



Article

A Cloud-Based Real-Time Mechanism to Protect End Hosts against Malware

Fu-Hau Hsu ¹, Chia-Hao Lee ¹, Ting Luo ¹, Ting-Cheng Chang ² and Min-Hao Wu ^{2,*}¹ Department of Computer Science and Information Engineering, National Central University, Taoyuan 32001, Taiwan² College of Information and Mechanical & Electrical Engineering, Ningde Normal University, Ningde 352100, China

* Correspondence: mhwu@csie.ncu.edu.tw

Received: 23 August 2019; Accepted: 1 September 2019; Published: 8 September 2019



Featured Application: Authors are encouraged to provide a concise description of the specific application or a potential application of the work. This section is not mandatory.

Abstract: Nowadays, antivirus is one of the most popular tools used to protect computer systems. Diverse antivirus vendors are established to protect their customers against malware. However, antivirus is facing some critical problems, such as significant detection windows, vulnerability inside antivirus, and long scanning time. In this paper, we recommend a cloud-based real-time defense mechanism named Skywalker to allow users to safely utilize antivirus without the above problems. After Skywalker is installed in a host, the host does not need to install any antivirus. However, Skywalker guarantees that the host only executes programs that have been verified by a cloud-based scanner, such as VirusTotal. VirusTotal uses 56 antivirus engines to check whether a program is malware. Research shows that the more antivirus engines are used, the more accurate the result is. Because the above scan is performed right before the execution of every program, Skywalker provides 24/7 real-time protection to a system. Besides, Skywalker eliminates the need to spend a lot of time scanning all files in a host. Experimental results show that after a program has been executed once, it takes Skywalker, at most, 0.47091 s to start the program again. Meanwhile, VirusTotal provides a secure protection to client hosts.

Keywords: malware; antivirus; real time detection; cloud system

1. Introduction

Diverse antivirus companies are established to defense various systems, because every day tremendous malware, worms, Trojans, and malicious content are created and spread over the Internet, which brings severe threats to computer systems and networks. According to a report published by Gartner [1] in 2015, worldwide security software revenue in 2014 is up to \$21.4 billion. The revenue has increased by 5.3 percent compared with the revenue in 2013. Antivirus software has become one of the most popular tools to protect end hosts.

Research [2] shows that the more antivirus engines are used, the more accurate the scanning results are. However, nowadays it is difficult to have more than one different antivirus engines installed on the same host at the same time without causing any conflict. In this paper, we recommend a mechanism, called Skywalker, to protect end hosts against malware attacks. Skywalker allows an end host to be protected by 56 antivirus engines at the same time. However, Skywalker does not need the end host to install any of them on it. As a result, Skywalker utilizes multiple antivirus engines to protect a host, but it relieves the end host from the overhead and risks introduced by antivirus software.

Even though antivirus engines provide powerful protection to hosts, they also encounter problems. The first problem is the usage problem. Maintaining and using antivirus causes non-trivial overhead. It usually takes several hours to scan an end host. Between two scans, there may be a window during which malware can reside on an end host without being detected. Besides, an end host needs to keep updating its virus database to obtain the latest malware signatures. If an end host does not have the signature of a piece of malware, the host is not able to withstand the attacks of the malware. Users not updating their virus signature databases or antivirus companies not creating the signature of a malicious program will result in a lack of the related signatures. Antivirus vendors often cannot generate virus signatures in time [3], because tremendous malicious applications are created by attackers every day. However, antivirus vendors need to catch and analyze a malicious application before creating its signatures, which takes at least few hours to several days. We call this time period the vulnerability window of the malicious application. During this window, end hosts are fragile to the malware. However, when more antivirus engines are used to scan malware, the length of the vulnerability window of a malicious application can be greatly decreased.

The second problem is the security problems of antivirus applications themselves. Research [4] shows that there are several approaches to disable antivirus software. Moreover, as described by CloudAV [5,6], antivirus cannot guarantee the security of the computers that install it. Furthermore, attackers can use tricks to bypass antivirus software, or even worse, use the vulnerability of antivirus to compromise end hosts. In the symposium on security for Asia network 2014 in Beijing, a researcher revealed that among the 17 most popular antivirus engines, 14 of them are vulnerable [7] and can be exploited locally or remotely. Those vulnerabilities include heap overflow, buffer overflow, remote vulnerabilities, and so on. Once attackers discover any of these vulnerabilities, they can easily exploit the vulnerabilities and compromise related hosts.

Skywalker utilizes a cloud-based scanner, such as VirusTotal [8], to scan every program that is going to be executed by an end host that installs Skywalker; hence, it provides 24/7 real time protection to end hosts. Skywalker does not scan programs that are not executed. As a result, there is no need to suspend an end host for several hours to wait for the completeness of a full system scan. Besides, because cloud-based scanners use multiple antivirus engines to check an executable and no antivirus engine is installed in an end host, Skywalker can avoid the antivirus vulnerability and antivirus maintenance problems. The Skywalker uses a variety of anti-virus engines to enable customers to determine whether a file is destructive by the detection outcomes of each anti-virus engine. The Skywalker uses a variety of anti-virus engines to enable customers to determine whether a file is destructive by the detection outcomes of each anti-virus engine. In other words, file scan that is originally done by one local antivirus engine is performed by multiple antivirus engines in the cloud through the network transmission between local hosts and the cloud.

