# Detecting & Defeating Split Personality Malware

Kalpa Vishnani, Alwyn R. Pais, Radhesh Mohandas
National Institute of Technology Karnataka, India
emails:kalpavishnani@gmail.com, alwyn.pais@gmail.com, radhesh@gmail.com

***Abstract*— Security analysts extensively use virtual machines to analyse sample programs and study them to determine if they contain any malware. In the process, if the malware destabilizes the guest OS, they simply discard it and load in a fresh image. This approach increases their productivity. Since naive users do not run virtual machines, malware authors have observed that it is a pretty good probability that their malware is being analysed if it is being run in a Virtual Machine (VM). When these analysis aware malware detect the presence of VMs, they behave in a benign manner thus escaping detection. A determined analyst will have to end up running the sample on a native machine that adds to his chase time. In this paper, we briefly discuss the techniques deployed to detect VM by the Analysis Aware Malware also known as the Split Personality Malware. We then introduce our tool that not only detects this category of malware but also fools it into believing that it is running on a native machine even when it is running on a virtualized one, forcing it to exhibit its malicious form. Most security analysts should find this tool really useful.**

***Keywords- Detecting Virtual Machines, Vmware, Analysis Aware Malware, Split Personality Malware, guest OS, host OS.***

## I. INTRODUCTION

Security researchers and analysts use a wide variety of tools to carry out malware analysis. Virtualization has emerged as a very useful technology in the field of security research and has gained widespread acceptance in the fraternity. It is very popular amongst malware researchers since they can intrepidly execute suspicious malware samples on the virtual machines without having their systems affected. Since many malware tend to destabilize the host systems, allowing them to run in a virtual environment increases the productivity of the analysts. This decreases the time and cost that the analysts need to study malware behaviours enabling them to build patches against the vulnerabilities that the malware exploit.

However, the malware developers have once again upped the ante by adding analysis awareness functionality into their malware. They detect the presence of malware analysis tools such as Virtual Machines (VM), debuggers and sandboxes and then either terminate execution or hide their malicious nature by executing like a benign application. As a result, they escape detection from a casual malware analyst. This category of malware is known as Analysis Aware malware or Split Personality malware.

The main subject of this paper is to tackle this class of malware. Current efforts mainly focus on flagging the Split Personality malware and once flagged they resort to analyzing them on a native machine to bring out their malicious nature. In this paper, we discuss our novel approach using which we detect the VM detection attempts and further trick the malware into believing that they are running on a host OS and hence make them exhibit their non-benign nature. We have developed a tool, VMDetectGuard for this purpose. We present the effective results obtained by means of this tool.

Our tool is currently built for VMware running the Windows platform. However, it is important to note that the solution we provide here is generic and can be easily tailored to cater to other OS platforms as well as VMs.

The remainder of this paper is organized as follows. Section 2 discusses the different VM detection techniques. In Section 3, we discuss related work and highlight their shortcomings. In Section 4, we present our approach for combating the VM-detecting Split Personality malware. In Section 5, we discuss the implementation details of our solution, VMDetectGuard. In Section 6, we present the analysis results obtained by running various VM detecting malware samples (both proof of concept and live malware) in the presence as well as in the absence of VMDetectGuard and noting down the behavioral changes in the malware. In Section 7, we conclude.

## II. VM DETECTION TECHNIQUES

There are various ways of VM detection, all of which can be classified in one of the following categories:

### A. Hardware Fingerprinting

Hardware Fingerprinting involves looking for specific virtualized hardware [1]. It can reveal a plethora of information about VM specific components required for reliable detection. In Table I, we have included the results of hardware fingerprinting which we obtained on a host OS and on a guest OS running on VMware. We carried out this fingerprinting using Windows Management Instrumentation (WMI) classes and APIs [2]

### B. Registry Check

The registry entries contain hundreds of references to the string "VMware" in the guest OS. Checking the registry values for certain keys clearly reveals the VM presence [1]. The following are a few examples:

```
HKEY_LOCAL_MACHINE\HARDWARE\DEVICEMAP\Scsi\Scsi
Port1\Scsi Bus 0\Target Id 0\Logical Unit Id
0\Identifier
  → VMware, VMware Virtual S1.0
```

 ISBN: 978-1-61208-010-9 

```
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control
\Class\{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000\DriverDesc
  → VMware SCSI Controller

HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control
\Class\{4D36E968-E325-11CE-BFC1-
08002BE10318}\0000\ProviderName
  → VMware, Inc.
```

### C. Memory Check

This technique involves looking at the values of specific memory locations after the execution of instructions such as SIDT (Store Interrupt Descriptor Table), SLDT (Store Local Descriptor Table), SGDT (Store Global Descriptor Table), and STR (Store Task Register) [1][3]-[7]. It is the most widespread detection technique employed by the present day VM detecting malware.

