Demystifying targeted malware used against Polish banks

welivesecurity.com/2017/02/16/demystifying-targeted-malware-used-polish-banks/

February 16, 2017

The purpose of this blog is to deliver technical details of an as-yet minimally documented malware that has made headlines in Poland.

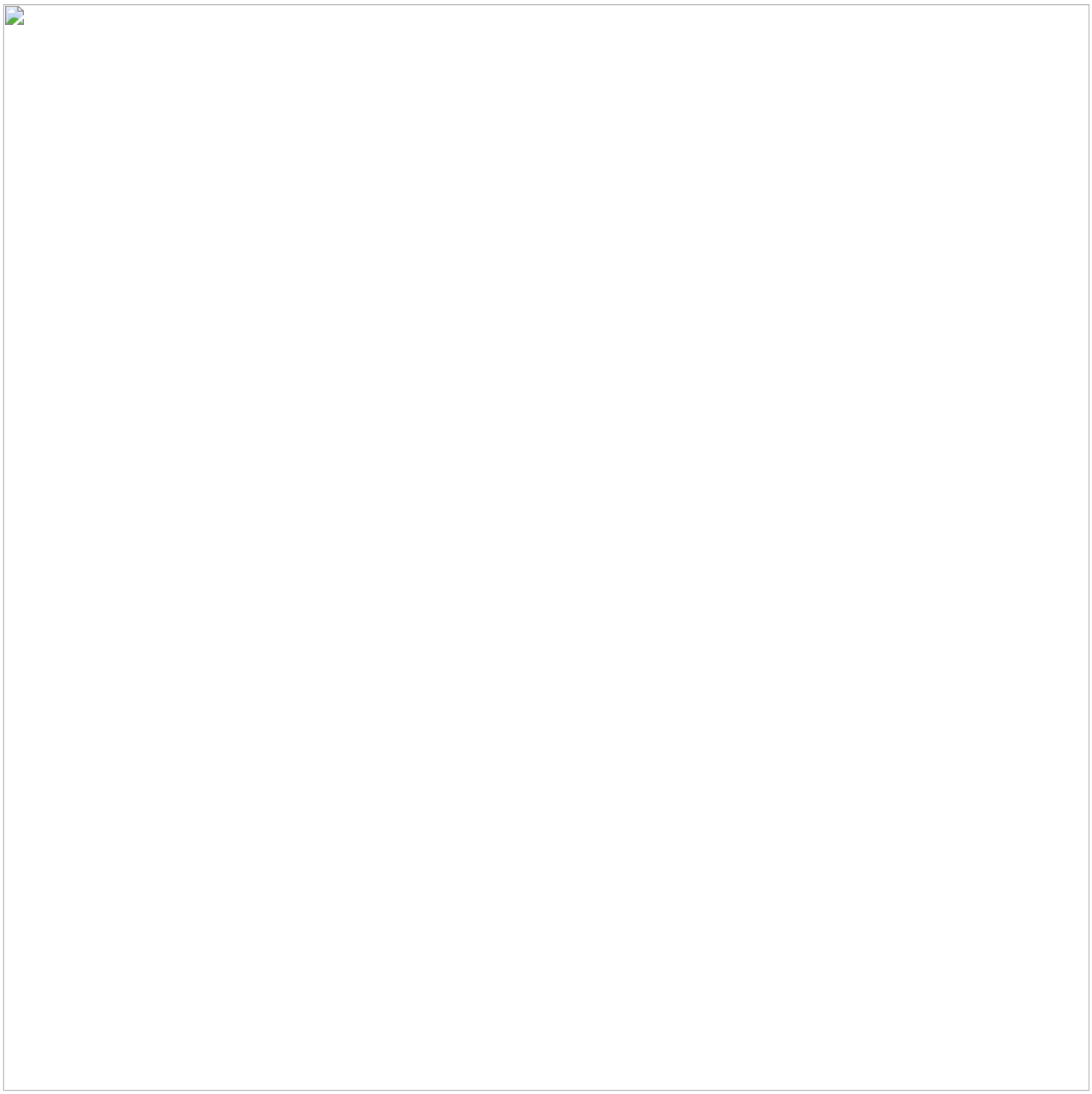
Peter Kálnai
16 Feb 2017 - 12:00PM

The purpose of this blog is to deliver technical details of an as-yet minimally documented malware that has made headlines in Poland.

