

Sporting Corruption Investigation
CLASSIFIED

Sam Heney 1700469
Year 4 BSc Ethical Hacking
CMP416 Digital Forensics 2

2021/21

Abstract

This report documents the findings of an investigation of several network traffic captures relating to a suspected corruption of an Olympic sport, Chess Boxing.

In Capture 1, the suspect Kim Ill-Song is found to be in possession of several files relating to Chess Boxing, a list of usernames, and a hidden Python-based cipher. There is also reference to the steganographic tool SilentEye.

They are then in Capture 2 found to be offering bribes to several representatives from various countries. Whether they were bribed or not and where they are located was determined in most cases, but in some there was not enough information to draw proper conclusions from.

Capture 3 contained some files downloaded over an FTP session, which when pieced together were found to be an image of a Chess board. Upon closer inspection, this image was found to contain steganographic data that could be extracted using SilentEye, then decoded using the Python cipher also found in capture 1. It was found to contain a password that didn't seem to be used for anything in the captures provided.

Finally in capture 4 a meeting was set up over text messaging, with the month and time provided but not the date, though it was hinted that that information was provided somehow. Further inspection revealed that the person Kim was meeting with crafted some fake location data, that when put onto a map spelled "17", indicating that they would be meeting on the 17th.

It seems that Ill-Song was indeed guilty of bribery and corruption, and this report goes into the full technical detail of how this incriminating information was uncovered.

Contents

1 Methodology and Findings	3
1.1 Capture 1	3
1.1.1 Traffic Analysis	3
1.1.2 Evidence Analysis	3
1.2 Capture 2	6
1.2.1 Traffic Analysis	6
1.2.2 Evidence Analysis	7
1.3 Capture 3	9
1.3.1 Traffic Analysis	9
1.3.2 Evidence Analysis	10
1.4 Capture 4	12
1.4.1 Traffic Analysis	12
1.4.2 Evidence Analysis	12
2 References	14
3 Appendices	15
3.1 Capture 1 Files	15
3.2 Capture 1 Fixed Cipher Code	17
3.3 Capture 2 Python IRC Decoder	18
3.4 Capture 2 IRC Log	19
3.5 Capture 3 Reconstructed Images	21
3.6 Capture 4 Messages Parser	22
3.7 Capture 4 Location Data	23

1. Methodology and Findings

1.1 Capture 1

1.1.1 Traffic Analysis

This incident entirely took place on July 11th, 2014 from 21:11:26 to 21:23:52 according to the timestamps within the captured packets.

The brief provided about this packet capture suggested that some files had been transferred and that these should be recovered. There are several protocols used for file transfer, but one of the most common is SMB. Filtering the capture by the SMB protocol shows there was substantial SMB traffic.

In packet 5857, host 172.29.1.23 announces itself to the network as FOX-WS. Likewise, in packet 5962, host 172.29.1.20 announces itself as DOG-WS. From now on, these hosts will be referred to by their hostname rather than their IP address.

Starting with packet 23838, the FOX-WS host established a connection to the IPC share on DOG-WS, allowing it to read the other shares available on the host. The account used to make the connection is `fox-ws\\test`, as seen in packet 23844. Beginning in packet 23972, the user reads all of the available shares on DOG-WS.

From 21:22:16 at packet 23897 to 21:22:18 at packet 24029, the user makes several requests reading the `\DOG-WS\DOCUMENTS` share. Then from 21:22:23 at packet 24034 to 21:22:28 at packet 24095 they browse to the `\DOG-WS\BLAH` share.

At 21:22:40 at packet 24186, the file `Documents.zip` was uploaded from FOX-WS to the `\DOG-WS\BLAH` share. This file could be a compressed archive containing the files mentioned in the brief.

There was some other activity relating to a file called `DOCUME~1.zip` which appears to be a copy of the `Documents.zip` file based on the limited data captured. FOX-WS also accessed several other files from the `\DOG-WS\DOCUMENTS` share, but they were only default files, and `Documents.zip` was the only file transferred.

1.1.2 Evidence Analysis

The `Documents.zip` file mentioned was recovered using tshark's ability to extract SMB objects using the `--extract-objects` flag. Figure 1.1 shows the full list of files and folders within the extracted folder. All of the files can be seen at appendix 3.1.

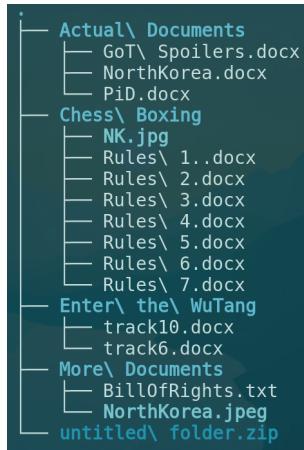


Figure 1.1: List of files within Documents.zip

Actual Documents

These are several files containing text encoded using base64. All three of these files were created by "Eric" and last modified by "Bryan Schmidt", according to the Word metadata.

`GoT Spoilers.docx` contains some spoilers for the TV show Game of Thrones.

`NorthKorea.docx` contains some Russian text referring to some insider information about technology being developed within the North Korean military, being sent to a person referred to as Obi-Wan.

Finally, `PiD.docx` is a letter from a person claiming to be William Campbell stating that they have replaced Paul McCartney.

Chess Boxing

This folder contains seven files that document the rules of Chess Boxing, indicating that this is the sport targeted for corruption. Again, these files were created by "Eric" and last modified by "Bryan Schmidt". `NK.jpg` is the North Korean flag, which potentially indicates the involvement of North Korea.

Enter the WuTang

This folder contains two files that again contain Base64 encoded contents. The files in this case were both created and last modified by "Bryan Schmidt".

`track10.docx` contains lyrics to the song "Protect Ya Neck", which is indeed track 10 on the album "Enter the WuTang" (Wu Tang Clan, 1994), referred to by the folder name.