### D. VM Communication Channel Check

This check involves detecting the presence of a host-guest communication channel. The IN instruction is a privileged instruction which when executed from ring 3 of a protected mode OS such as Windows, raises the exception "EXCEPTION PRIV INSTRUCTION"' [1]. However, when it is running on VMware, no such exception is generated. Instead, VMware initiates guest to host communication by calling the 'IN' instruction. If the magic number ('VMXh') is returned to the register EBX, then it is certain that the program is running inside VMware.

### E. Timing Analysis

An obvious yet rare attack against a Virtual Machine is to check a local time source, such as the "Time Stamp Counter" (TSC). We briefly restate the concept behind this attack discussed in a previous work [5].

Translation Lookaside Buffers (TLBs) can be explicitly flushed out and then the time to access a new page is determined by reading the TSC before and after the access. This duration can be averaged out over the number of TLBs to be filled. Next, the TLBs are filled with known data by accessing a set of present pages and the time to access a cashed page is determined as before. This value can also be averaged over the number of pages in the TLBs. Now, the CPUID instruction is executed. CPUID is the only VM sensitive instruction which on execution flushes out at least some of the TLBs as a side effect. Now each of the pages that were present in the VM is accessed again. If any of the page's access time matches that of a new page, the presence of a VM is revealed!

### F. Process & File Check

There are many VMware specific processes such as VMwareUser.exe, vmacthlp.exe, VMwareService.exe, VMwareTray.exe that constantly run in the background. There also exist some VMware specific files and folders [1]. Hence querying for these objects could also serve as a method for VM detection. Though this method could easily be fooled, when combined with other detection techniques, it could obtain more reliable results.

## III. Related Work

We found that the amount work done for the containment of Split Personality malware is not substantial. Very few researchers have provided solutions to counter the same. Moreover, most of them have focussed only on the detection of this class of malware [8][9][10]. Once this class of malware is detected, they propose to further analyse these malware on a host OS. The only approaches we found that aim at tricking the malware are proposed by Carpenter et al. [11] and Guizani et al [12].

Zhu & Chin [9] discuss two approaches to counter VM-aware malware. One approach professes the use of dynamic analysis to identify known virtual machine detection techniques. The authors have built an implementation for it called "Malaware". This is a mere detection approach. Once the malware is detected it is to be further analysed on a native machine. In the second method they propose the use of dynamic taint tracking to detect any impact caused by the input that changes the execution path of the malware. Although the authors claim that this approach will help to detect the already unknown VM detection techniques, we beg to differ. In this second approach they have addressed only two of the various VM detection techniques, Memory Check and Registry Check. Out of these, for countering Registry Check detection method they propose that, a check should be made to determine if the sample contains any conditional jump statement following a registry query. If so, they conclude that the sample is probably taking another execution path because it detected the presence of virtual machine. We further argue that this is not a good heuristic as a sample will not always be a split personality malware if it has a conditional statement after a registry access. Even a legitimate application could do that for other genuine reasons. For instance, the commercial software with trial periods have to extensively make use of this logic in order to check the registry values to see if the software has been registered by the user or not. If not then it must run in the trial mode. Moreover, this method does not give any solutions for other types of VM detection techniques such as Hardware Fingerprinting and Timing Analysis, both of which are gradually being adopted by the advanced Split Personality malware.

Carpenter et al. propose [11] two mitigation techniques. They aim at tricking the malware by, 1) changing the configuration settings of the .vmx file present on the host system and, 2) altering the magic value to break the guest-host communication channel. Out of these two techniques the first one has the following setbacks:

- The configuration options break the communication channel between guest and host not

just for the program trying to detect the VM, but for all the programs.

- Moreover the authors claim that these are undocumented features and that they are not aware of any side effects.

Their second technique is targeted only against VM Communication Channel Check method.

