INSTITUTO SUPERIOR TÉCNICO

Departamento de Engenharia Informática

Forensics Cyber Security

MEIC / METI 2018-2019 – 1st Semester

Digital Forensics Report

Ana Beatriz Nogueira	João Silveira	Martim Zanatti
82433	80789	82517

GROUP I – Network trace analysis

1

1.1 Indicate the complete URL of the original web request that led to the client being compromised

http://10.20.0.111:8080/banking.htm - We can find this in packet number 4647. In response to the GET request to this URL, the user received a *302 Moved* response to URL */banking.htm?UOjiXfyAbAISuH*. This response then triggered the request to the URL containing malicious javascript code.

This initial javascript code in the htm file (*original_banking.htm*) decodes the actual exploit code (*banking_next.js*) and executes it. For readability, we give a slightly modified version of this code (*clean_mod_banking_next.htm*) wrapped in the same hmtl code of the original file - this is the environment where the code executes. The objective of this code is very unclear. It seems to just create and change some variables, do some array and string operations and not execute or change anything relevant. However we found some parts of the code suspicious (event object, *srcElement* attribute, etc.) and that eventually led us to discover the vulnerability being exploited.

1.2 What file type was requested in the final web request to the malicious server?

e. GIF - The final web request to the malicious server (10.20.0.111) was a GET request to page */banking.htmTysdAWdqQEBybyCGKQkGJyVuQsNWvmIFg.gif*. This is a request for a GIF file.

1.3 What is the number of the first frame that indicates that the client has been compromised?

4722 - The user establishes a TCP connection with the malicious server. The user then receives a malicious file through this connection. We define this as the moment when the client was compromised because it means that the exploit worked and the malicious code is indeed executing.

1.4 At one point, the malicious server sends a malicious file to the client. What type of file it is?

a. Windows executable - There is a TCP communication between the malicious server and the user. The communication does not appear to contain anything interesting to the naked eye. Following the TCP stream we saved the content of the communication directed from the server to the user (*malware_file*). This file appeared to be just data, however after running *foremost* on this raw data file, we obtained the Windows executable file **malware.exe**.

```
root@kali:~/lab3/pcap1/artifacts# file malware.exe
malware.exe: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
```

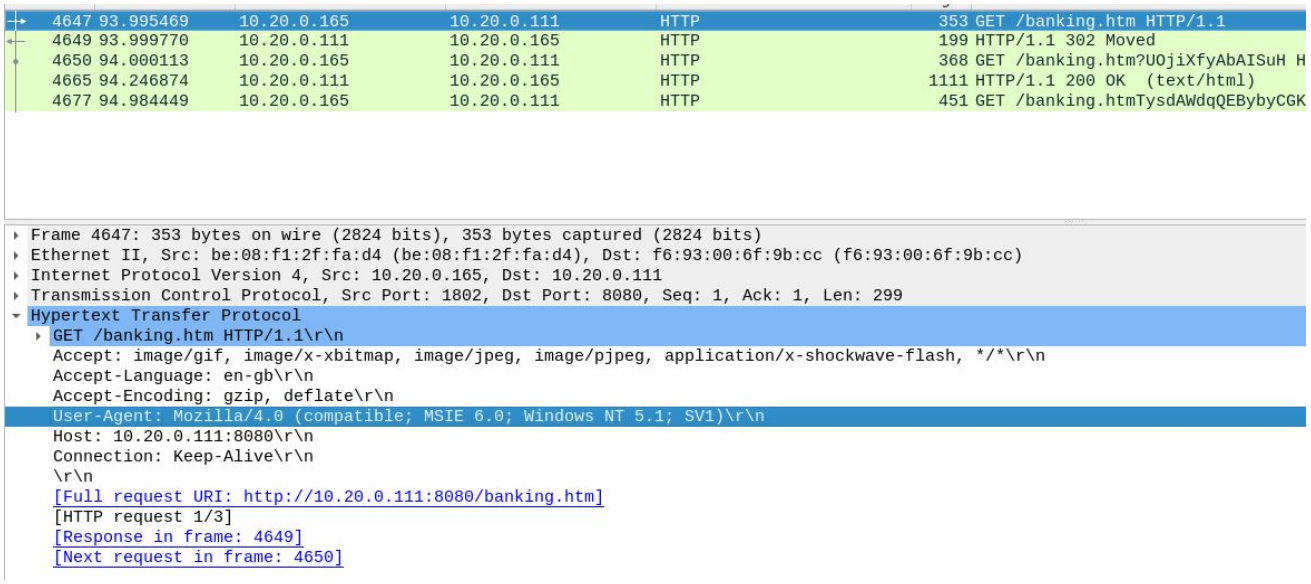
1.5 What is the SHA1 hash of the malicious file?

