the registry keys in "SOFTWARE\Microsoft\Windows\CurrentVersion\Run" in order to ensure the malware's survival in case of a reboot of the operating system. The function that creates the registry keys is presented in Figure 7.



```
if ( !RegCreateKeyExA(
        HKEY_CURRENT_USER,
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
v2 = strlen(&Data);
RegSetValueExA(hKey, "Adobe Flash Reader XI", 0, 1u, (const BYTE *)&Data, v2 + 1);
RegCloseKey(hKey);
```

Fig. 7. The malware creates a registry key

In Figure 8, we can also see that the malware is programmed to verify if the target system is protected by Bitdefender or if Windows Firewall is enabled. If one of them is enabled, the malware is set to sleep for a predetermined period of time in order to hide its presence.

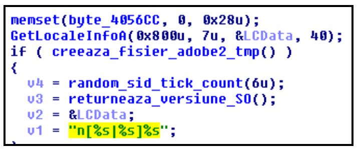


```
v1 = FindWindowA(0, "Windows Security Alert");
v2 = v1;
if ( v1 )
{
  SendMessageA(v1, 0x111u, 0x68u, 0);
  while ( IsWindow(v2) )
    Sleep(0x32u);
}
else
{
  v3 = FindWindowA(0, "BitDefender Firewall Alert");
```

Fig. 8. The malware checks if Bitdefender or Windows Firewall is enabled

Furthermore, we can find in the code the following sequences: "PASS", "NICK", "USER", "PONG", "JOIN", "PRIVMSG", "QUIT", which are commands used in IRC communication. With this information, we can deduce that, after infecting the system, the malware connects to the C&C server on an IRC channel and reports to the attacker.

In addition, we find the channel name is "jobs", and the "NICK" is set to be generated in a random fashion each time it connects to the server. The format of the nickname is presented in Figure 9: "n[%s|%s]%s" (for example n[USA|XP]395455), where the number is randomly generated each time based on processor tick – clock cycle, XP is the operating system version, and n[USA] is the same each time the malware connects to the server.



```
memset(byte_4056CC, 0, 0x28u);
GetLocaleInfoA(0x800u, 7u, &LCData, 40);
if ( creeaza_fisier_adobe2_tmp() )
{
  v4 = random_sid_tick_count(6u);
  v3 = returneaza_versiune_SO();
  v2 = &LCData;
  v1 = "n[%s|%s]%s";
```

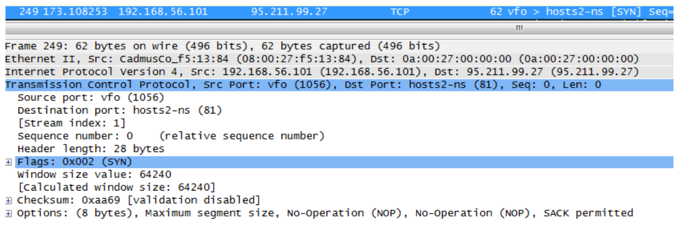
Fig. 9. Random generation of the nickname for IRC communication

In conclusion, we found that the malware is actually a botnet, the IP address of the C&C server, that it creates registry keys in order to ensure its survival and auto-start after reboot, that it checks if the target has an antivirus installed. With all this information, we can finish the analysis and compare our results with the report provided by the analysis done by Cuckoo.

B. Automated analysis using Cuckoo sandbox

In order to analyze the application using the sandbox, it is only necessary to launch the sandbox service (*python cuckoo.py)* and submit a job (*submit.py -machine guest_machine_name filename).* This automatically starts the Windows XP machine installed earlier, launches the application and analyzes its behavior. In the end, it provides a report containing the details of the analysis. It is important to note that the level of detail of the report is dependent to the number of services activated in Cuckoo.

The report of the analysis provided in two forms: an HTML report and a PCAP file containing the network traffic done by the analyzed application. From the PCAP we can see that, once executed, the application tries to connect to the C&Cserver, as presented in Figure 10.



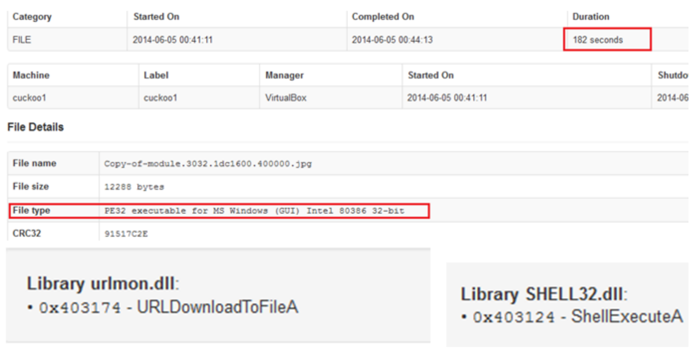
```
249 173.108253  192.168.56.101      95.211.99.27        TCP        62 vfo > hosts2-ns [SYN] Seq=

Frame 249: 62 bytes on wire (496 bits), 62 bytes captured (496 bits)
Ethernet II, Src: CadmusCo_f5:13:84 (08:00:27:f5:13:84), Dst: 0a:00:27:00:00:00 (0a:00:27:00:00:00)
Internet Protocol Version 4, Src: 192.168.56.101 (192.168.56.101), Dst: 95.211.99.27 (95.211.99.27)
Transmission Control Protocol, Src Port: vfo (1056), Dst Port: hosts2-ns (81), Seq: 0, Len: 0
  Source port: vfo (1056)
  Destination port: hosts2-ns (81)
  [Stream index: 1]
  Sequence number: 0    (relative sequence number)
  Header length: 28 bytes
⊞ Flags: 0x002 (SYN)
  Window size value: 64240
  [Calculated window size: 64240]
⊞ Checksum: 0xaa69 [validation disabled]
⊞ Options: (8 bytes), Maximum segment size, No-Operation (NOP), No-Operation (NOP), SACK permitted
```

Fig. 10. Wireshark capture created by Cuckoo

From the report generated by the sandbox analysis, we can observe that the duration of the analysis was approximately 3 minutes. Considering the fact that the manual analysis took roughly 4 hours, we can conclude that the sandbox is much faster in analyzing applications.