Hot news about successful attacks on Polish banks appeared recently on the Polish security portal ZaufanaTrzeciaStrona.pl (translated in English here). The impact of the attacks was described dramatically with adjectives like “the most serious”. The initial reports were very recently supported by two blogposts by Symantec and BAE Systems. The nationalities of affected institutions were extended also to Mexico and Uruguay, with even more high-profile targets in the attackers’ viewfinder that are located worldwide. There are many interesting aspects to these attacks starting from the targets, moving on to the vector of compromise, right up to the specific features of the malicious executables used. While the first two aspects have been quite thoroughly examined so far, the malicious binaries involved haven’t received much attention so far. The purpose of this blog post is to deliver technical details of this as-yet minimally documented malware.

Distribution channel

As mentioned on the Polish security portal, the threat is delivered sneakily, via a watering hole attack, whereby a trusted but compromised website redirects to a landing page booby-trapped with an exploit. In the case of the Polish attacks, the starting point was the official website of Komisja Nadzoru Finansowego (the Polish Financial Supervision Authority):



However, our data indicate that the website of the equivalent Mexican authority, Comisión Nacional Bancaria y de Valores (National Banking and Securities Commission), also served identical malicious redirects (unfortunately, information released by web tracking services or by the institution itself has not yet confirmed or made any mention of this). Based on our data, the redirects went from this subsite:

Stage 1: Dropper

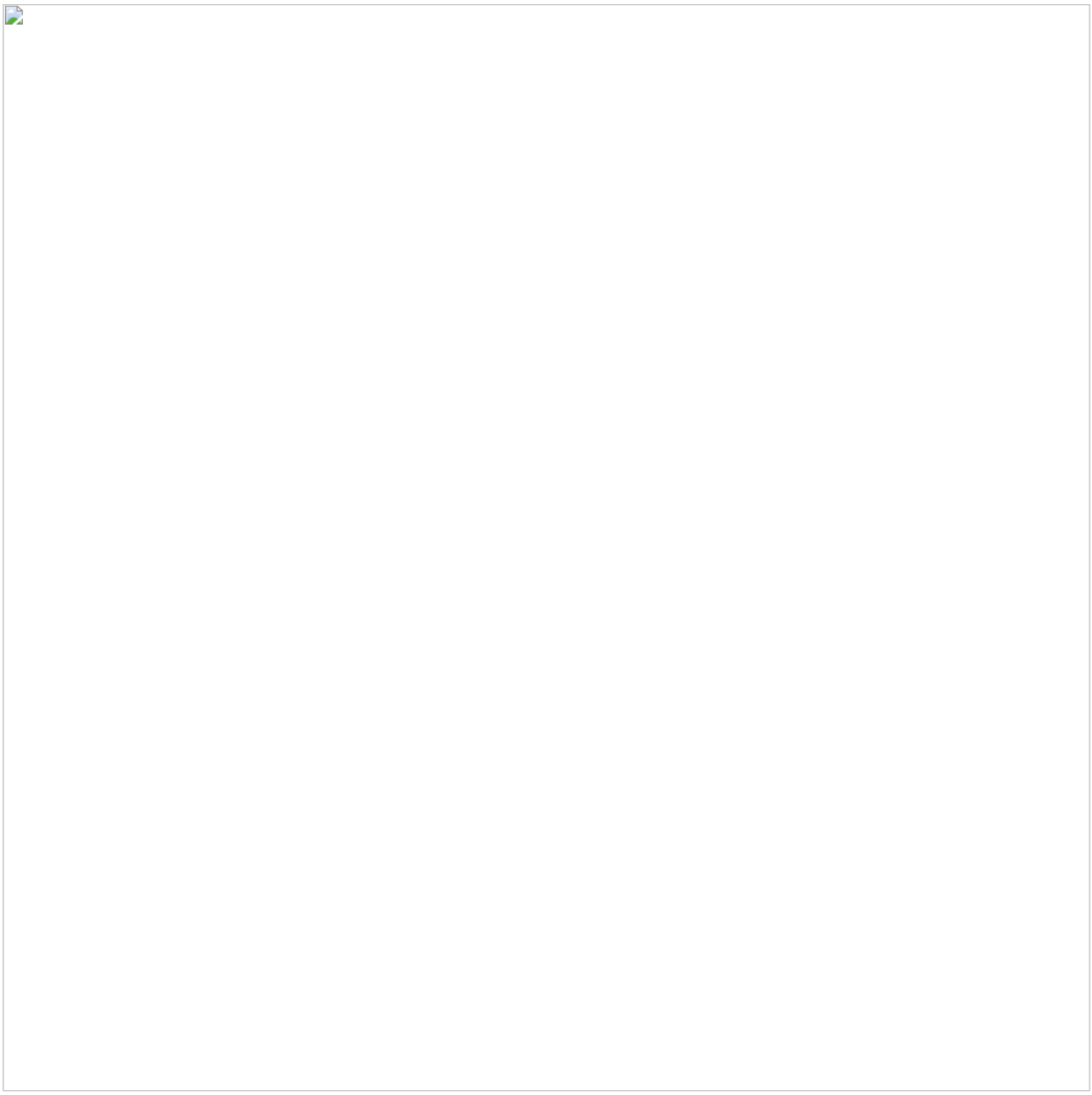
If the exploit kit successfully delivers the intended malware, the malicious payload – a 64-bit console application – is executed on the victim's computer. Unlike the dropper reported by BAE Systems, this program expects one of three switches in the argument list: *-l, -e, or -a* (section (2) in the following figure). While the *-l* switch has the same meaning, the remaining two are necessary to **e**xtract binaries of the next stage from resources (section (4) in the figure) and to (**a**uto)start one of them as a service (section (5)):

