The North Korean AV Anthology: A unique look on DPRK's Anti-Virus Market

Mark Lechtik & Ariel Jungheit / GReAT

Abstract

North Korea is a country known to strive for self-reliant economy. As part of that, a lot of its products are built and used exclusively within the country. Computer software, namely Antivirus programs, are no exception. One example outlined in a research from 2018 is Silivaccine, a known North Korean software to protect against foreign malware, which was indeed built in-house.

This concept is well demonstrated by Juche, North Korea's official ideology postulating that by becoming self-reliant and strong, a nation can achieve true socialism. The practice of Juche is firmly rooted in the ideals of sustainability and lack of dependency. Despite that, it is criticized by many as a mechanism for sustaining the totalitarian rule of the regime, justifying the country's 'Isolationism'. For them Juche is merely a facade.

Mark Lechtik is a Senior Security Researcher at Kaspersky`s GReAT, previously working as the Malware Research Team Leader at Check Point Research. He was born in Russia, but lives most of his life in Israel, where he graduated from Ben-Gurion University with a B.Sc in communication system engineering. Mark passion is reverse engineering malware, both as occupation and hobby. He enjoys deep diving into a variety of malwares from the worlds of both APT and crimeware, digging out their gory technical details and outlining their underlying stories and threat actors.

Ariel Jungheit is a security researcher on the Global Research and Analysis Team at Kaspersky Lab. Based in Germany, Ariel's interest in cybersecurity stems from his time in national military service, and before joining Kaspersky Lab, he worked for FireEye and iSight as a senior security analyst and an intelligence analyst respectively. At Kaspersky Lab, he`s contributing to GReAT's mission by helping to investigate the most active and advanced threat actors, targeted attacks, attacker tools and more. Ariel's professional passions includes reverse engineering malware, uncovering, tracking and analyzing APT campaigns and reporting all about it.

Introduction

Looking at the state of affairs regarding North Korea's sustainability, we see that even within the Antivirus market the nation cannot live up to its own ideology — from standard usage of 3rd-party libraries to abuse of other vendor's engines in their code, using foreign produce is a common practice. As we will outline in this research, several instances of this conduct can be witnessed by examining the products' code-bases and signatures.

Furthermore, it is interesting to note that Silivaccine is not the only Antivirus created, and possibly used, within the DPRK. As it turns out, there is a wealth of such programs developed internally. In the course of a decade we could witness 5 different Antivirus solutions originating in North Korea, which are naturally well hidden from the rest of the world.

This is quite surprising for a country that is sanctioned for many years and generally lacks a lot of resources. Such sanctions by world powers were originally intended to push North Korea away from its nuclear program, and at the same time punish it for cyberattacks, money laundering and violation of human rights. It is hard to assess how well these sanctions did in achieving their goal, but at least in the realm of information security and defense the effect was opposite.

The figure below depicts the range of known Anti-Viruses between 2002—2014, however we can provide an insight only about a subset of them which were made available to us for analysis. The reason the timeline halts in 2014 is that we were not able to obtain any further software created past that year.

In the following sections we are going to take a brief tour through several of these Antiviruses outlining their architecture, commonalities, differences and other interesting caveats. As this paper is meant to give an overview on the discussed products, deeper analysis details are reserved for another document. Also, we are actively looking for missing Antiviruses from our collection, needless to say that we welcome any contribution from the readers to accomplish it.

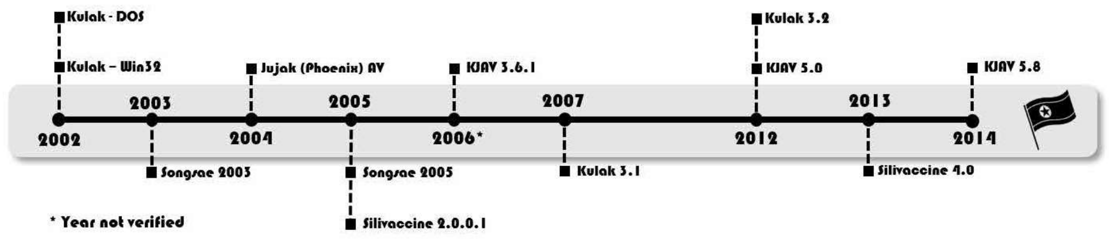


Figure 1. The timeline of known Antivirus products created in the DPRK

Songsae

Songsae Antivirus (성새 'Castle') was developed between 2003—2005 by "Kim Il Sung University". It appeared in 2 versions — one released in 2003 and the other in 2005, providing protection for both DOS and Windows operating systems. In this section we will discuss SongSae 2003, and particularly the parts of it that are compatible with Windows.

One of the first things notable about it is its very minimalistic user interface. Written in C++ using the MFC framework, the GUI is rendered from embedded HTML and JavaScript code with the help of the ActiveX technology. This is a rather outdated and uncommon way to write GUI, and may suggest that the developing team behind it was not highly experienced or acquainted with newer technologies.

Another point is the fact that all text in the product (including GUI strings and readme file) is written entirely in English. A possible explanation can be found in an analysis of North Korea's economic envoys in the early 2000's[1], indicating that the country was pushing towards expanded economic relationships with regards to their IT Technology, partly by holding exhibitions showcasing it (e.g. CeBIT in 2006).

Considering the above and having its readme file being signed off with the sentence "We are welcome to contact with other companies and expert groups", we are left to believe that the DPRK sought to create all kinds of partnerships with foreign vendors at that time. Having said that, we could observe that the subsequent Antiviruses slowly strayed away from English towards an exclusive usage of Korean.

In terms of capabilities, the Anti-Virus can scan memory, local and network drives as well as other 'low-level' parts of the disk (e.g. the boot sector and its components). Later on, in 2005, it also started to support the quarantine feature. The scanning is done based on signatures, which constitute a browsable virus database containing 1675 and 1836 records in versions 2003 and 2005 respectively.

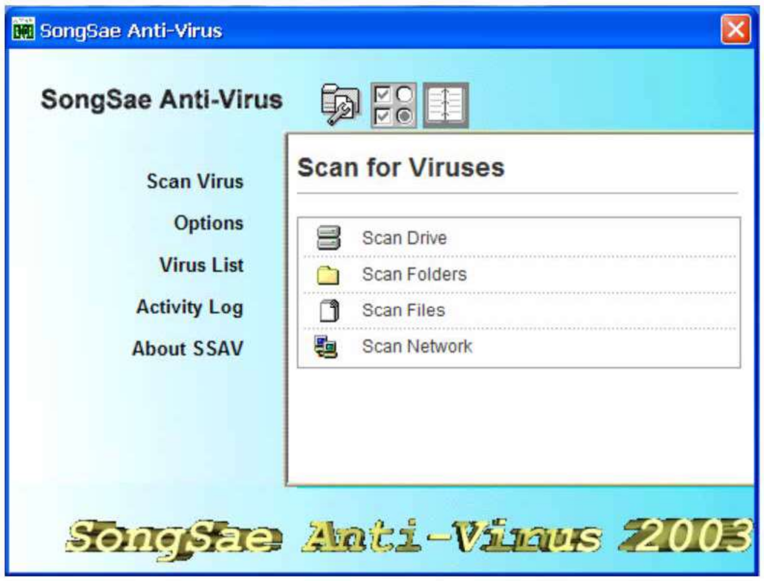


Figure 2. SongSag'e main GUI screen

Architecture

To be able to understand it a little better, we provide an outline of SongSae's architecture, followed by an overview on each component. This should give the reader an idea of how the components inter-operate, and generally convey the software's simplicity.

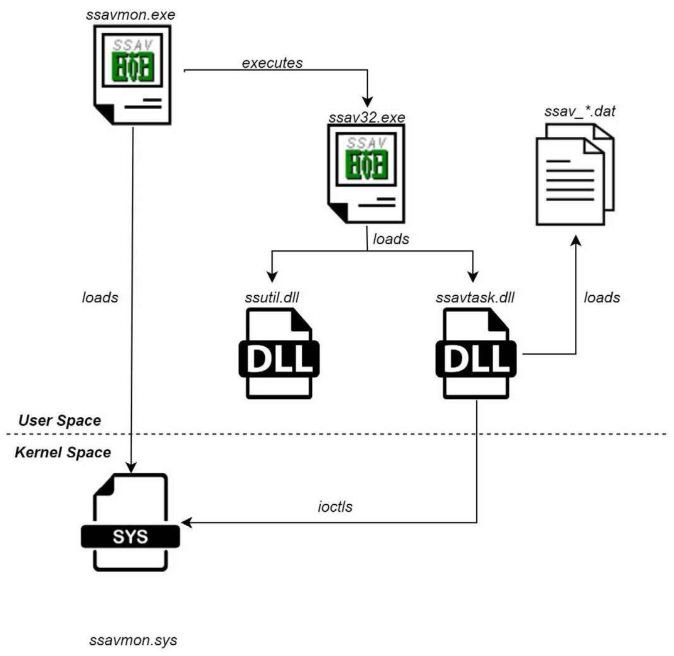


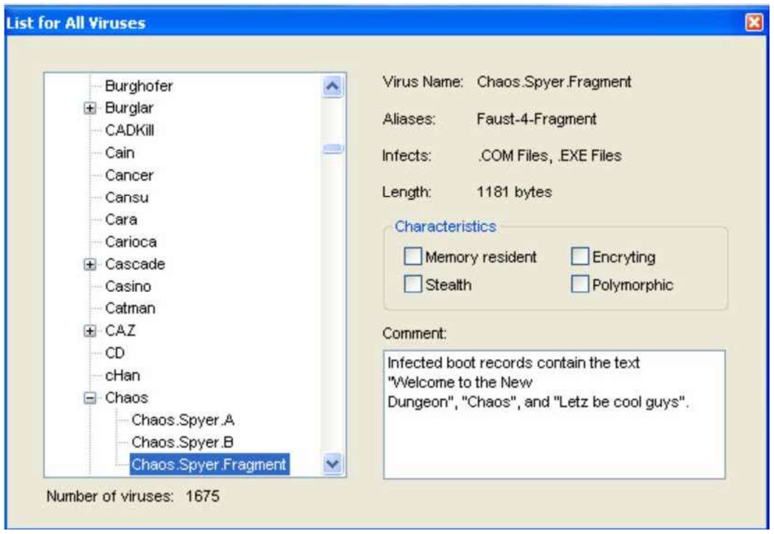
Figure 3. SongSae 2003 architecture

Components:

- **ssavmon.exe** — Process that manages the Antivirus' tray menu application. This allows opening the main GUI by spawning a new ssav32.exe process, and enabling or disabling the real-time monitoring driver (ssavmon.sys).
- **ssav32.exe** — Main GUI menu. All windows and their graphics are rendered from HTML and JS code, which is made possible using ActiveX & MFC. Loads ssavtask.dll (contains the scanning logic) and ssutil.dll.
- **ssavmon.sys** — Real time monitor which intercepts file related activity. It checks if a file ends with *vir* or *kln* extensions, and if not conducts a series of static checks against it. If the file was detected by one of these, access to it will be avoided by assigning the corresponding action's IRP with the status 0xC00000F (STATUS_NO_SUCH_FILE).
- **ssavtask.dll** — The core scanning engine of the AV, implemented as an ActiveX control module.
- **ssutil.dll** — Exports the function SS_BrowseFolder, which makes use of SHGetSpecialFolderLocation to obtain various directory paths.
- **ssav_d.dat** — Patterns\signatures related file.
- **ssav_m.dat** — Patterns\signatures related file.
- **ssav_v.dat** — Patterns\signatures related file.
- **ssav_i.dat** — List of signature descriptions.
- **ssav_n.dat** — List of signature names (xored with 0x21).