Figure 11 shows that among other findings, that the submitted application is a PE32 executable for Windows operating Systems, that it uses *urlmon.dll* in order to download a file.



Category	Started On	Completed On	Duration
FILE	2014-06-05 00:41:11	2014-06-05 00:44:13	182 seconds

Machine	Label	Manager	Started On	Shutdo
cuckoo1	cuckoo1	VirtualBox	2014-06-05 00:41:11	2014-06

File Details

File name	Copy-of-module.3032.1dc1600.400000.jpg
File size	12288 bytes
File type	PE32 executable for MS Windows (GUI) Intel 80386 32-bit
CRC32	91517C2E

Library urlmon.dll:
- 0x403174 - URLDownloadToFileA

Library SHELL32.dll:
- 0x403124 - ShellExecuteA

Fig. 11. Cuckoo analysis report

The report also provides a list of strings (Figure 12) found in the application.

```
SYSTEM\CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile\Authoriz
edApplications\List
SOFTWARE\Microsoft\Windows\CurrentVersion\Run
%temp%
windowsupdate
adbreader.exe
Adobe Flash Reader XI
BitDefender Firewall Alert
Windows Security Alert
CreateProcessA
website=1
\adobe2.tmp
```

Fig. 12. Information extracted by the sandbox

It is important to notice that the sandbox does not state in the report whether the analyzed file is a malicious or a legitimate file. This conclusion remains to be drawn by the analyst as in the case of a manual analysis. However, the incredibly low time required to analyze files in conjunction with the large number of information provided to the analyst can help in investigating suspicious files.

VI. Conclusions

Considering the increasing number of cybernetic attacks, the number of files that need to be analyzed and the time cost of analyzing a single file, we consider that the implementation of an automated analysis system is paramount in order to increase security within a network.

We designed and implemented a distributed firewall for Linux operating system, by using iptables for filtering, IPSec for securing network communication and a sandbox to automatically analyze URLs accessed by users in order to detect malicious applications. The sandbox was integrated in the distributed firewall solution in order to provide an automated mechanism for the detection of potential malware applications.

We presented how Cuckoo solution can be integrated into our distributed firewall and evaluated its functionality by analyzing a malware sample. We also performed manual malware analysis of the same sample. The automated Cuckoo solution produces the same results in a considerable smaller time.

The malware analysis solution proposed in this paper represents the extension of the Distfw scripts functionality [9][10]. We consider that the addition of the sandbox analysis and the extension of functionality of the Distfw scripts is a major leap forward in putting together a security enforcement solution for networks.

Considering that the results of the sandbox analysis have been promising, we plan to provide a method that will detect downloaded email attachments and automatically submit an analysis. A Verizon data breach report for 2013 [16] reveals that e-mail attachments are used as a main vector of attack in almost 80% of attacks related to espionage. Considering this alarming high number of attacks using email attachments as a means of attack, we consider that an automated attachment analysis will be an enhancement to our solution.

VII. Bibliography

[1] Mobile Threat Report Q1, http://www.f-secure.com/static/doc/labs_global/Research/Mobile_Threat_Report_Q1_2014.pdf (Last Access: August 2014)

[2] E. Skoudis, L. Zeltser, “Malware: Fighting Malicious Code”, Prentice Hall, 2003

[3] C. H. Malin, E. Casey, J. M. Aquilina, “Malware Forensics Field Guide for Linux Systems”, Syngress, 2014.

[4] M. Sikorski, A. Honig “Practical malware analysis”, Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software, No Starch Press, 2012.

[5] Cuckoo Sandbox documentation, http://docs.cuckoosandbox.org/en/latest/ (Last Access: August 2014)

[6] Zero Wine: Malware Behavior Analysis, http://zerowine.sourceforge.net/ (Last Access: August 2014)

[7] Anubis Malware Analysis Tool, https://anubis.iseclab.org/?action=about (Last Access: August 2014)

[8] Malwr, https://malwr.com/about/ (Last Access: August 2014)

[9] M. Vasilescu, A distributed firewall implementation, Master Thesis, University Politehnica of Bucharest, 2014.

[10] M. Potoroaca, Implementation of a distributed firewall over Windows platforms, Master Thesis, University Politehnica of Bucharest, 2014.

[11] S. Ioannidis, A.D. Keromytis, S.M. Bellovin, and J.M. Smith. Implementing a distributed firewall, ACM Conference on Computer and Communications Security, 2000.

[12] Kaspersky Security Bulletin 2013, http://www.securelist.com/en/analysis/204792318/Kaspersky_Security_Bulletin_2013_Overall_statistics_for_2013 (Last Access: August 2014)

[13] Dumpit, http://www.aldeid.com/wiki/Dumpit (Last Access: August 2014)

[14] Volatility, https://code.google.com/p/volatility/ (Last Access: August 2014)

[15] Ida disassembler, https://www.hex-rays.com/products/ida/ (Last Access: August 2014)

[16] Data Breach Investigations Report 2014, www.verizonenterprise.com/DBIR/2014/reports/rp_Verizon-DBIR-2014_en_xg.pdf (Last Access: August 2014)