The above description shows that there are four major goals that Skywalker wants to achieve to utilize the antivirus engine in a more efficient way. First, we expect to raise the detection rate of the latest malicious content. Second, we want to get rid of the time-consuming scanning work that is scheduled by an antivirus scheduler periodically. Third, we want to eliminate the overhead to maintain and update an antivirus engine. Lastly, we want to reduce the risks that are introduced by antivirus vulnerabilities. Experimental results show that after a program has been executed once, it takes Skywalker, at most, 0.47091 s to start the program again. Meanwhile, VirusTotal provides a secure protection to client hosts.

The rest of the paper is organized as follows. Section 2 introduces some background technologies that Skywalker uses. Section 3 describes the system design of Skywalker. Section 4 displays the experimental results to show the effectiveness and efficiency of Skywalker. Section 5 discusses related work. Section 6 concludes this paper.

2. Background

Section 2 introduces Windows system service, system service descriptor table, system service dispatcher, *system services descriptor table* (SSDT) hooking technique, and a free cloud-based antivirus scanner called VirusTotal, that Skywalker uses. Through this section, we can understand some background knowledge that Skywalker uses.

2.1. Windows System Service

A Windows system provides numerous system services for applications. Applications called Windows *Application Programming Interfaces* (APIs) are implemented in Win32 subsystem *Dynamic-link libraris* (DLLs) to use these system services. For instance, an application can be called CreateFile API to create a file or called WriteFile API to write output to a file.

Figure 1 illustrates the execution flow of invoking a system service from the user address space code of an application. When an application needs to execute a system service, it calls a Win32 API implemented in the system DLLs, such as kernel32.dll and user32.dll. Next, the Win32 API invokes a related system function in ntdll.dll. System functions include special functions supported by operating system or some system service stub functions. Instead of implementing real operations for system services, system service stub functions are only responsible for delivering critical parameters to the kernel and switching from user mode to kernel mode. So to switch to kernel mode, instruction int2e or sysenter is executed while KiFastSystemCall or KiIntSystemCall is invoked. Both functions are used to transform the current mode into kernel mode. The difference between them is that the former uses instruction int2e and the latter uses instruction sysenter. After transferring to kernel mode, the system service dispatch handler KiSystemService invokes the related system service routine according to the parameters receiving from the system service stub. When the related system service routine finishes, KiSystemService calls KiServiceExit to execute iretd or sysexit instructions to transfer the instructions of the application from the kernel address space back to a user address space.

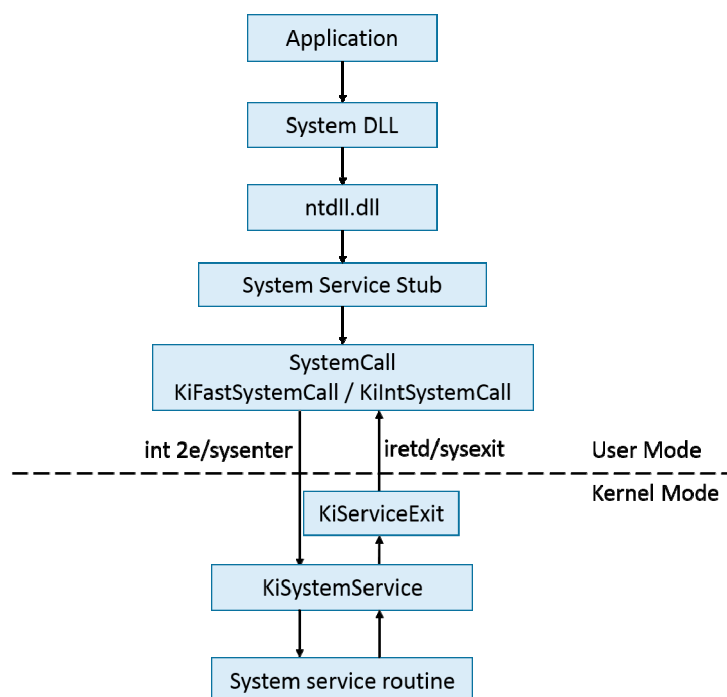


Figure 1. Execution flow of invoking a system service.

2.2. System Services Descriptor Table

A kernel array called system services descriptor table (SSDT) stores all native API addresses. SSDT is not only a huge address index table, but also includes other useful messages such as the numbers of system services and the byte allocations of the arguments of each system service. By modifying an SSDT entry, we can hook a critical Windows native API to intercept the parameters passed to the native API. Nowadays, most of the Antivirus software still uses this technique to monitor and catch the abnormal behavior in the kernel.

In a Windows system, a system service dispatcher gets a system call number as an index from a stub function and looks up the system service dispatch table (SSDT) to obtain the entry address of related system call service routine. After finding out the entry point, it jumps to the system call service routine to handle the request. Kernel function `KiSystemService`, shown in Figure 1, implements the system service dispatcher. However, before system service dispatcher gets the system call number, the stub uses EAX register, an extended (32-bit) processor register of x86 CPUs, to pass a system call number into the kernel.

A system call number is divided into three parts. The 12 least significant bits of it are an index of the SSDT. Bits 12–13 are an index of the service descriptor table (SDT). Windows system reserves two bits for service descriptor tables, which means that there should be four SDTs, but Windows only uses two SDTs. One of the SDT tables is `KeServiceDescriptorTable` and the other is `KeServiceDescriptorTableShadow`. Both service descriptor tables hold a critical kernel structure called `SystemServiceTable` to record important information about an SSDT. Figure 2 illustrates how the EAX register and service descriptor tables are used to find the entry point of a service routine. Two SSTs are declared in `KeServiceDescriptorTableShadow`. One is `SST0`, which points to the base address of `SSDT1`. The other one stores information about `SSDT2`. `SSDT2` is not used in our system. `KeServiceDescriptorTable` declares only `SST0`, which also points to the base address of `SSDT1`.