The work by Guizani et al. [12] provides an effective solution for Server-Side Dynamic Code Analysis. A small part of their solution deals with tricking the Split Personality malware employing Memory Check and VM Communication Channel detection techniques. However they do not address other detection techniques. It was their work that inspired us to build a complete solution for the containment of the all the VM detection methods and provide a more complete and robust solution.

The approach mentioned in the work by Balzarotti et al. [10] involves first running the sample on the reference system (physical system), logging its input and output values exchanged with the system and then running the same sample on the analysis system which runs a virtual environment where the output values, that were obtained on the reference system are simply replayed. Then, the differences in the sample's behaviour are observed. Thus, in this work too, the entire analysis of the detected Split Personality Malware is not carried out in the virtualized environment.

Hence we conclude that there does not exist any complete solution that effectively counters Split Personality malware.

## IV. OUR APPROACH

The main objective of this paper is to carry out the analysis, detection and containment of the Split Personality malware entirely on the virtualized system. We perform dynamic binary instrumentation of the sample under test in order to obtain its low level information as well as to intercept all the API calls made by it. We then check to see if the sample is trying to access any information which would help it in determining the VM presence. If a match is found with any of our monitored set of API calls or low level instructions, our tool logs the activity and provides fake values to the sample so as to make it feel that it is running on the native system. Fig. 1 illustrates the approach step by step.

**Step 1:** *Maintain a list of all the hardware as well as registry querying API calls. Also maintain a list of all the VM specific instructions such as SIDT, SLDT, SGDT, STR, IN.*

Following is a partial list of API calls to be monitored.

a) Hardware Querying APIs
   i) SetupDiEnumDeviceInfo()
   ii) SetupDiGetDeviceInstanceId()
   iii) SetupDiGetDeviceRegistryProperty()
   iv) WMI APIs

b) Registry Querying APIs
   i) RegEnumKey()
   ii) RegEnumValue()
   iii) RegOpenKey()
   iv) RegQueryInfoKeyValue()
   v) RegQueryMultipleValues()
   vi) RegQueryValue()

**Step 2:** *Perform dynamic binary instrumentation of the sample under test in order to obtain its low level information as well as to intercept all the API calls made by it.*

We perform dynamic binary instrumentation of the sample using the Pin framework [13]. It allows for monitoring all the API calls and low level instructions being executed by the sample.

**Step 3-12:** *Check to see if the sample under test makes a call or executes any of the monitored API calls or instructions respectively. If a match is found, set the OUTPUT to "Split Personality Malware Detected". Also, log the activity and provide fake values to the sample so as to make it feel that it is running on a host system.*

Let us consider the example of a sample that makes the following API calls with the given arguments:

```
RegOpenKeyEx(
       HKEY_LOCAL_MACHINE,
       TEXT("HARDWARE\\DEVICEMAP\\Scsi\\Sc
       si Port 0\\Scsi Bus 0\\Target Id
       0\\Logical Unit Id 0"),
       0,
       KEY_QUERY_VALUE,
       &hKey);

RegQueryValueEx(
       hKey,
       TEXT("Identifier"),
       NULL,
       NULL,
       (LPBYTE) PerfData,
       &cbData );
```

In the above case, the key value returned in a VMware machine will contain the string "VMware". Thus, we monitor the values returned by the OS in response to the API calls made by the sample. If it contains the string "VMware", the control passes to our replacement routine where we change the value to a more appropriate value such as "Miscrosoft" or to a value that would have been returned on a host Windows OS.

Similarly when VM specific instructions such as SIDT are at the verge of being executed by the sample, the control passes to our replacement routine where we set the value of the destination operand to a value that would be obtained on the host Windows OS.

 ISBN: 978-1-61208-010-9 

**Algorithm 1** VMDetectGuard to detect and trick Split Personality Malware

**Input:** Malware sample to be tested.
**Output:** Boolean Result: OUTPUT indicating Split Personality malware detected/not detected.

```
1: Maintain a list of API calls and low level instructions that help in VM Detection.
2: Run the sample under test.
3: Hook into the sample.
4: Set OUTPUT to NULL.
5: while the sample executes do
6:    Intercept the API calls and low level instructions being executed by the sample.
7:    if match is found with the monitored set of API calls or low level instructions.
      then
8:       Log the activity.
9:       Set the OUTPUT to Split Personality malware detected.
10:      Provide fake values to the running sample.
11:   else
12:      Do nothing.
13:   end if
14: end while
```

Figure 1. Our Approach for Countering Split Personality Malware

## V. IMPLEMENTATION

We have designed and implemented a solution to counter Split Personality malware that employ the various VM detecting techniques. In this section we present a detailed discussion of our implementation, VMDetectGuard.