Notes on the Signatures

Looking at the virus signature database file it's apparent that the description for the viruses were copied from virus information summary lists written throughout the 1990's, with the highest resemblance to Symantec's Norton Anti-Virus signature file dated 1994. Although we weren't able to conclude there was usage of any 3rd party software to create SongSae's engine, the fact that virus descriptions were copied from other sources makes it plausible that the signatures themselves were copied as well.



List for All Viruses

Burghofer
Burglar
CADKill
Cain
Cancer
Cansu
Cara
Carioca
Cascade
Casino
Catman
CAZ
CD
cHan
Chaos
Chaos.Spyer.A
Chaos.Spyer.B
Chaos.Spyer.Fragment

Virus Name: Chaos.Spyer.Fragment

Aliases: Faust-4-Fragment

Infects: .COM Files, .EXE Files

Length: 1181 bytes

Characteristics

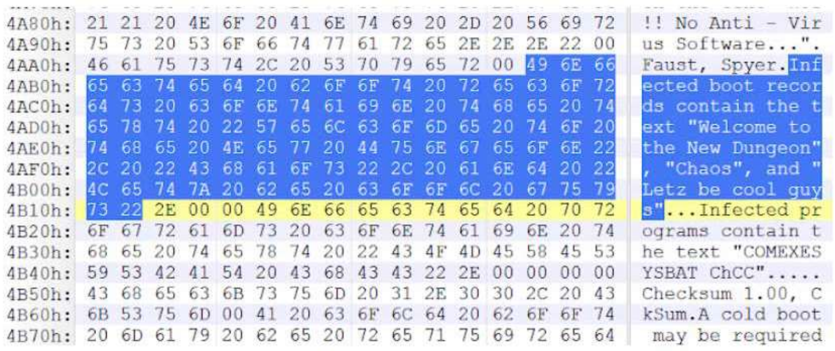
Memory resident ☐ Encryting ☐
Stealth ☐ Polymorphic ☐

Comment:

Infected boot records contain the text "Welcome to the New Dungeon", "Chaos", and "Letz be cool guys".

Number of viruses: 1675

Figure 4. Virus definition for 'Chaos' in Songsae 2003



```
4A80h: 21 21 20 4E 6F 20 41 6E 74 69 20 2D 20 56 69 72  !! No Anti - Vir
4A90h: 75 73 20 53 6F 66 74 77 61 72 65 2E 2E 2E 22 00  us Software...".
4AA0h: 46 61 75 73 74 2C 20 53 70 79 65 72 00 49 6E 66  Faust, Spyer.Inf
4AB0h: 65 63 74 65 64 20 62 6F 6F 74 20 72 65 63 6F 72  ected boot recor
4AC0h: 64 73 20 63 6F 6E 74 61 69 6E 20 74 68 65 20 74  ds contain the t
4AD0h: 65 78 74 20 22 57 65 6C 63 6F 6D 65 20 74 6F 20  ext "Welcome to 
4AE0h: 74 68 65 20 4E 65 77 20 44 75 6E 67 65 6F 6E 22  the New Dungeon"
4AF0h: 2C 20 22 43 68 61 6F 73 22 2C 20 61 6E 64 20 22  , "Chaos", and "
4B00h: 4C 65 74 7A 20 62 65 20 63 6F 6F 6C 20 67 75 79  Letz be cool guy
4B10h: 73 22 2E 00 00 49 6E 66 65 63 74 65 64 20 70 72  s"...Infected pr
4B20h: 6F 67 72 61 6D 73 20 63 6F 6E 74 61 69 6E 20 74  ograms contain t
4B30h: 68 65 20 74 65 78 74 20 22 43 4F 4D 45 58 45 53  he text "COMEXES
4B40h: 59 53 42 41 54 20 43 68 43 43 22 2E 00 00 00 00  YSBAT ChCC".....
4B50h: 43 68 65 63 6B 73 75 6D 20 31 2E 30 30 2C 20 43  Checksum 1.00, C
4B60h: 6B 53 75 6D 00 41 20 63 6F 6C 64 20 62 6F 6F 74  kSum.A cold boot
4B70h: 20 6D 61 79 20 62 65 20 72 65 71 75 69 72 65 64   may be required
```

Figure 5. Virus definition for 'Chaos' in Norton Antivirus 3.0 (1994)

KJAV

KJAV (Korean Juche-oriented Anti Virus) a.k.a "Singi" (신기) was developed by the Kim Chaek University of Technology in Pyongyang which is known to conduct projects for foreign companies. This was done partly in collaboration with Gwangmyong IT Center ("GeMyong Tech"), reportedly a spin-off of KCC (Korea Computer Center). This organization specializes in network software and security, developing Antivirus, data encryption, data recovery, and fingerprint software. KJAV has been in development since 2006 with Version 5.0 being released in 2012 and version 5.8 in 2014. According to news sources[2] it is still in development as of 2017, alongside a mobile version for smartphones.

KJAV introduces the option to scan files on hard disks and USB flash drives (including those that reside in archives), as well as to inspect network traffic for suspicious patterns that might be related to malicious activity. The latter is functionality that is not common to all North Korean Antiviruses, and enhances this one into a more complete security solution.

Having said that, there are only several network signatures we could witness, those were intended to catch a couple of worms — *Conficker* (W32.Net-Worm.Kido) which was a well known and widespread worm targeting MS08-67, and ***ChinaHacker*** (W32.ChnHacker.Email) which was used to spread via mass mail sent from each victim machine around 2002.

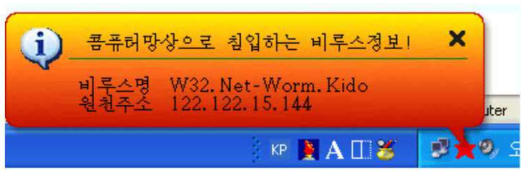
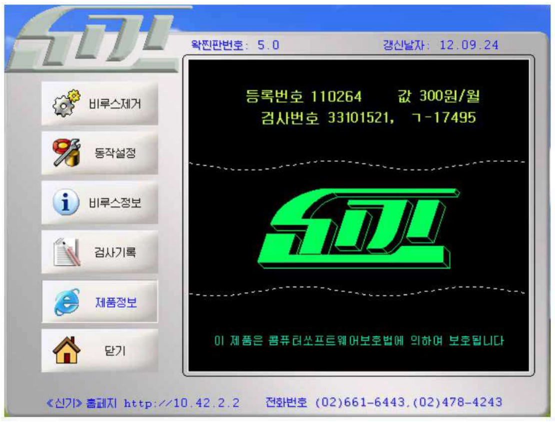

콤퓨터망상으로 침입하는 비루스경보!
비루스명 W32.Net-Worm.Kido
원천주소 122.122.15.144

Figure 6. Alert for Conficker detected traffic

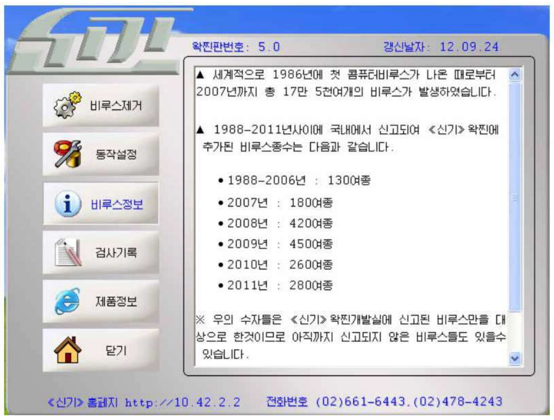
As for its engine, we assume that this Antivirus was developed almost entirely within the DPRK, given the fact that we haven't observed any conspicuous indication of the contrary. This appropriates its name, as the Juche dictates independence. The only similarities that we did observe were with malicious binaries widely perceived to be developed in the DPRK, which we will outline in subsequent sections.

Much to our surprise, KJAV is a licensed product that requires a license costing the user 300₩ / Month (~0.25$) as portrayed in the figure below. The reported average annual income in North Korea according to a 2013[3] estimate is thought to be $1,000 to $2,000. This makes KJAV affordable for computer users across North Korea, indicating that it might serve as a home Antivirus solution.



왁찐판번호: 5.0 갱신날자: 12.09.24
등록번호 110264 값 300원/월
검사번호 33101521, ㄱ-17495
이 제품은 콤퓨터쏘프트웨어보호법에 의하여 보호됩니다
《신기》 홈페지 http://10.42.2.2 전화번호 (02)661-6443,(02)478-4243

Figure 7. KJAV's main screen, containing license information



왁찐판번호: 5.0 갱신날자: 12.09.24

▲ 세계적으로 1986년에 첫 콤퓨터비루스가 나온 때로부터 2007년까지 총 17만 5천여개의 비루스가 발생하였습니다.

▲ 1988-2011년사이에 국내에서 신고되여 《신기》 왁찐에 추가된 비루스종수는 다음과 같습니다.