`track6.docx` however does not contain lyrics to a song, but contains a list titled "The Mystery of Chess Boxing: (usernames)". This could be the list of names/aliases of actors in the case referred to by the brief. The full list can be seen in appendix 3.1.

More Documents

This folder contained a text copy of the American Bill of Rights, as well as another jpeg image of the North Korean Flag.

Running binwalk on the image revealed that it contains a zip file with a python script inside, as seen in figure 1.2.

```
[sam@khaos:void:More Documents]$ binwalk NorthKorea.jpeg
DECIMAL      HEXADECIMAL      DESCRIPTION
-----      -----
0            0x0                JPEG image data, JFIF standard 1.01
3453          0xD7D              Zip archive data, at least v2.0 to extract, name: untitled/
3492          0xDA4              Zip archive data, at least v2.0 to extract, compressed size: 604, uncompressed size: 1397, name: untitled/broken.py
4263          0x10A7             End of Zip archive, footer length: 22
```

Figure 1.2: Binwalk revealing hidden zip file

The python file inside, named `broken.py`, is a Python script that has some functions that could be used as a cipher, making use of a text file to encode and decode information. The variable name used to refer to the text being used is "bill", likely referencing the Bill of Rights txt file.

Making use of the functions in the script, a tool was developed which can be used to encode and decode information using the cipher, shown in figure 1.3.

```
[sam@khaos:void:cipher]$ python2 fixed.py -e "heres some test data to encode"
b2490l2487 2500l2487a2501,2418a2501 2497l2495l2487,2418n2502l2487a2501n2502,2418a24
86 2483n2502 2483,2418n2502 2497,2418l2487d2496h2485 2497a2486l2487
[sam@khaos:void:cipher]$ python2 fixed.py -d "b2490l2487 2500l2487a2501,2418a2501 2
497l2495l2487,2418n2502l2487a2501n2502,2418a2486 2483n2502 2483,2418n2502 2497,2418
l2487d2496h2485 2497a2486l2487"
heres some test data to encode
[sam@khaos:void:cipher]$
```

Figure 1.3: Cipher in use

The full source code of the repaired cipher program can be seen at appendix 3.2.

untitled folder.zip

This zip file contained nested folders named "untitled folder", with the final folder being named "SilentEye". The full structure can be seen in figure 1.4. SilentEye is a steganography tool (Achor-ein, 2010), indicating some data hidden in the images found, but neither image contained SilentEye data.

```
[sam@khaos:void:Documents]$ tree untitled\ folder
untitled\ folder
└── untitled\ folder
    └── untitled\ folder
        └── untitled\ folder
            └── untitled\ folder
                └── SilentEye
```

Figure 1.4: SilentEye folder

1.2 Capture 2

1.2.1 Traffic Analysis

This incident entirely took place on June 17th, 2014 from 21:59:08 to 22:13:49 according to the timestamps within the captured packets.

The brief provided about this packet capture suggested that there was an IRC conversation that should be recovered and decoded. Before searching for IRC traffic, the capture was filtered for SMB traffic which showed a host announcement for FOX-WS, the same host that was involved in Capture 1.

Filtering the traffic by the IRC protocol showed that there was IRC traffic, part of this traffic can be seen in figure 1.5.

No.	Time	Source	Destination	Protocol	Length	Info
16	13.268894	172.29.1.17	185.30.166.35	IRC	96	Request (ISON)
17	13.454498	185.30.166.35	172.29.1.17	IRC	103	Response (303)
22	15.058733	172.29.1.17	185.30.166.35	IRC	276	Request (PRIVMSG)
28	22.548315	172.29.1.17	185.30.166.35	IRC	74	Request (PING)
30	22.733678	185.30.166.35	172.29.1.17	IRC	114	Response (PONG)
35	28.265886	172.29.1.17	185.30.166.35	IRC	96	Request (ISON)
37	28.451405	185.30.166.35	172.29.1.17	IRC	103	Response (303)
43	34.232842	185.30.166.35	172.29.1.17	IRC	220	Response (PRIVMSG)

Figure 1.5: IRC Traffic

The first IRC traffic seen is an ISON command requesting if the names Razor, Genius, Raekwon, Killah and Method are currently taken on the channel. Since this is some of the first traffic in the capture, the user making the requests joined the channel earlier than the beginning of the capture, possibly meaning that some of the discussion was missed.

The response to the ISON request shows that Ill_Song is the user sending the request. Since this host is FOX-WS, Ill_Song is also therefore the suspect involved in the Capture 1 case.

The ISON command is used by older IRC clients to check who is online on a friend list (Oikarinen & Reed, 1993), and given the frequency of the command being sent (around 20 seconds) it's likely that this is what's happening. The analysis of capture 1 showed that Ill_Song kept all of these usernames in a word document.

Looking at the traffic itself shows that the conversations were encoded using base64, as seen in figure 1.6.

No.	Time	Source	Destination	Protocol	Length	Info
342	256.642491	172.29.1.17	185.30.166.35	IRC	160	Request (PRIVMSG)
350	262.923782	172.29.1.17	185.30.166.35	IRC	74	Request (PING)
352	263.100145	185.30.166.35	172.29.1.17	IRC	106	Response (PONG)

Frame 342: 160 bytes on wire (1280 bits), 160 bytes captured (1280 bits)
Ethernet II, Src: Dell_fa:a6:cc (00:08:74:fa:a6:cc), Dst: Cisco_fa:52:2a (54:75:d0:ba:52:2a)
Internet Protocol Version 4, Src: 172.29.1.17, Dst: 185.30.166.35
Transmission Control Protocol, Src Port: 50588, Dst Port: 6667, Seq: 1717, Ack: 2251, Len: 106
Internet Relay Chat
Request: PRIVMSG Razor! :SkVRSE8yTE1OUVFHRVpKQUSGWENBNUR0T1ZSV1FJRFhOrjJHUUEVU5CU1NBWUxFTVJaR0s0M1RGWT09PT09P0==

Figure 1.6: Base64 encoded IRC message

In order to save the IRC log for further processing, tshark was used to follow the stream of all IRC traffic. Grep was then used to filter just the private messages sent to and from the client. Finally the output was written to the file `log.encoded`. The full command used is as follows:

```
tshark -r 'Capture 2.pcap' -q -z follow,tcp,ascii,0 | grep PRIVMSG > log.encoded
```

1.2.2 Evidence Analysis

In this section, the code snippets have been simplified to make them more readable. The full final code can be found at appendix 3.3.

