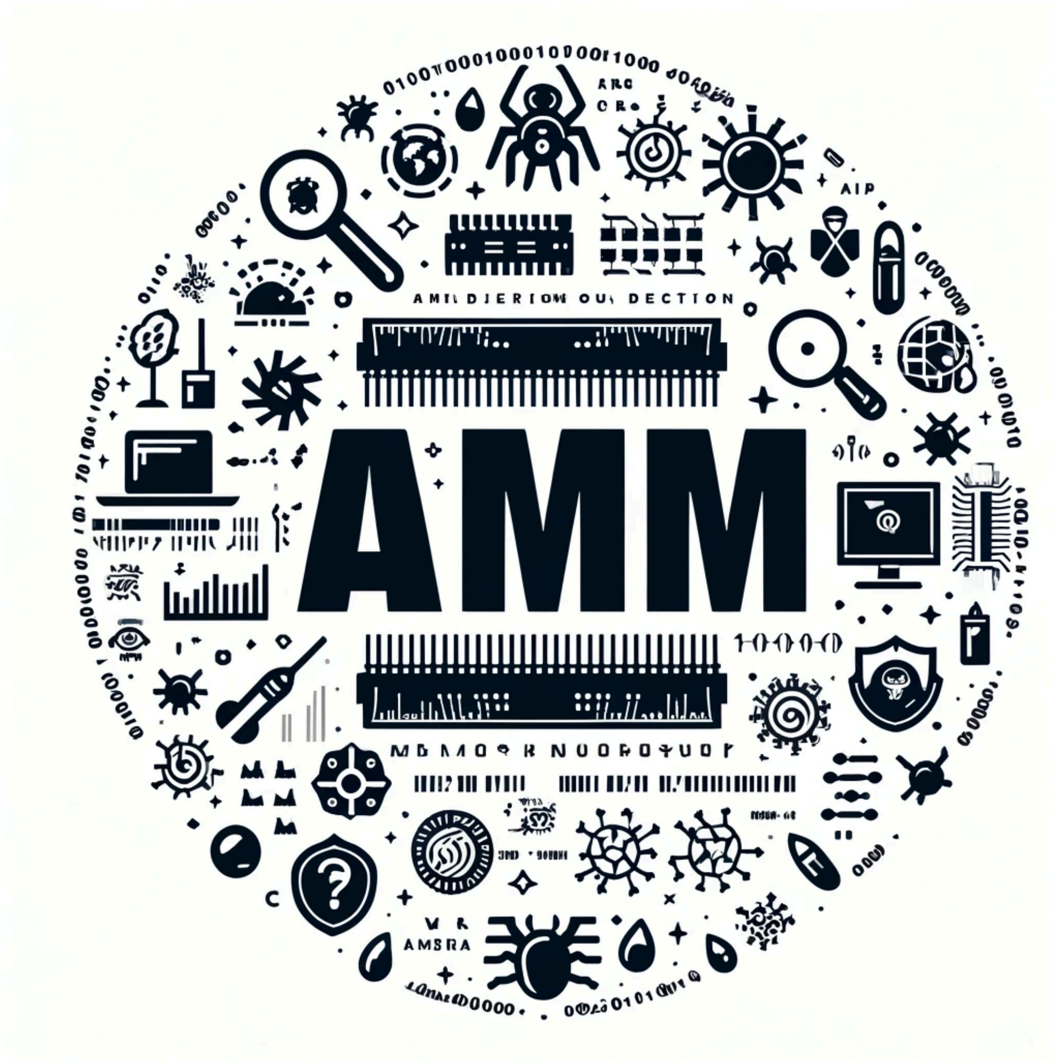
AMM - Memory and malware analysis
Prof. Juan Ramos
BA6 - 2023-2024



Complimenting Network Forensics with Memory Analysis

Memory and malware analysis

Annen Rayane **Ducommun Hugo** **Martins Alexis**

05.05.2024

Table des matières

1 - Context

We have received a memory dump of an infected machine, a packet capture of the traffic after infection, and a malware sample named "jackal.exe." This malware appears to be part of a large-scale APT attack. While it has already been analyzed online, our objective is to conduct our own analysis. The only information we possess is a tweet from someone stating, "jackal's c2 list is just base64 and xor," without any accompanying proof of concept or confirmation if they used the same executable.

The goal of this document is to provide an explanation of the malware's behavior, extract some artifacts and provide some remediations to avoid any further infection.

2 - Summary

The malware identified is called `jackal.exe`. The victim, "Michael Hale", has unfortunately clicked on a malicious link sent to him via the application Slack by a user named "Mike".

The malware exhibits behaviors typical of advanced persistent threats, including network communications via HTTP, downloading additional payloads, modifying system settings for persistence.

Once you download and execute 'jackal.exe', the malware immediately saves several encoded C2 servers in the Windows Registry. Next, it reaches out to one of these servers with a GET request. The server then replies with a port number to establish a connection, likely for deploying a reverse shell to compromise the system and extract data.

This analysis not only details the malware's operations and persistence but also traces the initial infection back to social engineering via Slack, pointing out the importance of comprehensive security training alongside technical defenses.

In the following figure a timeline summarizing the events:

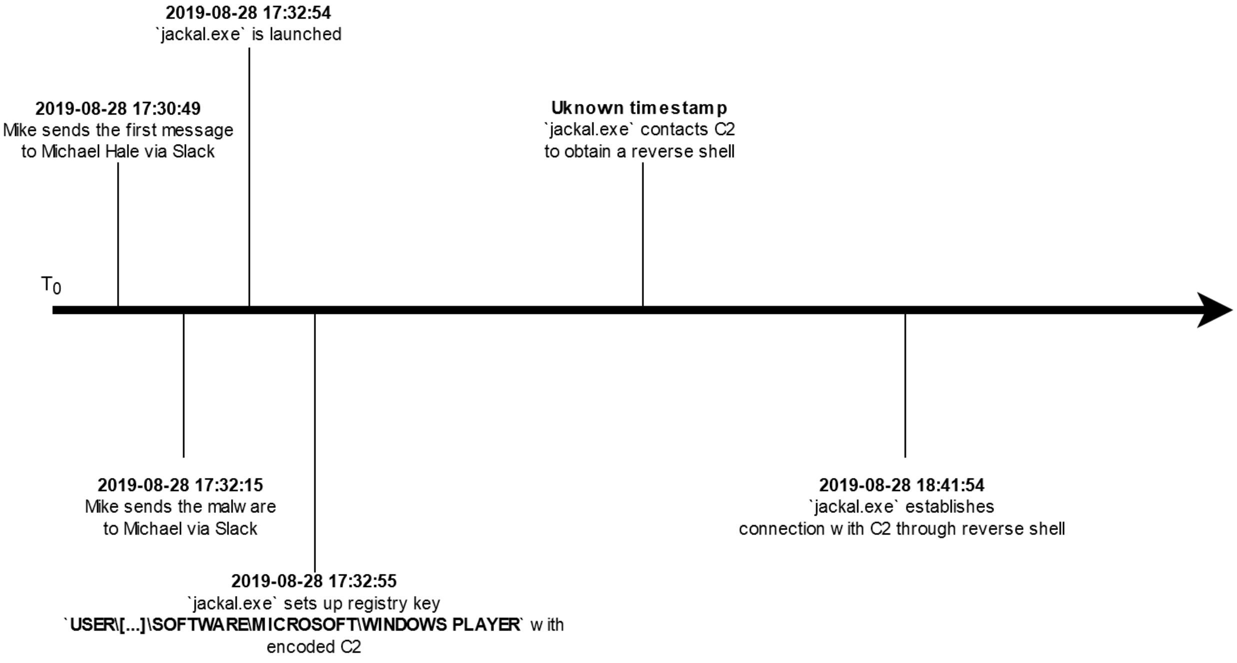

2019-08-28 17:32:54
`jackal.exe` is launched
2019-08-28 17:30:49
Mike sends the first message to Michael Hale via Slack
Uknown timestamp
`jackal.exe` contacts C2 to obtain a reverse shell
T_0
2019-08-28 17:32:15
Mike sends the malware to Michael via Slack
2019-08-28 18:41:54
`jackal.exe` establishes connection with C2 through reverse shell
2019-08-28 17:32:55
`jackal.exe` sets up registry key `USER\[...]\SOFTWARE\MICROSOFT\WINDOWS PLAYER` with encoded C2

Figure 1 - Timeline of events

3 - Analysis

3.1 - Pre-analysis

Before the analysis, we had to setup our Remnux VM and Volatility. We ran the `kdbgscan` and `psscan` tool, to create our `volatilityrc` file.

```
[DEFAULT]
PROFILE=Win10x64_17134
LOCATION=file:///home/remnux/Desktop/L7/Jackal/Jackal/memory/data.lime
KDBG=0xf8036e849d9c
DTB=0x00000000001ad002
```

3.2 - Process list

First we just simply list processes of the memory dump with `psxview` :

```
remnux@remnux$ vol.py psxview --apply-rules
Volatility Foundation Volatility Framework 2.6.1

Offset(P)          Name                    PID pslist psscan thrdproc pspcid
------------------ -------------------- ------ ------ ------ -------- ------
[ ... ]
0x0000000177e57680 jackal.exe             8628 True   True   True     True
[ ... ]
```

Besides having lots of process running or exited, we found the interesting process name `jackal.exe` with a PID of **8628**. After searching on the web, it looks clearly suspicious.

It hasn't been hidden at all by the attackers (because each process scanning techniques detect it).

3.3 - Dump process

To analyze the process, we simply dump it to a real executable with slack spaces using `procdump --memory` plugin from volatility:

```
remnux@remnux$ vol.py procdump --memory --pid=8628 -D dll
Volatility Foundation Volatility Framework 2.6.1

Process(V)         ImageBase          Name                 Result
------------------ ------------------ -------------------- ------
0xffffe50cc9644580 0x0000000000400000 jackal.exe           OK: executable.8628.exe
```

3.4 - PEframe

```
remnux@remnux$ file dll/executable.8628.exe
dll/executable.8628.exe: PE32 executable (console) Intel 80386, for MS Windows, UPX compressed
```

It is clearly a PE32 Windows executable that we can analyze with `peframe` . The most important and interesting part of the peframe is the `behavior` .

```
────────────────────────────────────────────────────────────────────────────────
Behavior
────────────────────────────────────────────────────────────────────────────────
anti dbg
Xor
network http
network dropper
escalate priv
win mutex
win registry
win token
win files operation
```

- **Anti dbg:** The malware tries to avoid being analyzed by detecting if it is being debugged.
- **Xor:** It uses a basic encryption method called XOR to hide its data.
- **Network HTTP:** The malware communicates over the internet using HTTP, a common web protocol.
- **Network dropper:** It can download and install other harmful programs from the internet.
- **Escalate priv:** The malware attempts to gain higher access rights on the computer to take full control.
- **Win mutex:** It uses a specific mechanism to prevent multiple instances of itself from running at the same time, which helps it avoid detection.
- **Win registry:** The malware makes changes to the Windows Registry, which can help it start automatically or hide from security software.
- **Win token:** It manipulates identifiers that control access rights, potentially allowing it to access restricted areas of the system.
- **Win files operation:** The malware can create, delete, or change files on the computer, which could harm the system or hide its activities.

3.5 - Strings

```
remnux@remnux$ strings -a -el executable.8628.exe
[ ... ]
FRRM
A3D7
ZXU6
[ ... ]
ISEqPSY9ISAgPSEjJDxndnpqajxgY39meD1yYGM=
IionPSEjKj0rKj0nIjxgdmF6fGA8e3J/cXZhPXtnfn8=
application/octet-stream
image/gif
text/*
__Dassara__
Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; The Jackal v4.2001
Software\Microsoft\Windows Player
>GET
C:\Users\Analyst\Downloads\jackal.exe
```

We can extract from the strings the path of the executable which is `C:\Users\Analyst\Downloads\jackal.exe`. It probably indicates that the user has downloaded it though a web navigator probably by clicking a malicious link (phishing …).

An additional information which is interesting is this registry path : `Software\Microsoft\Windows Player`. Because a legitimate application should probably not modify the registry of the Windows Player.

An HTTP request is done, we can see the `User-Agent` banner from Mozilla can also be a good artifact to detect the malware.

We also identify some base64 strings, which gives us some garbage by decoding it. However, it's important to keep these in mind because these will appear later in the registry and they are going to unlock some steps in the analysis.

The string `__Dassara__` is also weird. Maybe we will have more information about it later.

3.6 - Handles

```
remnux@remnux:~/Desktop/lab7$ vol.py handles --pid=8628 --object-type=Key
Volatility Foundation Volatility Framework 2.6.1

Offset(V)                Pid              Handle              Access Type              Details
-------------------     ------     -------------------     -------------------     -----------------

-------
[ … ]
0×ffff8801967b97d0     8628                  0×20c                  0×f003f  Key
USER\S-1-5-21-1485840275-1415126033-3747533036-1000\SOFTWARE\MICROSOFT\WINDOWS
PLAYER
[ … ]
```

By analyzing the handles on the malicious process, we can confirm that it opens this registry path.