We implemented our solution in the framework provided by the Pin tool [13] released by Intel Corporation. Pin is a tool for the instrumentation of programs. Pin allows a tool (such as ours) to insert arbitrary code in arbitrary places in the executable. The code is added dynamically while the executable is running.

### A. Methodology

As we stated earlier, all the VM detection methods fall under one or more categories of VM detection discussed in Section 2, we present our implementation methodology with respect to each VM detection category.

#### i) Countering Hardware Fingerprinting

We propose hardware emulation. The idea is to maintain a list of all the API calls that provide hardware information such as BIOS, Motherboard, Processor, Network Adapter etc. such that even if false values are supplied about them to a ring 3 application querying such information, the application would not crash. We created a proof of concept program to carry out hardware fingerprinting of a native as well as a virtual machine. Table I summarizes the results.

VMDetectGuard hooks into the sample under analysis and monitors the API calls it makes. Whenever a match is found with our set of monitored API calls, it logs this activity and provides fake values to the sample. In Table I we see how VM returns a value “none” for motherboard serial number. VMDetectGuard returns a more appropriate string such as “.16LV3BS.CN70166983G1XF” instead. Each time a match with the monitored set of APIs is found, our tool empowers the analyst who can choose either to modify or not to modify the values being returned to the sample, thus enabling him to notice any changes in the sample's behaviour.

**Caveat:** There are certain hardware components that cannot be emulated. For instance, the MAC address cannot be faked because the program requesting this value would be unable to carry out the desired networking tasks. In this case, we urge the malware analysts to change their MAC address in their VMware machine so that it does not match the VMware MAC address pattern. The guidelines for this are provided on the VMware forums [14].

#### ii) Countering Registry Check

VMDetectGuard also monitors registry querying APIs such as RegQueryInfoKeyValue, RegOpenKey, etc. It intercepts these API calls whenever they are executed by the sample. It then looks at the output values returned by the system. If the output contains the string "VMware", our tool replaces this string with a value that would have been returned on a non virtual system running the same OS.

#### iii) Countering Memory Check

For countering memory check we detect the presence SIDT, SLDT, and SGDT and STR instructions.

VMDetectGuard logs the activity whenever any of the above instructions is at the verge of being executed by the malware sample. It also appropriately modifies the values of the registers that are affected by these instructions after their execution making the sample feel that it is running on a native system.

Table II shows the different values obtained on a VMware and a host machine respectively on executing the above mentioned instructions.

TABLE II. VALUES OBTAINED ON EXECUTING MEMORY CHECK INSTRUCTIONS ON VMWARE AND HOST MACHINE (WINDOWS)

| Instruction | VMware | Host machine |
|---|---|---|
| SIDT | IDT is located typically at 0xffXXXXXX | IDT is located at a location lower than that. Around 0x80ffffff. |
| SLDT | Not located at 0xdead0000 | Located at 0xdead0000 |
| SGDT | GDT is located typically at 0xffXXXXXX | GDT is located at a location lower than that around 0x80ffffff. |
| STR | Selector segment value of TR register is value other than 0x40000000 | Selector segment value of TR register is 0x40000000 |

#### iv) Countering VM Communication Channel Check

We address this check in a way similar to countering Memory Check. We monitor execution of the IN instruction, and change the value of the magic number

 ISBN: 978-1-61208-010-9 

('VMXh') that was supplied as an input parameter by the sample under test to some other value.

*v) Countering Timing Analysis*

Our tool monitors the sample for instructions such as CPUID and RDTSC (Read Time Stamp Counter). Moreover it maintains the count of each type of instruction executed. So if a particular instruction is executed a large number of times which is above the threshold value for that type of instruction, it logs this activity too. This is because timing attacks are known to execute a single or a couple of instructions for a very large number of times as certain instructions when run for a large number of times on a virtualized system take considerably longer than on a native machine to execute. Such attacks also make use of the CPUID instruction. We counter this detection method by deleting the CPUID instruction just before its execution and then modifying the values of the general purpose registers that are affected by the CPUID instruction (ebx, ecx, edx).