94adf100411a80076192766a214e0ff92da13ab7 - This is the result of using the command *sha1sum* on the carved file (*malware.exe*) from the previous question.

```
root@kali:~/lab3/pcap1/artifacts# sha1sum malware.exe
94adf100411a80076192766a214e0ff92da13ab7  malware.exe
```

1.6 What vulnerable software has been exploited (in the following format, Firefox 3.5, Firefox 3.6, Word 2010, IE7, Safari2, Chrome2, AdobeReader, IE9)?

IE6 - We concluded it was Microsoft Internet Explorer 6.0. In the HTTP requests we can see that this is the browser the user is using (*MSIE 6.0*) and where the malicious javascript code must have executed.



After some research we also arrived at the vulnerability that was exploited - documented in **CVE-2010-0249**. This vulnerability, which behaves the same way we can observe in the trace, affects the Microsoft Internet Explorer 6.0 browser. This confirmed our hypothesis.

1.7 When the capture ends, is the client still connected to the malicious attacker?

Yes - There seem to be two ways to terminate a TCP connection: doing the FIN handshake, or sending the RST flag which will abort the connection. We could not find any of these packet patterns between the user (port 1804) and the attacker (port 4444) in the trace, so we conclude that the connection was not terminated.

1.8 Can you indicate the corresponding CVE security bulletin that covers the vulnerability that was exploited here (answer in form of CVE-$year-$number)?

CVE-2010-0249 - We were able to understand that this was the vulnerability exploited by the attacker by looking at the attack information we could extract from the trace and comparing it to the description of the CVE security bulletin. Also, the previously mentioned file, *clean_mod_banking_next.htm,* contains the code analogous to the exploit code sent by the attacker to the victim in https://www.exploit-db.com/exploits/11167 (this is an example code of how the vulnerability can be exploited).

1.9 From the capture, it is clear that the attacker gets a certain form of access (i.e. the interface), what (type of) access does the attacker "get" on the client?

Execution of arbitrary code - According to the available documentation on this vulnerability, the attacker gains access to execute arbitrary code with the privileges of the victim on the targeted machine (https://nvd.nist.gov/vuln/detail/CVE-2010-0249). We also found some evidence that the attacker might have access to the Windows command line - using grep with some common windows commands:

```
root@kali:~/lab3/pcap1/artifacts# grep "ping" malware.exe
Binary file malware.exe matches
root@kali:~/lab3/pcap1/artifacts# grep "console" malware.exe
Binary file malware.exe matches
root@kali:~/lab3/pcap1/artifacts# grep "assoc" malware.exe
Binary file malware.exe matches
root@kali:~/lab3/pcap1/artifacts# grep "cipher" malware.exe
Binary file malware.exe matches
root@kali:~/lab3/pcap1/artifacts# grep "fc" malware.exe
Binary file malware.exe matches
root@kali:~/lab3/pcap1/artifacts# grep "shutdown" malware.exe
Binary file malware.exe matches
```

2

2.1 What is the frame that indicates that something strange might be going on?

8 - Frame number 8 is the start of a long list of ping requests to all machines on a network - this is called a ping scan. IP 10.20.110 is trying to determine what machines are discoverable on the network. Even though this might not necessarily imply malicious activity, it is unusual behaviour.

2.2 What tool is generating this traffic?

nmap - After examining the SYN scans made in the captures, and doing some research online, we realised that the window sizes for the SYN packets are small and fixed (1024), which is uncommon for regular SYN packets. Also this is considered a common trait of nmap's scan packets, seen here. Also, as we mention below, we see the User-Agent field in the HTTP requests as Nmap Scripting Engine, which also gives it away.

