Annual ADFSL Conference on Digital Forensics, Security and Law

2013
Proceedings

Jun 12th, 9:25 AM

A Forensic Study of the Effectiveness of Selected Anti-Virus Products Against SSDT Hooking Rootkits

Sami Al-Shaheri
Department of Faculty of Professional Education, Concordia University College of Alberta, al-shaheri_sami@hotmail.com

Dale Lindskog
Department of Faculty of Professional Education, Concordia University College of Alberta, dale.lindskog@concordia.ab.ca

Pavol Zavarsky
Department of Faculty of Professional Education, Concordia University College of Alberta, pavol.zavarsky@concordia.ab.ca

Ron Ruhl
Department of Faculty of Professional Education, Concordia University College of Alberta, ron.ruhl@concordia.ab.ca

A FORENSIC STUDY OF THE EFFECTIVENESS OF SELECTED ANTI-VIRUS PRODUCTS AGAINST SSDT HOOKING ROOTKITS

Sami Al-Shaheri (Al-shaheri_sami@hotmail.com)
Dale Lindskog (dale.lindskog@concordia.ab.ca)
Pavol Zavarsky (pavol.zavarsky@concordia.ab.ca)
Ron Ruhl (ron.ruhl@concordia.ab.ca)

Department of Faculty of Professional Education
Concordia University College of Alberta
Edmonton AB T5B 4E4
Canada

ABSTRACT

For Microsoft Windows Operating Systems, both anti-virus products and kernel rootkits often hook the System Service Dispatch Table (SSDT). This research paper investigates the interaction between these two in terms of the SSDT. To investigate these matters, we extracted digital evidence from volatile memory, and studied that evidence using the Volatility framework. Due to the diversity in detection techniques used by the anti-virus products, and the diversity of infection techniques used by rootkits, our investigation produced diverse results, results that helped us to understand several SSDT hooking strategies, and the interaction between the selected anti-virus products and the rootkit samples.

Keywords: System Service Dispatch Table (SSDT), Anti-virus, Rootkits, Memory Analysis, Volatility

1. INTRODUCTION

SSDT hooking is a prevalent method employed by some security tools, in order to set restrictions on accessing a system's resources [12]. For example, anti-virus products often hook the SSDT in order to scan the newly launched program [6][7]. Anti-virus products usually achieve this hooking by altering addresses stored in the Native SSDT functions, causing them to point to the anti-virus' routines. The anti-virus software then checks and verifies the system call source, blocking all suspicious calls, but otherwise invokes the SSDT functions without any changes to the system call [15]. Rootkits usually do something similar. In either case, knowing the table address of the SSDT is required in order to index the target functions, and to perform the SSDT hooks.

SSDT is "write-protected in Windows XP and later version of Windows" and that the "write protect (WP) bit in the CR0 control register" [7]. Thus, in order to perform the SSDT hooking, some rootkits modify the protection of the SSDT as a first step of attacking the SSDT, by clearing specific bits of the control CR0 register [8][4]. In [3], the authors illustrate two techniques for disabling SSDT write protection, and also note that "to subvert the write protection on the SSDT, we need to temporarily clear the WP flag" [3]. Rootkits usually use the function *NtDeleteValueKey* to change the value of a registry key, in order to modify the SSDT's protection. Rootkit developers use several SSDT hooking methods in order to compromise processes and system files, or to modify records in the SSDT, causing it to point to the rootkit itself [3][13][4].

Volatility is a powerful framework that can be used to investigate SSDT hooks. Volatility uses thrdscan to scan ETHREAD objects and thus to detect when rootkits make copies of the existing

SSDTs and assign them to a particular thread such as ETHREAD.Tcb [24]. Volatility also uses the ssdt_ex plugin and the HookedSSDT tags to determine which SSDT functions are hooked [13][24][1].

In this research paper we systematically investigate the way in which representative anti-virus software and kernel rootkits interact with the SSDT, and with each other in terms of the SSDT. Our experimentation was conducted in four stages. In the first stage, we explored SSDT hooking by anti-virus products, independently of any SSDT hooking by the rootkits. Then we studied rootkits independently of anti-virus products. Next, we investigated the effects of SSDT hooking rootkits in an antivirus protected environment. Finally, we investigated machines that had been infected with a rootkit which an anti-virus product was attempting to clean. Our results show that there is a broad range in the effectiveness of anti-virus products in their ability to protect against a rootkit's strategic SSDT hooking techniques, and that a deep and current understanding of that strategy is essential to anti-virus development.

In the next section we review related research. In the third section, we describe our experimental methodology and results. Section 4 contains our analysis of those results, and based on this, our recommendations. Section 5 is our conclusion.

2. REVIEW OF RELATED RESEARCH

In 2010, the Matousec.com team conducted a study to determine whether and how anti-virus products can be evaded. They tested 35 anti-virus products, and found that every anti-virus product in that sample which implements SSDT hooks was vulnerable, including big names such as Kaspersky Internet Security and Norton Internet Security [15]. Even subsequent to disclosure [16], Matousec.com found that only some of the anti-virus software developers had fixed their vulnerabilities. They developed what they call the Kernel Hook Bypassing Engine (KHOBE) attack, which allows the malicious codes to bypass the anti-virus's protection mechanisms [15]. KHOBE has two components the 'attacker' which attempts to invoke the system service; and a 'faker thread' which attempts to modify the parameters such as CLIENT_ID. If the modification occurs after the security check by the anti-virus and before the original service gets invoked, the attacker will invoke the service, and the bypass attack will be successful [15]. The Matousec.com research illustrated a combined attack where KHOBE uses three components attacker and two faker threads, in this case the attacker needs a scheduler to switch between the faker threads [15].

Matousec.com did not investigate SSDT hooking methods from a forensics perspective, nor did they study computer memory in order to verify their claim and provide evidences. While Matousec.com implemented an attack to demonstrate their claims about the vulnerabilities of SSDT hooking by anti-virus products, Corregedor and Solms implemented two rootkits that "could collectively disable antimalware programs" [4]. The first rootkit was designed to sabotage a Windows OS, and the second to disable antimalware programs. The paper discusses SSDT hooking with a focus on the rootkit's effect on the KeServiceDescriptorTable. However, there are four System Service Dispatch Tables, and other rootkits have different techniques; for example, the rootkit Blackenergy sometimes uses more than one table [2][10]. Furthermore, the authors (Corregedor and Solms) stated that there are "no other papers that [specifically explore] how rootkits are implemented", and they requested further efforts investigating additional malware to gain greater knowledge of how rootkits work [4]. Similarly to Matousec.com's research, Corregedor and Solms also demonstrated that SSDT hooking by anti-virus products is vulnerable to manipulation by rootkits. Whereas Matousec.com conducted an attack to verify this, Corregedor and Solms demonstrated the steps required to implement these two rootkits. In both cases, a forensics investigation analyzing the environment of the attack was not conducted.

Hsu et al. developed a rootkit that they called "antivirus terminator" [9]. They proposed a mechanism, called ANtivirus Software Shield (ANSS), to prevent anti-virus software from being terminated. They tested their developed rootkit on five anti-virus products, with the result that it successfully terminated

all five. They tested the same five with their anti-virus protection mechanism (ANSS) installed, with the result that ANSS in each case protected the anti-virus software from termination. The operative component of the ANSS is its filter, which has many rules, such as the rule that any invocation of SSDT functions should be through the ANSS. In addition, the ANSS filter prohibits applications from using the function NtTerminateProcess to terminate the anti-virus software, and also prohibits any modification or deleting of registry keys via functions like NtDeleteKey, NtSetValueKey, and NtCreateKey. This work is similar to Matousec.com's research in that both showed that antivirus products are vulnerable when they hook the SSDT. It is also similar to that of Corregedor and Solms in that they developed rootkits. Again, a forensics approach was not conducted to collect evidence from memory.