With the chat log fully encoded, it would need to be processed by a script that could go through each message, decode them, and output them. Python was determined to be most suitable for the task. Firstly the file was loaded in and each line was parsed in order to pass just the encoded part of each line to a decoder function:

```
for line in encoded:
    if line.startswith('PRIVMSG'):
        print(decode(line[line.find(':') + 1:]))
    else:
        print(decode(line[line.find('Illi_Song :') + 10:]))

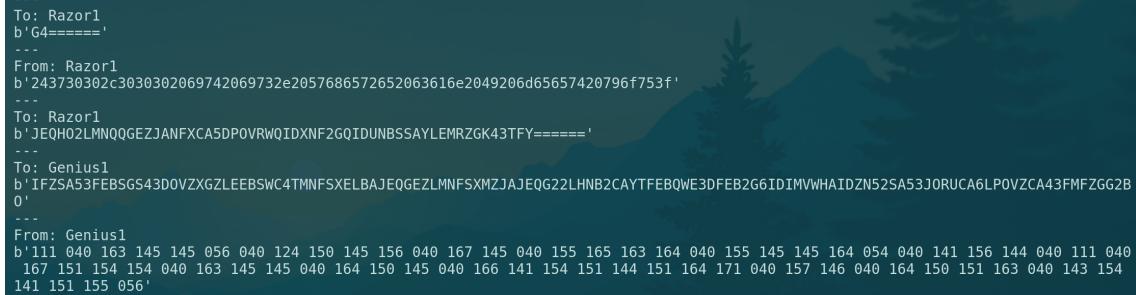

```

The lines had a different format depending on whether they were outgoing or incoming, making the if/else necessary. Since each line seemed to be base64 encoded, the `decode()` function just needed to decode the base64:

```
def decode(line):
    return base64.b64decode(line)


```

Executing this revealed that beyond the base64 encoding, the messages were encoded with several different encodings, seen in figure 1.7



The screenshot shows a terminal window with several lines of encoded text. The first line starts with 'To: Razor1' followed by a base64 encoded string. The second line starts with 'From: Razor1' followed by another base64 encoded string. The third line starts with 'To: Genius1' followed by a base64 encoded string. The fourth line starts with 'From: Genius1' followed by a hex dump of the message content.

```
-->
To: Razor1
b'G4====='
...
From: Razor1
b'243730302c3030302069742069732e2057686572652063616e2049206d65657420796f753f'
...
To: Razor1
b'JEQH02LMNQ0GEZJANFXCA5DP0VRWQIDXNF2GQIDUNBSSAYLEMRZGK43TFY====='
...
To: Genius1
b'IFZSA53FEBSGS43DOVZXGZLEEBSWC4TMNFSXELBAJEQGEZLMNFSXMZJAJEQG22LHNB2CAYTFEBQWE3DFEB2G6IDIMVWHAIDZN52SA53J0RUCA6LPOVZCA43FMFZGG2B
0'
...
From: Genius1
b'111 040 163 145 145 056 040 124 150 145 156 040 167 145 040 155 165 163 164 040 155 145 145 164 054 040 140 141 156 144 040 111 040
167 151 154 154 040 163 145 145 040 164 150 145 040 166 141 154 151 144 151 164 171 040 157 146 040 164 150 151 163 040 143 154
141 151 155 056'
```

Figure 1.7: Various encodings across several messages

Studying the encoded messages revealed that certain users used certain encodings consistently. The `decode()` function was changed to take the username of the sender, and this was used to create custom decoding algorithms per user.

Ill Song's messages were determined to be base32 encoded, so their decoder was implemented as follows:

```
if sender == 'Illi_Song':
    line = base64.b32decode(line).decode()


```

Razor1 and Raekwon's messages are hex encoded and have Windows-1251 ascii encoding, which was a little more complicated:

```

elif sender == 'Razor1' or sender == 'Raekwon' or sender == 'Method':
    line = bytearray.fromhex(line.decode()).decode('cp1251')

```

Killah and Genius1's messages are octal encoded which was significantly more complicated to deal with than the other two encodings:

```

elif sender == 'Killah' or sender == 'Genius1':
    new_line = ''
    line = line.split()
    for i in range(len(line)):
        new_line += chr(int(line[i].decode(), 8))
    line = new_line

```

Once all of the decoders had been implemented, the log could be read. The full log can be found at appendix 3.4.

The first conversation is with Razor, where they accept a bribe of \$700,000. Ill_Song also mentions that Razor is based in "the City of Love", commonly used to refer to Paris.

Next there is a discussion with Genius. Genius refers to a location with an md5 hash, and Ill_Song responds "not here" implying that is their location. Cracking the hash using hashcat reveals it to be Caracas. Genius mentions being able to help with Ill_Song's "search". There is no mention of a bribe.

After that is Method. Ill_Song attempts to initiate a discussion about a bribe with Method, but they turned it down. There was no mention of their location.

Next is Killah. Immediately, Ill_Song reveals that Killah is based in Qatar. When offered the chance to bribe, Killah turned it down across two messages which seem to imply that there are multiple people operating the same account.

Finally, Raekwon. Raekwon is aware of the bribe bought by Razor, and is revealed by Ill_Song to be an official on the executive committee of the ICBA. They end up accepting a bribe of 20 million Rubles.

The following is a table showing the aliases of the officials, their locations if known and whether or not they were bribed:

Alias	Location	Bribed
Razor	Paris, France	Yes
Genius	Caracas, Venezuela	Unknown
Method	Unknown	No
Killah	Qatar	No
Raekwon	Russia	Yes

1.3 Capture 3

1.3.1 Traffic Analysis

This incident entirely took place on July 3rd, 2014 from 21:33:35 to 21:40:03 according to the timestamps within the captured packets.