2.3 What does this frame constitute the beginning of?

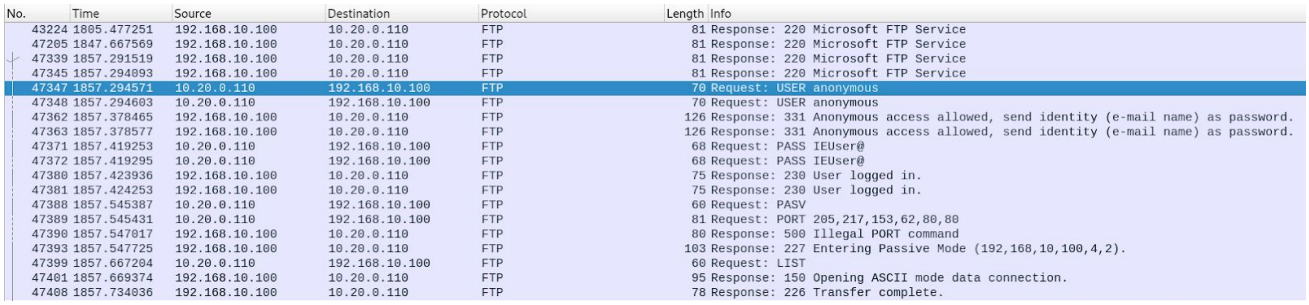
d. Ping Scan - Frame number 8 begins a ping scan, as described in answer 2.1 above.

2.4 A switch was removed from the command to improve the speed, what was this switch (just the letters, case-sensitive)?

-sU - This flag activates a UDP scan. At 11 minutes a new scan begins but, as opposed to the scan made at 4-5 minutes, no UDP scan is performed.

2.5 What switch was added to the final scan (case-sensitive)?

-A - This flag enables OS detection, version detection, script scanning and traceroute. We found evidence of the presence of the four options enabled by this switch. For *OS detection* we identified several of the packet characteristics found in https://nmap.org/book/osdetect-methods.html. For *version detection* we found the UDP probes containing arbitrary strings (e.g. frame 47284) and probes for testing whether SSL is running and to identify RPC-based services. For *script scanning* we analysed the User-Agent in the HTTP requests in the end of the capture which showed "Nmap Scripting Engine" (this is the engine used when running scripts with nmap). Also, we found evidence of execution of at least one of the *default* scripts (which are the ones run in script scanning) - *ftp-anon*. This script tries to check if an FTP server allows authentication by anonymous users and then makes a LIST request.



No.	Time	Source	Destination	Protocol	Length	Info
43224	1805.477251	192.168.10.100	10.20.0.110	FTP	81	Response: 220 Microsoft FTP Service
47205	1847.667569	192.168.10.100	10.20.0.110	FTP	81	Response: 220 Microsoft FTP Service
47339	1857.291519	192.168.10.100	10.20.0.110	FTP	81	Response: 220 Microsoft FTP Service
47345	1857.294093	192.168.10.100	10.20.0.110	FTP	81	Response: 220 Microsoft FTP Service
47347	1857.294571	10.20.0.110	192.168.10.100	FTP	70	Request: USER anonymous
47348	1857.294603	10.20.0.110	192.168.10.100	FTP	70	Request: USER anonymous
47362	1857.378465	192.168.10.100	10.20.0.110	FTP	126	Response: 331 Anonymous access allowed, send identity (e-mail name) as password.
47363	1857.378577	192.168.10.100	10.20.0.110	FTP	126	Response: 331 Anonymous access allowed, send identity (e-mail name) as password.
47371	1857.419253	10.20.0.110	192.168.10.100	FTP	68	Request: PASS IEUser@
47372	1857.419295	10.20.0.110	192.168.10.100	FTP	68	Request: PASS IEUser@
47380	1857.423936	192.168.10.100	10.20.0.110	FTP	75	Response: 230 User logged in.
47381	1857.424253	192.168.10.100	10.20.0.110	FTP	75	Response: 230 User logged in.
47388	1857.545387	10.20.0.110	192.168.10.100	FTP	60	Request: PASV
47389	1857.545431	10.20.0.110	192.168.10.100	FTP	81	Request: PORT 205,217,153,62,80,80
47390	1857.547017	192.168.10.100	10.20.0.110	FTP	80	Response: 500 Illegal PORT command
47393	1857.547725	192.168.10.100	10.20.0.110	FTP	103	Response: 227 Entering Passive Mode (192,168,10,100,4,2).
47399	1857.667204	10.20.0.110	192.168.10.100	FTP	60	Request: LIST
47401	1857.669374	192.168.10.100	10.20.0.110	FTP	95	Response: 150 Opening ASCII mode data connection.
47408	1857.734036	192.168.10.100	10.20.0.110	FTP	78	Response: 226 Transfer complete.