Arnold [2] conducted a comparative analysis of rootkit detection techniques against several rootkits. Unlike the aforementioned studies, Arnold did conduct a forensics analysis. He used a hybrid approach, which included viewing the processor's utilization on the infected system and comparing it to a clean system, and analyzing the output of network-based detection tools (e.g., netstat and nmap) for both the infected and clean systems. In addition, Arnold examined the system files' locations and registry modifications of the infected system. Arnold's approach did not provide significant evidence about the functions of the SSDT that were hooked. Rather, it provided indirect indicators and warnings, such as the processor utilization, and presented statistics concerning captured network packets. While Arnold investigated the CPU and the network, we investigate the memory, as that is where the most direct evidence of an SSDT attack is located.

Alzaidi et al. [1] extracted digital evidence from volatile memory. They performed their work in a virtualized environment, and they compared two forensic techniques, live response and memory image analysis, by examining the detection capabilities of two forensic utilities, Redline and Volatility, "when the SSDT has been hooked by a rootkit". They showed "that the limitations of this live response utility [Redline] are due to the fact that it relies on system calls for detection of SSDT hooks". When Alzaidi et al. used Volatility, it was much more effective, and even detected that the live response utility Redline was infected by Blackenergy's hook. They did not discuss the hooking and release of SSDT functions by anti-virus software, and they called for additional efforts to be made in analyzing SSDT "function hooking by antivirus software in cases where rootkits are also in place" [1].

Our research has pursued an approach similar to [1], in that we investigate digital evidence extracted from volatile memory using the Volatility framework as a memory image forensics tool. The following section provides an introduction to the SSDT's structure and the SSDT hooking methods used by anti-virus and rootkit software.

3. METHODOLOGY AND RESULTS

We employed well-known anti-virus products [19] and a set of publicly available rootkits that target the SSDT. The rootkit samples were collected from many forums, such as KernelMode.info [20] and Offensive Computing [18]. Our experimentation was conducted in four stages; each stage involved a number of individual experiments using virtual machines, where some of these machines were acting as cases and one virtual machine as the control. The purpose of the control machine was for comparison: it was our 'clean' machine, and allowed us to easily identify the SSDT hooks arising as a result of rootkit and anti-virus installation and interaction. Windows XPSP3 was installed on all machines (cases and control) in each stage. The host machine was running Window 7, was equipped with an Intel i7 Core, 2.20 GHz CPU, and 16 GB RAM. A 1.5 TB external drive was used to store the memory images. In each stage memory images were taken from VMware workstation 9.0 machines, and memory analysis was performed on the control machine.

We selected five anti-virus products (AVG, Kaspersky, McAfee, Avast, Trend Micro) and three rootkits (Blackenergy, Haxdoor, Papras). For Stage 1, we analyzed the SSDT hooking methods of the

selected anti-virus products. For Stage 2, we analyzed the SSDT hooking methods of the selected rootkits. Stage 3 studied the interactions between the anti-virus products and rootkits, when the former were first installed, and then the latter. Finally, in Stage 4, we did the converse of Stage 3: the rootkits were first installed, and then the anti-virus products. Let us now look at the experimental results in detail.

First stage: The primary goal of this stage was to analyze the selected anti-virus' SSDT hooking methods. In our first experiment we installed AVG Anti-virus, and discovered that it hooked the function *NtOpenProcess* which "opens a handle to a process and sets the access rights" [17]. AVG terminated threads by calling *NtTerminateThread*, and *NtWriteVirtualMemory* was called in order to prevent any unauthorized write to or overwrite of virtual memory [22]. We also noted that AVG hooked certain functions related to keyboard inputs, such *NtUserGetKeyState* and *NtUserGetAsyncKeyState.* Such functions can help an anti-virus product to prevent malicious code from reading keyboard related information located in memory or the keyboard buffer [32]. Table 1 in Appendix A shows more fully the SSDT hooks by AVG.

In the second experiment we found that Kaspersky hooked a huge number of functions, but in this paper we focus our attention only on those functions related to the operation of the selected rootkits. Kaspersky hooked functions more critical than AVG, apparently in order to prevent registry manipulation using function calls such as *NtRestoreKey, NtDeleteValueKey*, and *NtDeleteKey*. In addition, Kaspersky hooked functions like *NtAdjustPrivilegesToken*, in order to enable or disable the access privileges to a specified token that contains information for a logon session [17]. In addition to hooking those functions related to registry manipulation, Kaspersky also hooked *NtClose* to prevent malicious attempts to close handles to critical objects such as processes, e.g., an installation process, or even to shut down the system. Table 2 in Appendix A shows more fully the SSDT hooks by Kaspersky.

McAfee Anti-virus was the subject of our third experiment. Volatility was unable to detect any SSDT hooking by McAfee. We concluded that McAfee did not at all hook the SSDT.

In the fourth experiment during this stage, Avast was also found to be hooking many critical functions, such as the *NtDeleteValueKey*. In addition, it created a key using the function *NtCreateKey*, and then hooked the function *NtDeleteValueKey*, in order to prevent any modification to that registry key. Avast in fact hooks more functions related to registry keys than the other anti-virus products.

In our fifth and last experiment during this stage we tested TrendMicro. The SSDT hooking method found here was similar to Avast's: many critical functions were hooked by TrendMicro. With TrendMicro, all hooks point to a hidden driver, and any call of these functions is routed through that hidden driver.

Second stage: The primary goal of this stage was to analyze the SSDT hooking methods of rootkits, in order to prepare to investigate the interaction between those rootkits and the anti-virus products in the third and the fourth stages of our experimentation. The focus at this stage was on rootkits that employ SSDT hooking as part of their exploitation techniques.

Our first experiment at this stage was to launch the rootkit Blackenergy in a new virtual machine. We found that Blackenergy hooks the SSDT. Blackenergy starts the SSDT hooking process with the function *NtDeleteValueKey*; this function is typically used by rootkits to modify or add values in a specified registry key. Blackenergy hooked this function in order to break the protection of the SSDT. Blackenergy also hooked functions like *NtOpenKey* and *NtSetValueKey* in order to gain write permission to the registry [3]. *NtTerminateThread* was hooked, the purpose of which might be for thread injection [5], and the *NtWriteVirtualMemory* function was also hooked, to write to or overwrite virtual memory, in order to address injected code [22]. This SSDT hooking method by Blackenergy

allows it to avoid detection and deletion. In fact, Blackenergy attempts to hide its driver, as shown in Figure 1 below.

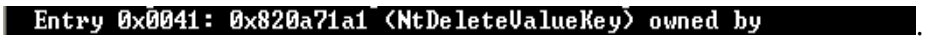

Figure 1 Blackenergy is pointing SSDT function to a hidden driver

The rootkit Haxdoor hooks fewer functions; notably, it hooks *NtOpenKey* in order to manipulate a registry key [17]. There are many versions of this rootkit available in public; some of these samples just use the *NtCreateProcess* function to create a new process [14]. We selected a Haxdoor version that hooks more functions, as shown in Appendix B Figure 2. Its also notable is the fact that, while Blackenergy was able to hide its driver, this was not so with Haxdoor, as shown in Figure 2 below.

Figure 2 Haxdoor is pointing SSDT function to the driver *vbagz.sys*

We used the rootkit Papras in our last experiment of this stage, and we found Papras was hooking the functions *NtQueryDirectoryFile* and *NtQuerySystemInformation* in order to retrieve information from a specific file. Papras can therefore be used to retrieve the credential information, e.g. an online banking login id that may be stored in a buffer [23]. Papras does something similar to Blackenergy in terms of hiding its driver, as shown in Figure 3 below.

Figure 3 Papras is pointing SSDT function to a hidden driver

Third stage: During this stage our goal was to observe and analyze the interactions between rootkits and anti-virus products. Here we studied the effect of SSDT hooking by rootkits operating in an antivirus protected environment. The anti-virus products were installed first, and the rootkits were launched subsequently. We examined the SSDT functions hooked either by the selected anti-virus products (as discovered in the first stage of our experimentation) or by the rootkits (as discovered in the second stage), in order to observe how the anti-virus products and rootkits interact as defender and attacker within the system.