In section (5) in the figure above, the dropper tries to change the configuration of a system service in order to load the dropped loader as a service. The service is configured to start automatically by the service control manager during system startup. Administrator privileges are necessary to achieve the goal.

Unlike the subsequent stages, the threat does not hide itself very carefully during the first stage. It even contains verbose statements that provide information about the status of execution (in this case, about extracting encrypted resources; however, the debug-info like original function names are not present).

The dropper employs dynamic API loading instead of having Windows functions in its import table (well explained in the Novetta report "Operation Blockbuster" on the Lazarus Group, page 34). Section (3) in the figure above displays a wrapper of this feature, going one system library after another.

It seems that the attackers denote the second stage as "loader" and the third stage, containing the main malware functionality, as "module". The loader is decrypted, while the module is just extracted and dropped as-is. To reduce their visibility during forensic analysis, the files borrow their creation time from the system's shlwapi.dll. An interesting feature of the encryption algorithm employed is that it is quite a recent RC4-like stream cipher called Spritz (https://people.csail.mit.edu/rivest/pubs/RS14.pdf, 2014). C and Python implementations of Spritz are already available and they correspond to the following disassembled code from the dropper:



Stage 2: Loader

There is a further indication of the intent to preserve the low profile of the threat. The loader is protected by a commercial packer called Enigma Protector and, as we learned, the module is stored in an encrypted state, waiting for the loader to unleash it. Having taken a closer look at this protection, we found that an unregistered copy of 64-bit Enigma v. 1.31 was used. Entirely as we would expect, since malware authors at this level of proficiency would not normally make such elementary mistakes as potentially revealing their identity by using a properly registered copy. (However, it is not unusual for criminals to abuse a leaked or pirated registered application if available.) Attackers intending to build a large botnet do not, in general, use commercial packers because a certain proportion of anti-malware vendors detect those generically. They therefore restrict the potential size of the botnet. However, in the case of a targeted attack there are advantages to using such protection. One that comes to mind is that the reconstruction of the original binary – i.e. as it was before it was subjected to the binary camouflage process – is hardly ever easy.