For *traceroute* we identified the ICMP packets.

Artifacts

We present below the list of the most relevant files found and produced during the investigation, a short description of their contents and their respective MD5 fingerprints.

File name	File contents	MD5
malware_file	Raw data file extracted from Wireshark. Sent by the attacker to the victim.	`da1447c27196f9d8bcbfff0dcd14780a`
malware.exe	Windows executable which is the malware that compromises the user. Carved from *malware_file*.	`216c9d064601b4d2066f4573ac20d656`
original_banking.htm	HTML page that contains the first level of the exploit javascript code.	`1bc7a66b75dd15129c61a8992f3ceaf4`
banking_next.js	Decoded javascript exploit code.	`e3e4cff30a9155d8336111f798540b39`

GROUP II – Website fingerprinting over Tor

1 Indicate: (a) the selected feature set, and (b) the accuracy of your classifier using this feature set. Give an account of other features you have tried before converging into this feature set. Explain how you've modified the source code in order to compute your proposed feature set.

(a) The final set of features we decided for was computed by the following feature extractors:

- **sum_sizes:** returns the sum of all the sizes in the sequence;
- **duration:** returns the total duration of the sequence, i.e. the last timestamp;
- **num_switches:** returns the number of times the direction of packets changed, i.e. the number of occurrences of two consecutive cells with different sizes;
- **longest_time_idle:** returns the largest time difference between two consecutive cells;
- **num_inc_out_cells:** returns two values - the number of incoming (-1) and the number of outgoing (1) cells;
- **inc_out_duration:** returns two values - the total duration of incoming and outgoing communications;
- **out_packet_ordering:** returns the indexes of all outgoing packets as features. For this we use a parameter that defines the size of the returned list. In case the sequence has less outgoing cells than the indicated size, the missing values are replaced with the character 'X';
- **out_burst_stats:** returns three features with information about outgoing bursts. A burst is defined as two or more consecutive cells in the same direction. This feature extractor returns the number of outgoing bursts, the size of the largest outgoing burst and the mean size of the outgoing bursts in the sequence.

Our approach to developing this feature set was by thinking about which traffic patterns cannot be obscured by the Tor system. The standard request-response interaction between a client and server is a relevant vector for trying to classify Tor traffic, so the variations in the incoming and outgoing cells will be good indicators for website fingerprinting and any features derived from that.

In contrast, timestamps might not be as relevant, at least on their own - this both as a consequence of how unpredictable traffic routing on a network is, but also about the Tor specificities regarding to the relays.

(b) For the closed world setting, using a parameter of K=5 for the number of neighbours, we obtained an accuracy of approximately 80% using the previously described feature set. We experimented with different feature sets, either adding or removing feature extractors. In the end we decided for the above as it was the one representing the better trade-off between accuracy and dimension of the feature space. We describe below the different feature extractors we implemented but ended up not using. This does not necessarily mean they are worse! With the limited time we had we only managed to try a few configurations, however there might be a different combination of our feature extractors that yields a feature set able to achieve a higher accuracy.

Here we describe the feature extractors we implemented but did not use:

- **num_cells:** returns the total amount of cells in the sequence;
- **num_first_inc_out_cells:** returns two values - the number of incoming (-1) and the number of outgoing (1) cells found in the first *n* cells;
- **out_packet_ordering_and_diffs:** besides returning the same as *out_packet_ordering*, it returns the number of incoming cells between every two outgoing cells. Thus, this returns a list of size double the parameter;

- **out_concentration:** we have two integer parameters for this one, *window* and *length*. So for every *window* packets we return a feature representing the number of outgoing packets in this window. This is done for the first *length* packets.

We must note that we drew inspiration and used solutions found in a number of scientific papers on Website Fingerprinting. We mainly looked through the list of papers found in Free Haven's Selected Papers in Anonymity that focused on this topic. The paper we used the most was "Effective Attacks and Provable Defenses for Website Fingerprinting" by Wang et al., which can be found here.