- 1988-2006년 : 130여종
- 2007년 : 180여종
- 2008년 : 420여종
- 2009년 : 450여종
- 2010년 : 260여종
- 2011년 : 280여종

※ 우의 수자들은 《신기》 왁찐개발실에 신고된 비루스만을 대상으로 한것이므로 아직까지 신고되지 않은 비루스들도 있을수 있습니다.

《신기》 홈페지 http://10.42.2.2 전화번호 (02)661-6443,(02)478-4243

Figure 8. An outline of the new malware families the developers witnessed each year by 2012

Architecture

Once again, to get a better idea of the product we provide a brief overview of its architecture. This can be divided into 4 essential parts:

- Anti-Virus
- Host Based IPS & Update Utility
- Utilities
- Data Files

The components under each one are shown in the figure below and described in the subsequent paragraph.

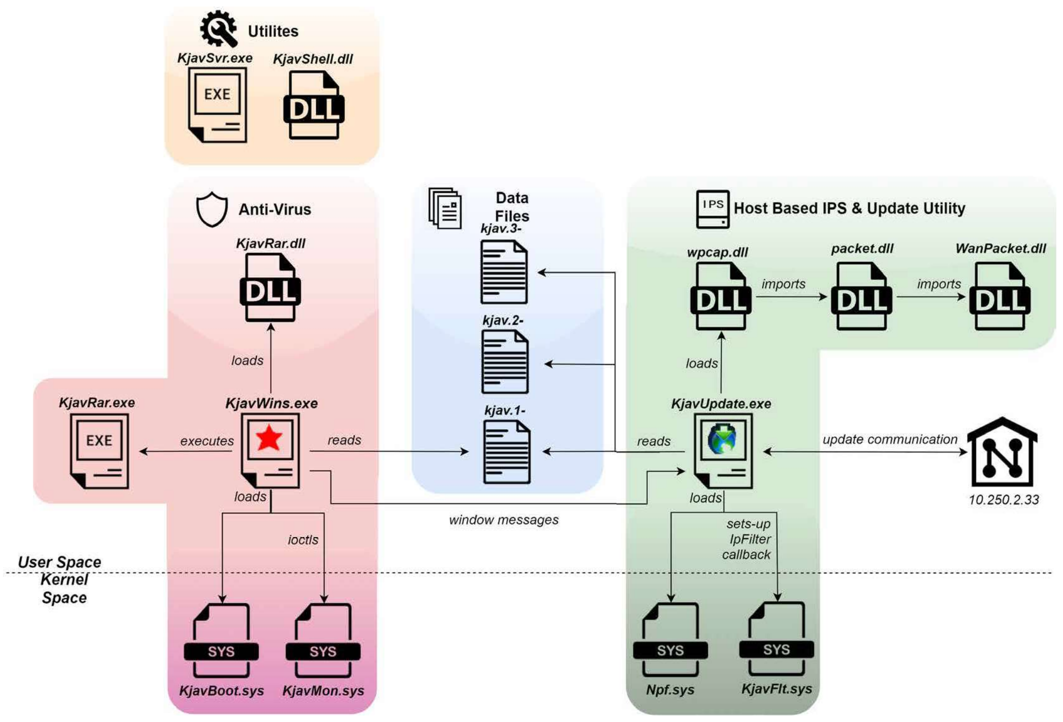


Figure 9. Architecture of KJAV

Components:

1. Anti-Virus

- **KjavWins.exe** — the main executable of the whole software suite. It contains all other components as resource binaries and deploys them in the system upon execution. The other part of it serves as a monolithic core component of the AV, containing the GUI (written using MFC), the scanning engine and hardcoded signatures. As KJAV appears to be licensed software and contains code for verifying user keys, this binary is protected with Themida.
- **KjavBoot.sys** — a driver invoked during startup which deletes all files under the registry key `\Registry\Machine\SYSTEM\CurrentControlSet\Control\Session Manager\DelFile`.

- **KjavMon.sys** — modified version of a driver used in a legacy SysInternals tool named FileMon. This tool allowed monitoring file system activity, and later was combined with another deprecated tool named RegMon to create Process Monitor. In this case, the referenced version of filemon.sys is 4.28, dating to the year 2000, and intended to intercept file access so that the Antivirus can conduct real time monitoring of it before it is executed.
- **KjavRar.exe** — public command line version of WinRar 3.51.
- **KjavRar.dll** — public documented UnRar.dll, used for reading archive contents and decompressing .rar files. Also part of the WinRar software.

2. Host Based IPS & Update Utility

- **KjavUpdate.exe** — this is the main executable for this component of the AV, which registers a window class named "KjavUpdate" and carries out various functionalities upon receiving window messages. In this sense, its purpose is twofold — one is to serve as an update utility and the other is functioning as a host based network IPS.
  In the former case its able to receive window messages with update server IPs (or use a hardcoded fallback one, set to 10.250.2.33 — this was also used as the update server for SiliVaccine 2013), get commands to issue the current engine and signature files to a server and deploy new variants of these when received.
  The latter functionality allows real time monitoring of IP packets and their inspection against predefined network signatures. According to various strings in the binary, this part of the product was named NetVirusTracer by the authors.
  The inspection can be done in 2 ways — one is by using an existing Windows feature whereby a special callback can be registered to the ipfltdrv.sys driver, allowing the OS to invoke this function each time an IP packet is processed.
  Another way is by utilizing a combination of WinPcap components which allow setting an optional capture filter and intercepting packets as they are passed through the system. During interception, packets are scanned against the network ruleset.
- **KjavFlt.sys** — a network filter driver which registers an IpFilter callback — a hook that gets executed each time an IP packet goes in or out of the system. The hook contains logic for scanning the packet against network signatures that reside in the "`Kjav.3-`" file.
- **WinPcap components** — these include npf.sys (Netgroup Packet Filter Driver), `wpcap.dll`, `packet.dll` and `WanPacket.dll`. All are 3rd party components which allow packet capture, injection, programmable filtering and monitoring. All of these work together to facilitate the creation of a host based network IPS as specified above.

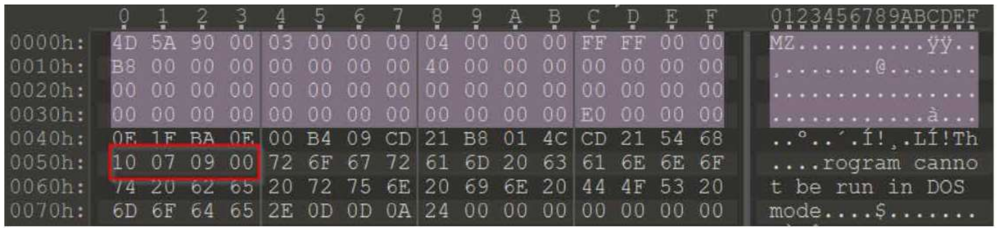
3. Utilities

- **KjavSvr.exe** — can create or destroy a service named HdSvr, which upon request deletes files in startup folders containing "`~.exe`" as a substring in their name.
- **KjavShell.dll** — a COM class serving as a shell extension, which is used to register a context menu handler for KJAV. The entry in the context menu allows scanning a single file when right-clicking it.

4. Data Files

- **Kjav.1-** — main Antivirus signature file, containing both metadata on the currently deployed engine and version of the signatures, as well as the corresponding virus patterns.
- **Kjav.2-** — peculiar malformed DLL file (originally named PatchDll.dll) which is supposedly packed with UPX. It is impossible to run because of its format corruption, however it can be witnessed that KjavUpdate reads a field from its DOS header and fills it as data sent to the update server, under the parameter 'DllUpdateDate'. On another instance, the file is overwritten with a part of the server response that is followed by the keyword 'GetDllData'.

- **Kjav.3-** — network signature file, encoded with a single byte XOR (0xf8). In the case of the 2012 version we could find only one signature with the name W32.NetWorm.Kido, which is a name some vendors gave at the time to Conficker. In the 2014 version we found an additional signature corresponding to the ChinaHacker worm.



```
        0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0000h: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
0010h: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
0020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0030h: 00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00  ............à...
0040h: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
0050h: 10 07 09 00 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  ....rogram canno
0060h: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
0070h: 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
```

Figure 10. DOS header of kjav.2- containing a field used during an HTTP request to the update server

Notes on Signatures

One of the interesting signatures we could find inside the Antivirus was one intended to detect JML. This is a file infector that was developed around 1997 as part of a graduation thesis of a student named "Cho Myung-rae" at Mirim University about the militarization capabilities of the computer virus[4]. Named after Cho's initials (alternatively spelled "Jo Myung-lae") — Cho's achievement was well acclaimed, forming a research group centered around it in North Korea's Computer Technology Institute, where he served as a high-ranking military officer for years to come — according to an interview with a defector.

Cho soon became a decorated entity in North Korea, with Kim Jong-il awarding him for teaching him "the true value of computing" and ordering concentrated efforts to be given to promote research in the field. Cho's achievements may have been overly exaggerated by the regime, and while this alone had little-to-none impact on the outside, it led the children of the elite seeing Hacking as a desirable career path. With mottos such as: 'Developing viruses is the best way to secure important jobs' and 'cyber-warriors' enjoying perks such as luxury apartments, virus development was perceived as a road to power rather than a crime.

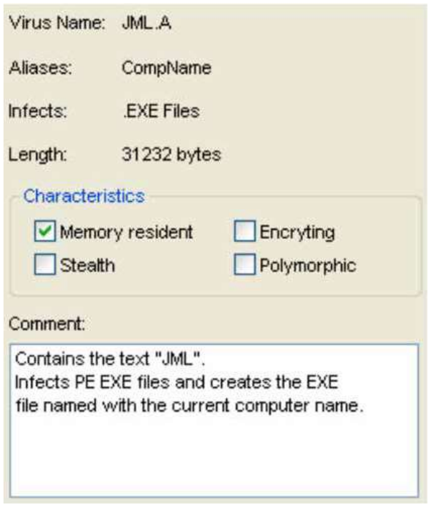
As for JML's functionality, it serves 2 purposes — one is spreading further via other executables on the local file system and network shares by scanning for files with the .exe extension, writing a piece of shellcode to them and redirecting execution to it by modifying the entry point. The shellcode will attempt to execute another malicious component and then resume execution to the original entry point. Another functionality allows looking for files in the infected machine and uploading them to a C2. Otherwise, further payload can be downloaded by it and executed. This is done via a predefined network protocol.