3.7 - Registry

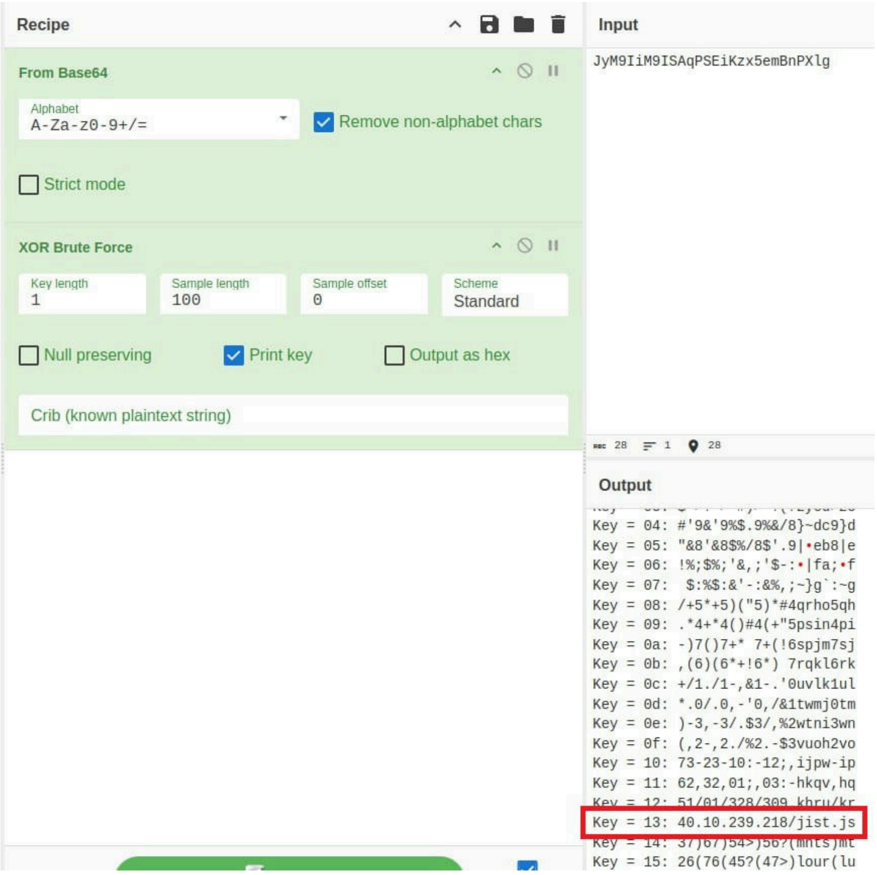
```
remnux@remnux:~/Desktop/lab7$ vol.py printkey -K 'Software\Microsoft\Windows Player'
Volatility Foundation Volatility Framework 2.6.

Legend: (S) = Stable   (V) = Volatile

----------------------------
Registry: \??\C:\Users\Analyst\ntuser.dat
Key name: Windows Player (S)
Last updated: 2019-08-28 17:32:55 UTC+0000

Subkeys:

Values:
REG_SZ        DB1L            : (S) IionPSEjKj0rKj0nIjxgdmF6fGA8e3J/cXZhPXtnfn8=
REG_SZ        WN33            : (S) ISEqPSY9ISAgPSEjJDxndnpqajxgY39meD1yYGM=
[ … ]
REG_SZ                          A3D7                                   :   (S)
ISElPSEjIj0iJCA9JCE8e3×8d2R6fXg8RXJ9cHxmZXZhPGV6YHpnPGBmfn52YT17Z35/
REG_SZ        FRRM            : (S) JyM9IiM9ISAqPSEiKzx5emBnPXlg
```

We recognize the weird 30 base64 strings that we found before. They are actually stored in this registry. The registry key names are those 4-letter words (DB1L, WN33 …).

As we recall the initial tweet, this list is just the C2s XORed and encoded with base64.

The registry key could be a persistence mechanism for the C2 list. We should try to decode and decrypt it to see if we can get some IP addresses.

First, we tried to XOR the base64 decoded string with their registry key name, but it didn't work. We decided to bruteforce the key of the XOR, and we found out that the key for each text is `0×13`. Below is an example for a chosen registry key and its output.



Recipe

From Base64

Alphabet: A-Za-z0-9+/=

☑ Remove non-alphabet chars

□ Strict mode

XOR Brute Force

Key length: 1 | Sample length: 100 | Sample offset: 0 | Scheme: Standard

□ Null preserving ☑ Print key □ Output as hex

Crib (known plaintext string)

Input

```
JyM9IiM9ISAqPSEiKzx5emBnPXlg
```

28 1 28

Output

```
Key = 04: #'9&'9%$.9%&/8}~dc9}d
Key = 05: "&8'&8$%/8$'.9|•eb8|e
Key = 06: !%;$%;'&,;'$-:•|fa;•f
Key = 07:  $:%$:&'-:&%,;~}g`:~g
Key = 08: /+5*+5)("5)*#4qrho5qh
Key = 09: .*4+*4()#4(+"5psin4pi
Key = 0a: -)7()7+* 7+(!6spjm7sj
Key = 0b: ,(6)(6*+!6*) 7rqkl6rk
Key = 0c: +/1./1-,&1-.'0uvlk1ul
Key = 0d: *.0/.0,-'0,/&1twmj0tm
Key = 0e: )-3,-3/.$3/,%2wtni3wn
Key = 0f: (,2-,2./%2.-$3vuoh2vo
Key = 10: 73-23-10:-12;,ijpw-ip
Key = 11: 62,32,01;,03:-hkqv,hq
Key = 12: 51/01/328/309 khru/kr
Key = 13: 40.10.239.218/jist.js
Key = 14: 37)67)54>)56?(mnts)mt
Key = 15: 26(76(45?(47>)lour(lu
```

Figure 2 - CyberChef recipe and its output

Upon decrypting the entire list of C2s found in the registry key, we obtained the following output:

```
194.209.89.41/serios/halber.html
229.5.233.207/teiyy/spluk.asp
185.229.157.168/onrecyeho/verbal.txt
168.84.198.248/seen/yelp.html
54.139.180.138/roars/varioud/alternation.html
253.224.171.39/news/comm.php
184.48.143.117/waiting.asp
212.43.140.152/sport/haphazard.js
252.229.193.227/stock/trading.html
233.172.180.163/rancher/windows.php
146.82.77.59/s91911/klsja11/filter.txt
166.20.53.219/Burbank/Lucid/jacks.html
159.232.102.158/passing/source/home.asp
202.19.197.131/october/saturn.js
67.85.248.25/waifs/juno.html
128.134.176.126/exists/Pasadena.doc
108.115.99.86/yellowhammer/reports
32.24.248.178/goodish/fellow.html
105.71.237.16/super/clabbers.xml
90.35.234.248/orient/bakers.html
13.82.151.215/lopper/dumbbell.txt
101.93.92.167/ostensory/wicked/all
178.85.191.52/eeprom/severe.html
218.113.178.71/weather.html
227.22.157.5/usa/soccer.html
45.151.183.149/kasher/Xeroxing.js
221.165.164.56/pullback/yeshivah.txt
200.144.50.212/resampled/before.html
226.201.173.72/hoodwink/Vancouver/visit/summer.html
40.10.239.218/jist.js
```

3.8 - Mutex

First we decided to use the plugin `mutantscan`, but the output was terrible and there was way too much information for us to understand it correctly. We decided to go for a different strategy and use the `handles` plugin again and the output was more readable.

```
remnux@remnux:~/Desktop/L7/Jackal/Jackal$  vol.py  handles  -p  8628  --object-type=Mutant
Offset(V)              Pid            Handle             Access Type             Details
___________________  ______  ___________________  ___________________  _________________  _______
0xffffe50ccad25550    8628                    0x11c               0x1f0001  Mutant  SM0:8628:168:WilStaging_02
0xffffe50cc83d7ec0   8628            0x200            0x1f0001 Mutant            __Dassara__
0xffffe50cc7f64080    8628                    0x3b0               0x1f0001  Mutant  ZonesLockedCacheCounterMutex
0xffffe50cc7ecfb10    8628                    0x3b4               0x1f0001  Mutant  ZonesCacheCounterMutex
0xffffe50ccaa013f0    8628                    0x3c0               0x1f0001  Mutant  SM0:8628:64:WilError_01
```