2

2.1 Can you spot a trend in the accuracy of the classifier as k increases? Explain the variations observed.

k	1	2	5	10	15
ACC	91.5	87.6	81.0	73.6	67.4

Table 1

We observe that the accuracy decreases as the number of neighbours increases. This happens in part due to the implementation of our *k*-NN classifier. In this approach, if any of the *k* neighbours disagree, the testing point is assigned to the non-monitored class, even if all of the *k* neighbours point to monitored pages (which always happens in the closed-world scenario). So, when we increase the number of neighbours, the probability of including more than one class in the neighbour set, also increases, thus leading to a more frequent classification of pages as non-monitored.

We conclude that our feature set still has room for improvement. We struggle to separate pages in the feature space when we consider the 15 neighbours, which is a relatively small number in comparison with the 100 instances of each page. Still, for a smaller number of neighbours, the classifier achieves a very high accuracy, so we are definitely in the right path.

However, our feature set might behave differently in a conventional implementation of *k*-NN, where the actual majority vote is used for classification.

2.2 Can you spot any trend in the TPR/FPR trade-off alongside the variation of k and nm? Justify.

nm%	1	2	5	10	15
0.1	99.26/20.01	78.29/6.77	64.67/0.88	52.25/0.38	45.43/0.22
0.2	99.17/20.23	77.81/7.54	60.81/0.99	50.05/0.33	39.10/0.11
0.4	98.83/18.89	76.39/6.39	60.65/0.72	50.11/0.22	41.16/0.11
0.6	98.61/15.83	81.00/6.73	63.31/0.99	54.51/0.22	45.83/0.11
0.8	98.19/14.24	80.06/4.60	64.97/0.72	54.93/0.28	47.78/0.11

Table 2 (TPR% / FPR%)

With the increase in the number of neighbours the evolution of the TPR is similar to the evolution of the accuracy explained above. Once again, since all neighbours have to agree, increasing the number of neighbours will necessarily increase the probability of disagreement, thus reducing the probability of a monitored page of being classified as such. We can say that the TPR in the open-world scenario is analogous to the accuracy in closed-world. However, the TPR is in general lower here than the accuracy was before because we now introduce the non-monitored pages which will add noise when classifying monitored pages.

When analysing the variation of the FPR in relation to the number of neighbours, the relation is almost the complement of the previous analysis. When more neighbours are used, the probability of classifying a non-monitored page as monitored decreases because we are increasing the probability of the neighbours disagreeing, which leads to classification as non-monitored. So the FPR decreases with the increase of the number of neighbours.

Now we look at the variation of the percentage of non-monitored pages included in the training set.

We see that increasing this percentage does not show a clear variation on the TPR. We can see however that, in general, the values of the TPR along the vertical axis are quite similar and do not vary much. We can conclude that our features are able to separate monitored from non-monitored pages with the same effectiveness almost independently of the non-monitored pages used for training.

The FPR clearly decreases with the increase of the nm%. This is expectable as increasing the number of non-monitored pages will increase the probability of the neighbours disagreeing or agreeing on a non-monitored page. Also, the feature set should also be able to approximate non-monitored pages.

To sum up, we see that in general changing a parameter to improve one of the ratios will negatively affect the other.

2.3 According to your results in Table 2, which configuration leads to a more successful attack? Do you think an attack with such a TPR / FPR trade-off may be effective in practice? Justify.

We conclude that the best configuration overall is the $k = 2$ and $nm\% = 0.8$. This configuration represents the best trade-off amongst our results: it shows a high TPR of 80% and a low FPR of 4.6%. This way we can identify most accesses to monitored pages while keeping away from wrongly accusing most innocent users.

If this attack was to be carried out in practice, its exact context should be taken into strong consideration. It is really important to measure the importance and impact of favoring the TPR in detriment of the FPR, and vice-versa.

As briefly mentioned in the handout, it might be a serious problem to accuse an innocent person of visiting a child pornography website. In this case it might even be more suitable to use $k >= 5$, as for these configurations the FPR is minimal. This also means that we are necessarily giving less importance to finding guilty users.

However, depending on the context of the investigation, it might be reasonable to add a large set of falsely identified individuals to the set of suspects, as it might be possible to do a more conclusive investigation on all of them without causing harm to the innocent. In this last case we would wish to favor TPR over FPR, thus we should probably go for a configuration using a k of 1.

In short, the analysis of the trade-off between TPR and FPR should be directly related to the trade-off between the impacts caused by favouring each of the ratios.