*vi) Countering File & Process Check*

These checks are countered in the similar way as the Registry Check. APIs for File/Folder/Process queries are monitored. If the sample makes querying request for VMware files, folders or processes, the tool sends out the 'file/process not found' error.

Thus our tool takes complete control of the sample and governs the output values to be fed to it.

### B. VMDetectGuard Output

VMDetectGuard produces various log files along with the Boolean Result: Split Personality malware detected/ not detected.

It generates instruction trace, system call trace, instruction count log, opcode mix log as well as a VM specific log. This VM specific log contains all the API calls as well as the low level instructions that were executed by the sample under test. In case the sample is not a Split Personality malware, the VM specific log remains empty. All these logs can be used for further analysis of the sample.

## VI. Results & Analysis

In order to test the effectiveness of our tool VMdetectGuard, we ran various VM detecting malware samples (both, proof of concept samples and live malware captured from the internet) on VMware in the presence as well as absence of VMDetectGuard; to observe if there were any notable changes in their behaviour. Table III summarizes the results of our analysis.

Fig. 2 and Fig. 3 illustrate the changes in the behaviours of redpill.exe [6] and scoopyNG.exe [7] respectively when ran on VMware in the presence and absence of VMDetectGuard respectively. It can be seen how VMDetectGuard fools both the binaries into believing that they are not running on a Virtual Machine.

We also analysed some samples of live malware captured from the internet. Amongst these, Backdoor.Win32.SdBot.fmn was found to employ both, Timing Analysis as well as Memory Check. When run in the absence of VMDetectGuard, the application displays a message, "Sorry, this application cannot run in a Virtual Machine". However on running it in the presence of VMDetectGuard, it runs and ultimately shuts the instance of OS running on VMware! While analyzing the logs generated by this malware sample we noted that it executed RDTSC 487 times, CPUID once. It also executed SIDT and SLDT instructions. But since our tool provided it with fake values it continued to act malicious and ultimately shut the OS. By means of VMDetectGuard, we also obtained its low level trace as well as system call trace for further analysis. Fig. 4 shows how Backdoor.Win32.SdBot.fmn refuses to run in a virtual machine when run in the absence of VMDetectGuard. Fig. 5 is a snapshot of the low level information of Backdoor.Win32.SdBot.fmn obtained while tricking it using VMDetectGuard. It shows the use of the Memory Check method (SLDT instruction) made by the malware sample.

## VII. Conclusion

Split Personality malware is on a gradual rise and proactive measures are necessary to curb them before they become uncontrollable.

We found lack of research in this field. Moreover there does not exist any full-fledged tool to counter Split Personality malware.

We have designed and implemented VMDetectGuard, a tool that detects as well as tricks Split Personality malware. Our experimental results demonstrate that the tool effectively detects as well as tricks the split personality binaries leading to their effective analysis in the virtualized environment.

Although we have tested VMDetectGuard for several VM Detecting malware, we are still in the testing phase to ensure the completeness of our solution. Moreover, we are yet to carry out its performance evaluation to make it more efficient. We are working on it.

Our solution is currently built for VMware and Windows OS. We now seek to extend the support to other Operating Systems as well as Virtual Machines such as VirtualBox, Virtual PC, Xen, Hydra, Qemu etc. Similar techniques can also be used to counter anti-debugging tricks.

 ISBN: 978-1-61208-010-9 

TABLE I COMPARISON RESULTS OF HARDWARE FINGERPRINTING OBTAINED ON A WINDOWS VIRTUAL AND A NATIVE MACHINE RESPECTIVELY

| **Hardware component** | **Attribute queried** | **VMware** | **Native Machine** |
|---|---|---|---|
| Motherboard | Serial No. | None | .2GTP3BS.CN7016697MG1DN. |
| Processor | SocketDesignation | CPU Socket #0 | Microprocessor |
| SCSI Controller | Caption | VMware SCSI Controller | Microsoft iSCSI Initiator |
| BIOS | Serial Number | VMware-56 4d 68 4c f9 e5 62 f4-fb 4d f0 5b 88 28 29 d9 | 2GTP3BS |
| USB Controller | Caption | 1. Intel(R) 82371AB/EB PCI to USB Universal Host Controller<br>2. Standard Enhanced PCI to USB Host Controller | 1. Intel(R) ICH9 Family USB Universal Host Controller – 2936<br>2. Intel(R) ICH9 Family USB Universal Host Controller – 2938<br>3. Intel(R) ICH9 Family USB Universal Host Controller – 2937 |
| Network Adapter | Caption | 1. VMware Accelerated AMD PCNet Adapter | 1. WAN Miniport (SSTP)<br>2. WAN Miniport (IKEv2)<br>3. WAN Miniport (L2TP) |
| Network Adapter | Mac Address | 00:0C:29:28:29:D9 (This MAC address falls in VMWare Mac Address Range) | 50:50:54:50:30:30 |