The mutexes starting with `SM0 ...` seem legitimate, but the remaining three are more suspicious. Our guess for the three suspicious mutexes are :

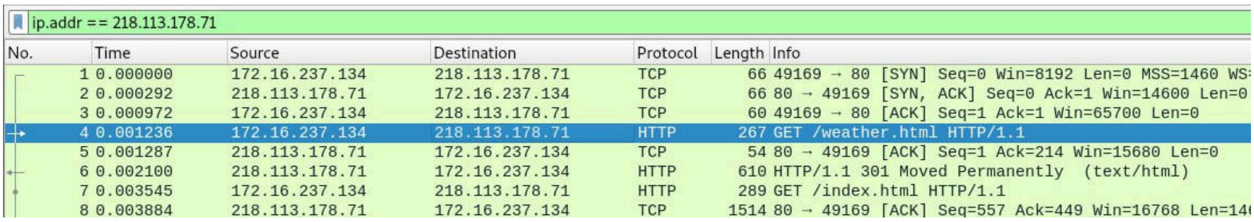
- **ZonesCacheCounterMutex :** Created to make sure only one instance of the malware is running.
- **ZonesLockedCacheCounterMutex :** Created to make sure only one instance of the malware is running.
- **Dassara :** We didn't found any information on the internet/VirusTotal about this one, but it probably has the same utility or similar as the two previous mutexes.

3.9 - Network

We've been given a network traffic file of the infected machine called `lab.pcap`.

We know that the malware is communicating through an HTTP GET request to one of the IP addresses of the C2 decrypted list.

By testing all the IP one by one we get a hit on a certain packet:



ip.addr == 218.113.178.71

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.16.237.134	218.113.178.71	TCP	66	49169 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=
2	0.000292	218.113.178.71	172.16.237.134	TCP	66	80 → 49169 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0
3	0.000972	172.16.237.134	218.113.178.71	TCP	60	49169 → 80 [ACK] Seq=1 Ack=1 Win=65700 Len=0
4	0.001236	172.16.237.134	218.113.178.71	HTTP	267	GET /weather.html HTTP/1.1
5	0.001287	218.113.178.71	172.16.237.134	TCP	54	80 → 49169 [ACK] Seq=1 Ack=214 Win=15680 Len=0
6	0.002100	218.113.178.71	172.16.237.134	HTTP	610	HTTP/1.1 301 Moved Permanently (text/html)
7	0.003545	172.16.237.134	218.113.178.71	HTTP	289	GET /index.html HTTP/1.1
8	0.003884	218.113.178.71	172.16.237.134	TCP	1514	80 → 49169 [ACK] Seq=557 Ack=449 Win=16768 Len=14

Figure 3 - HTTP request to a C2 server

C2 server :

- IP : `218.113.178.71`
- Endpoint : `weather.html`

To have more information, we decided to right click on the packet and do `Follow ⇒ HTTP Stream` to see the complete HTTP stream exchange.

First request just redirects the client to `index.html`, the `weather.html` is probably a decoy:

```
GET /weather.html HTTP/1.1
Accept: text/*, image/gif, application/octet-stream
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; The Jackal v4.2001
Host: 218.113.178.71
Cache-Control: no-cache

HTTP/1.1 301 Moved Permanently
Date: Mon, 11 Mar 2013 18:56:28 GMT
Server: Apache/2.2.17 (Ubuntu)
Location: http://218.113.178.71/index.html
Vary: Accept-Encoding
Content-Length: 320
Content-Type: text/html; charset=iso-8859-1
```

The client then send another request to get an HTML page :

```
GET /index.html HTTP/1.1
Accept: text/*, image/gif, application/octet-stream
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; The Jackal v4.2001
Host: 218.113.178.71
Cache-Control: no-cache
Connection: Keep-Alive

HTTP/1.1 200 OK
Date: Mon, 11 Mar 2013 18:56:28 GMT
Server: Apache/2.2.17 (Ubuntu)
Last-Modified: Mon, 11 Mar 2013 16:29:01 GMT
ETag: "43348-29df-4d7a8abacef5c"
Accept-Ranges: bytes
Content-Length: 10719
Vary: Accept-Encoding
Keep-Alive: timeout=15, max=99
Connection: Keep-Alive
Content-Type: text/html

<!doctype html><html itemscope="itemscope" itemtype="http://schema.org/WebPage"><head><meta content="Search the world's information, including webpages, images, videos and more. Google has many special features to help you find exactly what you're looking for." name="description"><meta content="noodp" name="robots"><meta itemprop="image" content="/images/google_favicon_128.png">
<title>Google</title>
<!--j4ckal:YHt2f38zKiMqIw==-->
<script>(function(){
[ ... ]
```

The HTML page seems to be a Google home page copy but we can identify a suspicious HTML comment `<!--j4ckal:YHt2f38zKiMqIw==-->` which seems to be a hidden message for the client encoded in base64.

Decode that with base64 gives us garbage, we just simply use the same mechanism (XOR with key 0x13) to recover the following message :

```
Key = 13: shell 9090
```

The message is probably a reverse shell and the associated port for the C2 to connect to. Of course, we verified that with `netscan`. The infected machine indeed listen on port 9090 :

```
0xe50cc8a79180     TCPv4    0.0.0.0:9090                 0.0.0.0:0           LISTENING
8628      jackal.exe     2019-08-28 18:41:54 UTC+0000
```

This is done by using one of the C2 available, but we can imagine that maybe other C2 could have other commands to execute or maybe, it could be used as a backup of the main download point.

3.10 - Source of infection

Now that we know a bit what the malware is doing, the important part is to determine how he came here. at the beginning of our analysis, we ran few process checkers plugins like `pslist` where we noticed that `jackal.exe` was the child of a `firefox.exe` process. This was very suspicious at that time, but it's even worse now that we know what Jackal does. We also noticed that the `Jackal.exe` path was in the `Downloads` folder which indicates that is potentially comes from a web browser.

```
remnux@remnux:~/Desktop/L7/Jackal/Jackal$ vol.py pslist | grep 2004
0xffffe50ccb09e580 firefox.exe            2004   7540     62        0      1      0 2019-08-28 15:50:41 UTC+0000
...
0xffffe50cc9644580 jackal.exe             8628   2004      2        0      1      1 2019-08-28 17:32:54 UTC+0000
```

Now we'll try to find a relevant information directly in the strings of the memory dump by searching for `jackal.exe`. The output is not so big we can parse it by hand to find something interesting :

```
remnus@remnus:$ strings -a -el data.lime > strings_data_lime.out
remnux@remnux:$ cat strings_data_lime.out | grep jackal.exe
[...]
https://slack-redir.net/link?url=http://67.205.163.62/jackal.exe&v=3
[...]
```

Actually we get a `Slack` redirection URL to a weird IP address `67.205.163.62`. So the user has probably received a message from a malicious user in Slack to download this malware. To be sure about the guess, we decided to go further and check the Firefox history hoping to have a better comprehension of the situation.