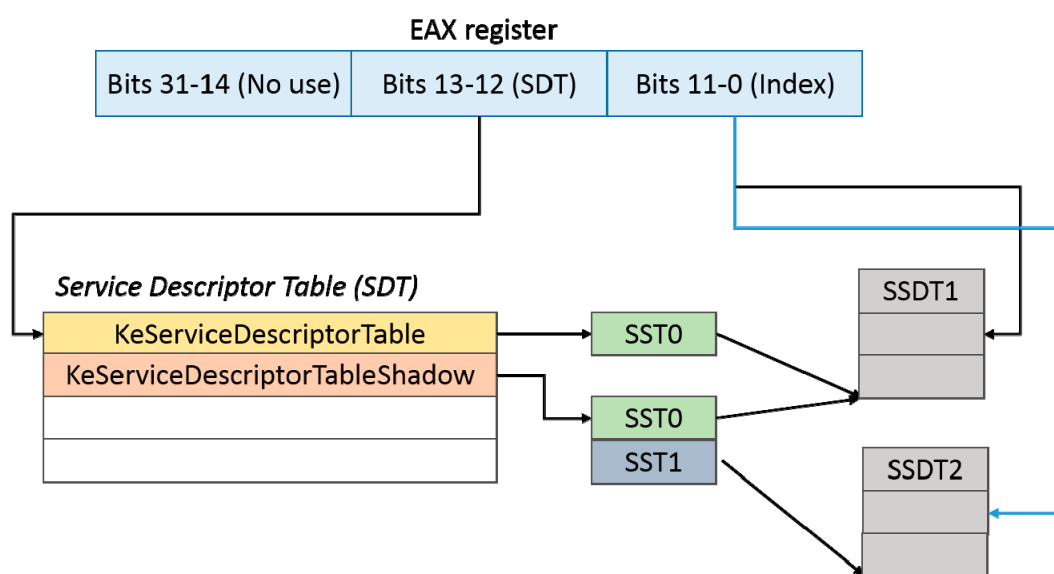


Figure 2. Relationships among EAX register (a 32-bit processor register of x86 CPUs), service descriptor table (SDT), and system service dispatch table (SSDT)1.

2.3. SSDT Hooking

SSDT hooking [9–11] is a technique that allows a user to modify an SSDT entry related to a system call service routine with the function address of a hooked function so that every invocation of the original system call service routine will call the hooked function instead. The hooked function can call the original system call service routine; hence, it can intercept every parameter past to the original system call service routine. However, SSDT is a read-only memory block. In a Windows system, part

of the bits stored in the cr0 register represent a segment selector, which points to a segment descriptor in the global descriptor table (GDT). The Write Protect bit (WP flag) in cr0 register is a flag used to protect a read-only memory from being written to. If WP flag is set to 0, the read-only memory can be written to in kernel mode. On the other hand, if WP flag is set to 1, the access right of the physical memory is determined by the user/supervisor flag and R/W flag in a page directory entry and page table entry. The WP flag only takes effect in kernel mode. Hence, we need to modify the WP flag to 0 inside the kernel before starting hooking SSDT.

2.4. VirusTotal

The prototype of Skywalker uses VirusTotal [8] as its cloud-based scanner. However, Skywalker can switch to any other cloud-based scanner that provides similar functionality. VirusTotal aims to help users to check the legitimacy of web pages or files up to 128 KB (32 MB for developers using public API). A user uploads an image file or provides a Uniform Resource Locator (URL) to VirusTotal first. Then VirusTotal analyzes the file or the web page pointed by the URL using 56 antivirus engines or website scanners to scan and identify Trojans, viruses, worms, and other malicious content in the file or the web page. According to the official website [12] of VirusTotal, VirusTotal live updates its infection signature data sources and blacklists. Additionally, VirusTotal supplies comprehensive results from each scanner. Moreover, VirusTotal can supply detail information from each scanner. Furthermore, VirusTotal provides public [13] and private APIs for developers to use its services. The public API has a request limitation and private API does not have a request limitation, but needs to pay to use it. The prototype of Skywalker uses public API due to limited budget. Uploading files and getting responses from VirusTotal are quite time-consuming. Users may need to wait for an irregular time to retrieve the scanning results from VirusTotal due to the transmit limitations for developers. Table 1 shows some limitations of VirusTotal public API.

Table 1. Limitations of VirusTotal public Application Programming Interface (API).

Parameter	Setting
Privileges	public key
Request rate	4 requests/minute
Daily quota	5760 requests/day
Monthly quota	178,560 requests/month
Status	Key enabled

The scanning precedence of VirusTotal is determined by the API that a user utilizes. The private API has a higher priority than the public API. Two tips are strongly recommended by VirusTotal to save time and bandwidth efficiently. First, use hash values of files to retrieve file scan reports that already exist in the VirusTotal database. Second, rescan files that have already been sent by others in the past, because these files have already existed in VirusTotal, so these files do not need to be uploaded again, but time to wait for scanning these files is still needed. Thus, retrieving file scan reports before rescanning files that have already been sent is the most efficient way to save time and bandwidth. By using both tips, developers can retrieve file scan reports first, and then determine whether to rescan files or not in accordance with the report dates. If those reports are recent ones, files that have been rescanned can be ignored. Hence, time and bandwidth can be saved simultaneously.

3. System Design

The overall system architecture of Skywalker is shown in Figure 3. Letters (a)–(j) beside the arrows in the figure represent the execution order of related operation. Skywalker consists of three major components, *SSDT Hooker*, *Image Controller*, and *Image Handler*. Moreover, Skywalker consists of two phases. Phase 1 is SSDT hooking, implemented by *SSDT Hooker* and *Image Handler*. Phase 2 is image scanning, implemented by *Image Controller* and *Image Handler*.