The brief provided about this packet capture suggested that there was some FTP traffic that might contain some forensically obfuscated files. As in the last capture, an SMB host announcement for FOX-WS was found at packet 6005. This again proves that Ill_Song was the suspect here.

Filtering for FTP traffic shows that there was an FTP exchange. At 21:36:35 in packet 5854, user Ill_Song connects to the server named "Super Secret Server" with the password "Ill_Song". As a sample of the FTP traffic found, this exchange can be seen in figure 1.8.

No.	Time	Source	Destination	Protocol	Length	Info
5852	179.750171	172.29.1.21	172.29.1.23	FTP	79	Response: 220 Super Secret Server
5853	179.750921	172.29.1.21	172.29.1.23	FTP	79	Response: 220 Super Secret Server
5854	179.767158	172.29.1.23	172.29.1.21	FTP	69	Request: USER Ill_Song
5856	179.767169	172.29.1.21	172.29.1.23	FTP	88	Response: 331 Please specify the password.
5857	179.767408	172.29.1.23	172.29.1.21	FTP	69	Request: USER Ill_Song
5859	179.767420	172.29.1.21	172.29.1.23	FTP	88	Response: 331 Please specify the password.
5860	179.767658	172.29.1.23	172.29.1.21	FTP	69	Request: PASS Ill_Song
5861	179.767666	172.29.1.23	172.29.1.21	FTP	69	Request: PASS Ill_Song
5864	179.887813	172.29.1.21	172.29.1.23	FTP	77	Response: 230 Login successful.

Figure 1.8: FTP Traffic

During this FTP session, two files were downloaded: `sandofwhich.zip` and `ojd34.zip`. This can be seen in figure 1.9.

```
104 Response: 227 Entering Passive Mode (172,29,1,21,210,232).
76 Request: RETR sandofwhich.zip
130 Response: 150 Opening BINARY mode data connection for sandofwhich.zip (24792 bytes).
78 Response: 226 Transfer complete.
62 Request: TYPE I
85 Response: 200 Switching to Binary mode.
60 Request: PASV
103 Response: 227 Entering Passive Mode (172,29,1,21,121,89).
70 Request: RETR ojd34.zip
124 Response: 150 Opening BINARY mode data connection for ojd34.zip (24714 bytes).
78 Response: 226 Transfer complete.
```

Figure 1.9: Two files downloaded over FTP

Upon inspection of these files, discussed further in the next section, it was found that they were incomplete. Searching for other files revealed an email exchange between `kim.illsong <kim.illsong@aol.com>` and `The Gza <da.genius36@aol.com>`. Ill_Song's full alias Kim Ill-Song is likely a reference to the founder of North Korea Kim Il-sung.

The Gza opens with "You have made a bold claim but i'd like to see some proof.". Kim Ill Song responds with "Ask and you shall receive. You know where to find it.". After this exchange, Kim Ill-Song sent two zip files by email, which were recovered from the raw tcp stream with file carving.

1.3.2 Evidence Analysis

All of the zip files recovered were found to contain many different files with a .jpg extension. Upon further inspection, they contain image data but only some of them contain the start bytes for a jpg image, indicating that this is fragmented image data.

The brief suggested that an Edward Snowden quote might help. The file names of all the images are individual words that seem like they could make up a sentence, and this quote seemed to contain the words within the filenames:

“I can’t in good conscience allow the U.S. government to destroy privacy, internet freedom and basic liberties for people around the world with this massive surveillance machine they’re secretly building.” - Edward Snowden

The images with the words required to make up the entire quote came from each of the four zip files. Once assembled by sequencing all the files into one image file, the complete image could be recovered and can be seen in figure 1.10.



Figure 1.10: Image assembled from the fragments

The vertical line artefacts across the image suggested some steganography had been applied to the image. Using the tool referred to by a file on Ill-Song’s system in capture one, SilentEye, some data was recovered as seen in figure 1.11.

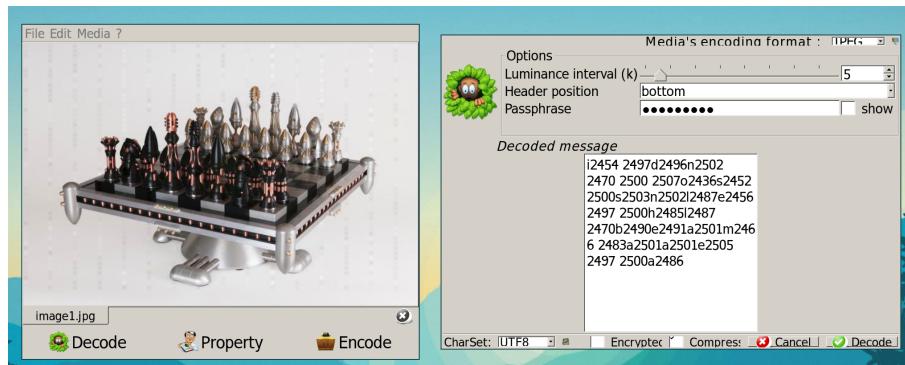


Figure 1.11: Data recovered from the image using SilentEye

This data appeared to be in the format of the python cipher also from capture one, and putting it through the cipher revealed the message to be "Dontry2BruteForceThisPassword", seen in figure 1.12. It's unclear what this text/password might be used for, but this is most likely the encoded communication referred to by the brief.

```
[sam@khaos:void:cipher]$ python fixed.py -d "i2454 2497d2496n2502 2470 2500 2507o2436  
s2452 2500s2503n2502l2487e2456 2497 2500h2485l2487 2470b2490e2491a2501m2466 2483a2501  
a2501e2505 2497 2500a2486"  
DontTry2BruteForceThisPassword
```

Figure 1.12: Encoded text decoded by the cipher

The remaining image fragments still seemed to make up some pictures, but the words weren't aligned with any particular quote. In figure 1.13, the method used to brute force the order of the words can be seen, where the image was refreshed with each new fragment added.