We began with AVG and Blackenergy. We observed AVG requesting notification about registry key changes using the functions *NtNotifyChangeKey* and *NtNotifyChangeMultipleKeys*. Notification was positive and the registry key was changed and the SSDT compromised. The reaction by AVG was to hook the same functions back. For example, the function *NtOpenProcess* was reclaimed by AVG. Unfortunately, Blackenergy was able to return the favour, and change the registry key again; it then hooked *NtDeleteValueKey*. The reaction by AVG this time was different: AVG used the *NtUserGetAsyncKeyState* in order to return the status of all keys at a given moment. Figure 2 in Appendix C shows the reaction of AVG in order to prevent the SSDT attack by Blackenergy.

This kind of reaction by AVG is limited in its effectiveness, as the rootkit will continue deleting the values of the keys in the registry using the function *NtDeleteValueKey*; it seems that AVG might avoid this vulnerability by hooking or disabling the function *NtDeleteValueKey*. However, many processes were belonging to AVG were infected, carrying out the rootkit's functions. Blackenergy was able to camouflage itself as vmtoolsd.exe, and also took control of the process avgwdsvc.exe, the AVG Watchdog Service. AVG's SSDT hooking method, therefore, was ineffective at protecting the system's processes or even its own processes. Figures 3 and 4 in Appendix C show the infected processes.

We executed the same experiment with Kaspersky and Blackenergy. As we know from the first stage, Kaspersky hooks critical functions in order to prevent registry manipulation, such as manipulation of NtDeleteValueKey. We found that Blackenergy was unable to hook any SSDT function, because, as we knew from our previous experiments in the second stage, Blackenergy relies on

NtDeleteValueKey, and this function and other critical functions were already hooked by Kaspersky. In general, we found Kaspersky was able to protect all processes, including its own processes, such as avp.exe. See Figure 5, Appendix C.

Our third experiment at this stage employed McAfee and Blackenergy. As we know from the first stage, McAfee does not use SSDT hooking. After installing Blackenergy, McAfee reported that it had detected and was able to remove the rootkit, yet this appeared to be only partly true, since McAfee continued to report this even after it had apparently attempted to remove the rootkit. We investigated this further from a process perspective. We found to be infected the process Mcagent.exe, which is a process belonging McAfee, designed to ensure that its virus definitions are up to date. Further, Blackenergy was able to infect other processes that belong to McAfee, such as the Mcshield.exe, which is McAfee's process to monitor computer processes, files and the registry, in order to detect and prevent virus infection. Similarly, McSvHost.exe, known as McAfee Service Host, became infected, as was Mcpvtray.exe, McAfee's AntiTheft process. Finally, MOBKbackup.exe, the McAfee Online Backup Service, was also infected. Figures 7-11 in Appendix C show these infected processes.

Avast and Trend Micro were able to protect the SSDT and prevent these Blackenergy attacks, due to the fact that, like Kaspersky, Avast and Trend Micro hooked many critical functions, such as NtDeleteValueKey. Since the findings were similar to that of Kaspersky, we do not show the details in this paper.

The five selected Anti-virus products were able to protect against the other two rootkits, Haxdoor and Papras. Volatility didn't show any SSDT hooking by Haxdoor or Papras when any of the selected anti-virus products were installed. For example, this sample of Volatility output is from a machine where Haxdoor and MacAfee were both in place. The SSDT tables are not occupied, because MacAfee is not using the SSDT, and yet Haxdoor was still unable to function while the anti-virus software was running.

Created: 2012-12-13 00:34:30
Exited: 2012-12-13 00:49:37
Owning Process: 0x81caf928 "
Attached Process: 0x81caf928 "
State: Terminated
BasePriority: 0x8
Priority: 0x10
TEB: 0x00000000
StartAddress: 0x7c8106e9
ServiceTable: 0x80552f60
[0] 0x80501b8c
[1] 0xbf999b80
[2] -
[3] -

Fourth stage: The purpose of this stage was, like the previous stage, to observe and analyze the interactions between rootkits and anti-virus products. We investigated machines that had been first infected with a rootkit, which we then attempted to clean with an anti-virus product.

We began by launching Blackenergy and then installing AVG. However, the installation process did not complete and the system began an automatic shutdown. Our analysis of the memory image revealed that Blackenergy was calling the function *NtShutdownSystem*; this explains why the system was closing down (see Figure 1, Appendix D). We explored further, from a process perspective, concentrating on setup.exe and explorer.exe. We found that AVG was unable to execute setup.exe and that it was not running, and we further found that explorer.exe was infected by Blackenergy. Figures 2 and 3 of Appendix D show the details.

Interactions between Kaspersky and Blackenergy were similar, but slightly different from the foregoing. Kaspersky does not hook the function *NtShutdownSystem*, and consequently the same thing happened here as with AVG: the system shut down, and Blackenergy didn't allow Kaspersky to be installed. Our analysis of the memory image produced unexpected findings: Kaspersky was not successfully installed, but attempted regardless to hook the SSDT, presumably during the installation process. Blackenergy, however, already had control over the Native SSDT functions (KeServiceDescriptorTable), and maintained that control, while Kaspersky took control of the GUI SSDT functions (KeServiceDescriptorTableShadow). For details, see Figures 4 and 5 in Appendix D. TrendMicro was similar to the foregoing: Blackenergy called *NtShutdownSystem* to thwart installation of the anti-virus. Avast was quite different, however: Avast indeed hooked *NtShutdownSystem*, so that Blackenergy could not shut down the system, but in addition hooked various functions in order to ensure the completion of its installation without disruption. For example, Avast hooked *NtSetBootOptions*, *NtModifyBootEntry*, and *NtAddBootEntry*. See Figure 6 in Appendix D.

Finally, we note here in passing that all five selected anti-virus products were able to clean the other two rootkits, Haxdoor and Papras. We omit the details from this paper.

4. DISCUSSION AND RECOMMENDATIONS

We investigated rootkits that target the SSDT, and we found that these rootkits usually use more than one strategy to conceal an attack. For example, Blackenergy manipulated the registry in order to break the protection of the SSDT, and used the function NtDeleteValueKey to change the value of registry keys, in order to modify the SSDT's protection. In our fourth stage we observed Blackenergy closing down the system in order to stop the installation process of the anti-virus product. Haxdoor and Papras employed SSDT hooking in order to steal sensitive information, using the functions NtQueryDirectoryFile and NtQuerySystemInformation. Some Anti-virus products employ the SSDT hooking to set restrictions on accessing a system's resources. For example, some will hook the SSDT to scan any new launched program [6][7][21].

Anti-virus products like Kaspersky, Avast and TrendMicro protect registry keys by hooking the function NtDeleteValueKey, which can be effective in preventing manipulation of registry keys to break SSDT protection. Avast and TrendMicro created their own key using the function NTCreateKey, and then protected the created key and its value using the functions NtDeleteKey and NtDeleteValueKey. This may be effective in preventing attacks against the unused SSDT tables, and makes it difficult for rootkits to modify the protection of the SSDT by clearing a specific bit of the processor's CR0 register [3][4]. Instead of using this well-known value, Avast and TrendMicro create a new key with a new value.

Generally, hooking critical SSDT functions is essential for Anti-virus products. In our experiments we found that the SSDT hooking decisions by Kaspersky, Avast, and TrendMicro were most effective in terms of protecting the SSDT. On the other hand, AVG missed many critical functions, and SSDT hooking was not used at all by McAfee. Since what constitutes a critical function depends to a great degree on malware design, we emphasize that SSDT hooking by anti-virus products should be based on a precise understanding of current rootkit design. It is noteworthy that a recent paper published by Anti-virus team members did not show a full understanding of Blackenergy's current design [11].

5. CONCLUSION

We investigated the effectiveness of selected anti-virus products in defending the SSDT against malicious hooking, and exhibited the use of forensics tools and techniques for the investigation of rootkit attacks in the presence of anti-virus software. We recommend careful attention to rootkit SSDT hooking design when developing anti-virus products.