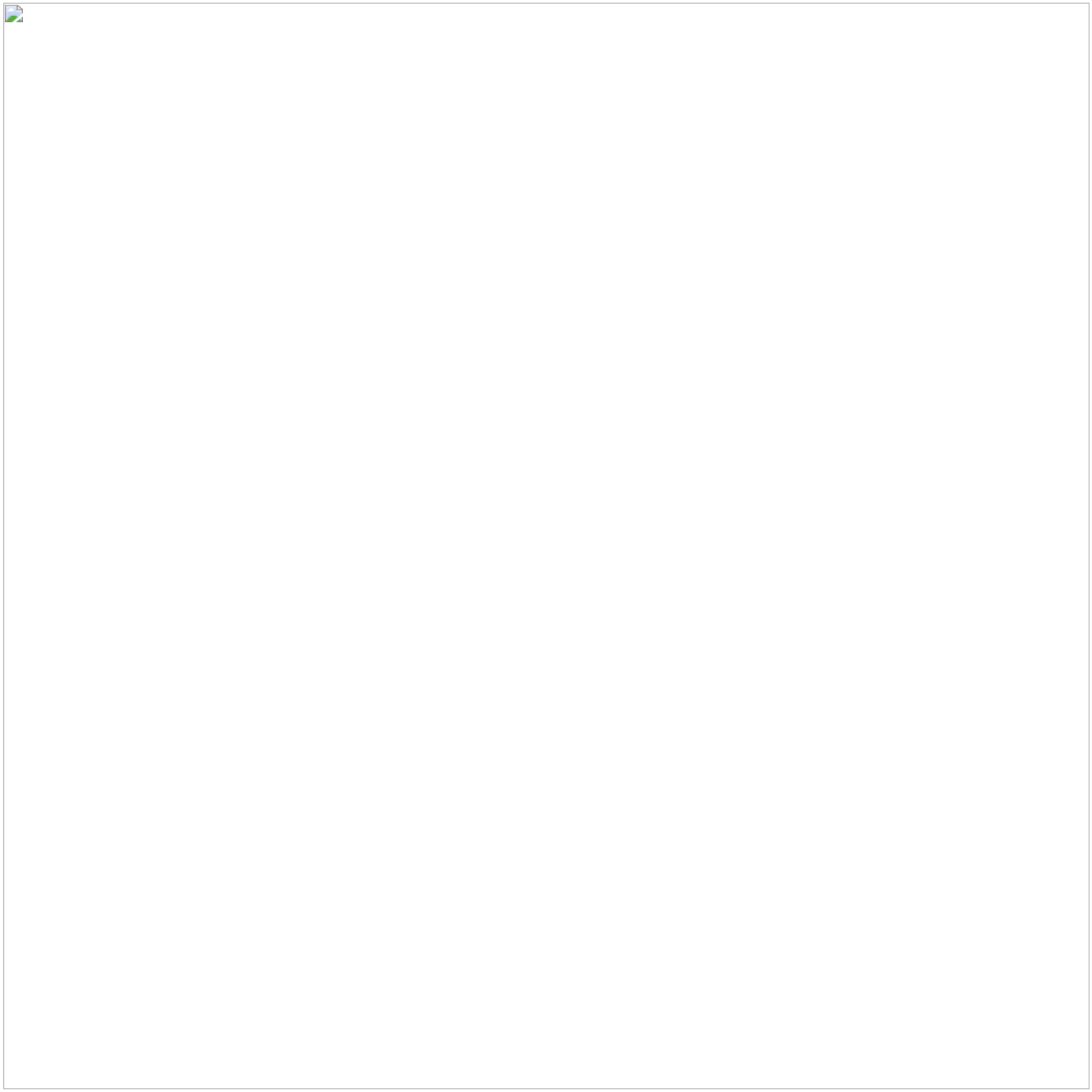
The impression sometimes given that only 64-bit Windows machines can be targeted by this threat is wrong. A 32-bit variant has also been extracted from computers in some affected institutions. Though it has the same overall structure, the 32-bit variant is not just a recompilation of the 64-bit one's source, but differs slightly: the dropper and the loader stages are combined into one, classic RC4 encryption is used rather than Spritz, and the module stage is stored in the registry instead of in the filesystem. Also, the version of the Enigma protector applied is 3.7, with a single developer license, and apparently used to protect the binary on 11th of January 2017.

Stage 3: Module

The third and final stage is the relatively large module (~730 KB) that contains the main features of the malware: to communicate with the C&C and receive orders from operators.

The module injects itself into all running sessions on the compromised Windows system.

The next screenshot shows the situation after loading the module into the disassembling tool IDA Pro. The upper bar shows various parts of the binary: the code sections in blue, and the data sections in gray-and-yellow. The difference between cyan and dark blue sections is that cyan represents code statically linked from existing libraries. Besides the usual C runtime, we identified the linkage of an open source multiprotocol file transfer library called libcurl (version 7.47.1, released on 8th of February, 2016), together with chunks of code from projects like OpenSSL and XUnzip. The color effect in the bar is not generated automatically: in this case, we had to explicitly mark parts that we considered as linked library code and we imported all the function names. The dark blue sections represent the code written by the attackers.