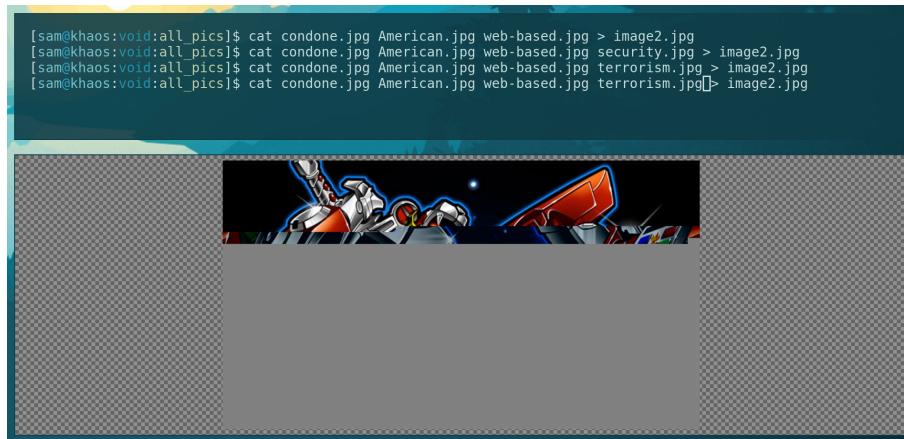


Figure 1.13: Brute force method to find images

Eventually the images could be fully reconstructed, they can be seen at appendix 3.5. They don't seem to be relevant to the case other than the second image being of Kim Jong-un, the Supreme Leader of North Korea.

1.4 Capture 4

1.4.1 Traffic Analysis

This incident entirely took place on July 2nd, 2014 from 17:38:50 to 17:52:32 according to the timestamps within the captured packets.

The brief for this capture mentioned that a conversation had taken place, but didn't mention a protocol. Searching for "Ill-Song" revealed a conversation over HTTP requests, seen in figure 1.14.

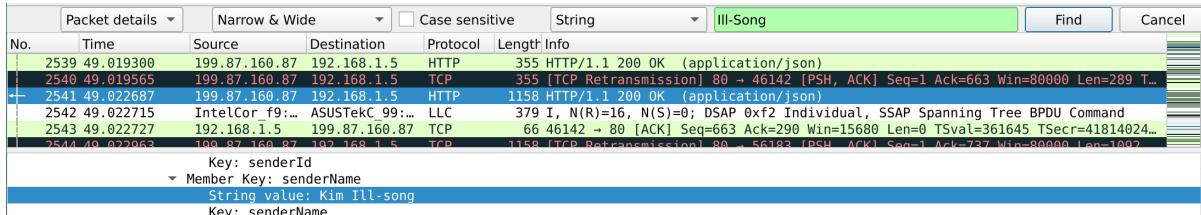


Figure 1.14: Searching for "Ill-Song"

Looking at the X-Requested-With header revealed that it was sent by an app called "TextFree" and the user agent showed that it was likely a Nexus 7, shown in figure 1.15. A cursory look at the text data shows that this device belongs to "Ann Dercover" and she's sending and receiving messages from "Kim Ill-Song", a familiar name.

```
X-Requested-With: com.pinger.textfree\r\nUser-Agent: Mozilla/5.0 (Linux; U; Android 4.2.2; en-us; Nexus 7 Build.
```

Figure 1.15: Some key headers

All of the message data was send to and received from a specific ip address, they were all http requests and they were in json. Putting all of this in a tshark filter allowed just the contents of those requests to be extracted:

```
tshark -r "Capture 4.pcap" \
-Y "http && json && (ip.src == 199.87.160.87 || ip.addr == 199.87.160.87)" \
-T fields -e http.file_data > messages
```

Right after the conversation there was a sequence of location data posted to a map api. Filtering by requests sent to that endpoint with tshark allowed for the data to be extracted:

```
tshark -r Capture\ 4.pcap -Y "http.host == mob.mapquestapi.com" > locations
```

1.4.2 Evidence Analysis

The message data was obscured across many large JSON blobs. A python script was developed to extract the conversation from the blobs, and the full code can be seen at appendix 3.6.

The script outputs the conversation in csv along with the relevant phone numbers. The full clean dialogue can be seen in figure 1.16.

```

Kim Ill-song: Good afternoon, Ann.
Ann: who is this?
Kim Ill-song: Castling.
Ann: where are you?
Kim Ill-song: I know I can't tell you that.
Ann: Do you know that there are people investigating Kim Ill-Song?
Kim Ill-song: Of course. However, they will never know it is me behind the bribes.
Ann: still we should be careful. Pay attention. I want to meet in September at 5PM.
Kim Ill-song: At our old meetup spot?
Ann: yes
Kim Ill-song: What day?
Ann: I told you to pay attention.

```

Figure 1.16: Recovered chat log

The chat log reveals that Kim Ill-Song is actually someone named "Castling", although it's likely that this is another pseudonym given the relevance of that name to chess. Ann Dercover mentions that they should meet in September at 5PM at their "old meetup spot". She does not mention a day in September, and when Ill-Song asks her for that she mentions that he should have paid attention.

The other data present was the location data, which was parsed with some simple vim commands. Once parsed into a CSV format, the data could be imported to Google Earth, the result of which is in figure 1.17. The full location data in CSV format can be seen at appendix 3.7.