REFERENCES

[1] Alzaidi, M., Alasiri, A., Lindskog, D., Zavarsky, P., and Ruhl, R. (2011). *The study of SSDT Hook through a comparative analysis between live response and memory image*. Information Systems Security Department, Concordia University College of Alberta, Unpublished Master Thesis.

[2] Arnold, Martin T. (2011). *A comparative analysis of rootkit detection techniques*. Faculty of Science, University of Houston Clear Lake, Unpublished Master Thesis.

[3] Blunden, B. (2009). *The Rootkit Arsenal: Escape and evasion in the dark corners of the system*. Wordware Publishing Inc., Texas.

[4] Corregedor, M., and Von Solms S. (2011). Implementing Rootkits to address operating system vulnerabilities. *Information Security South Africa* (ISSA), August 13-15, 2011, Johannesburg, South Africa.

[5] Csrss. (2011). Native thread injection into the session manager subsystem. *Code Project*, May 11, 2009.

[6] Dolan, B. (2012). Auditing the system call table. Retrieved from http://moyix.blogspot.com/2008/08/auditing-system-call-table.html on August 20, 2012.

[7] Davis, A. M., Bodmer, S., and LeMasters, A. (2009). Kernel-Mode Rootkits. *Hacking Exposed Malware & Rootkits*, McGraw-Hill, USA.

[8] Hoglund, G. G., and Butler, J. (2006). The age-old art of hooking. *Rootkits: Subverting the Windows kernel*, Personal Education, Boston.

[9] Hsu, F. H, Wu M. H., Tso, C. K., Hsu, C. H., and Chen, C. W. (2012). Antivirus software shield against antivirus terminators. *IEEE Transactions on Information Forensics and Security*, October 2012, Jhongli, Taiwan.

[10] Jogie, N. (2010). Rootkit analysis: Hiding SSDT hooks. Retrieved from http://www.securabit.com/wp-content/uploads/2010/03/Rootkit-Analysis-Hiding-SSDT-Hooks1.pdf on June 30, 2012.

[11] Kapoor, A., and Mathur, R. (2011). Predicting the future of stealth attacks. *Virus Bulletin Conference*, October 2011, Oregon, USA.

[12] Kumar Uday, E. (2006). Battle with the unseen-understanding Rootkits on Windows. *The 9th AVAR International Conference*, Authentium, USA.

[13] Ligh, M. H., Blake Hartstein, S. A., and Richard, M. (2011). Memory forensics: Rootkits. *Malware analyst's cookbook and DVD: Tools and techniques for fighting malicious cod*. Wiley Publishing Inc., Indianapolis.

[14] Lobo, D. (2010). Identifying Rootkit infections using data mining. *ICISA Conference*, April 23-24, 2010, Ballarat, Victoria, Australia.

[15] Matousec Team, KHOBE 8.0 Earthquake for Windows desktop security software. (2012). Retrieved from http://www.matousec.com/info/articles/khobe-8.0-earthquake-for-windows-desktop-security-software.php on May 20, 2012.

[16] Matousec Team, Plague in (security) software drivers. (2012). Retrieved from http://www.matousec.com/info/articles/plague-in-security-software-drivers.php on May 20, 2012.

[17] Microsoft Developer Network. (2012). Retrieved from http://msdn.microsoft.com on May 29-December 20, 2012.

[18] Open Malware. (2012). Retrieved from http://www.offensivecomputing.net on May 30 and December 15 2012.

[19] OPSWAT. (2012). Market share report June 2012. Retrieved from http://www.opswat.com/about/media/reports/antivirus-june-2012 on August 20, 2012.

[20] phpBB: A Forum for Kernelmode exploration. (2012). Retrieved from http://www.kernelmode.info on May 30 and December 15 2012.

[21] Rie, C. (2006). Inside Windows Rootkit. *VigiantMinds Inc.*, May 22, 2006. Retrieved from http://www.thehackademy.net/madchat/vxdevl/library/Inside%20Windows%20Rootkits.pdf

[22] Sinha, P., Boukhtouta A., Heber Belarde, V., and Debbabi, M. (2011). Insights from the analysis of the Mariposa Botnet, *NCFTA*, Oct 10-13, 2012. Concordia University, Montreal, Canada.

[23] ThreatExpert. (2012). Retrieved from http://www.threatexpert.com/ on May 30-December 15. 2012.

[24] Volatile Systems. (2012). Volatility framework. Retrieved from http://code.google.com/p/volatility on May 30 and February 13, 2012.

APPENDICES

Appendixes A, B, C, and D provide detail regarding the results of our four stages of experimentation.

Appendix A Anti-virus SSDT hooking

Table A-1 SSDT functions hooked by AVG

Function	Driver name
NtOpenProcess	AVGIDSShim.Sys
NtTerminateProcess	AVGIDSShim.Sys
NtTerminateThread	AVGIDSShim.Sys
NtWriteVirtualMemory	AVGIDSShim.Sys
NtUserGetAsyncKeyState	AVGIDSShim.Sys
NtUserGetKeyboardState	AVGIDSShim.Sys
NtUserGetKeyState	AVGIDSShim.Sys
NtUserSetWindowsHookEx	AVGIDSShim.Sys

Table A-2 SSDT functions hooked by Kaspersky

Function	Driver name
NtAdjustPrivilegesToken	klif.sys
NtClose	klif.sys
NtConnectPort	klif.sys
NtCreateEvent	klif.sys
NtCreateMutant	klif.sys
NtCreatePort	klif.sys
NtCreateProcess	klif.sys
NtCreateProcessEx	klif.sys
NtCreateSection	klif.sys
NtCreateSemaphore	klif.sys
NtCreateThread	klif.sys
NtCreateWaitablePort	klif.sys
NtDebugActiveProcess	klif.sys
NtDeleteKey	klif.sys
NtDeleteValueKey	klif.sys
NtDeviceIoControlFile	klif.sys
NtDuplicateObject	klif.sys
NtEnumerateKey	klif.sys
NtEnumerateValueKey	klif.sys
NtLoadDriver	klif.sys
NtLoadKey	klif.sys
NtLoadKey2	klif.sys
NtMapViewOfSection	klif.sys
NtNotifyChangeKey	klif.sys
NtOpenEvent	klif.sys
NtOpenMutant	klif.sys
NtOpenProcess	klif.sys
NtOpenSection	klif.sys
NtOpenSemaphore	klif.sys
NtOpenThread	klif.sys
NtQueryKey	klif.sys
NtQueryMultipleValueKey	klif.sys
NtQueryObject	klif.sys
NtQueryValueKey	klif.sys
NtQueueApcThread	klif.sys
NtRenameKey	klif.sys
NtReplaceKey	klif.sys
NtReplyPort	klif.sys
NtReplyWaitReceivePort	klif.sys
NtReplyWaitReceivePortEx	klif.sys
NtRequestWaitReplyPort	klif.sys
NtRestoreKey	klif.sys
NtResumeThread	klif.sys
NtSaveKey	klif.sys
NtSaveKeyEx	klif.sys
NtSaveMergedKeys	klif.sys
NtSecureConnectPort	klif.sys
NtSetContextThread	klif.sys
NtSetInformationToken	klif.sys
NtSetSystemInformation	klif.sys

Table A-3 SSDT functions hooked by Avast

Function	Driver name
NtAllocateVirtualMemory	aswSP.SYS
NtClose	aswSP.SYS
NtCreateKey	aswSP.SYS
NtCreateSection	aswSP.SYS
NtDeleteKey	aswSP.SYS
NtDeleteValueKey	aswSP.SYS
NtDuplicateObject	aswSP.SYS
NtFreeVirtualMemory	aswSP.SYS
NtLoadDriver	aswSP.SYS
NtOpenKey	aswSP.SYS
NtOpenProcess	aswSP.SYS
NtOpenThread	aswSP.SYS
NtProtectVirtualMemory	aswSP.SYS
NtQueryValueKey	aswSP.SYS
NtRenameKey	aswSP.SYS
NtRestoreKey	aswSP.SYS
NtSetValueKey	aswSP.SYS
NtTerminateProcess	aswSP.SYS
NtUnloadDriver	aswSP.SYS
NtWriteVirtualMemory	aswSP.SYS