To do so, we tried the various plugins included in Volatility to check web browsers history, but none of them worked. So we had to be more creative and to go dump the file where the history is stored.

Doing a quick search on the internet and we found that this file is `places.sqlite`. To dump this file, we have to use the plugin `dumpfiles`. This will dump every file with the word `places.sqlite` in it.

```
vol.py dumpfiles -p 2004 -r "places.sqlite" -D output
DataSectionObject           0xffffe50cc530e7d0                                     2004 \Device\HarddiskVolume3\Users\Analyst\...\places.sqlite-shm
DataSectionObject           0xffffe50cc8791aa0                                     2004 \Device\HarddiskVolume3\Users\Analyst\...\places.sqlite-wal
SharedCacheMap 0xffffe50cc8791aa0   2004   \Device\HarddiskVolume3\Users\Analyst\...\places.sqlite-wal
DataSectionObject           0xffffe50ccae00ef0                                     2004 \Device\HarddiskVolume3\Users\Analyst\...\places.sqlite
```

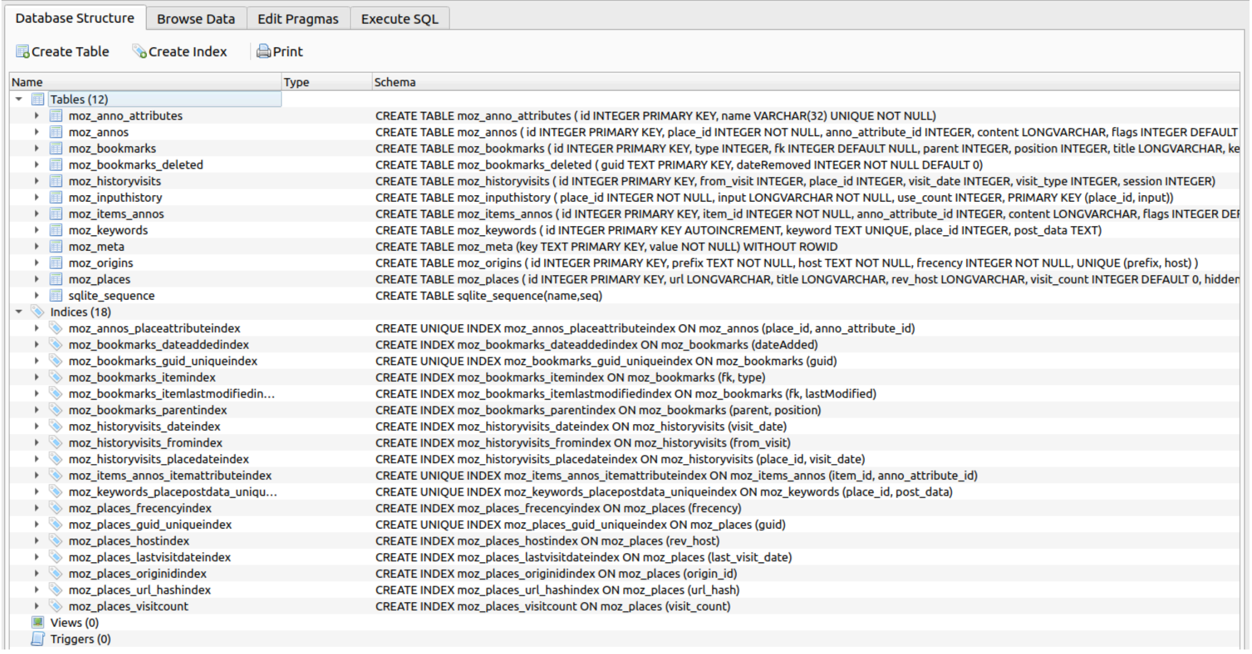
We get 4 files, but they are all unreadable and we didn't know what to do at first. But using the command `file` on Linux allowed us to have more information about each of these files.

```
remnux@remnux:~/Desktop/L7/Jackal/Jackal/output$ file *
file.2004.0xffffe50cc556ea20.dat:  data
file.2004.0xffffe50cc561ced0.dat:  SQLite 3.x database, user version 52, last written using SQLite version 3028000
file.2004.0xffffe50cc7e8a500.dat:  SQLite Write-Ahead Log, version 3007000
file.2004.0xffffe50ccaddf660.vacb: SQLite Write-Ahead Log, version 3007000
```

The second file is the most interesting for us, because it's a database file and that's clearly what we are looking for. We only have one problem with this file at this state, it's not readable by any sqlite viewer. There is one last step, to convert.

```
sqlite3 file.2004.0×ffffe50cc561ced0.dat ".recover" > places.sqlite | sqlite3
places.sql < places.sqlite
```

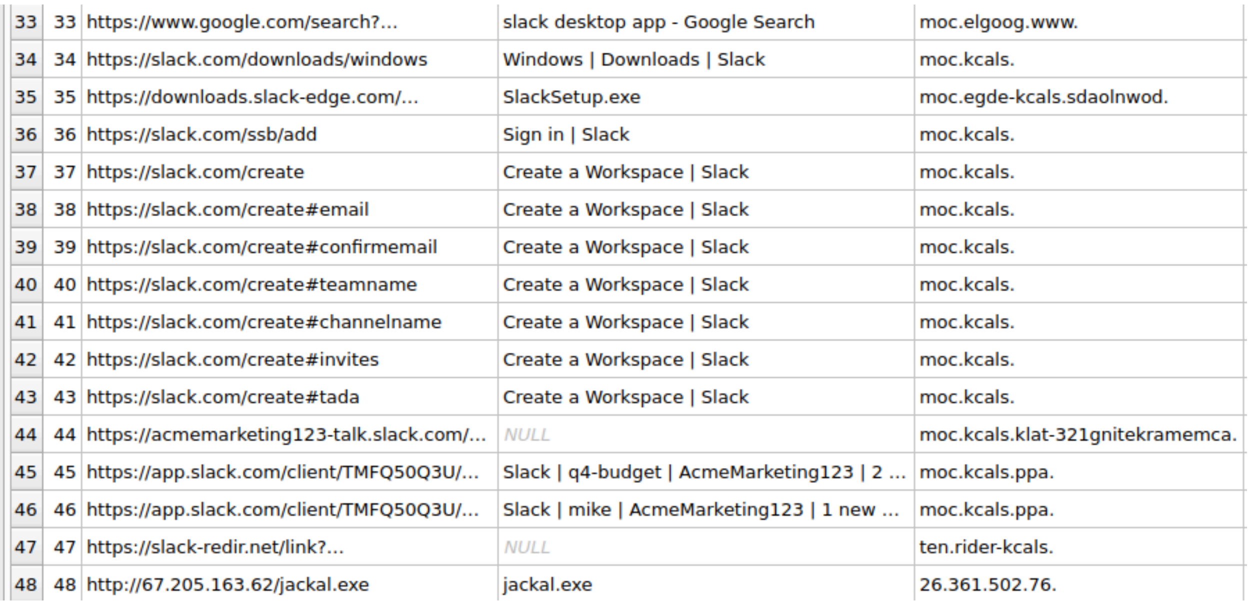
The output file is totally readable by a sqlite broswer like `sqlitebrowser` tool.