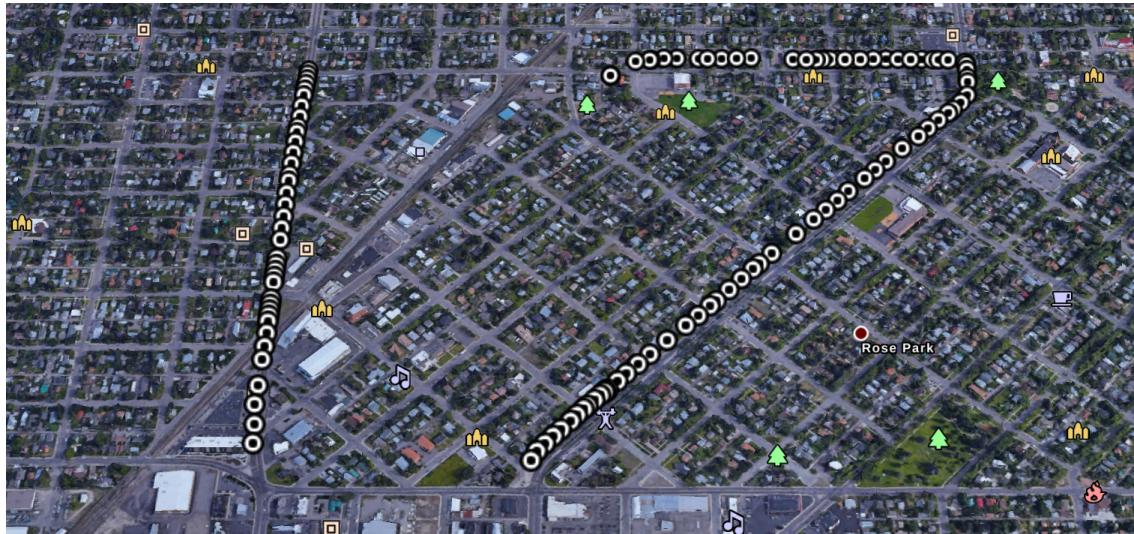


Figure 1.17: Location data mapped out

Since this data spells out the number 17, it can be inferred that Ann Dercover deliberately crafted this location to convey that they should meet on the 17th of September at 5PM.

2. References

- Achorein, (2010), *SilentEye*, Available at: <https://achorein.github.io/silenteye/>
- Oikarinen & Reed, (1993), *Internet Relay Chat Protocol*,
Available at: <https://tools.ietf.org/html/rfc1459#section-5.8>
- Wu Tang Clan, (1994), *Enter The Wu-Tang (36 Chambers)*,
<https://www.discogs.com/Wu-Tang-Clan-Enter-The-Wu-Tang-36-Chambers/release/670735>

3. Appendices

3.1 Capture 1 Files

Jon Snow burns down Winterfell (again) and the Wall.

Hodor kills Theon.

Daenerys gets eaten by a dragon.

Stannis falls in love with Tyrion.

Figure 3.1: "GoT Spoilers.docx" decoded

Для кого это может касаться:

Я был свидетелем, что Ким Чен Ун и правительство Северной Кореи разработали программу, которая позволяет им путешествовать во времени. С использованием этой технологии, я считаю, что они намерены двигаться вперед и изменить результаты войны в Корее.

Пожалуйста, Оби-Ван, ты моя единственная надежда.

Translation:

For whom it may concern:

I have witnessed Kim Jong-un and the North Korean government develop a program that allows them to travel in time. With this technology, I believe they intend to move forward and change the outcome of the Korean War.

Please, Obi-Wan, you are my only hope.

Figure 3.3: "NorthKorea.docx" decoded and translated

Dear Ed,

Yeah I totally took over for Paul after he died in '66. You got me. As you can see, we don't even look that much alike:



We aren't even the same height! What can I say, people are stupid.

Thanks for the inquiry,

William Campbell
(Paul McCartney)

Figure 3.2: "PiD.docx" decoded

1. SUMMARY OF RULES. MAIN POINTS. TOUCH MOVE rule strictly applies.

- If a piece is touched, then it must be moved (if a legal move is available)
- If an opponent's piece is touched, it must be taken (if legal). COUNTDOWN IF STALLING FOR TIME.In general a player manages how much or little time to take for each move, and this is fine! However, if a player clearly plays far too slowly for the specific position, for example when he is facing unavoidable checkmate, the arbiter will do a countdown. He will point at the board, and warn the player by counting to 10 with his hands (just like a boxing referee). If the player has not moved by the count of 10, he loses the game and the match. Note there is no minimum time to make a move! Also, even if there is only 1 legal move, the player should be allowed some time to psychologically compose themselves. It should be considered that

Figure 3.4: Sample of the decoded Chess Boxing rule files



Figure 3.5: "NK.jpg"

"Protect Ya Neck"
 "So what's up man?
 Cooling man"
 "Chilling chilling?"
 "Yo you know I had to call, you know why right?"
 "Why?"
 "Because, yo, I never ever call and ask, you to play something right?"
 "Yeah"
 "You know what I wanna hear right?"
 "What you wanna hear?
 I wanna hear that Wu-Tang joint"
 "Wu-Tang again?"
 "Ah yeah, again and again!"
 [sounds of fighting]

Figure 3.6: Sample of the decoded "track10.docx" file

The Mystery of Chess Boxing: (usernames)

Mr. Method

Kim Ill-Song

Mr. Razor

Mr. Genius

Mr. G. Killah

Matt Cassel

Mr. I. Deck

Mr. M Killa

Mr. O.D.B.

Mr. Raekwon

Mr. U-God

Mr. Cappadonna (possibly)

John Woo?

Mr. Nas

Figure 3.7: Chess Boxing Mystery Aliases



Figure 3.8: "NorthKorea.jpeg"

The Bill of Rights: A Transcription
The Preamble to The Bill of Rights
 Congress of the United States
 begun and held at the City of New-York, on
 Wednesday the fourth of March, one thousand seven hundred and eighty nine.
 THE Conventions of a number of the States, having at the time of their adopting th
 e Constitution, expressed a desire, in order to prevent misconstruction or abuse o
 f its powers, that further declaratory and restrictive clauses should be added: An
 d as extending the ground of public confidence in the Government, will best ensure
 the beneficent ends of its institution.
 RESOLVED by the Senate and House of Representatives of the United States of Americ
 a, in Congress assembled, two thirds of both Houses concurring, that the following
 Articles be proposed to the Legislatures of the several States, as amendments to
 the Constitution of the United States, all, or any of which Articles, when ratifie
 d by three fourths of the said Legislatures, to be valid to all intents and purpos
 es, as part of the said Constitution; viz.