Table A-4 SSDT functions hooked by TrendMicro

Function	Driver name
NtCreateKey	Hidden
NtCreateMutant	Hidden
NtCreateProcess	Hidden
NtCreateProcessEx	Hidden
NtCreateSymbolicLinkObject	Hidden
NtCreateThread	Hidden
NtDeleteKey	Hidden
NtDeleteValueKey	Hidden
NtDuplicateObject	Hidden
NtLoadDriver	Hidden
NtOpenProcess	Hidden
NtOpenSection	Hidden
NtOpenThread	Hidden
NtRenameKey	Hidden
NtRestoreKey	Hidden
NtSetSystemInformation	Hidden
NtSetValueKey	Hidden
NtTerminateProcess	Hidden
NtTerminateThread	Hidden
NtWriteVirtualMemory	Hidden
NtUserSetWindowsHookAW	Hidden
NtUserSetWindowsHookEx	Hidden

Appendix B Rootkit SSDT hooking

```
Volatile Systems Volatility Framework 2.1_alpha
  Entry 0x0041: 0x820a71a1 (NtDeleteValueKey) owned by
  Entry 0x0047: 0x820a6e39 (NtEnumerateKey) owned by
  Entry 0x0049: 0x820a6f52 (NtEnumerateValueKey) owned by
  Entry 0x0077: 0x820a6d6f (NtOpenKey) owned by
  Entry 0x007a: 0x820a6aa9 (NtOpenProcess) owned by
  Entry 0x0080: 0x820a6b31 (NtOpenThread) owned by
  Entry 0x0089: 0x820a73e6 (NtProtectVirtualMemory) owned by
  Entry 0x0091: 0x820a75bd (NtQueryDirectoryFile) owned by
  Entry 0x00ba: 0x820a72fa (NtReadVirtualMemory) owned by
  Entry 0x00d5: 0x820a6cfc (NtSetContextThread) owned by
  Entry 0x00f7: 0x820a708f (NtSetValueKey) owned by
  Entry 0x00fe: 0x820a6c89 (NtSuspendThread) owned by
  Entry 0x0102: 0x820a6c16 (NtTerminateThread) owned by
  Entry 0x0115: 0x820a7370 (NtWriteVirtualMemory) owned by
```

Figure B-1 SSDT functions hooked by Blackenergy

```
Volatile Systems Volatility Framework 2.1_alpha
  Entry 0x002f: 0xf8bd2fe9 (NtCreateProcess) owned by vbagz.sys
  Entry 0x0030: 0xf8bd2a86 (NtCreateProcessEx) owned by vbagz.sys
  Entry 0x0077: 0xf8bd2467 (NtOpenKey) owned by vbagz.sys
  Entry 0x007a: 0xf8bd2799 (NtOpenProcess) owned by vbagz.sys
  Entry 0x0091: 0xf8bd27ef (NtQueryDirectoryFile) owned by vbagz.sys
C:\Volatility 2.0\volatility\plugins\malware.py:3035: DeprecationWar
```

Figure B-2 SSDT functions hooked by Haxdoor

```
 Entry 0x0049: 0x8207c58a (NtEnumerateValueKey) owned by
 Entry 0x0091: 0x8207c6b6 (NtQueryDirectoryFile) owned by
 Entry 0x00ad: 0x8207c85c (NtQuerySystemInformation) owned by
```