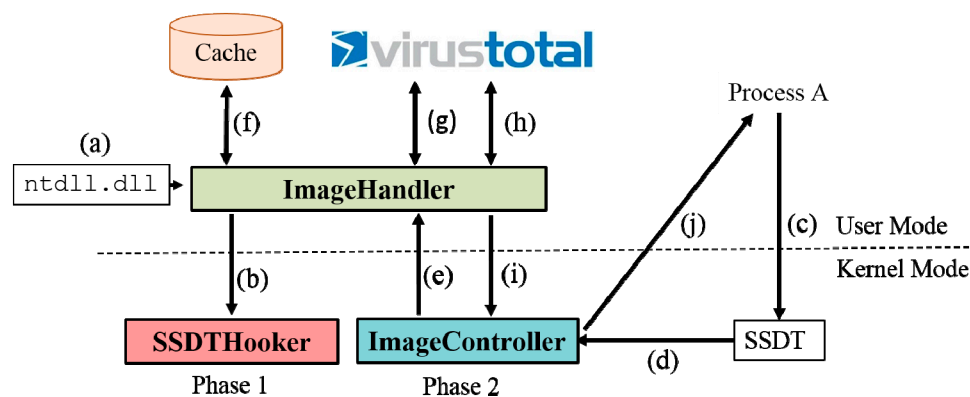


Figure 3. System structure and execution flow of major Skywalker components.

Component SSDT Hooker activates in phase 1. It is responsible for hooking Image Controller into SSDT. Skywalker uses a Windows driver to implement SSDT Hooker. SSDT Hooker clears the WP flag of cr0 register first. Then, SSDT Hooker receives the system call number and the address of stub function `ZwCreateUserProcess` from Image Handler. The system call number is an SSDT index from which we can obtain the location that stores the address of the hooked function `nt!NtCreateUserProcess`. Thus, the address of native API `nt!NtCreateUserProcess` in SSDT is replaced by our component, Image Controller. After the hooking, once a user clicks an executable image, the execution control flow of kernel is transferred to Image Controller before the executable is executed. Image Controller gets the full path name of the executable and sends the name to Image Handler, which in turn gets the image of the executable and sends it to VirusTotal for scanning. Finally, after getting the result from VirusTotal, if the executable is not malware, Image Handler notifies Image Controller to give the control to the `nt!NtCreateUserProcess` to create a process and execute the executable. Otherwise, Image Handler notifies Image Controller to block the execution of the executable and send the user a warning message.

Component Image Controller activates in phase 2. It is in charge of both the retrieval of the path name of an executable that is going to be executed and execution of the executable. When a user double clicks an executable file, Windows system calls stub function `ZwCreateUserProcess` first. Then, the system executes the hooded Image Controller. Image Controller uses a named pipe to communicate with Image Handler. A named pipe connection is created by `API PipeConnection`. After creating and setting a named Pipe, Image Controller can use `SendImagePathFromPipe` to send the path name of a target executable to Image Handler. Finally, `ReceiveOrderFromPipe` is invoked to wait for the response result from the Image Handler. If the result is 1, Image Handler passes all parameters that it receives from the stub function to native API `NtCreateUserProcess` and transfers the control flow to it. `NtCreateUserProcess` creates a new process to execute the executable. However, if the result is 0, it means the executable is malware. Hence, Image Controller blocks the execution of the executable and returns execution flow to the stub function. Image Controller also sends a warning message to the user.

Component Image Handler resides in user address space of a Skywalker process and takes different roles in phase 1 and phase 2. The main task in phase 1 is to pass the memory address of stub function `ZwCreateUserProcess` into SSDT Hooker. Image Handler loads `ntdll.dll` into the memory. The stub function `ZwCreateUserProcess` is exported from `ntdll.dll`, thus we can use `GetProcAddress` to obtain the memory address of it. Image Handler then uses `DeviceIOControl` to communicate with SSDT Hooker. The memory address is transmitted to SSDT Hooker in the kernel address space. Image Handler then enters into phase 2.

The main task of Image Handler in phase 2 is (1) to wait for Image Controller to send it the path name of an executable, (2) to retrieve the executable, (3) to calculate the hash value of the executable and check whether the executable has been sent to VirusTotal before. If the executable has been sent to VirusTotal, Image Handler uses its previous scan result to decide whether it can execute the executable. Otherwise, Image Handler sends the executable to VirusTotal for scanning and retrieves the final scan

report of the executable from VirusTotal. Finally, based on the report, Image Handler notifies Image Controller whether it can execute the executable.

As a named pipe server, Image Handler makes use of overlapped procedures [14,15] to service simultaneous links to numerous pipe clients [16]. Picture Trainer develops a set number of pipeline circumstances. Each pipe instance can be attached to a different pipeline client. Image Controls develops pipe customers to communicate with Image Handler. Image Controls creates pipe clients to communicate with Image Handler. Image Handler maintains three queues—scan queue, report queue, and done queue—to cooperate with VirusTotal. First, all pipe instances of a named pipe are in `CONNECTING_STATE`. If a pipe client successfully establishes a connection, its pipe instance enters into `READING_STATE`. In `READING_STATE`, a pipe client can retrieve the path name of an executable from the overlapped Input/Output(I/O) buffer. The path name is packaged to a scan request and sent into the scan queue, then the pipe instance is changed to `WRITING_STATE`. For this pipe instance, Image Hand keeps asking VirusTotal the positive value of the related executable file. The positive value will be discussed later.