Figure 3.9: "BillOfRights.txt" sample

3.2 Capture 1 Fixed Cipher Code

```
import re
import sys

def fileToString(pathToFile):
    f = open(pathToFile, "r")
    strs = ""
    #adds each line of the file to the strs string
    for line in f.readlines():
        strs+=line
    return strs
def ASCII():
    #number of ASCII characters
    NumOfASCII = 150
    #returns list of all ASCII characters
    return "".join([chr(i) for i in range(NumOfASCII)])
def sumName(name):
    sums=0
    #sums the indices in ASCII of all the characters in name
    for x in name:
        sums+=ord(x)
    return sums
def indexInFile(password,name):
    indices = []
    ASCIIArray = ASCII()
    #populates an array of indices to be used by the encoder
    for chrs in password:
        indices.append(ASCIIArray.index(chrs)+sumName(name)*2)
    return indices
def indexInASCII(name,encoded):
    indices = []
    ASCIIArray = ASCII()
    #split on all non-numeric characters
    #remove first index because it is blank
    indexList = re.split("[^\d]",encoded)[1:]
    #converts encoded characters to ASCII
    for index in indexList:
        indices.append(ASCIIArray[int(index) - (sumName(name)*2)])
    #returns decoded message
    return "".join(indices)
def encode(name,password):
    #returns a list of indices to be used for encoding
    indices = indexInFile(password,name)
    #convert file associated with name to a string
    bill = fileToString("./%s.txt"%name)
    encoded = ""
    #add letter in file plus index of the letter in the file to the encoded string
    for index in indices:
        encoded+=bill[index]+str(index)

    return encoded

if __name__ == "__main__":
    name = "BillOfRights"
```

```

if (len(sys.argv) != 3):
    print("use '-e [message]' to encode a message or '-d [encoded text]' to decode some text.")
elif (sys.argv[1] == "-d"):
    print(indexInASCII(name, sys.argv[2]))
elif (sys.argv[1] == "-e"):
    print(encode(name, sys.argv[2]))
else:
    print("use '-e [message]' to encode a message or '-d [encoded text]' to decode some text.")

```

3.3 Capture 2 Python IRC Decoder

```

import base64

def decode(line, sender):
    if line.startswith(" . . . . ."):
        line = line[17:]

    # all messages are base64 encoded
    line = base64.b64decode(line)

    # Ill_Song's messages are base32 encoded
    if sender == 'Ill_Song':
        try:
            line = base64.b32decode(line).decode()
        except:
            # Strange edge case where a message was partially sent by Ill_Song
            # in base64, then the full message was sent in base32
            line = line.split()
            line = base64.b32decode(line[1]).decode()

    # Razor1 and Raekwon's messages are hex encoded and have weird ascii encoding
    elif sender == 'Razor1' or sender == 'Raekwon' or sender == 'Method':
        line = bytearray.fromhex(line.decode()).decode('cp1251')

    # Killah and Genius1's messages are octal encoded
    elif sender == 'Killah' or sender == 'Genius1':
        new_line = ''
        line = line.split()
        for char in line:
            new_line += chr(int(char.decode(), 8))
        line = new_line

    return line

if __name__ == '__main__':
    encoded = open('log.encoded', 'r')

    print('---')
    for line in encoded:
        if line.startswith('PRIVMSG'):
            print('To:', line[8:line.find(':')])
            print(decode(line[line.find(':') + 1:], 'Ill_Song'))

```

```
    else:
        sender = line[1:line.find("!")]
        print('From:', sender)
        print(decode(line[line.find('I_ll_Song :') + 10:], sender))
    print('---')
```

3.4 Capture 2 IRC Log

—
To: Razor1

Mr. Razor, I am excited about the prospect of the Chess Boxing world title coming to Pyongyang.

—
From: Razor1

Well the decision is not final yet.

—
To: Razor1

Pyongyang is beautiful this time of year. Perhaps you would like to visit and experience what Best Korea has to offer.

—
From: Razor1

I am a very busy man, but perhaps I could be persuaded to visit. See if Pyongyang is the right place for the World Title.

—
To: Razor1

Perhaps not. How about I send you a gift? Something to get you out of the City of Love and take your own vacation somewhere.

—
From: Razor1

Somewhere expensive, I hope.

—
To: Razor1

5

—
From: Razor1

9

—
To: Razor1

7

—
From: Razor1

\$700,000 it is. Where can I meet you?

—
To: Razor1

I will be in touch with the address.

—
To: Genius1

As we discussed earlier, I believe I might be able to help you with your search.

—
From: Genius1

I see. Then we must meet, and I will see the validity of this claim.

—
To: Genius1

I can be in c9fa5b8cb3b197ae5ce4baf8415a375b within the week.

—
From: Genius1
No. Not here. Can I not go to you?

—
To: Genius1
I am afraid that would be unwise. I will send you a message with the date and location through a more secure form of communication.

—
To: Method
Mr. Method, I am excited about the prospect of the Chess Boxing world title coming to Pyongyang.

—
From: Method
I am not sure who you are, but I have an idea. Either way, I am not interested.

—
To: Method
I am just hopeful. It would mean so much to have the Title here. Please consider it.

—
From: Method
Do not speak to me again.

—
To: Killah
How is the weather in Qatar, Mr. Killah?

—
From: Killah
Hot, as always. Who is this?

—
To: Killah
I am a fan of Chess Boxing. I would love to see the Title held in Korea.

—
From: Killah
We will have to see how the bid turns out.

—
To: Killah
Is there anything that I could do to help make your decision easier?

—
From: Killah
No! The great nation of Qatar would never be swayed so easily.

—
From: Killah
Nor would I. We do not take kindly to this pathetic notion of bribery.