There is only one encrypted URL stored in the module. Communication is encrypted. We haven't recorded any communication, because the remote server wasn't responding at the time of analysis. The module supports quite a lot of commands, with more than enough of the kind to clearly characterize it as a remote access Trojan (RAT). The dictionary of the commands is like this: "SLEP", "HIBN", "DRIV", "DIR", "DIRP", "CHDR", "RUN", "RUNX", "DEL", "WIPE", "MOVE", "FTIM", "NEWF", "DOWN", "ZDWN", "UPLD", "PVEW", "PKIL", "CMDL", "DIE", "GCFG", "SCFG", "TCON", "PEEX", "PEIN". Many of the commands are self-explanatory (SLEP is to sleep, PKIL is to kill a process, UPLD is to exfiltrate data, DOWN is to download, DEL is to delete a file, and so on). It is possible the original libcurl functions have been customized to meet the needs of the attackers. However, libcurl is a huge project with hundreds of contributors, tens of thousands lines of code and hundreds of options. The precise inspection and analysis of the linkage is in progress at the time of writing.

Lazarus-like toolkits

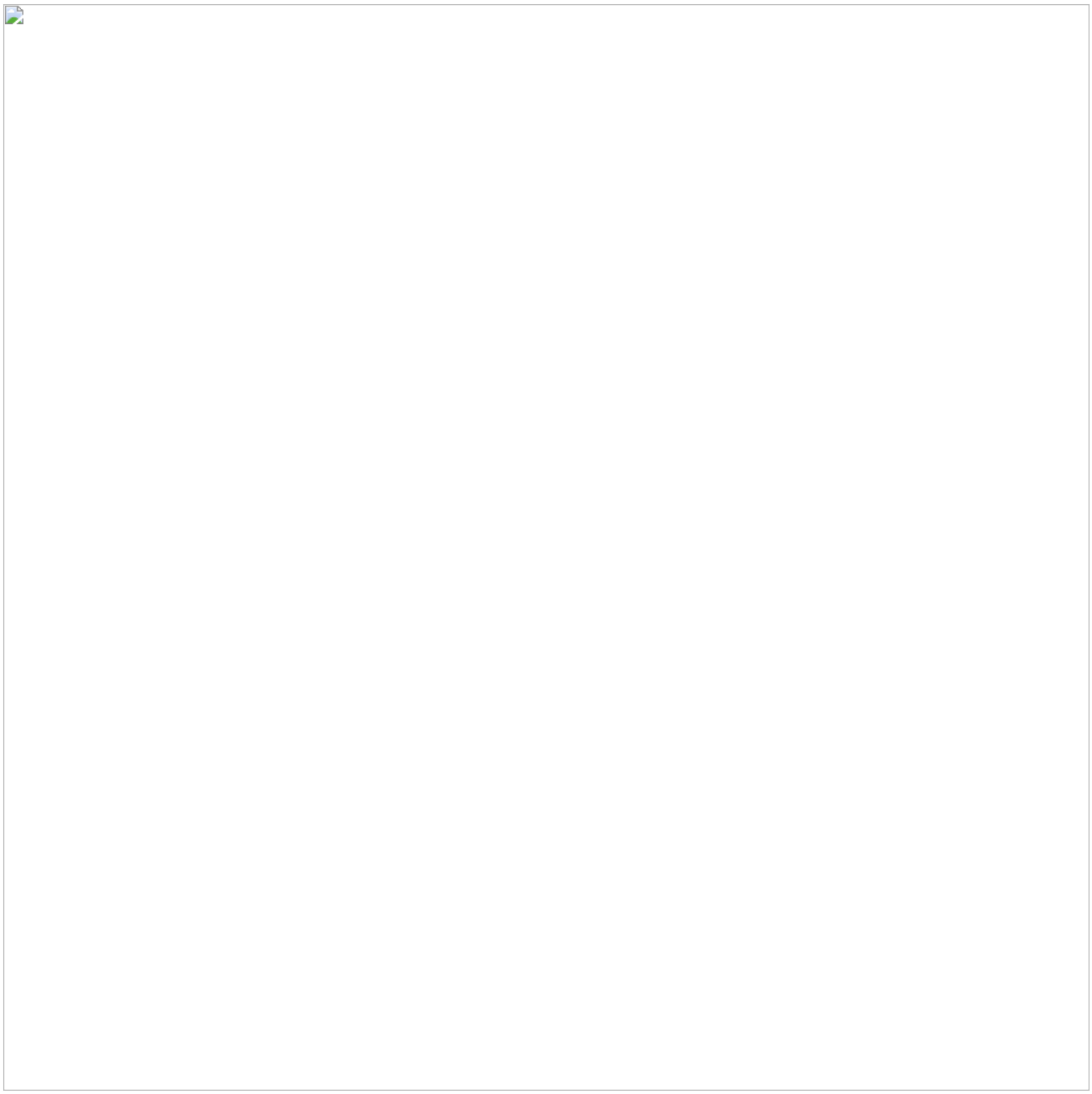
The researchers from BAE Systems refer to the Enigma-protected 32-bit dropper as "Once unpacked it drops a known malware variant, which has been seen as part of the Lazarus group's toolkit…". Moreover, Symantec states "Some code strings seen in the malware used shares commonalities with code from malware used by the threat group known as Lazarus." One can find a connection also in the report by Novetta, such as the already-mentioned dynamic API loading. All these signs motivate us to describe actual crucial properties of a Lazarus-like toolkit as follows:

(1) multi-staged malware that executes in a cascade
(2) the initial stage is a console application expecting at least one parameter
(3) WINAPIs are loaded dynamically
(4) RC4 or similar with a long key used for the decryption of the next stage
(5) the consequent stage(s) are dynamically linked libraries that are loaded as a service with the SERVICE_AUTO_START start type (administrator privileges are required for this action)

Our data show activity of various Lazarus-like malware in-the-wild recently. However, to provide a clearer picture of the whole case, we need time to collect further relevant information.

Strange discovery

During our research, we found another interesting sample that belongs to the same malware family. A console application expecting four parameters called fdsvc.exe ((2) check), that executes in a cascade ((1) check). Moreover, it decrypts the next stage using RC4 with a 32-byte key ((4) check). It doesn't have the last two properties. On the other hand, it injects the payload into all running Windows sessions. Moreover, the payload has statically linked libcurl v. 7.49.1. What makes this sample especially interesting, is how the final stage parses commands from operators. The operators are using commands in Russian language presented in a translit, which is a method of encoding Cyrillic letters with Latin ones:



But we have to be careful with attribution. The language used might very well be a false flag! One quick reason that comes to mind: malware authors usually implement commands via numbers or English shortcuts. Having a twelve-letter command is a bit impractical.

Conclusion

Considering the artifacts in the code, we venture to say that this is neither some reuse of code existing long before these recent Polish banking attacks, nor a forgotten, discontinued project. Moreover, we have observed occurrences of malware resembling this example in the past few weeks. The attackers behind the threat have a good knowledge of what they are doing, so incident response teams in financial institutions or other high-profile targets might not be having much untroubled sleep in the near future. Actually, that is their job these modern days: to suffer sleepless nights!

IoCs

Samples involved in the attacks

SHA1	Detection	Note

SHA1	Detection	Note
bedceafa2109139c793cb158cec9fa48f980ff2b	Win64/Spy.Banker.AX	Dropper;gpsvc.exe
aa115e6587a535146b7493d6c02896a7d322879e	Win64/Spy.Banker.AX	Enigma-protected loader
a107f1046f5224fdb3a5826fa6f940a981fe65a1	Win64/Spy.Banker.AX	Enigma-protected module; RAT; libcurl v. 7.47.1
4f0d7a33d23d53c0eb8b34d102cdd660fc5323a2	Win32/Spy.Banker.ADQH	32-bit Enigma-protected dropper;gpsvc.exe

Malware with a translit

SHA1	Detection	Note
fa4f2e3f7c56210d1e380ec6d74a0b6dd776994b	Win64/Spy.Banker.AX	Dropper;fdsvc.exe
11568dffd6325ade217fbe49ce56a3ee5001cbcc	Win64/Spy.Banker.AX	Encrypted module;fdsvc.dll
e45ca027635f904101683413dd58fbd64d602ebe	Win64/Spy.Banker.AX	Decrypted module; RAT;libcurl v. 7.49.1 (*)
50b4f9a8fa6803f0aabb6fd9374244af40c2ba4c	Win32/Spy.Banker.ADRO	32-bit module; RAT;libcurl v. 7.49.1

16 Feb 2017 - 12:00PM