TABLE III SPLIT PERSONALITY MALWARE ANALYSIS RESULTS OBTAINED USING VMDETECTGUARD

| **No.** | **VM detecting program sample** | **VM detection method employed** | **VMware run with VMDetectGuard turned off** | **VMware run with VMDetectGuard turned on** |
|---|---|---|---|---|
| 1 | RedPill [6] | Memory | Detected VMware | Could not Detect VMware |
| 2 | ScoopyNG [7] | Memory | Detected VMware | Could not Detect VMware |
| 3 | VmDetect [15] | Memory | Detected VMware | Could not detect VMware |
| 4 | Worm.win32.autorun.pga | Timing Analysis | Displayed message saying "not a valid win32 application" | Ran maliciously |
| 5 | Trojan-Spy.Banker.pcu | Memory | Immediately terminated execution | Ran maliciously |
| 6 | Trojan-Spy.Win32.Bancos.zm | Memory | Ran benignly | Ran maliciously |
| 7 | Backdoor.Win32.SdBot.fmf | Memory | Ran benignly | Ran maliciously |
| 8 | Backdoor.Win32.SdBot.fmn | Memory, Timing Analysis | Displays a message, "This application cannot run under a Virtual Machine" | Ran maliciously |

 ISBN: 978-1-61208-010-9 

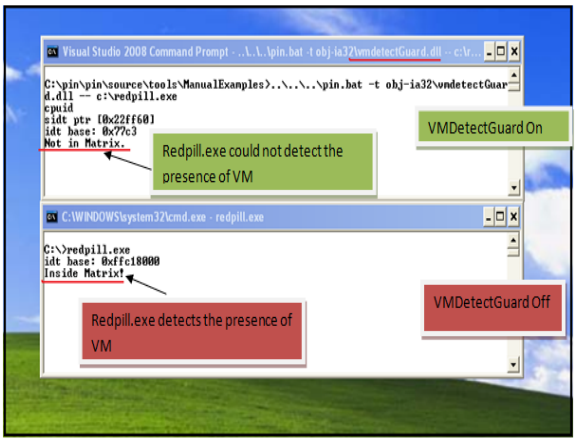



Figure 2. Redpill.exe executed in the presence and absence of VMDetectGuard resp.

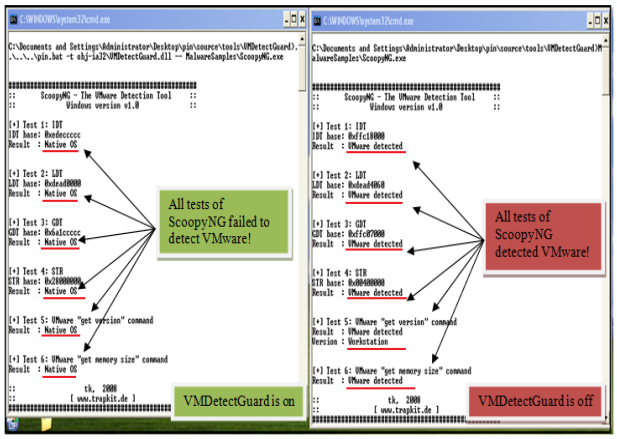



Figure 3. ScoopyNG.exe executed in the presence and absence of VMDetectGuard resp.