A scan queue is implemented by a doubly linked list. It uses an object to store information of all executables that are ready to be scanned by VirusTotal. Image Handler uses API `VtScan` to send executables to VirusTotal. After VirusTotal receives an executable, VirusTotal arranges the file into its own scanning schedule. VirusTotal returns a response message to Image Handler. Image Handler retrieves the scan ID from the response and moves the object into its report queue.

The report queue of Image Handler is responsible for obtaining the reports from VirusTotal using scan IDs. If a scheduled executable is not scanned completely yet, VirusTotal returns a “queued for analysis” message to Image Handler. Thus, Image Handler has to request the report of an executable periodically until it obtains the result. In our prototype, the duration time we set to request a scan report automatically is 17 s, due to the restriction of the total scanning times of using public VirusTotal API. According to VirusTotal, a developer using the free service of VirusTotal only can make four scan requests per minute.

When Image Handler acquires a response from VirusTotal, it retrieves the positive value from the response and moves the related object into its done queue. A positive value represents the numbers of virus engines in VirusTotal that regard the executable as a malicious file. Skywalker sets a threshold to one-third for the positive value of a scanned file to identify whether a file is malicious or not. Hence, if the positive value of a file is zero or lower than the threshold value, Image Handler notifies Image Controller that Image Controller can execute the file. On the other hand, if the positive value of a file exceeds the threshold value, Skywalker regards this file as a suspicious file which could probably harm a user’s computer. Thus, Image Handler reports to the user immediately and asks the user whether he still wants to execute the file. The user can choose to execute the file or not. Finally, Image Handler sends the user decision to Image Controller. If the user insists on executing the file that VirusTotal has judged as a malicious file, Image Handler still sends a command to Image Controller to execute the file. Otherwise, Image Controller cancels the execution of the file.

4. Evaluation

Various experiments have been made to evaluate the effectiveness and efficiency of Skywalker. This section introduces our experimental environment and test files. Then it discusses the scan registration time and scan result time of executables. Finally, it discusses the total extra time, called total preparation time, that Skywalker takes to start the execution of a file. It also compares the total preparation time of an executable when it is first executed and the total preparation time of an executable when it has been executed before.

We implemented Skywalker on a Windows 7 32-bit computer, and the specification of our host is shown in Table 2. Moreover, we chose VirusTotal as our cloud-based scanner.

Table 2. Hardware specification.

Processor	Intel® Core™ i5-4430 CPU 3.00G HZ/3.00G HZ
Memory	DDR 1 GB
Disk	60 GB

There are 12 executables used in our experiments. Table 3 shows the names and file sizes of these files. We want to learn the time spent on scan requests and report requests. The file size is one of the main factors that decides the time, because it takes more time to transmit and scan a larger file. Hence, what kinds of files we choose are not important, but what sizes of files we choose may lead to different results. The sizes of test files vary between 0.018 MB and 17.405 MB.

Table 3. Information of test files.

	File Name	Size (MB)
1.	Rgui.exe	0.018
2.	Aero Color Show.exe	0.300
3.	Magnifier v2.4	0.245
4.	Fiddler.exe	1.099
5.	WinRAR.exe	1.375
6.	RandPass	1.442
7.	Enigeo v4.1.0	3.381
8.	Spotify.exe	6.400
9.	Cmake-gui.exe	9.359
10.	Filezilla	12.128
11.	Line.exe	13.167
12.	Evernote.exe	17.405

The major impact introduced by Skywalker to a system is the extra time, called total preparation time hereafter, to start an executable. When Skywalker is enabled, for an executable that is first executed, the total preparation time comes from two major sources, scan registration time and scan result time. Scan registration time consists of the time to retrieve and upload a local executable to VirusTotal and the time to wait for VirusTotal to send a preliminary report. The preliminary report contains a scan ID to identify a scan task. After obtaining a scan ID for an executable, Skywalker repeats using the scan ID to ask VirusTotal whether the final report of the executable is available. The time spending on the above operation is called scan result time. However, for an executable that is not executed for the first time, its total preparation time comes from the time to calculate the hash of the executable. The following subsections discuss the registration time, the scan result time, and total preparation time of test files.

4.1. Scan Registration Time

A scan task begins when Image Handle receives the path name of an executable from Image Controller. A scan task ends when Image Handler receives a preliminary report from VirusTotal. A preliminary scan report contains lots of important information, including file size, scan ID, scan date, and so on.

To get the scan registration time of various executables with different sizes, we sent each test file individually to VirusTotal 100 times and recorded the average scan registration time. Figure 4 shows the results. The file sizes of the test files vary from 0.018 MB to 17.405 MB. The average scan registration time changes from 2.34 s to 43.12 s. Figure 4 shows that the scan registration time highly depends upon the size of an executable. The larger a file size is, the longer its scan registration time takes. After all, it takes longer time to transmit and scan a larger file.