Figure B-3 SSDT functions hooked by Papras

Appendix C SSDT hooking interaction when anti-virus is installed before rootkit

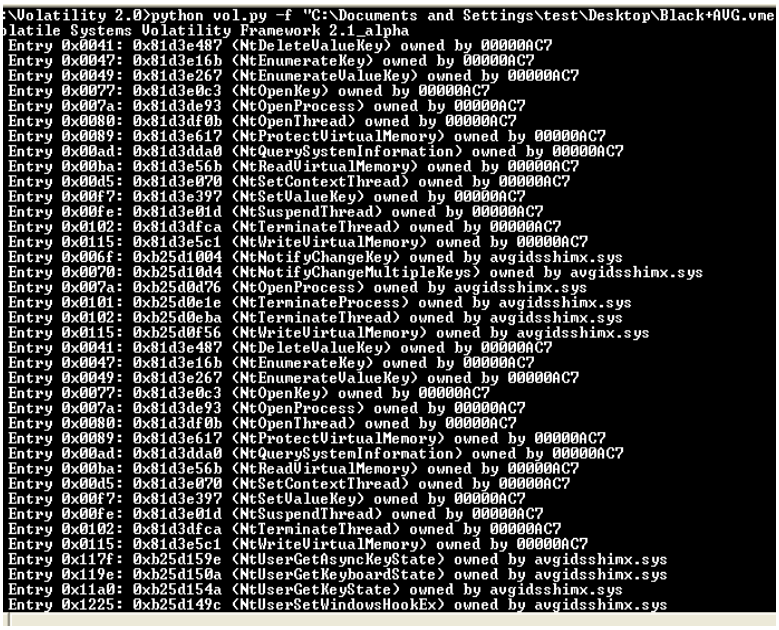

Figure C-1 SSDT hooking with AVG and Blackenergy

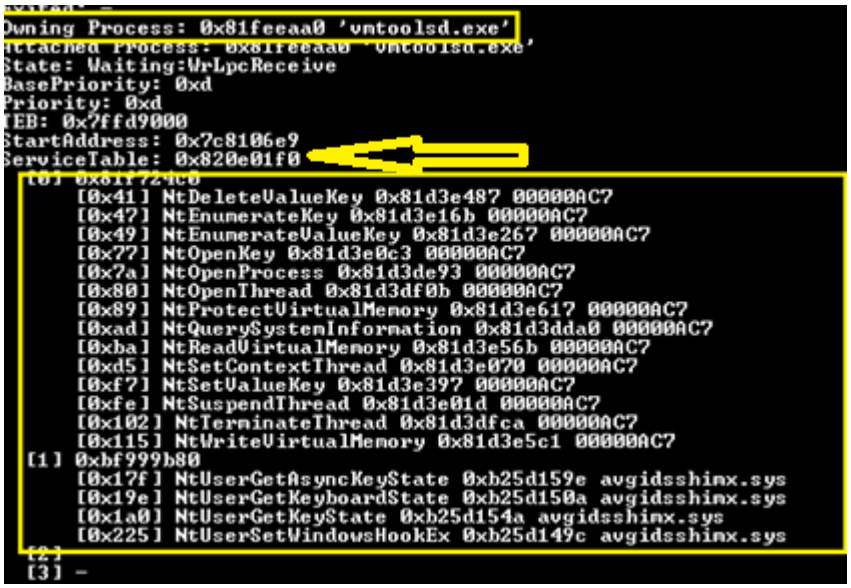

Figure C-2 AVG's vmtoolsd.exe infected

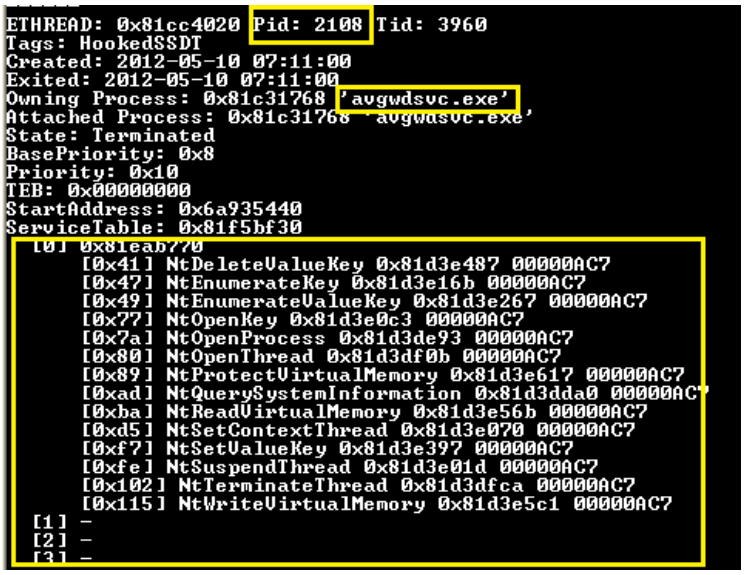

Figure C-3 AVG's avgwdsvc.exe infected

Figure C-4 Kaspersky protects its avp.exe process from Blackenergy

Figure C-5 Kaspersky protects its Datamn~1.exe process from Blackenergy

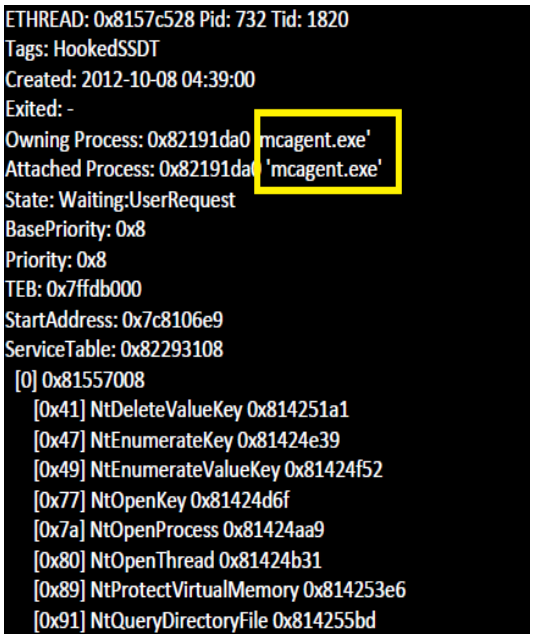

Figure C-6 MacAfee's mcagent.exe process infected by Blackenergy

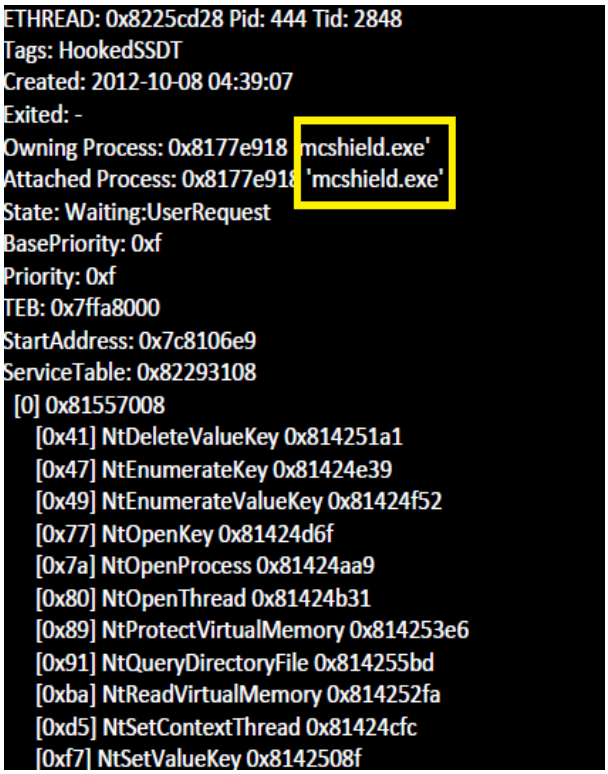

Figure C-7 MacAfee's mcshield.exe process infected by Blackenergy

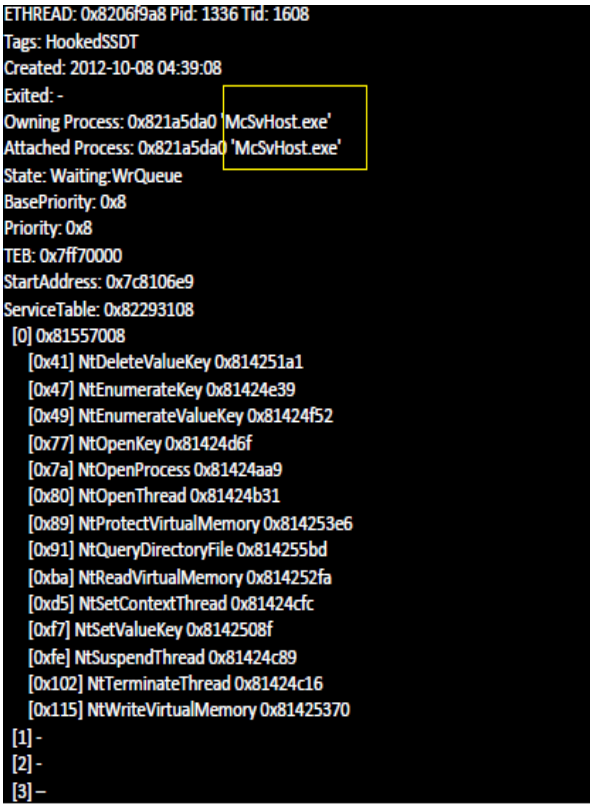

Figure C-8 McAfee's McSvHost.exe process infected by Blackenergy

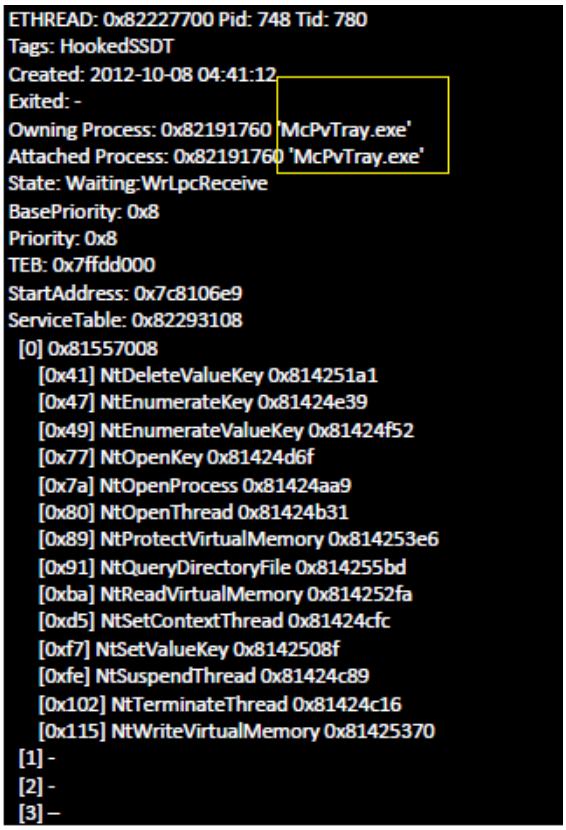

Figure C-9 MacAfee's McPvTray.exe process infected by Blackenergy

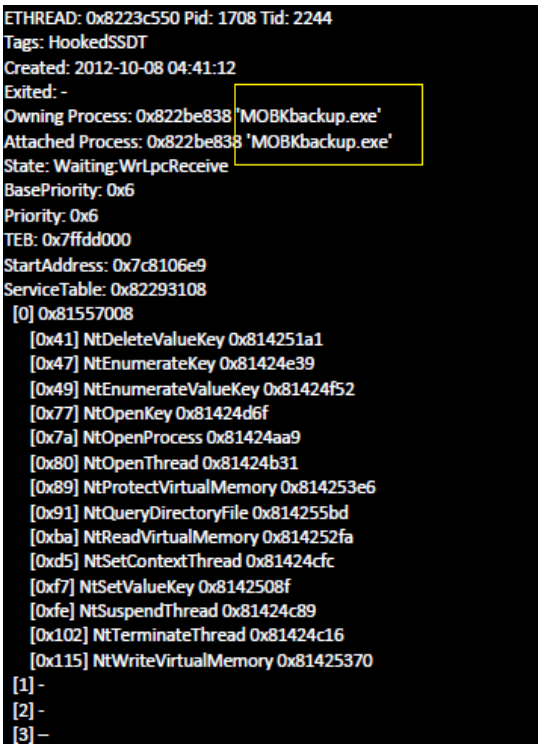

Figure C-10 MacAfee's MOBKbackup.exe process infected by Blackenergy

Appendix D SSDT hooking interaction when rootkit is installed before anti-virus

```
Entry 0x0041: 0x823ba1a1 (NtDeleteValueKey) owned by
Entry 0x0047: 0x823b9e39 (NtEnumerateKey) owned by
Entry 0x0049: 0x823b9f52 (NtEnumerateValueKey) owned by
Entry 0x0077: 0x823b9d6f (NtOpenKey) owned by
Entry 0x007a: 0x823b9aa9 (NtOpenProcess) owned by
Entry 0x0080: 0x823b9b31 (NtOpenThread) owned by
Entry 0x0089: 0x823ba3e6 (NtProtectVirtualMemory) owned by
Entry 0x0091: 0x823ba5bd (NtQueryDirectoryFile) owned by
Entry 0x00ad: 0x823b9956 (NtQuerySystemInformation) owned by
Entry 0x00ba: 0x823ba2fa (NtReadVirtualMemory) owned by
Entry 0x00d5: 0x823b9cfc (NtSetContextThread) owned by
Entry 0x00f7: 0x823ba08f (NtSetValueKey) owned by
Entry 0x00f9: 0x823b7ca8 (NtShutdownSystem) owned by
Entry 0x00fe: 0x823b9c89 (NtSuspendThread) owned by
Entry 0x0102: 0x823b9c16 (NtTerminateThread) owned by
Entry 0x0115: 0x823ba370 (NtWriteVirtualMemory) owned by
```

Figure D-1 Blackenergy using NtShutdownSystem

```
State: Waiting:UserRequest
BasePriority: 0xd
Priority: 0xf
TEB: 0x7ffdf000
StartAddress: 0x7c8106f5
ServiceTable: 0x824543a8
[0] 0x8248f898
[0x41] NtDeleteValueKey 0x823ba1a1
[0x47] NtEnumerateKey 0x823b9e39
[0x49] NtEnumerateValueKey 0x823b9f52
[0x77] NtOpenKey 0x823b9d6f
[0x7a] NtOpenProcess 0x823b9aa9
[0x80] NtOpenThread 0x823b9b31
[0x89] NtProtectVirtualMemory 0x823ba3e6
[0x91] NtQueryDirectoryFile 0x823ba5bd
[0xad] NtQuerySystemInformation 0x823b9956
[0xba] NtReadVirtualMemory 0x823ba2fa
[0xd5] NtSetContextThread 0x823b9cfc
[0xf7] NtSetValueKey 0x823ba08f
[0xf9] NtShutdownSystem 0x823b7ca8
[0xfe] NtSuspendThread 0x823b9c89
[0x102] NtTerminateThread 0x823b9c16
[0x115] NtWriteVirtualMemory 0x823ba370
[1] 0xbf999b80
[2] -
[3] -
```

Figure D-2 The process explore.exe infected by Blackenergy

```
Owning Process: 0x82068d60 'setup.exe'
Attached Process: 0x82068d60 'setup.exe'
State: Waiting:UserRequest
BasePriority: 0x8
Priority: 0x8
TEB: 0x7ffa8000
StartAddress: 0x7c8106e9
ServiceTable: 0x8201fef0
[0] 0x82012328
[0x41] NtDeleteValueKey 0x823121a1
[0x47] NtEnumerateKey 0x82311e39
[0x49] NtEnumerateValueKey 0x82311f52
[0x77] NtOpenKey 0x82311d6f
[0x7a] NtOpenProcess 0x82311aa9
[0x80] NtOpenThread 0x82311b31
[0x89] NtProtectVirtualMemory 0x823123e6
[0x91] NtQueryDirectoryFile 0x823125bd