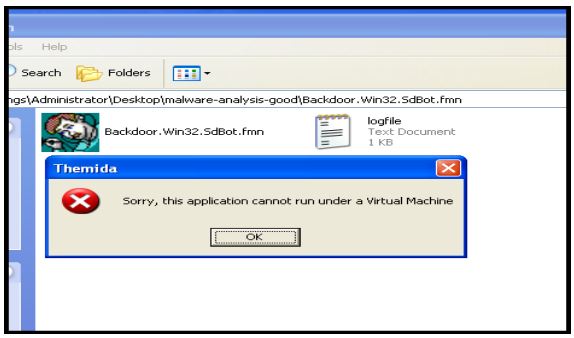
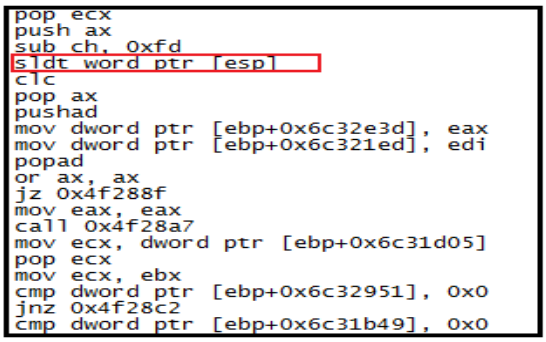



Figure 4. Backdoor.Win32.SdBot.fmn run in the absence of VMDetectGuard



```
pop ecx
push ax
sub ch, 0xfd
sldt word ptr [esp]
clc
pop ax
pushad
mov dword ptr [ebp+0x6c32e3d], eax
mov dword ptr [ebp+0x6c321ed], edi
popad
or ax, ax
jz 0x4f288f
mov eax, eax
call 0x4f28a7
mov ecx, dword ptr [ebp+0x6c31d05]
pop ecx
mov ecx, ebx
cmp dword ptr [ebp+0x6c32951], 0x0
jnz 0x4f28c2
cmp dword ptr [ebp+0x6c31b49], 0x0
```

Figure 5. Low level information of ackdoor.Win32.SdBot.fmn obtained while tricking it using VMDetectGuard

## REFERENCES


[1] Liston T. and Skoudis E. (2006). "On the Cutting Edge: Thwarting Virtual Machine Detec-tion" [Online]. Available: http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf (Nov 1, 2010)

[2] WMI Classes (2008). "WMI Classes Windows" [Online]. Available: http://msdn.microsoft.com/en-us/library/aa394554%28v=vs.85%29.aspx (Dec 30, 2010)

[3] Quist D. and Smith V. (2005). "Detecting the Presence of Virtual Machines Using the Local Data Table" [Online] Available: http://www.offensivecomputing.net/files/active/0/vm.pdf (Nov 14, 2010)

[4] Omella A. (2006). "Methods for Virtual Machine Detection" [Online]. Available: http://www.s21sec.com (Nov 24, 2010)

[5] Ferrie P. "Attacks on Virtual Machines". In the Proceedings of the Association of Anti-Virus Asia Researcher Conference, 2007.

[6] Rutkowska J. (2004). "Red Pill" [Online]. Available: http://invisiblethings.org/papers/redpill.html (Nov 4, 2010 )

[7] Klein T. (2005). "Scooby Doo - VMware Fingerprint suite" [Online]. Available: http://www.trapkit.de/research/vmm/scoopydoo/index.html (Nov 20, 2010)

[8] Lau B. and Svajcer V. "Measuring virtual machine detection in malware using DSD tracer". In the Proceedings of Virus Bulletin, 2008, pp. 181-195.

[9] Zhu D. and Chin E. (2007). "Detection of VM-Aware Malware" [Online]. Available: http://radlab.cs.berkeley.edu/w/uploads/3/3d/Detecting_VM_Aware_Malware.pdf (Dec 10, 2010)

[10] Balzarotti D., Cova M., Karlberger C., Kruegel C., Kirda E., and Vigna G. "Efficient Detection of Split Personalities in Malware". In the Proceedings of 17th Annual Network and Distributed System Security Symposium (NDSS 2010), Feb 2010

[11] Carpenter M., Liston T., and Skoudis E. "Hiding Virtualization from Attackers and Malware". IEEE Security and Privacy, June 2007, pp. 62-65.

[12] Guizani, W., Marion J.-Y., and Reynaud-Plantey D. "Server-Side Dynamic Code Analysis". Analysis, 2009

[13] Pin (2004). "Pin - A Dynamic Binary Instrumentation Tool" [Online]. Available: http://www.pintool.org/ (Jan 10, 2010)

[14] VMware (2010), "VMware KB: Changing a MAC address in a Windows virtual machine" [Online]. Available: http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1008473 (Jan 15, 2010)

[15] VmDetect (2005), "Detect if your program is running inside a Virtual Machine - CodeProject" [Online]. Available: http://www.codeproject.com/KB/system/VmDetect.aspx (Jan 4, 2010)


 ISBN: 978-1-61208-010-9 