Around 2003, JML was discovered by "Ahn Cheol-soo" in South Korea. There exists some correlation between a file infector backdoor malware named Win32/Weird and JML[5]. Win32/Weird was first detected in July 1999[6]. AhnLab changed a couple detection names from 'Win32/JML' to 'Win32/Weird' in 2003. Win32/Weird.b detected in 2002, was first reported in South Korea and has not been detected elsewhere at the time of the writing (neither Symantec nor McAfee have profiles on Win32/Weird.b). It's worth noting that there's a possibility that the date 1997 is incorrect and Cho may have modified Win32/Weird to create JML.

With the rise of computers in North Korea reaching ca. 4 Million (counting both official and personal) computers distributed throughout the country in 2013[7], where half of them situated in Pyongyang, North Korea did not envision this rapid change and thus security had been loose.

The young computer students studying Cho's thesis were developing new mutations of the virus which propagated through the state's intranet, leading to North Korea sporadically being a target of accidental internal hacking attempts[8]. This may well be the reason for finding multiple signatures attempting to detect JML within KJAV as of the 2012 version.

Despite the above, we can see a signature for JML existing in SongSae back in July 2003. According to external resources[9] the JML virus spread outside of North Korea in early March 2003, and around that time we observed a signature for JML to exist only at AhnLab's Antivirus. This could also suggest that while writing SongSae, the developers tried to observe which signatures existed in its South Korean counterpart.



Virus Name: JML.A

Aliases: CompName

Infects: .EXE Files

Length: 31232 bytes

Characteristics

☑ Memory resident ☐ Encryting
☐ Stealth ☐ Polymorphic

Comment:

```
Contains the text "JML".
Infects PE EXE files and creates the EXE
file named with the current computer name.
```

Figure 11. JML description on SongSae

Another signature that stands out is one prefixed with the string "KjavCrack". As mentioned before, KJAV is licensed and therefore requires an activation key that is generated per device and is based on CPU & HDD related fields. As with a lot of other licensed programs, there will always be someone attempting to avoid paying the license fee and bypass its protection by cracking it or generating activation keys.

We've observed that KJAV contains a set of signatures looking for such scenarios, e.g. ***KjavCrack.****, ***KjavCrack.KeyGen.****, ***KjavCrack.Dll.****, ***KjavCrack.Exe.****, ***KjavCrack.Mem.****, ***KjavCrack.Rebuild.****. This begs the question — why? Could it be that cracking scene inside North Korea exists, posing a threat to KJAV's revenue? Surely such crime will be severely punished and is not worth saving the rather low monthly price of the license.

Taking this into consideration, there might be another plausible explanation. Although the appearance of KJAV outside of North Korea is scarce, it's apparent that an older version has leaked. It could very well be the case that it was cracked by an external entity and made its way back to North Korea at some point. In turn, the developers might have considered such cracks to be potential threat or just wanted to avoid any abuse of their intellectual property domestically.

Apart from the above, the signatures of KJAV cover popular worm infectors of that time such as Virut, Sality and Conficker. Considering the JML testimony, it may very well be that other file infectors circumvented inside North Korea's computer systems, either by studying and attempting to modify said malware in order to weaponize it or simply by accidently getting infected due to poor security practices.

Code Similarity

Another interesting point about KJAV concerns code similarity that was found between one of its components and pieces of malware, some of which are attributed to North Korea. By comparing the Antiviruses' binaries against a large set of malicious binaries, we were able to locate 2 main similarities — one is between a C function in KjavUpdate.exe and Lazarus affiliated malware named Manuscrypt, and another between an unknown and supposedly proprietary library function in KjavUpdate and a malware called KimJongRAT.

As for the former, it seems that both KjavUpdate.exe and Manuscrypt share a very similar function for RC4 decryption. While RC4 relies on a rather simple algorithm, its implementation in this case is pretty unique such that we couldn't trace it back to any open source repository. Namely, we see a very similar flow of code, whereby all of the RC4 decryption steps (i.e. S-Box creation, key scheduling algorithm and pseudo-random generation algorithm) are resident within a single function, creating identical call flow graphs.

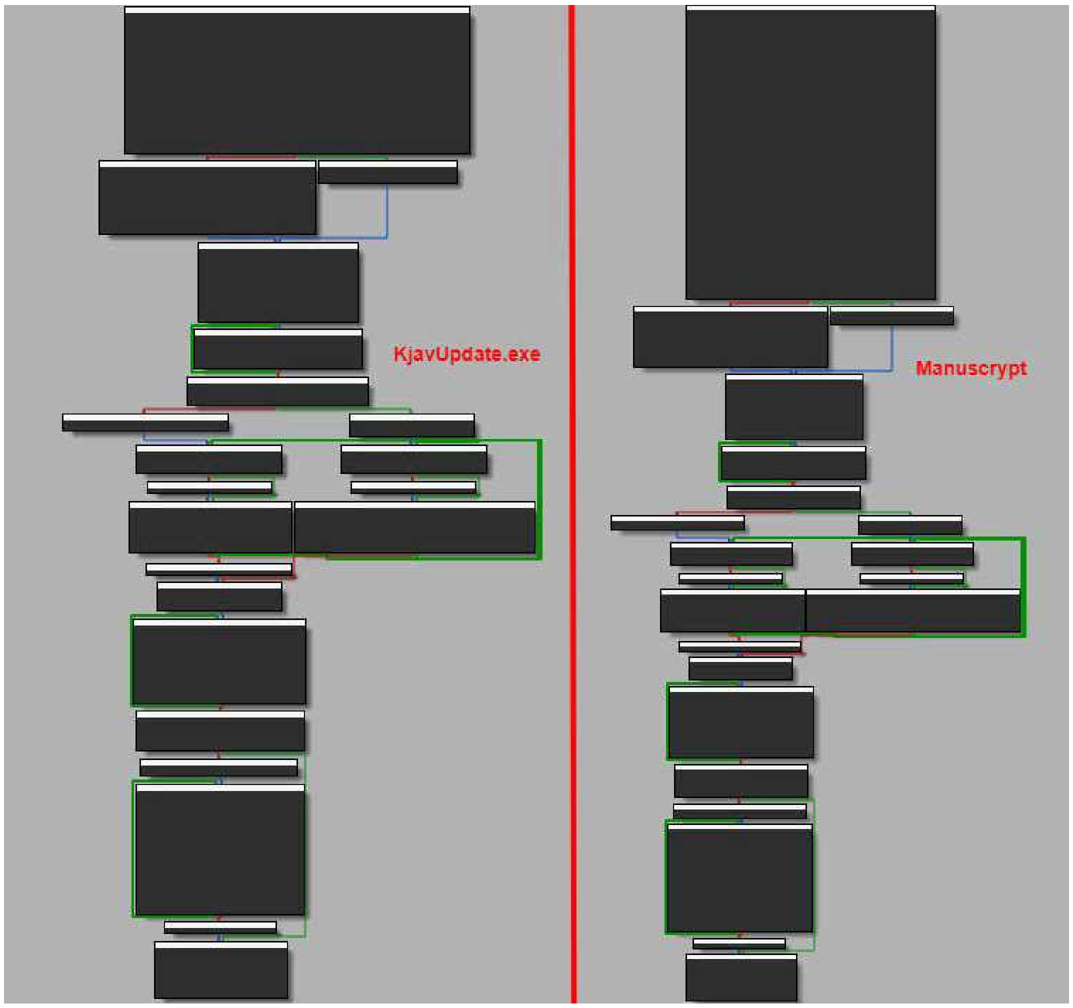
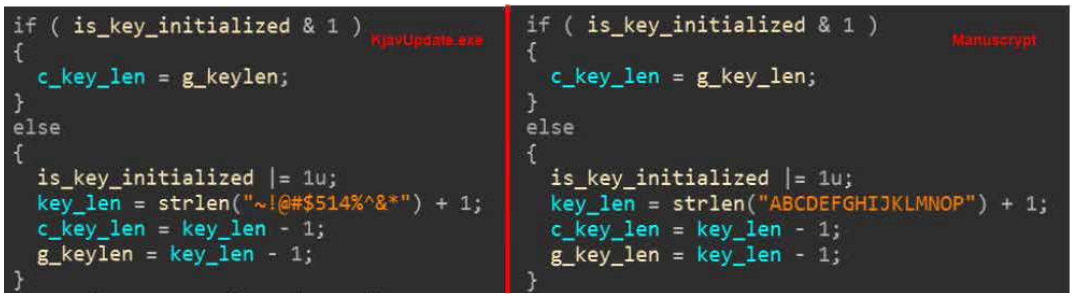


Figure 12. CFG similarity between KjavUpdate.exe and Manuscrypt's implementation of RC4

Apart from that there is usage of non-RC4 related fields in both functions. For example, both check for a global that indicates if an RC4 key was initialized. If it wasn't, a hardcoded fallback key is used. This is outlined in the figure below.