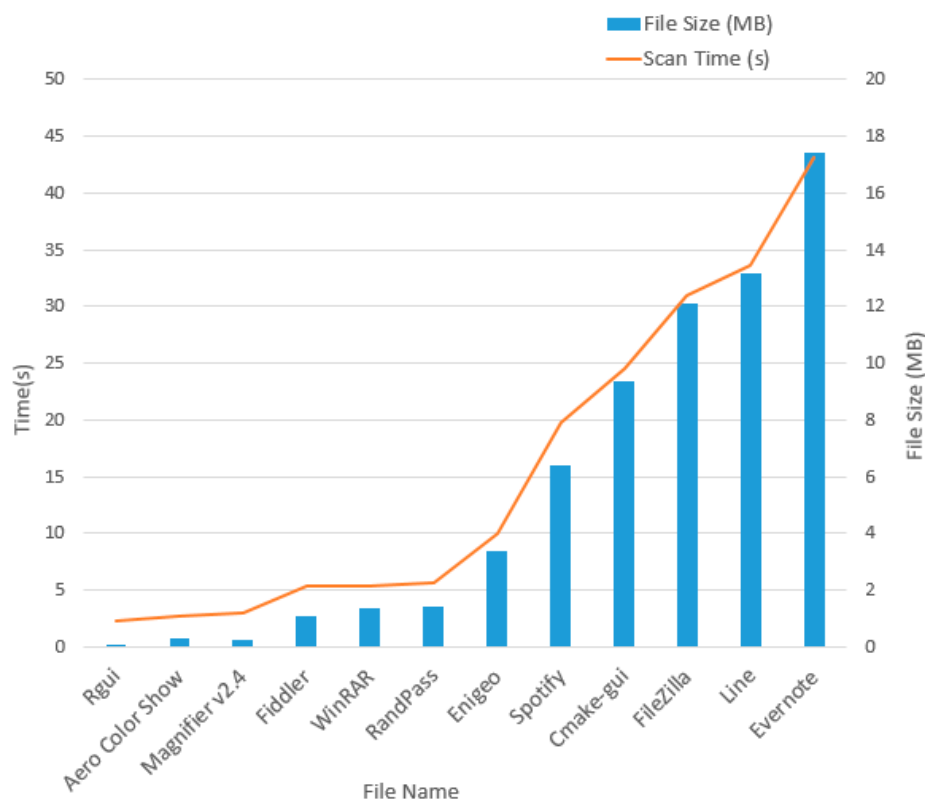


Figure 4. Average scanning time of various executables with different file sizes.

4.2. Scan Result Time

If a user uses the free VirusTotal service, usually the user needs to wait for a non-predictable time to get the final scan report for an executable. Hence, Skywalker will not stop sending a query to VirusTotal periodically with a scan ID for an executable until VirusTotal sends it the final scan report of the executable. A final scan report uses a positive value to show how many anti-virus engines think that the executable is malicious.

Similarly, to evaluate the scan result time of various executables with different sizes, we sent each test file individually to VirusTotal 100 times and recorded the average scan result time. As shown in Figure 5, Evernote.exe has the highest average scan result time, 184 s. WinRAR.exe has the lowest average scan result time, 152 s. The average scan report time of the 12 test files is 164 s, which is approximately equivalent to 2.7 min. For each test file, Figure 5 also shows the average number of queries that Skywalker made to get the final scan report.

In Figure 5, the test files are listed in ascending order from left to right according to their file sizes. The rightmost file, Evernote.exe, has the largest file size. The average scan result time of all 12 test files is over 140 s. The scan result time of the two smallest files, Rgui.exe and Aero Color Show.exe, is close to the scan result time of the largest file, Evernote.exe. Hence, we think that the average scan result time is uncorrelated with the file size. In fact, the average scan report time highly depends on the scheduling strategy of VirusTotal.

Figure 5 also shows the average numbers of queries that the 12 test files made to get their final scan reports. For instance, on average, Skywalker needs to send 11.53 queries to VirusTotal to get the final scan report of Rgui.exe. The average number of queries that all the 12 test files made to get their final scan reports is 10.6 times. In other words, each time when a user clicks an executable, on average, Skywalker requires querying VirusTotal 10.6 times to get the final scan report of the executable.

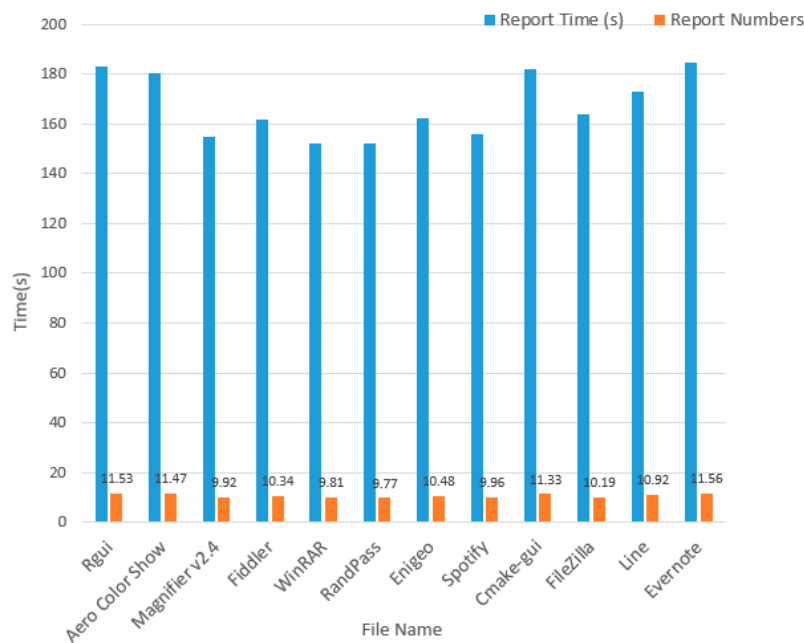


Figure 5. Average scan report time of 12 different test files and average numbers of queries that the 12 test files made to get their final scan reports.

4.3. Total Preparation Time

To get the total preparation time of Skywalker to start the execution of an executable, we recorded the time that Skywalker spent to start the execution of our test files. The total preparation time of a program starts at the time when Image Controller detects that the program is going to be executed. The total preparation time ends at the time when Image Handler sends a command to Image Controller to execute the file. We measured the total preparation time of each of the 12 test files 100 times and calculated the average time. As shown in Figure 6, Skywalker spends the least amount of time on Magnifier, and Skywalker spends the most amount of time on Evernote.exe. The average total preparation time of all the test files is 188.15 s, which exceeds over 3 min. Obviously, if a program is first executed, it takes Skywalker a long time to start it. However, this only occurs on a program that is first executed.