[0xba] NtReadVirtualMemory 0x823122fa
[0xd5] NtSetContextThread 0x82311cfc
[0xf7] NtSetValueKey 0x8231208f
[0xfe] NtSuspendThread 0x82311c89
[0x102] NtTerminateThread 0x82311c16
[0x115] NtWriteVirtualMemory 0x82312370
[1] -
[2] -
[3] -
```

Figure D-3 The process setup.exe compromised by Blackenergy

```
Entry 0x0041: 0x823121a1 (NtDeleteValueKey) owned by
Entry 0x0047: 0x82311e39 (NtEnumerateKey) owned by
Entry 0x0049: 0x82311f52 (NtEnumerateValueKey) owned by
Entry 0x0077: 0x82311d6f (NtOpenKey) owned by
Entry 0x007a: 0x82311aa9 (NtOpenProcess) owned by
Entry 0x0080: 0x82311b31 (NtOpenThread) owned by
Entry 0x0089: 0x823123e6 (NtProtectVirtualMemory) owned by
Entry 0x0091: 0x823125bd (NtQueryDirectoryFile) owned by
Entry 0x00ba: 0x823122fa (NtReadVirtualMemory) owned by
Entry 0x00d5: 0x82311cfc (NtSetContextThread) owned by
Entry 0x00f7: 0x8231208f (NtSetValueKey) owned by
Entry 0x00fe: 0x82311c89 (NtSuspendThread) owned by
Entry 0x0102: 0x82311c16 (NtTerminateThread) owned by
Entry 0x0115: 0x82312370 (NtWriteVirtualMemory) owned by
Entry 0x0041: 0x823121a1 (NtDeleteValueKey) owned by
Entry 0x0047: 0x82311e39 (NtEnumerateKey) owned by
Entry 0x0049: 0x82311f52 (NtEnumerateValueKey) owned by
Entry 0x0077: 0x82311d6f (NtOpenKey) owned by
Entry 0x007a: 0x82311aa9 (NtOpenProcess) owned by
Entry 0x0080: 0x82311b31 (NtOpenThread) owned by
Entry 0x0089: 0x823123e6 (NtProtectVirtualMemory) owned by
Entry 0x0091: 0x823125bd (NtQueryDirectoryFile) owned by
Entry 0x00ba: 0x823122fa (NtReadVirtualMemory) owned by
Entry 0x00d5: 0x82311cfc (NtSetContextThread) owned by
Entry 0x00f7: 0x8231208f (NtSetValueKey) owned by
Entry 0x00fe: 0x82311c89 (NtSuspendThread) owned by
Entry 0x0102: 0x82311c16 (NtTerminateThread) owned by
Entry 0x1007: 0xb1d4aec8 (NtGdiAlphaBlend) owned by klif.sys
Entry 0x100d: 0xb1d4a640 (NtGdiBitBlt) owned by klif.sys
Entry 0x10bf: 0xb1d4ae82 (NtGdiGetPixel) owned by klif.sys
Entry 0x10e3: 0xb1d4a716 (NtGdiMaskBlt) owned by klif.sys
Entry 0x10ed: 0xb1d4a786 (NtGdiPlgBlt) owned by klif.sys
Entry 0x1124: 0xb1d4a6aa (NtGdiStretchBlt) owned by klif.sys
Entry 0x112a: 0xb1d4b016 (NtGdiTransparentBlt) owned by klif.sys
Entry 0x1133: 0xb1d4abbe (NtUserAttachThreadInput) owned by klif.sys
Entry 0x1143: 0xb1d4a60c (NtUserCallOneParam) owned by klif.sys
Entry 0x117a: 0xb1d4a374 (NtUserFindWindowEx) owned by klif.sys
Entry 0x117f: 0xb1d4a168 (NtUserGetAsyncKeyState) owned by klif.sys
Entry 0x119e: 0xb1d4a56a (NtUserGetKeyboardState) owned by klif.sys
Entry 0x11a0: 0xb1d4a1b8 (NtUserGetKeyState) owned by klif.sys
Entry 0x11cc: 0xb1d4a2bc (NtUserMessageCall) owned by klif.sys
Entry 0x11db: 0xb1d4a208 (NtUserPostMessage) owned by klif.sys
Entry 0x11dc: 0xb1d4a260 (NtUserPostThreadMessage) owned by klif.sys
Entry 0x11ea: 0xb1d4ac78 (NtUserRegisterHotKey) owned by klif.sys
Entry 0x11eb: 0xb1d4a4ea (NtUserRegisterRawInputDevices) owned by klif.sys
Entry 0x11f6: 0xb1d4a320 (NtUserSendInput) owned by klif.sys
Entry 0x1211: 0xb1d4aa4a (NtUserSetParent) owned by klif.sys
Entry 0x1220: 0xb1d49fbe (NtUserSetWindowLong) owned by klif.sys
Entry 0x1225: 0xb1d4a018 (NtUserSetWindowsHookEx) owned by klif.sys
Entry 0x1228: 0xb1d4a0c0 (NtUserSetWinEventHook) owned by klif.sys
Entry 0x1240: 0xb1d4ad90 (NtUserUnregisterHotKey) owned by klif.sys
Entry 0x1250: 0xb1d4a474 (NtUserWindowFromPoint) owned by klif.sys
```