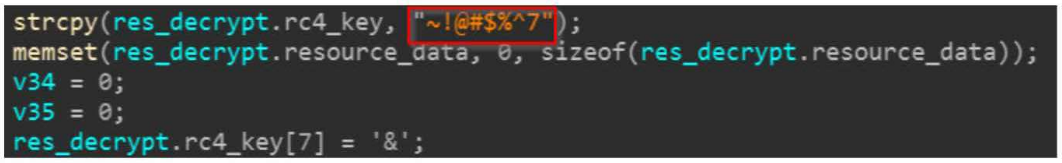


```
if ( is_key_initialized & 1 )
{
  c_key_len = g_keylen;
}
else
{
  is_key_initialized |= 1u;
  key_len = strlen("~!@#$514%^&*") + 1;
  c_key_len = key_len - 1;
  g_keylen = key_len - 1;
}
```

```
if ( is_key_initialized & 1 )
{
  c_key_len = g_key_len;
}
else
{
  is_key_initialized |= 1u;
  key_len = strlen("ABCDEFGHIJKLMNOP") + 1;
  c_key_len = key_len - 1;
  g_key_len = key_len - 1;
}
```

Figure 13. Usage of hardcoded RC4 fallback keys in KjavUpdate.exe and Manuscrypt

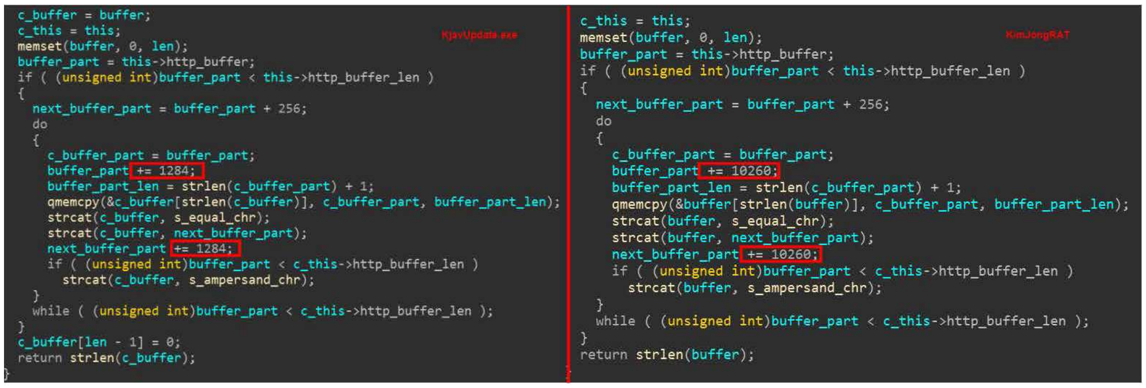
On the same note, we can spot a usage of a very similar RC4 key in another malware named KimJongRAT. This malware, which was initially discovered and analyzed[10] in 2013, was not officially attributed to North Korea until recently, when a fresh version of it was discovered in connection to the BabyShark malware, described[11] by Unit42. While the structure of the malware's key ('!@#$%^7', which is then modified to '~!@#$%^&') resembles that of *KjavUpdate.exe* ('~!@#$514%^&*'), the underlying RC4 implementation is different.



```
strcpy(res_decrypt.rc4_key, "~!@#$%^7");
memset(res_decrypt.resource_data, 0, sizeof(res_decrypt.resource_data));
v34 = 0;
v35 = 0;
res_decrypt.rc4_key[7] = '&';
```

Figure 14. RC4 key used in KimJongRAT, which is similar to the one used in KjavUpdate.exe

Still it's possible to find some code similarities between both binaries. In particular, we were able to spot overlaps in one C++ class which is in charge of HTTP communication. The implementation of this library component varies between those 2 instances, but still shares some similarities in its core functions as depicted in one example below.



```
c_buffer = buffer;
c_this = this;
memset(buffer, 0, len);
buffer_part = this->http_buffer;
if ( (unsigned int)buffer_part < this->http_buffer_len )
{
  next_buffer_part = buffer_part + 256;
  do
  {
    c_buffer_part = buffer_part;
    buffer_part += 1284;
    buffer_part_len = strlen(c_buffer_part) + 1;
    qmemcpy(&c_buffer[strlen(c_buffer)], c_buffer_part, buffer_part_len);
    strcat(c_buffer, s_equal_chr);
    strcat(c_buffer, next_buffer_part);
    next_buffer_part += 1284;
    if ( (unsigned int)buffer_part < c_this->http_buffer_len )
      strcat(c_buffer, s_ampersand_chr);
  }
  while ( (unsigned int)buffer_part < c_this->http_buffer_len );
}
c_buffer[len - 1] = 0;
return strlen(c_buffer);
}
```

```
c_this = this;
memset(buffer, 0, len);
buffer_part = this->http_buffer;
if ( (unsigned int)buffer_part < this->http_buffer_len )
{
  next_buffer_part = buffer_part + 256;
  do
  {
    c_buffer_part = buffer_part;
    buffer_part += 10260;
    buffer_part_len = strlen(c_buffer_part) + 1;
    qmemcpy(&buffer[strlen(buffer)], c_buffer_part, buffer_part_len);
    strcat(buffer, s_equal_chr);
    strcat(buffer, next_buffer_part);
    next_buffer_part += 10260;
    if ( (unsigned int)buffer_part < c_this->http_buffer_len )
      strcat(buffer, s_ampersand_chr);
  }
  while ( (unsigned int)buffer_part < c_this->http_buffer_len );
}
return strlen(buffer);
```

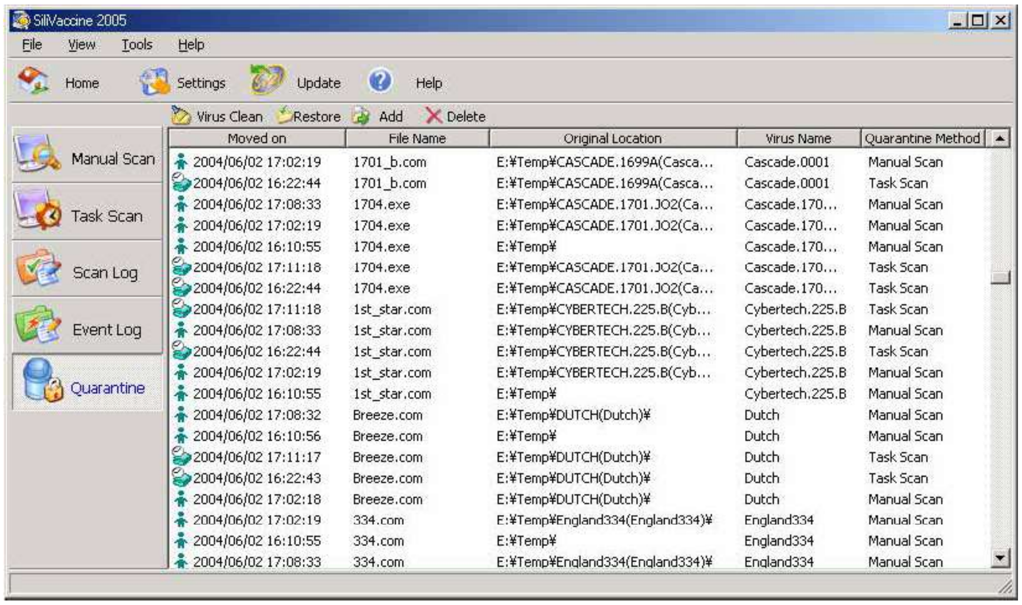
Figure 15. Code similarity between two (supposedly) proprietary library functions in KimJongRAT and KjavUpdate.exe. Only difference appears to be in a referenced chunk size within an internal buffer

Silivaccine

Silivaccine is yet another Antivirus produced in North Korea, which we could observe through 2 versions — one was released in 2005 (v2.0.0.1) and another in 2013 (v5.0). Both versions were created by "STS Tech Service" (also named 626 or Silibank[12]), which is a financial institution existing since 2001, believed[13] to work from the Chilbosan Hotel in Shenyang. An interesting caveat is that this is reportedly the base of operations for North Korea's Unit 121 (Electronic Reconnaissance Bureau's Cyber Warfare Guidance Bureau) which hand-picks hackers graduating from Mirim University, according to reports[14] by defectors and the US military.

The 2013 version was also being developed in conjunction with an entity named PGI (Pyonyang Gwangmyong Information Technology), the same entity that collaborated on KJAV.

Since the 2013 version was a subject of a thorough research done by Check Point in 2018[15], we will not reiterate those but mention the main findings — the fact Silivaccine used a ripped-off Trend Micro engine (vsapi32.dll) and the fact it was done in a way that suggests the authors might have had access to Trend Micro proprietary resources when writing their version of the engine.

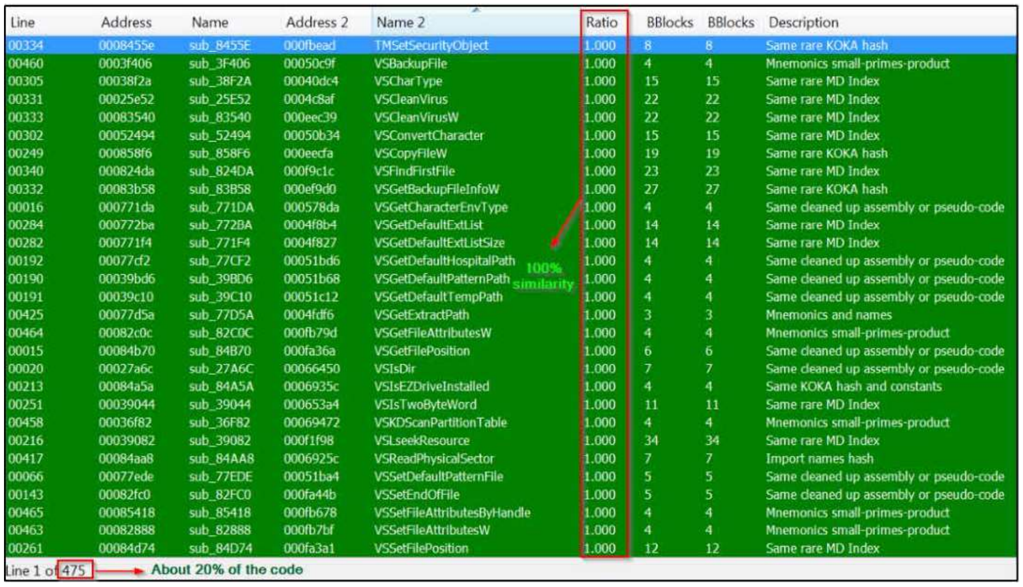


Moved on	File Name	Original Location	Virus Name	Quarantine Method
2004/06/02 17:02:19	1701_b.com	E:¥Temp¥CASCADE.1699A(Casca...	Cascade.0001	Manual Scan
2004/06/02 16:22:44	1701_b.com	E:¥Temp¥CASCADE.1699A(Casca...	Cascade.0001	Task Scan
2004/06/02 17:08:33	1704.exe	E:¥Temp¥CASCADE.1701.JO2(Ca...	Cascade.170...	Manual Scan
2004/06/02 17:02:19	1704.exe	E:¥Temp¥CASCADE.1701.JO2(Ca...	Cascade.170...	Manual Scan
2004/06/02 16:10:55	1704.exe	E:¥Temp¥	Cascade.170...	Manual Scan
2004/06/02 17:11:18	1704.exe	E:¥Temp¥CASCADE.1701.JO2(Ca...	Cascade.170...	Task Scan
2004/06/02 16:22:44	1704.exe	E:¥Temp¥CASCADE.1701.JO2(Ca...	Cascade.170...	Task Scan
2004/06/02 17:11:18	1st_star.com	E:¥Temp¥CYBERTECH.225.B(Cyb...	Cybertech.225.B	Task Scan
2004/06/02 17:08:33	1st_star.com	E:¥Temp¥CYBERTECH.225.B(Cyb...	Cybertech.225.B	Manual Scan
2004/06/02 16:22:44	1st_star.com	E:¥Temp¥CYBERTECH.225.B(Cyb...	Cybertech.225.B	Task Scan
2004/06/02 17:02:19	1st_star.com	E:¥Temp¥CYBERTECH.225.B(Cyb...	Cybertech.225.B	Manual Scan
2004/06/02 16:10:55	1st_star.com	E:¥Temp¥	Cybertech.225.B	Manual Scan
2004/06/02 17:08:32	Breeze.com	E:¥Temp¥DUTCH(Dutch)¥	Dutch	Manual Scan
2004/06/02 16:10:56	Breeze.com	E:¥Temp¥	Dutch	Manual Scan
2004/06/02 17:11:17	Breeze.com	E:¥Temp¥DUTCH(Dutch)¥	Dutch	Task Scan
2004/06/02 16:22:43	Breeze.com	E:¥Temp¥DUTCH(Dutch)¥	Dutch	Task Scan
2004/06/02 17:02:18	Breeze.com	E:¥Temp¥DUTCH(Dutch)¥	Dutch	Manual Scan
2004/06/02 17:02:19	334.com	E:¥Temp¥England334(England334)¥	England334	Manual Scan
2004/06/02 16:10:55	334.com	E:¥Temp¥	England334	Manual Scan
2004/06/02 17:08:33	334.com	E:¥Temp¥England334(England334)¥	England334	Manual Scan