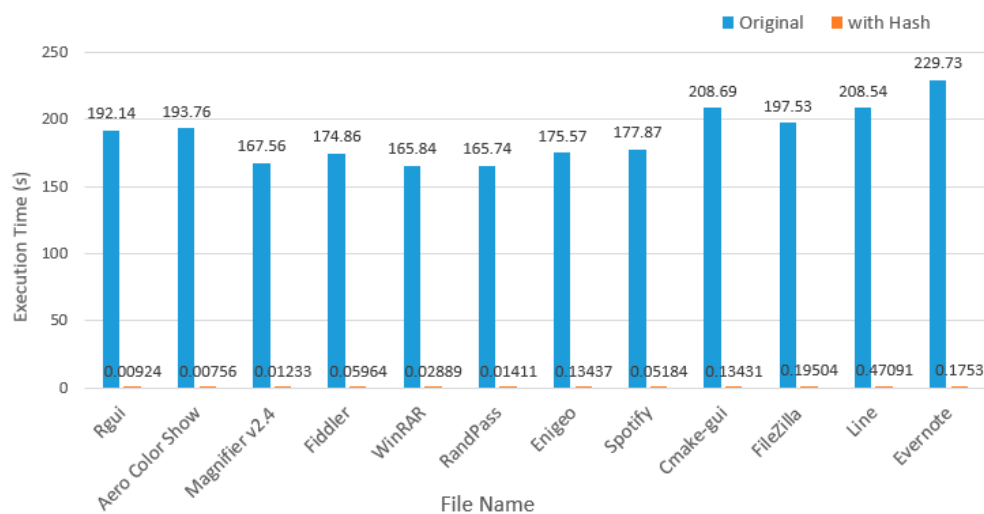


Figure 6. Total preparation time of an executable when the executable is first executed and total preparation time of the executable when it has been executed.

After calculating the hash value of a program, if Skywalker finds that the program has been executed before, Skywalker does not need to send the program to VirusTotal. As a result, the total preparation time of executables should decrease dramatically. Figure 6 confirms this inference. The total preparation time reduces the interval to between 0.00756 s and 0.47091 s.

4.4. Effective Evaluation

To analyze the effectiveness of Skywalker, 200 malware that we collected from the Internet were downloaded to a host with Skywalker. When clicking the malware to execute these malicious programs, Skywalker successfully blocked their execution and showed warning messages to the users.

4.5. Limitations

Experimental results that Skywalker is an effective and efficient solution. But it also has some limitations. First, Skywalker utilizes antivirus engines to detect malware. Hence, Skywalker cannot provide hosts protection that antivirus engines do not provide. Besides, when a program is going to be executed, it takes non-trivial extra time to start the program. A possible solution is to send a program to VirusTotal when it is installed in an end host; hence, when a user executes the program first time, he does not need to wait for a period of time. If Skywalker wishes to speed up the phrase of startup, we can conduct different balances in this trade-offs for this limitation. For instance, we can set an “offline mode” which uses larger memory in replace of time; that is, we may establish some meaningful data or patterns in the memory earlier than the startup stage. This is a tradeoff, just like the training phrase of supervisor learning in machine learning field. If Skywalker wants to gain greater precision or precision, we must need to invest other resources. Besides, hosts in a Local Area Network (LAN) can also share their VirusTotal scan results through hash values of programs. Hence, if a host in an LAN has submitted a program to VirusTotal for analysis, other hosts in the LAN can just use the scan result to decide whether it is safe to execute the program. Finally, as mentioned in Section 2, using the private API of VirusTotal for developers can also reduce the longer scanning time problem.

5. Related Work

Jon Oberheide [6] et al. first proposed and implemented a cloud service with multiple antivirus engines in 2008. Their system, called CloudAV [5], runs a lightweight process as a host agent to catch executables on user systems and send them to the network service they deployed, which contained 10 antivirus engines and two behavioral detection engines. However, CloudAV does not provide real-time detection for local hosts. It spends a long period of time on scanning files, as conventional antivirus software does, because a scheduled scan performed by conventional antivirus software is always time-consuming. Furthermore, only 10 antivirus engines were employed on CloudAV. On the contrary, Skywalker provides a real-time mechanism to reduce the time spent on cloud scanning and has a higher detection rate by taking advantage of VirusTotal.

MIDeA [17] is a multi-parallel intrusion detecting mechanism on a high-speed network. SEER [18] was another research project that aimed to leverage cloud resources to detect memory virus. Similar to Skywalker, it proposes a concept of virus scanning as a service to defeat modern malware efficiently. The cloud-based defense mechanism also extends to mobile security. Researchers like Xuesen Lin [19] gave a detailed discussion about cloud-based security and some frameworks in 2011.

6. Conclusions

Millions of malwares are created and rapidly spread all over the Internet every day. Malware bring severe security threats to every host in the world. In order to defeat malware, many antivirus engines have arisen and have gradually taken on an important role in protecting user computers. To fully utilize the advantages of antivirus engines while avoiding suffering its disadvantages, we designed Skywalker. Experimental results show that Skywalker has achieved the following four major goals. First, Skywalker raises the detection rate of the latest malicious content, because Skywalker use

multiple antivirus engine to analyze programs. Second, Skywalker gets rid of the time-consuming scanning work that is scheduled by an antivirus scheduler periodically. Third, Skywalker eliminates the overhead to maintain and update an antivirus engine. Last, Skywalker reduces the risks that may be introduced by antivirus vulnerabilities. Experimental results show that after a program has been executed, it takes Skywalker, at most, 0.47091 s to start its execution. Meanwhile, VirusTotal provides a secure protection to client hosts.