—
To: Raekwon
Mr. Raekwon, have you spoken with Mr. Razor?

—
From: Raekwon
I have, but I won't be bought so easily.

—
To: Raekwon
Bought? Of course not. You are an official on the executive committee of the ICBA. I just want you to know that I am here to help make your decision as easy as possible.

—
From: Raekwon
I would need at least 20 million Rubles.

—
To: Raekwon
Consider it done. I will send you the information for the drop-off point soon.

3.5 Capture 3 Reconstructed Images



Figure 3.10: Reconstructed Image 2



Figure 3.11: Reconstructed Image 3

3.6 Capture 4 Messages Parser

```
import csv
import json

class parser:
    def __init__(self):
        self.ids = []
        self.out_file = open('readable.csv', 'w', newline='')
        self.writer = csv.writer(self.out_file)
        #self.writer.writerow(['Sender', 'Sender Number', 'Recipient Number', 'Message'])

    def write(self, message, text):
        #self.writer.writerow([message['senderName'], message['senderId'], message['recipientId'], message['text']])
        self.writer.writerow([message['senderName'], text])

    def parse(self, file):
        for line in file:
            parsed = json.loads(line.replace("\n", "").replace("\r", ""))

            try:
                for message in parsed['result']['recMessages']:
                    if message['messageId'] not in self.ids:
                        self.ids.append(message['messageId'])
                        self.write(message, message['messageText'])
            except:
                pass

            try:
                self.write(parsed, parsed['messageTxt'])
            except:
                pass

if __name__ == '__main__':
    p = parser()

    with open('messages.json', 'r') as file:
        p.parse(file)
```

3.7 Capture 4 Location Data

Latitude,Longitude	
46.85661315917969,-114.01860809326172	46.864051818847656,-114.00627899169922
46.85693359375,-114.01863098144531	46.864051818847656,-114.00605773925781
46.85727310180664,-114.01868438720703	46.864051818847656,-114.00592803955078
46.857601165771484,-114.01866912841797	46.86405944824219,-114.00563049316406
46.858055114746094,-114.01866149902344	46.86405944824219,-114.00534057617188
46.8582878112793,-114.01864624023438	46.86405563354492,-114.00506591796875
46.858524322509766,-114.01863861083984	46.864051818847656,-114.00477600097656
46.858734130859375,-114.01864624023438	46.864051818847656,-114.00452423095703
46.858943939208984,-114.01864624023438	46.864044189453125,-114.0042724609375
46.859046936035156,-114.01864624023438	46.864044189453125,-114.00414276123047
46.85914993286133,-114.01864624023438	46.86404037475586,-114.00392150878906
46.859466552734375,-114.01864624023438	46.86393356323242,-114.0035171508789
46.85957717895508,-114.01864624023438	46.86381912231445,-114.00352478027344
46.85969161987305,-114.01864624023438	46.863643646240234,-114.0035400390625
46.85980987548828,-114.01864624023438	46.86354446411133,-114.00354766845703
46.85993194580078,-114.01864624023438	46.86325454711914,-114.00360107421875
46.86029052734375,-114.01863098144531	46.86309051513672,-114.00376892089844
46.86052322387695,-114.01863861083984	46.86293411254883,-114.00396728515625
46.86098861694336,-114.01863098144531	46.86286163330078,-114.00408172607422
46.861228942871094,-114.01863861083984	46.862701416015625,-114.00432586669922
46.86147689819336,-114.01863098144531	46.86253356933594,-114.00457763671875
46.86159896850586,-114.01863098144531	46.86210632324219,-114.00520324707031
46.86183547973633,-114.01862335205078	46.86148452758789,-114.00609588623047
46.862064361572266,-114.01861572265625	46.86122131347656,-114.00647735595703
46.862281799316406,-114.01860046386719	46.86103057861328,-114.00672912597656
46.86248779296875,-114.01860046386719	46.86065673828125,-114.00727081298828
46.86260223388672,-114.01859283447266	46.86037063598633,-114.0076675415039
46.86282730102539,-114.0185775756836	46.859989166259766,-114.00820922851563
46.86306381225586,-114.0185775756836	46.85979080200195,-114.00848388671875
46.863426208496094,-114.0185546875	46.85969161987305,-114.00862121582031
46.86355209350586,-114.01854705810547	46.859500885009766,-114.00887298583984
46.86367416381836,-114.01853942871094	46.85930252075195,-114.00914001464844
46.8637809753418,-114.01853942871094	46.8590087890625,-114.0095443725586
46.86387252807617,-114.0185317993164	46.858829498291016,-114.00979614257813
46.86370849609375,-114.01163482666016	46.858646392822266,-114.01005554199219
46.864017486572266,-114.01107025146484	46.858375549316406,-114.01044464111328
46.864044189453125,-114.01074981689453	46.858123779296875,-114.01079559326172
46.86404800415039,-114.01071166992188	46.85795211791992,-114.01103973388672
46.86408996582031,-114.01042175292969	46.85765838623047,-114.0114517211914
46.86408996582031,-114.01012420654297	46.857513427734375,-114.01164245605469
46.864078521728516,-114.00962829589844	46.85749053955078,-114.01168823242188
46.86406707763672,-114.00910186767578	46.85747146606445,-114.01171112060547
46.86407470703125,-114.00875854492188	46.857418060302734,-114.01179504394531
46.86408233642578,-114.0084228515625	46.857181549072266,-114.01212310791016
46.864044189453125,-114.00716400146484	46.85708236694336,-114.01225280761719
46.864044189453125,-114.00694274902344	46.85697937011719,-114.01237487792969
46.86404800415039,-114.00680541992188	46.856834411621094,-114.01256561279297
46.86405563354492,-114.00670623779297	46.85672378540039,-114.01271057128906
46.864051818847656,-114.00662231445313	46.856597900390625,-114.01287078857422
46.864051818847656,-114.00646209716797	46.85647201538086,-114.01302337646484