Figure 16. Main GUI window for Silivaccine 2005

The 2005 version shares a lot of similarities with its newer version, more specifically a lot of the components are analogous (e.g. SVUpdate.exe, SVDealer, SVTray.exe, SVShell.exe etc.) and the architecture is somewhat similar. Much like its successor, v2.0.0.1 used a Trend Micro engine, only this one appeared as a driver named SVKernel.sys which was based on a Trend Micro driver named vsapint.sys.

Once again it is possible to see that there is 100% similarity in a great number of functions (~500, which is about 20% of Silivaccine's engine code) between the two engines, while a lot of the rest being proprietary North Korean code. This suggests that as in the 2013 version, the developers didn't use the engine "as is" but rather rewrote their own version using parts of the original.

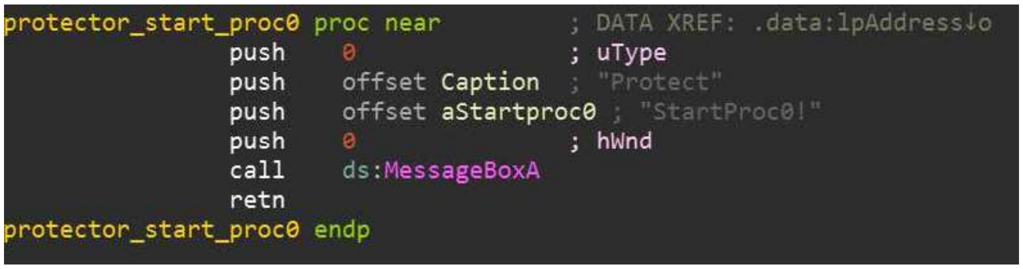


Line	Address	Name	Address 2	Name 2	Ratio	BBlocks	BBlocks	Description
00334	0008455e	sub_8455E	000fbead	TMSetSecurityObject	1.000	8	8	Same rare KOKA hash
00460	0003f406	sub_3F406	00050c9f	VSBackupFile	1.000	4	4	Mnemonics small-primes-product
00305	00038f2a	sub_38F2A	00040dc4	VSCharType	1.000	15	15	Same rare MD Index
00331	00025e52	sub_25E52	0004c8af	VSCleanVirus	1.000	22	22	Same rare MD Index
00333	00083540	sub_83540	000eec39	VSCleanVirusW	1.000	22	22	Same rare MD Index
00302	00052494	sub_52494	00050b34	VSConvertCharacter	1.000	15	15	Same rare MD Index
00249	000858f6	sub_858F6	000eecfa	VSCopyFileW	1.000	19	19	Same rare KOKA hash
00340	000824da	sub_824DA	000f9c1c	VSFindFirstFile	1.000	23	23	Same rare MD Index
00332	00083b58	sub_83B58	000ef9d0	VSGetBackupFileInfoW	1.000	27	27	Same rare KOKA hash
00016	000771da	sub_771DA	000578da	VSGetCharacterEnvType	1.000	4	4	Same cleaned up assembly or pseudo-code
00284	000772ba	sub_772BA	0004f8b4	VSGetDefaultExtList	1.000	14	14	Same rare MD Index
00282	000771f4	sub_771F4	0004f827	VSGetDefaultExtListSize	1.000	14	14	Same rare MD Index
00192	00077cf2	sub_77CF2	00051bd6	VSGetDefaultHospitalPath	1.000	4	4	Same cleaned up assembly or pseudo-code
00190	00039bd6	sub_39BD6	00051b68	VSGetDefaultPatternPath	1.000	4	4	Same cleaned up assembly or pseudo-code
00191	00039c10	sub_39C10	00051c12	VSGetDefaultTempPath	1.000	4	4	Same cleaned up assembly or pseudo-code
00425	00077d5a	sub_77D5A	0004fdf6	VSGetExtractPath	1.000	3	3	Mnemonics and names
00464	00082c0c	sub_82C0C	000fb79d	VSGetFileAttributesW	1.000	4	4	Mnemonics small-primes-product
00015	00084b70	sub_84B70	000fa36a	VSGetFilePosition	1.000	6	6	Same cleaned up assembly or pseudo-code
00020	00027a6c	sub_27A6C	00066450	VSIsDir	1.000	7	7	Same cleaned up assembly or pseudo-code
00213	00084a5a	sub_84A5A	0006935c	VSIsEZDriveInstalled	1.000	4	4	Same KOKA hash and constants
00251	00039044	sub_39044	000653a4	VSIsTwoByteWord	1.000	11	11	Same rare MD Index
00458	00036f82	sub_36F82	00069472	VSKDScanPartitionTable	1.000	4	4	Mnemonics small-primes-product
00216	00039082	sub_39082	000f1f98	VSLseekResource	1.000	34	34	Same rare MD Index
00417	00084aa8	sub_84AA8	0006925c	VSReadPhysicalSector	1.000	7	7	Import names hash
00066	00077ede	sub_77EDE	00051ba4	VSSetDefaultPatternFile	1.000	5	5	Same cleaned up assembly or pseudo-code
00143	00082fc0	sub_82FC0	000fa44b	VSSetEndOfFile	1.000	5	5	Same cleaned up assembly or pseudo-code
00465	00085418	sub_85418	000fb678	VSSetFileAttributesByHandle	1.000	4	4	Mnemonics small-primes-product
00463	00082888	sub_82888	000fb7bf	VSSetFileAttributesW	1.000	4	4	Mnemonics small-primes-product
00261	00084d74	sub_84D74	000fa3a1	VSSetFilePosition	1.000	12	12	Same rare MD Index

Line 1 of 475 → About 20% of the code

Figure 17. Code similarity between vsapint.sys driver from Trend Micro (right columns — Address 2\ Name 2) and Silivaccine's SVKernel.sys

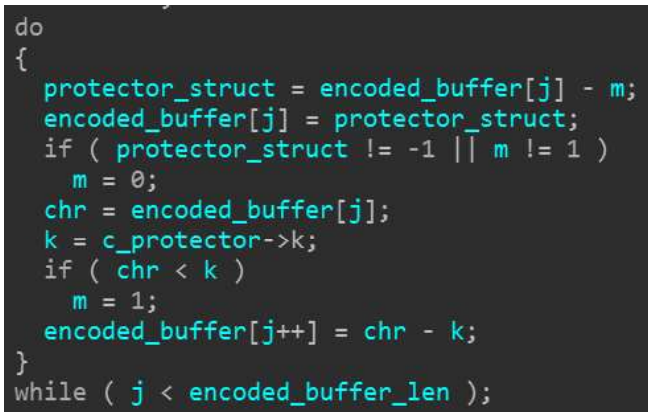
The 2013 version had some features that could not be found in the 2005 version. One of them is the way some of the binaries are packed. While in 2013 there was usage of Themida to protect various Antivirus components, the earlier version used a much simpler protector. That protector seems to be a homebrewed one, which would encode functions chosen by the developer at compile time. This is evident by seeing each encoded part preceded with a simple stub function that pops a message box with the string "`Start Proc%d!`" (where %d would be some integer value), and followed by a similar stub with a string of the format "`EndProc%d!`".



```
protector_start_proc0 proc near          ; DATA XREF: .data:lpAddress↓o
                push    0                ; uType
                push    offset Caption   ; "Protect"
                push    offset aStartproc0 ; "StartProc0!"
                push    0                ; hWnd
                call    ds:MessageBoxA
                retn
protector_start_proc0 endp
```

Figure 18. Stub that precedes each protected function in Silivaccine 2005

The algorithm for decoding each piece of code is quite straightforward and incomparable to Themida's level of obfuscation. It is by no means an encryption algorithm (as there is no key) and can be easily decoded given nothing but the encoded blob. The decoding function is being called during startup, i.e. invoked from the binary's entry point, and it has the logic outlined in the figure below. In this sense, Silivaccine has made tremendous progress between 2005 and 2013.



```
do
{
  protector_struct = encoded_buffer[j] - m;
  encoded_buffer[j] = protector_struct;
  if ( protector_struct != -1 || m != 1 )
    m = 0;
  chr = encoded_buffer[j];
  k = c_protector->k;
  if ( chr < k )
    m = 1;
  encoded_buffer[j++] = chr - k;
}
while ( j < encoded_buffer_len );
```

Figure 19. Routine used for decoding each protected function in Silivaccine 2005

Apart from that, it's interesting to note that some features existed in the early version but were removed later on. A notable component that existed in Silivaccine 2005 is called SVCharge (comprised of an executable and a DLL with the same name) which is intended to provide a real-time scan functionality for mail attachments as well as files downloaded from the internet.

This was achieved by setting up a window hook of type WH_CBT[16] from ***SVCharge.dll***, which would get invoked by the system before various window events (e.g. creating, closing, minimizing, maximizing a window etc.). When the hook function runs, it would execute in the context of the window's process, hence it can query the process image name and see if it corresponds to that of Outlook \ Outlook Express or Internet Explorer.