Author Contributions: Conceptualization, F.-H.H., and T.L.; methodology, F.-H.H., T.L. and M.-H.W.; software, T.L., C.-H.L. and M.-H.W.; validation, T.L., C.-H.L. and M.-H.W.; formal analysis, F.-H.H., T.L., C.-H.L. and M.-H.W.; investigation, F.-H.H., T.L., C.-H.L. and M.-H.W.; resources, F.-H.H., T.L., C.-H.L. and M.-H.W.; data curation, F.-H.H., T.L., C.-H.L. and M.-H.W.; writing—original draft preparation, F.-H.H., and T.L.; writing—review and editing, F.-H.H., C.-H.L., T.-C.C. and M.-H.W.; visualization, C.-H.L., T.-C.C. and M.-H.W.; supervision, M.-H.W. and T.-C.C.; project administration, F.-H.H., M.-H.W., and T.-C.C.; funding acquisition, F.-H.H., and T.-C.C.

Funding: This research was supported by grants from Recruiting High Level Talent program of Ningde Normal University (2018Y22), the Key Laboratory of Ecotourism and Leisure Agriculture of Fujian Province and the Big Data Institute of Rehabilitation and Nursing of Fujian Province for the Elderly.

Acknowledgments: Special thanks go to the reviewers for their valuable suggestions. The authors also would like to thank their College of Information and Mechanical & Electrical Engineering at Ningde Normal University for research design, etc.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Worldwide Security Software Market Grew 5.3 Percent in 2014. Available online: <https://www.gartner.com/en/newsroom/press-releases/2015-05-27-gartner-says-worldwide-security-software-market-grew-5-percent-in-2014> (accessed on 1 August 2019).
- Bishop, P.; Bloomfield, R.; Gashi, I.; Stankovic, V. Diversity for security: A study with off-the-shelf antivirus engines. In Proceedings of the 2011 IEEE 22nd International Symposium on Software Reliability Engineering (ISSRE), Hiroshima, Japan, 29 November–2 December 2011; pp. 11–19.
- Cyveillance, Malware Detection Rates for Leading AV Solutions. Available online: https://www.cyveillance.com/web/docs/WP_MalwareDetectionRates (accessed on August 2010).
- Hsu, F.H.; Wu, M.H.; Tso, C.K.; Hsu, C.H.; Chen, C.W. Antivirus Software Shield against Antivirus Terminators. *IEEE Trans. Inf. Forensics Secur.* **2012**, *7*, 1439–1447. [CrossRef]
- Oberheide, J.; Cooke, E.; Jahanian, F. CloudAV: N-Version Antivirus in the Network Cloud. In Proceedings of the USENIX Security Symposium, San Jose, CA, USA, 28 July–1 August 2008; pp. 91–106.
- Oberheide, J.; Cooke, E.; Jahanian, F. Rethinking Antivirus: Executable Analysis in the Network Cloud. In Proceedings of the HotSec, Boston, MA, USA, 6–10 August 2007.
- Koret, J. Breaking Antivirus Software. In Proceedings of the Symposium on Security for Asia Network (SyScan), Singapore, 31 March–3 April 2014.
- VirusTotal on Wiki. Available online: <http://en.wikipedia.org/wiki/VirusTotal> (accessed on 1 August 2019).
- Blunden, B. *The Rootkit Arsenal*; Jones&Bartlett: Burlington, MA, USA, 2009.
- Jogie, N. Rootkit Analysis: Hiding SSDT Hooks. 2010. Available online: <https://securabit.com/wp-content/uploads/2010/03/Rootkit-Analysis-Hiding-SSDT-Hooks1> (accessed on 1 August 2019).
- Lukan, D. Hooking the System Service Dispatch Table (SSDT). Available online: <http://resources.infosecinstitute.com/hooking-system-service-dispatch-table-ssdt/> (accessed on 1 August 2019).
- VirusTotal. Available online: <https://www.virustotal.com/> (accessed on 1 August 2019).
- VirusTotal Public API in C. Available online: <https://github.com/VirusTotal/c-vtapi> (accessed on 1 August 2019).
- Microsoft. Named Pipe Server Using Overlapped I/O. Available online: <https://msdn.microsoft.com/en-us/library/windows/desktop/aa365603%28v=vs.85%29.aspx> (accessed on 1 August 2019).
- Hart, J.M. *Windows System Programming*, 4th ed.; Addison-Wesley: Boston, MA, USA, 2010.
- Microsoft. Named Pipe Client. Available online: <https://msdn.microsoft.com/en-us/library/windows/desktop/aa365592%28v=vs.85%29.aspx> (accessed on 1 August 2019).

17. Vasiliadis, G.; Polychronakis, M.; Ioannidis, S. MIDeA: A multi-parallel intrusion detection architecture. In Proceedings of the 18th ACM Conference on Computer and Communications Security, Chicago, IL, USA, 17–21 October 2011; pp. 297–308.
18. Gionta, J.; Azab, A.; Enck, W.; Ning, P.; Zhang, X. SEER: Practical memory virus scanning as a service. In Proceedings of the 30th Annual Computer Security Applications Conference, New Orleans, LA, USA, 8–12 December 2014; pp. 186–195.
19. Lin, X. Survey on cloud based mobile security and a new framework for improvement. In Proceedings of the 2011 IEEE International Conference on Information and Automation (ICIA), Shenzhen, China, 6–8 June 2011; pp. 710–715.



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).