Figure D-4 Kaspersky installation unsuccessful, but trying to hook the SSDT

```
Owning Process: 0x81f6a020 'explorer.exe'
Attached Process: 0x81f6a020 'explorer.exe'
State: Waiting:UserRequest
BasePriority: 0x8
Priority: 0x9
TEB: 0x7ffd4000
StartAddress: 0x7c8106e9
ServiceTable: 0x82321188
  [0] 0x81f676a8
      [0x41] NtDeleteValueKey 0x823121a1
      [0x47] NtEnumerateKey 0x82311e39
      [0x49] NtEnumerateValueKey 0x82311f52
      [0x77] NtOpenKey 0x82311d6f
      [0x7a] NtOpenProcess 0x82311aa9
      [0x80] NtOpenThread 0x82311b31
      [0x89] NtProtectVirtualMemory 0x823123e6
      [0x91] NtQueryDirectoryFile 0x823125bd
      [0xba] NtReadVirtualMemory 0x823122fa
      [0xd5] NtSetContextThread 0x82311cfc
      [0xf7] NtSetValueKey 0x8231208f
      [0xfe] NtSuspendThread 0x82311c89
      [0x102] NtTerminateThread 0x82311c16
      [0x115] NtWriteVirtualMemory 0x82312370
  [1] 0xbf999b80
      [0x7] NtGdiAlphaBlend 0xb1d4aec8 klif.sys
      [0xd] NtGdiBitBlt 0xb1d4a640 klif.sys
      [0xbf] NtGdiGetPixel 0xb1d4ae82 klif.sys
      [0xe3] NtGdiMaskBlt 0xb1d4a716 klif.sys
      [0xed] NtGdiPlgBlt 0xb1d4a786 klif.sys
      [0x124] NtGdiStretchBlt 0xb1d4a6aa klif.sys
      [0x12a] NtGdiTransparentBlt 0xb1d4b016 klif.sys
      [0x133] NtUserAttachThreadInput 0xb1d4abbe klif.sys
      [0x143] NtUserCallOneParam 0xb1d4a60c klif.sys
      [0x17a] NtUserFindWindowEx 0xb1d4a374 klif.sys
      [0x17f] NtUserGetAsyncKeyState 0xb1d4a168 klif.sys
      [0x1a0] NtUserGetKeyState 0xb1d4a1b8 klif.sys
      [0x1cc] NtUserMessageCall 0xb1d4a2bc klif.sys
      [0x1db] NtUserPostMessage 0xb1d4a208 klif.sys
      [0x1dc] NtUserPostThreadMessage 0xb1d4a260 klif.sys
      [0x1ea] NtUserRegisterHotKey 0xb1d4ac78 klif.sys
      [0x1eb] NtUserRegisterRawInputDevices 0xb1d4a4ea klif.sys
      [0x1f6] NtUserSendInput 0xb1d4a320 klif.sys
      [0x211] NtUserSetParent 0xb1d4aa4a klif.sys
      [0x220] NtUserSetWindowLong 0xb1d49fbe klif.sys
      [0x225] NtUserSetWindowsHookEx 0xb1d4a018 klif.sys
      [0x228] NtUserSetWinEventHook 0xb1d4a0c0 klif.sys
      [0x240] NtUserUnregisterHotKey 0xb1d4ad90 klif.sys
      [0x250] NtUserWindowFromPoint 0xb1d4a474 klif.sys
```

Figure D-5 explorer.exe under control of Blackenergy's Nativc SSDT functions; Kaspersky taking the GUI

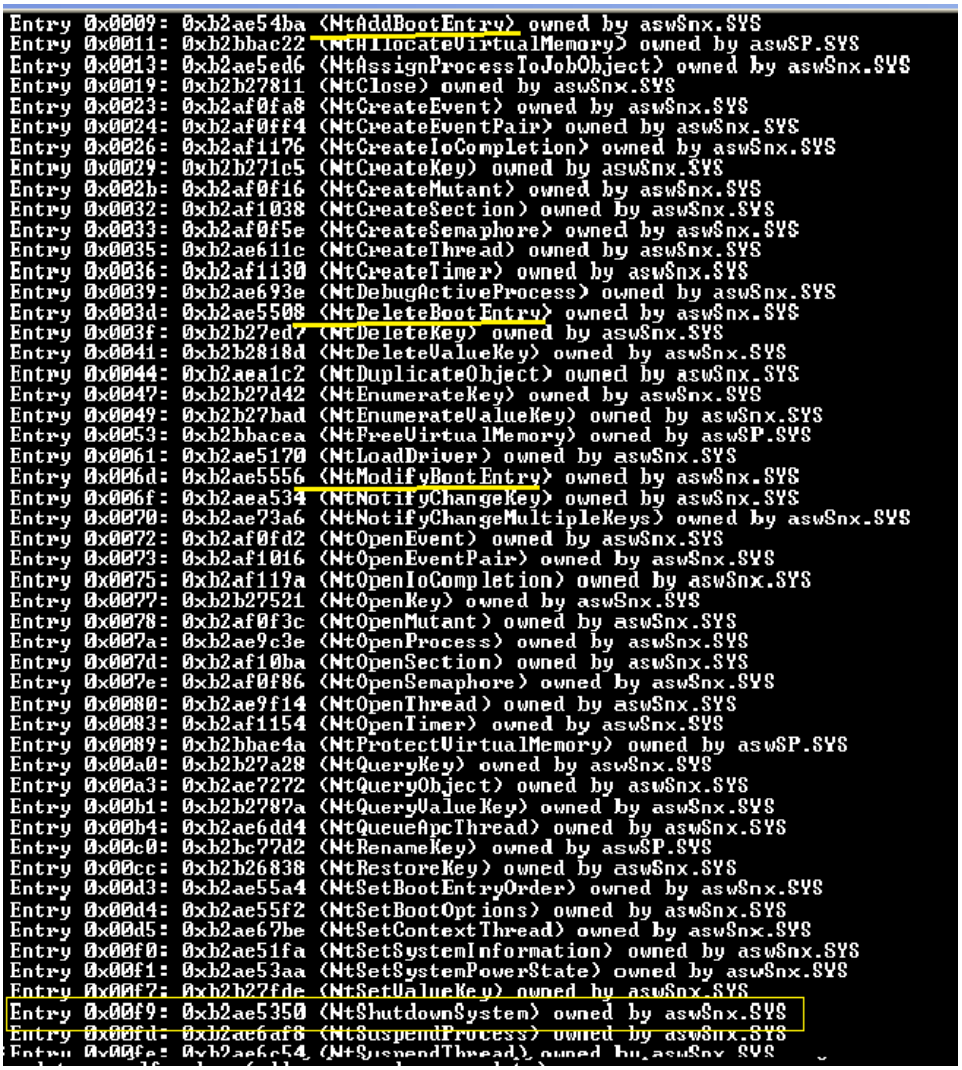

Figure D-6 Avast using booting functions and NtShutdownSystem