In either one of these cases the window hook will check and load the SVCharge.dll to the process (if not already loaded), and then install a hook function for WSOCK32.dll's '***connect***' function. When connect's hook is invoked, it checks the function arguments to verify if the outgoing port is 80 or 110 (HTTP or POP3 accordingly), and in each case gathers some information on the connection. This gets sent via a window message to the SVCharge window, which runs in the context of ***SVCharge.exe***. The latter can intercept the connection via a POP3 local server or a proxy server set-up for inspection beforehand.

This functionality is interesting mainly for the fact that it's intended to intercept mail and web traffic and due to the fact that the underlying engine and signatures come from Trend Micro, which would mostly detect malware that is common outside of the DPRK. It makes it plausible to assume that the threats caught by such real time monitoring would be ones found in internet traffic, rather than in the home based intranet.

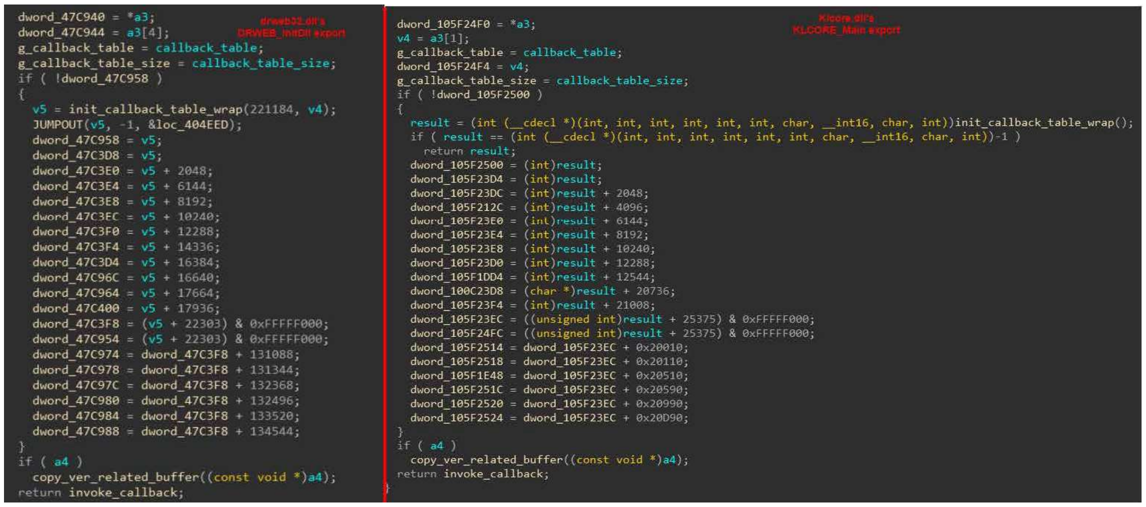
In this case it would raise the question as to who might use this software, given that access to the internet in North Korea is highly restricted. As it is assumed that users with unlimited access to the internet are high ranking officials, members of NGOs or government ambassadors, we can speculate that Silivaccine is intended for 'enterprise' use (e.g. by universities, military and other government institutions).

Kulak

Kulak is probably the longest standing Antivirus in the North Korean industry. Developed by Kim Il Sung University and the Ministry of People's Security (North Korea LEA), Kulak started its way as a DOS based software, which quickly evolved into a Windows based one. Specifically, we were able to trace 3 versions released in 2002, 2007 and 2012, both from openly-available resources and actual installers in our possession.

One of the main observations that can be made with regards to Kulak is that it's more advanced than its counterparts in architecture and sophistication, and seems to be overall a more solid security product. This partially reminds of Silivaccine, which is also quite complex at a glance, as having a Trend Micro based engine is by far more advanced than those of Songsae and KJAV.

Considering this, it may not come as a surprise that Kulak also makes use of a third party engine, at least in its 2012 version. In this case, the abused engine is that of Dr.Web, a long standing and prominent manufacturer of security software from Russia. Kulak's counterpart engine was named Klcore.dll, and by tracking particular functions which provide metadata on the engine, it's possible to conclude that it's based on version 4.29 of the *drweb32.dll* library.



```
dword_47C940 = *a3;
dword_47C944 = a3[4];
g_callback_table = callback_table;
g_callback_table_size = callback_table_size;
if ( !dword_47C958 )
{
  v5 = init_callback_table_wrap(221184, v4);
  JUMPOUT(v5, -1, &loc_404EED);
  dword_47C958 = v5;
  dword_47C3D8 = v5;
  dword_47C3E0 = v5 + 2048;
  dword_47C3E4 = v5 + 6144;
  dword_47C3E8 = v5 + 8192;
  dword_47C3EC = v5 + 10240;
  dword_47C3F0 = v5 + 12288;
  dword_47C3F4 = v5 + 14336;
  dword_47C3D4 = v5 + 16384;
  dword_47C96C = v5 + 16640;
  dword_47C964 = v5 + 17664;
  dword_47C400 = v5 + 17936;
  dword_47C3F8 = (v5 + 22303) & 0xFFFFF000;
  dword_47C954 = (v5 + 22303) & 0xFFFFF000;
  dword_47C974 = dword_47C3F8 + 131088;
  dword_47C978 = dword_47C3F8 + 131344;
  dword_47C97C = dword_47C3F8 + 132368;
  dword_47C980 = dword_47C3F8 + 132496;
  dword_47C984 = dword_47C3F8 + 133520;
  dword_47C988 = dword_47C3F8 + 134544;
}
if ( a4 )
  copy_ver_related_buffer((const void *)a4);
return invoke_callback;
```

```
dword_105F24F0 = *a3;
v4 = a3[1];
g_callback_table = callback_table;
dword_105F24F4 = v4;
g_callback_table_size = callback_table_size;
if ( !dword_105F2500 )
{
  result = (int (__cdecl *)(int, int, int, int, int, int, char, __int16, char, int))init_callback_table_wrap();
  if ( result == (int (__cdecl *)(int, int, int, int, int, int, char, __int16, char, int))-1 )
    return result;
  dword_105F2500 = (int)result;
  dword_105F23D4 = (int)result;
  dword_105F23DC = (int)result + 2048;
  dword_105F212C = (int)result + 4096;
  dword_105F23E0 = (int)result + 6144;
  dword_105F23E4 = (int)result + 8192;
  dword_105F23E8 = (int)result + 10240;
  dword_105F23D0 = (int)result + 12288;
  dword_105F1DD4 = (int)result + 12544;
  dword_100C23D8 = (char *)result + 20736;
  dword_105F23F4 = (int)result + 21008;
  dword_105F23EC = ((unsigned int)result + 25375) & 0xFFFFF000;
  dword_105F24FC = ((unsigned int)result + 25375) & 0xFFFFF000;
  dword_105F2514 = dword_105F23EC + 0x20010;
  dword_105F2518 = dword_105F23EC + 0x20110;
  dword_105F1E48 = dword_105F23EC + 0x20510;
  dword_105F251C = dword_105F23EC + 0x20590;
  dword_105F2520 = dword_105F23EC + 0x20990;
  dword_105F2524 = dword_105F23EC + 0x20D90;
}
if ( a4 )
  copy_ver_related_buffer((const void *)a4);
return invoke_callback;
}
```

Figure 20. Comparison of Dr.Web's and Kulak's engine initialization functions

Another prominent similarity was witnessed between a Dr.Web user mode component named spidercpl.exe (referred to as SpIder Guard) and a Kulak driver named Klhook.exe (internally referenced as Kulak Protect). The former serves as a real time monitoring component, intended to scan process memory and opened files upon access. It's Kulak counterpart is apparently made to make the same, achieving it via the kernel space. Following are a few examples for some of the similarities between them.

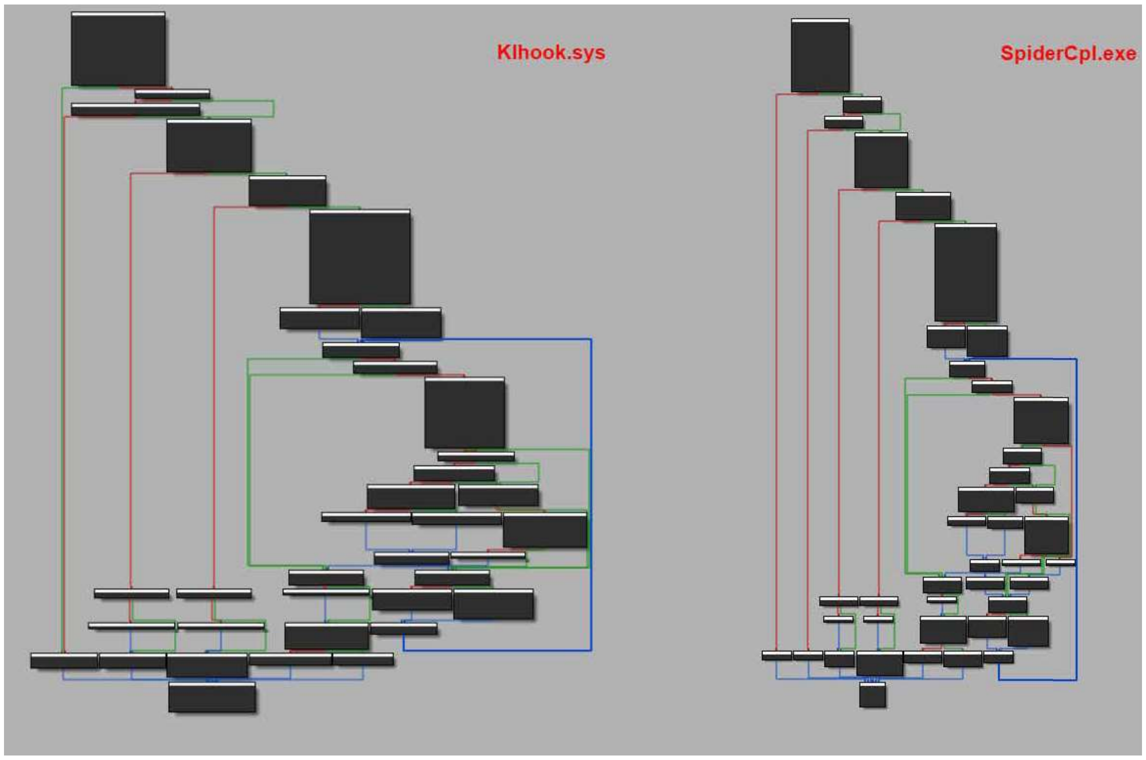
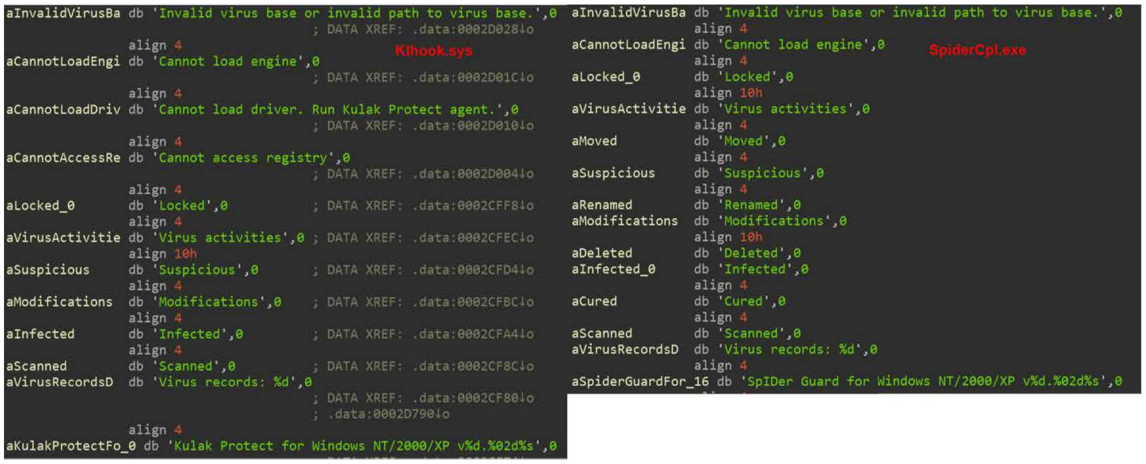