Name	Type	Schema
Tables (12)		
moz_anno_attributes		CREATE TABLE moz_anno_attributes (id INTEGER PRIMARY KEY, name VARCHAR(32) UNIQUE NOT NULL)
moz_annos		CREATE TABLE moz_annos (id INTEGER PRIMARY KEY, place_id INTEGER NOT NULL, anno_attribute_id INTEGER, content LONGVARCHAR, flags INTEGER DEFAULT
moz_bookmarks		CREATE TABLE moz_bookmarks (id INTEGER PRIMARY KEY, type INTEGER, fk INTEGER DEFAULT NULL, parent INTEGER, position INTEGER, title LONGVARCHAR, ke
moz_bookmarks_deleted		CREATE TABLE moz_bookmarks_deleted (guid TEXT PRIMARY KEY, dateRemoved INTEGER NOT NULL DEFAULT 0)
moz_historyvisits		CREATE TABLE moz_historyvisits (id INTEGER PRIMARY KEY, from_visit INTEGER, place_id INTEGER, visit_date INTEGER, visit_type INTEGER, session INTEGER)
moz_inputhistory		CREATE TABLE moz_inputhistory (place_id INTEGER NOT NULL, input LONGVARCHAR NOT NULL, use_count INTEGER, PRIMARY KEY (place_id, input))
moz_items_annos		CREATE TABLE moz_items_annos (id INTEGER PRIMARY KEY, item_id INTEGER NOT NULL, anno_attribute_id INTEGER, content LONGVARCHAR, flags INTEGER DEF
moz_keywords		CREATE TABLE moz_keywords (id INTEGER PRIMARY KEY AUTOINCREMENT, keyword TEXT UNIQUE, place_id INTEGER, post_data TEXT)
moz_meta		CREATE TABLE moz_meta (key TEXT PRIMARY KEY, value NOT NULL) WITHOUT ROWID
moz_origins		CREATE TABLE moz_origins (id INTEGER PRIMARY KEY, prefix TEXT NOT NULL, host TEXT NOT NULL, frecency INTEGER NOT NULL, UNIQUE (prefix, host))
moz_places		CREATE TABLE moz_places (id INTEGER PRIMARY KEY, url LONGVARCHAR, title LONGVARCHAR, rev_host LONGVARCHAR, visit_count INTEGER DEFAULT 0, hidden
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq)
Indices (18)		
moz_annos_placeattributeindex		CREATE UNIQUE INDEX moz_annos_placeattributeindex ON moz_annos (place_id, anno_attribute_id)
moz_bookmarks_dateaddedindex		CREATE INDEX moz_bookmarks_dateaddedindex ON moz_bookmarks (dateAdded)
moz_bookmarks_guid_uniqueindex		CREATE UNIQUE INDEX moz_bookmarks_guid_uniqueindex ON moz_bookmarks (guid)
moz_bookmarks_itemindex		CREATE INDEX moz_bookmarks_itemindex ON moz_bookmarks (fk, type)
moz_bookmarks_itemlastmodifiedin...		CREATE INDEX moz_bookmarks_itemlastmodifiedindex ON moz_bookmarks (fk, lastModified)
moz_bookmarks_parentindex		CREATE INDEX moz_bookmarks_parentindex ON moz_bookmarks (parent, position)
moz_historyvisits_dateindex		CREATE INDEX moz_historyvisits_dateindex ON moz_historyvisits (visit_date)
moz_historyvisits_fromindex		CREATE INDEX moz_historyvisits_fromindex ON moz_historyvisits (from_visit)
moz_historyvisits_placedateindex		CREATE INDEX moz_historyvisits_placedateindex ON moz_historyvisits (place_id, visit_date)
moz_items_annos_itemattributeindex		CREATE UNIQUE INDEX moz_items_annos_itemattributeindex ON moz_items_annos (item_id, anno_attribute_id)
moz_keywords_placepostdata_uniqu...		CREATE UNIQUE INDEX moz_keywords_placepostdata_uniqueindex ON moz_keywords (place_id, post_data)
moz_places_frecencyindex		CREATE INDEX moz_places_frecencyindex ON moz_places (frecency)
moz_places_guid_uniqueindex		CREATE UNIQUE INDEX moz_places_guid_uniqueindex ON moz_places (guid)
moz_places_hostindex		CREATE INDEX moz_places_hostindex ON moz_places (rev_host)
moz_places_lastvisitdateindex		CREATE INDEX moz_places_lastvisitdateindex ON moz_places (last_visit_date)
moz_places_originidindex		CREATE INDEX moz_places_originidindex ON moz_places (origin_id)
moz_places_url_hashindex		CREATE INDEX moz_places_url_hashindex ON moz_places (url_hash)
moz_places_visitcount		CREATE INDEX moz_places_visitcount ON moz_places (visit_count)
Views (0)		
Triggers (0)		

Figure 4 - Sqlitebrowser overview

If we search through the tables, one of them has the history we were looking for, it's `moz_places` and this is its content.



33	33	https://www.google.com/search?...	slack desktop app - Google Search	moc.elgoog.www.
34	34	https://slack.com/downloads/windows	Windows | Downloads | Slack	moc.kcals.
35	35	https://downloads.slack-edge.com/...	SlackSetup.exe	moc.egde-kcals.sdaolnwod.
36	36	https://slack.com/ssb/add	Sign in | Slack	moc.kcals.
37	37	https://slack.com/create	Create a Workspace | Slack	moc.kcals.
38	38	https://slack.com/create#email	Create a Workspace | Slack	moc.kcals.
39	39	https://slack.com/create#confirmemail	Create a Workspace | Slack	moc.kcals.
40	40	https://slack.com/create#teamname	Create a Workspace | Slack	moc.kcals.
41	41	https://slack.com/create#channelname	Create a Workspace | Slack	moc.kcals.
42	42	https://slack.com/create#invites	Create a Workspace | Slack	moc.kcals.
43	43	https://slack.com/create#tada	Create a Workspace | Slack	moc.kcals.
44	44	https://acmemarketing123-talk.slack.com/...	NULL	moc.kcals.klat-321gnitekramemca.
45	45	https://app.slack.com/client/TMFQ50Q3U/...	Slack | q4-budget | AcmeMarketing123 | 2 ...	moc.kcals.ppa.
46	46	https://app.slack.com/client/TMFQ50Q3U/...	Slack | mike | AcmeMarketing123 | 1 new ...	moc.kcals.ppa.
47	47	https://slack-redir.net/link?...	NULL	ten.rider-kcals.
48	48	http://67.205.163.62/jackal.exe	jackal.exe	26.361.502.76.