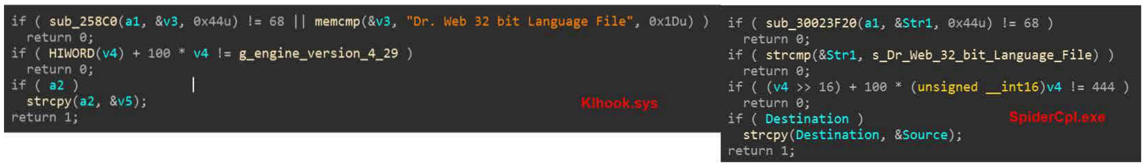
Figure 21. Comparison of a proprietary function call flow graph in SpiderCpl and Klhook



```
aInvalidVirusBa db 'Invalid virus base or invalid path to virus base.',0
                                        ; DATA XREF: .data:0002D028↓o
                align 4
aCannotLoadEngi db 'Cannot load engine',0
                                        ; DATA XREF: .data:0002D01C↓o
                align 4
aCannotLoadDriv db 'Cannot load driver. Run Kulak Protect agent.',0
                                        ; DATA XREF: .data:0002D010↓o
                align 4
aCannotAccessRe db 'Cannot access registry',0
                                        ; DATA XREF: .data:0002D004↓o
                align 4
aLocked_0       db 'Locked',0           ; DATA XREF: .data:0002CFF8↓o
                align 4
aVirusActivitie db 'Virus activities',0 ; DATA XREF: .data:0002CFEC↓o
                align 10h
aSuspicious     db 'Suspicious',0       ; DATA XREF: .data:0002CFD4↓o
                align 4
aModifications  db 'Modifications',0    ; DATA XREF: .data:0002CFBC↓o
                align 4
aInfected       db 'Infected',0         ; DATA XREF: .data:0002CFA4↓o
                align 4
aScanned        db 'Scanned',0          ; DATA XREF: .data:0002CF8C↓o
aVirusRecordsD  db 'Virus records: %d',0
                                        ; DATA XREF: .data:0002CF80↓o
                                        ; .data:0002D790↓o
                align 4
aKulakProtectFo_0 db 'Kulak Protect for Windows NT/2000/XP v%d.%02d%s',0
```

```
aInvalidVirusBa db 'Invalid virus base or invalid path to virus base.',0
                align 4
aCannotLoadEngi db 'Cannot load engine',0
                align 4
aLocked_0       db 'Locked',0
                align 10h
aVirusActivitie db 'Virus activities',0
                align 4
aMoved          db 'Moved',0
                align 4
aSuspicious     db 'Suspicious',0
                align 4
aRenamed        db 'Renamed',0
aModifications  db 'Modifications',0
                align 10h
aDeleted        db 'Deleted',0
aInfected_0     db 'Infected',0
                align 4
aCured          db 'Cured',0
                align 4
aScanned        db 'Scanned',0
aVirusRecordsD  db 'Virus records: %d',0
                align 4
aSpiderGuardFor_16 db 'SpIDer Guard for Windows NT/2000/XP v%d.%02d%s',0
```

Figure 22. Comparison of some used strings in SpiderCpl and Klhook. As we can see in the Klhook sys, some are changed from 'SpIder Guard' to 'Kulak Protect'

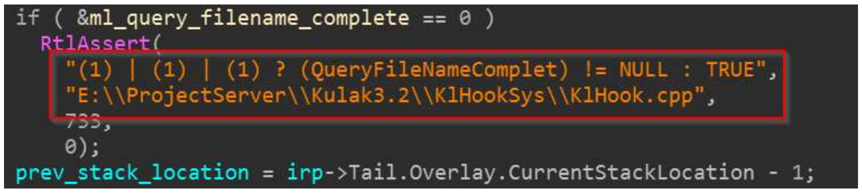


```
if ( sub_258C0(a1, &v3, 0x44u) != 68 || memcmp(&v3, "Dr. Web 32 bit Language File", 0x1Du) )
  return 0;
if ( HIWORD(v4) + 100 * v4 != g_engine_version_4_29 )
  return 0;
if ( a2 )
  strcpy(a2, &v5);
return 1;
```

```
if ( sub_30023F20(a1, &Str1, 0x44u) != 68 )
  return 0;
if ( strcmp(&Str1, s_Dr_Web_32_bit_Language_File) )
  return 0;
if ( (v4 >> 16) + 100 * (unsigned __int16)v4 != 444 )
  return 0;
if ( Destination )
  strcpy(Destination, &Source);
return 1;
```

Figure 23. Function checking for a signature file string artifact in SpiderCpl.exe and Klhook.sys. Note that Kulak is actually looking for a Dr.Web string

Like in the case of Silivaccine, we are able to witness some evidence pointing towards the fact that the engine was built using a proprietary resource. In this case, we can look at calls to RtlAssert to see the path from which the driver was compiled. It is evident that the underlying source file belongs to Kulak.



```
if ( &ml_query_filename_complete == 0 )
  RtlAssert(
    "(1) | (1) | (1) ? (QueryFileNameComplet) != NULL : TRUE",
    "E:\\ProjectServer\\Kulak3.2\\KlHookSys\\KlHook.cpp",
    733,
    0);
prev_stack_location = irp->Tail.Overlay.CurrentStackLocation - 1;
```

Figure 24. Source code path that was used to compile Klhook.sys, according to an RtlAssert string argument

As for the question on how the Dr.Web engine and monitor components found their way into Kulak, there are several speculations that come to mind. For instance, it's possible that a Dr.Web technology which was previously licensed to a South Korean company named NWI was leaked in some form to the North.

At the same time, it's hard to explain how SpIder Guard code got incorporated into a driver that was supposedly compiled from a North Korean code base. In that case, it is possible that there was some leak of proprietary resources belonging to Dr.Web that could aid the North Korean developers when building this component.

Either way, we can't indicate what happened in reality without a shadow of a doubt. Instead, it is worth noting that whatever the scenario might have been, Kulak provides more evidence that North Korea is not entirely independent when creating their own programs, but in some cases is heavily reliant on foreign produce.

Summary

Within a decade we observed 5 unique Antivirus solutions being developed by at least 6 entities — some seem to be educational facilities (e.g. Kim Il Sung, Chaek and Mirim Universities), others appear as government owned entities (e.g. Ministry of People's Security, PGI and STS Tech-Service). Such a great investment in an internal security product may not be surprising given the fact that the country is heavily sanctioned and limited in resources. This is exactly what is portrayed in DPRK's own ideology — Juche, in which the main values are unity, independence and self-reliance.

In spite of this, we can see that these values are not reflected from the products that we analyzed. In a few instances we observed usage of 3rd party engines and signatures belonging to big vendors who claim to have no formal ties to North Korea. This suggests that the DPRK is in more need of foreign produce and financial connections than it is willing to admit

In terms of progress, it is possible to observe an evolution in the development capabilities of North Korean programmers. While the earlier AVs had a rather simple architecture and often bulky code, the later products seem more orderly and rich in features. This is no surprise, especially in cases in which products were built around an external engine. This required a great deal of attention to the engine's internals so that the in-house built components will properly inter-operate with it.

On the same note, we could witness a development in software protections used to burden the analysis of the Antiviruses' components. The first product we described — SongSae version 2003, contained no protection whatsoever, while Silivaccine 2005 was already protected by a rudimentary homebrew protector. Starting from at least 2012 (with KJAV 5.0) all products were protected using Themida, which introduced a fairly high level obstacle when attempting to reverse engineer them.

Analysing these Themida packers we observed two licenses in use. One was shared between KJAV 5.0, 5.8 and Kulak 3.2, and the other used in Silivaccine 2012. This reuse of software may either suggest that the North Korean developers referred to same resources to obtain it or possibly cooperated to some extent when writing the products.

Furthermore, hunting for other programs using these licenses, we could find both legitimate software as well as more questionable ones (e.g. Chinese phone tools, games and game NoCD patches). We conclude that these licenses were leaked by looking at the detection rate of other packed programs, since once a license leaks "Antivirus companies will blacklist that Taggant signature" — according to 'Oreans', the company behind Themida.

To sum things up, we will note that this research is still a work in progress. As Antivirus software contains a lot of code, there are still a lot of details and information to go through. These could provide further insights on top of the ones presented in this paper. We intend to keep investigating the software we gathered so far, with the hope of obtaining new ones to expand our understanding of the subject. Any fresh observations and expanded detail will hopefully be published in separate documents.