Figure 5 - Content of moz_places tables

We can read the history from the top to the bottom to understand what happen on the victim's computer. He downloaded Slack and created his account, then he got invited to a Slack canal called `AcmeMarketing123`. He checked the `q4-budget` and then, he received a message from a `mike`. After this message, he opened a link that we already found earlier (https://slack-redir.net/link?url=http://67.205.163.62/jackal.exe). We can deduce that `mike` probably sent him this link and that `mike` is the attacker, we should check further to validate our guess.

Now that we clearly know that Slack has been the entry point, we should be able to recover the original message and/or the sender.

We first simply search in the strings of the memory dump itself the download URL :

```
remnux@remnux:$ cat strings/strings_data_lime.out | grep "http://67.205.163.62/jackal.exe" -n
164142:https://slack-redir.net/link?url=http://67.205.163.62/jackal.exe&v=3
[...]
3083609:check out this tool: <LINK:START http://67.205.163.62/jackal.exe>http://67.205.163.62/jackal.exe<LINK:END>
```

The line 3083609 probably corresponds to the message the malicious user sent to his target.

We should try to dig in this way :

```
remnux@remnux:$ cat strings/strings_data_lime.out | grep "check out this tool:" -n
960616:{"ok":true,"latest":"1567013535.001000","oldest":"1567013449.000200","messages":[{"client_msg_id":"e3fe0f43-6e90-4328-be3c-1734acef0513","type":"message","text":"check out this tool: <http:\/\/67.205.163.62\/jackal.exe>","user":"UMT81Q5LG","ts":"1567013535.001000","team":"TMFQ50Q3U"},{"client_msg_id":"a4b2569a-b48e-48fb-a91f-4519381aa58c","type":"message","text":"hello, what's up?","user":"UMVDCGU6A","ts":"1567013478.000400","team":"TMFQ50Q3U"},{"client_msg_id":"94314470-24b0-475c-a963-098903fd79a3","type":"message","text":"hey there!","user":"UMT81Q5LG",
"ts":"1567013449.000200","team":"TMFQ50Q3U"}],"has_more":false,"pin_count":0}
2055082:{"ok":true,"latest":"1567013535.001000","oldest":"1567013449.000200","messages":[{"client_msg_id":"e3fe0f43-6e90-4328-be3c-1734acef0513","type":"message","text":"check out this tool: <http:\/\/67.205.163.62\/jackal.exe>","user":"UMT81Q5LG","ts":"1567013535.001000","team":"TMFQ50Q3U"},{"client_msg_id":"a4b2569a-b48e-48fb-a91f-4519381aa58c","type":"message","text":"hello, what's up?","user":"UMVDCGU6A","ts":"1567013478.000400","team":"TMFQ50Q3U"},{"client_msg_id":"94314470-24b0-475c-a963-098903fd79a3","type":"message","text":"hey there!","user":"UMT81Q5LG",
"ts":"1567013449.000200","team":"TMFQ50Q3U"}],"has_more":false,"pin_count":0}
3032238:{"ok":true,"oldest":"1567013535.001000","messages":[{"client_msg_id":"e3fe0f43-6e90-4328-be3c-1734acef0513","type":"message","text":"check out this tool: <http:\/\/67.205.163.62\/jackal.exe>","user":"UMT81Q5LG",
"ts":"1567013535.001000","team":"TMFQ50Q3U"}],"has_more":false,"pin_count":0}
3083609:check out this tool: <LINK:START http://67.205.163.62/jackal.exe>http://67.205.163.62/jackal.exe<LINK:END>
3240032:{"ok":true,"oldest":"1567013535.001000","messages":[{"client_msg_id":"e3fe0f43-6e90-4328-be3c-1734acef0513","type":"message","text":"check out this tool: <http:\/\/67.205.163.62\/jackal.exe>","user":"UMT81Q5LG",
"ts":"1567013535.001000","team":"TMFQ50Q3U"}],"has_more":false,"pin_count":0}
[...]
```

If we format the JSON at one of the first two lines, we get a simple conversation with some user IDs :

```
{
   "ok":true,
   "latest":"1567013535.001000",
   "oldest":"1567013449.000200",
   "messages":[
      {
         "client_msg_id":"e3fe0f43-6e90-4328-be3c-1734acef0513",
         "type":"message",
         "text":"check out this tool: <http:\/\/67.205.163.62\/jackal.exe>",
         "user":"UMT81Q5LG",
         "ts":"1567013535.001000",
         "team":"TMFQ50Q3U"
      },
      {
         "client_msg_id":"a4b2569a-b48e-48fb-a91f-4519381aa58c",
         "type":"message",
         "text":"hello, what's up?",
         "user":"UMVDCGU6A",
         "ts":"1567013478.000400",
         "team":"TMFQ50Q3U"
      },
      {
         "client_msg_id":"94314470-24b0-475c-a963-098903fd79a3",
         "type":"message",
         "text":"hey there!",
         "user":"UMT81Q5LG",
         "ts":"1567013449.000200",
         "team":"TMFQ50Q3U"
      }
   ],
   "has_more":false,
   "pin_count":0
}
```

To go a bit further we will try to recover the victim's identity by searching in the memory dump by his user ID which is `UMVDCGU6A` :

```
remnux@remnux:$ cat strings/strings_data_lime.out | grep "UMVDCGU6A" -n
[ ... ]
49193540:{{"type":"hello","flannel":true,"server_version":"flannel: build-id:2971-ceda03c0cadada51ad049f8003c93b0f5b6b30db-build-worker-dev-ops-vpc-vlp8-2019-07-25T13:05:42-07:00 golang:go1.12.7","host_id":"flannel-iad-k3bu-8784","region":"us-east-1","self":{"id":"UMVDCGU6A","team_id":"TMFQ50Q3U",
"name":"michael.hale","deleted":false,"color":"9f69e7","real_name":"michael.hale",
"tz":"America\/Chicago","tz_label":"Central Daylight Time","tz_offset":-18000,"profile":{"title":"","phone":"","skype":"",
"real_name":"michael.hale","real_name_normalized":"michael.hale","display_name":"",
"display_name_normalized":"","fields":null,"status_text":"","status_emoji":"",
"status_expiration":0,"avatar_hash":"g518edf6d36b","status_text_canonical":"",
"team":"TMFQ50Q3U"},"is_admin":true,"is_owner":true,"is_primary_owner":true,
"is_restricted":false,"is_ultra_restricted":false,"is_bot":false,
"is_app_user":false,"updated":1567008570},"start":{"rtm_start":{"ok":true,"url":"wss:\/\/cerberus-xxxx.lb.slack-msgs.com\/websocket\/VUSp4QuB08wlSM4JGoUvjJTpaP5H5I0M5INoXE3YMfnv_Nx26jsKmiZnQH0iQiUVRSUbNgYA-r8WfCEVYexz5TpP2-D688Jxtp3-5prY1qs="}}}
[ ... ]
```

By formatting it, we can recover the victim's name which is : **Michael Hale**

```
{
  {
    [...]
    "self":{
       "id":"UMVDCGU6A",
       "team_id":"TMFQ50Q3U",
       "name":"michael.hale",
       "deleted":false,
       "color":"9f69e7",
       "real_name":"michael.hale",
       "tz":"America\/Chicago",
       "tz_label":"Central Daylight Time",
       "tz_offset":-18000,
       [...]
    },
    [...]
  }
}
```

4 - Remediation

We will now enumerate all the IOCs we found to prevent any further infection.

4.1 - IP address

IP address of the server where he has downloaded Jackal :

```
67.205.163.62
```

C2 IP list to blacklist on the Firewall :

```
194.209.89.41/serios/halber.html
229.5.233.207/teiyy/spluk.asp
185.229.157.168/onrecyeho/verbal.txt
168.84.198.248/seen/yelp.html
54.139.180.138/roars/varioud/alternation.html
253.224.171.39/news/comm.php
184.48.143.117/waiting.asp
212.43.140.152/sport/haphazard.js
252.229.193.227/stock/trading.html
233.172.180.163/rancher/windows.php
146.82.77.59/s91911/klsja11/filter.txt
166.20.53.219/Burbank/Lucid/jacks.html
159.232.102.158/passing/source/home.asp
202.19.197.131/october/saturn.js
67.85.248.25/waifs/juno.html
128.134.176.126/exists/Pasadena.doc
108.115.99.86/yellowhammer/reports
32.24.248.178/goodish/fellow.html
105.71.237.16/super/clabbers.xml
90.35.234.248/orient/bakers.html
13.82.151.215/lopper/dumbbell.txt
101.93.92.167/ostensory/wicked/all
178.85.191.52/eeprom/severe.html
218.113.178.71/weather.html
227.22.157.5/usa/soccer.html
45.151.183.149/kasher/Xeroxing.js
221.165.164.56/pullback/yeshivah.txt
200.144.50.212/resampled/before.html
226.201.173.72/hoodwink/Vancouver/visit/summer.html
40.10.239.218/jist.js
```

4.2 - Registry check

The following registry path must be monitor and checked to verify that there is no weird base64 strings store in multiple keys :

- `Software\Microsoft\Windows Player`

Here are the exhaustive list of the 30 keypaires containing the encoded C2 list added by `jackal` malware :

```
DB1L       : IionPSEjKj0rKj0nIjxgdmF6fGA8e3J/cXZhPXtnfn8=
WN33       : ISEqPSY9ISAgPSEjJDxndnpqajxgY39meD1yYGM=
4H2N       : IismPSEhKj0iJiQ9IiUrPHx9YXZwanZ7fDxldmFxcn89Z2tn
MRRU       : IiUrPSsnPSIqKz0hJys8YHZ2fTxqdn9jPXtnfn8=
HNFY       : Jic9IiAqPSIrIz0iICs8YXxyYWA8ZXJhenxmdzxyf2d2YX1yZ3p8fT17Z35/
IEUH       : ISYgPSEhJz0iJCI9ICo8fXZkYDxwfH5+PWN7Yw==
1AUR       : IisnPScrPSInID0iIiQ8ZHJ6Z3p9dD1yYGM=
47SG       : ISIhPScgPSInIz0iJiE8YGN8YWc8e3Jje3JpcmF3PXlg
FAU1       : ISYhPSEhKj0iKiA9ISEkPGBnfHB4PGdhcnd6fXQ9e2d+fw==
2LHL       : ISAgPSIkIT0iKyM9IiUgPGFyfXB7dmE8ZHp9d3xkYD1je2M=
5WYY       : IiclPSshPSQkPSYqPGAqIioiIjx4f2B5ciIiPHV6f2d2YT1na2c=
KYKG       : IiUlPSEjPSYgPSEiKjxRZmFxcn14PF9mcHp3PHlycHhgPXtnfn8=
Q810       : IiYqPSEgIT0iIyE9IiYrPGNyYGB6fXQ8YHxmYXB2PHt8fnY9cmBj
M65P       : ISMhPSIqPSIqJD0iICI8fHBnfHF2YTxgcmdmYX09eWA=
0GF9       : JSQ9KyY9IScrPSEmPGRyenVgPHlmfXw9e2d+fw==
IYCD       : IiErPSIgJz0iJCU9IiElPHZremBnYDxDcmByd3Z9cj13fHA=
78OI       : IiMrPSIiJj0qKj0rJTxqdn9/fGR7cn5+dmE8YXZjfGFnYA==
YBTI       : ICE9ISc9IScrPSIkKzx0fHx3emB7PHV2f398ZD17Z35/
THRG       : IiMmPSQiPSEgJD0iJTxgZmN2YTxwf3JxcXZhYD1rfn8=
PXDT       : KiM9ICY9ISAnPSEnKzx8YXp2fWc8cXJ4dmFgPXtnfn8=
FYNO       : IiA9KyE9IiYiPSEiJjx/fGNjdmE8d2Z+cXF2f389Z2tn
IWT5       : IiMiPSogPSohPSIlJDx8YGd2fWB8YWo8ZHpweHZ3PHJ/fw==
6NLE       : IiQrPSsmPSIqIj0mITx2dmNhfH48YHZldmF2PXtnfn8=
XOJV       : ISIrPSIiID0iJCs9JCI8ZHZyZ3t2YT17Z35/
XP8X       : ISEkPSEhPSImJD0mPGZgcjxgfHBwdmE9e2d+fw==
3EDQ       : JyY9IiYiPSIrID0iJyo8eHJge3ZhPEt2YXxren10PXlg
ONON       : ISEiPSIlJj0iJSc9JiU8Y2Z/f3FycHg8anZge3plcns9Z2tn
ZXU6       : ISMjPSInJz0mIz0hIiE8YXZgcn5jf3Z3PHF2dXxhdj17Z35/
A3D7       : ISElPSEjIj0iJCA9JCE8e3x8d2R6fXg8RXJ9cHxmZXZhPGV6YHpnPGBmfn52YT17Z35/
FRRM       : JyM9IiM9ISAqPSEiKzx5emBnPXlg
```

4.3 - Mutex

The malware uses a mutex called `__Dassara__` which is in clear text in the malicious executable downloaded. It can be blocked on IDS.

4.4 - Malware banner

This `User-Agent` banner must be analyzed and blocked in Firewalls or IDS :

`Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; The Jackal v4.2001`

4.5 - User training

This malware has been downloaded by an employee through a Slack message from someone sending a malicious link. He probably didn't verify the correctness of the link, clicked and downloaded the malware.

A training could be a good control to update collaborator's sensitivity about phishing, malicious messages and so on.

5 - Conclusion

This investigation into the `jackal.exe` malware has provided profound insights into the sophisticated nature of Advanced Persistent Threats (APTs) and their mechanisms of attack. Our analysis revealed how the malware leveraged social engineering, persistence, and encrypted communications to control and manipulate the infected system discreetly.

The findings underscore the necessity for organizations to enhance their cybersecurity strategies. This includes the implementation of advanced endpoint detection, regular employee training, and the promotion of secure browsing practices.

By detailing the operations of `jackal.exe`, this report contributes to a deeper understanding of APTs and aids in improving defensive measures against them. We advocate for continuous improvement in security practices and international cooperation to effectively combat and mitigate the impact of such sophisticated cyber threats.