VIRTUALIZATION RESISTANT MALWARE BEHAVIOR ANALYSIS

Manish Choudhary
Georgia Institute of Technology

Abstract

Malware is one of the most dangerous threats to the cyber world and cyber economy. Malware analysis is performed to analyse the behavior of the malicious code for the purpose of detection and prevention of similar attacks in future. Malware authors have started to implement software armoring techniques like anti-debugging, anti-disassembly and anti-virtualization to evade detection and analysis. The objective of this study is to analyse the behavior of such anti-analysis and virtualization resistant malware. Anti-debugging and anti-VM techniques implemented by the modern crypters have been analysed in detail.

1. INTRODUCTION

Malware, a malicious piece of software, is plaguing the network with threats like phishing, spamming, personal information theft, botnet and denial of service attacks. Malware is a growing threat with hundreds of thousands of new samples reported per day. Collection, analysis and reverse engineering the malware samples are integral part of strategical combat against these threats. Malware analysis provides information about the behaviour of the malware, how to measure and contain its damage and the detection patterns to generate signatures and cleanup tools. There have been various tools and technologies produced to perform static and dynamic malware analysis. The most important of these are debuggers, disassemblers and virtual environments. While on one side, disassemblers are used for the static analysis to understand the malicious code, on the other side, debuggers are the tools used to perform dynamic analysis to learn the behavior of malware by executing it in a controlled environment. Virtualization has emerged as a very useful technology in the field of security research for anti-malware researchers for analysing the behaviour of unknown malware samples. It provides a safe way for malware analysis as once the analysis is complete, the virtual environment can be reverted or destroyed without any risk to the host.

As more security researchers rely on virtual machine emulators and the analysis tools, malware authors have started implementing software armoring techniques as packing, runtime obfuscation, virtual machine and debugger detection. Modern malware contains a myriad of anti-debugging, anti-instrumentation, anti-disassembly and anti-VM techniques to evade detection. The environment aware malware are sensitive to the presence of emulators and other tools, and thus, behaves in a benign manner or even refuse to run. This report is a study of such anti-analysis techniques implemented by modern malware. For this purpose, RDG Tejon Crypter by RDG Max has been considered as the target. The study focuses on the anti-debugging, anti-VM, anti-Qemu, anti-anubis and other obfuscation techniques implemented by the subject.

The rest of the report has been divided as follows. Section 2 discusses about the related work in this field, section 3 covers the description of the environment created for study along with the details of the subject. Section 4 discusses the static and dynamic analysis performed. Section 5 discussed about the anti-debugging and anti-virtualization techniques implemented by RDG Tejon Crypter. Section 6 covers the author's views followed by future work and conclusion.

2. RELATED WORK

There has been a lot of research on the intelligent modern malware, aware of the presence of debugging tools and virtual environments. Many papers have been published discussing the various techniques used by these malware. A lot of techniques are available on open forums for detection of virtual environments like VMWare. Qemu and KVM are comparatively new technologies and thus, less publicly known techniques exist currently. As per the author's knowledge, no published study has been carried out specific to the techniques implemented by RDG Tejon Crypter

even though many of the techniques used by the crypter might have been published before.

3. ENVIRONMENT

To study the techniques implemented by the crypter, 'notepad.protected.exe' packed with the default features was taken. The virtual environment considered was KVM, the kernel based virtual machine. KVM is a virtualization solution integrated with vanilla linux kernel. KVM uses full virtualization to run virtual machines and leverages the facilities provided by hardware support for virtualization i.e. relies on virtualization capable CPU with either Intel VT or AMD SVM extensions. Qemu is used by KVM as a per VM process to provide hardware/device emulation. The study has been conducted on Windows XP SP3 x86 virtual machine.

The obvious choice of debugger for most of the security researchers is Ollydbg and that has been used throughout the study of the subject. The disassembler used is IDA-PRO along with some other tools to perform static and dynamic analysis of the executable packed with the crypter. RDG Crypter uses Visual Basic and is thus, highly dependent of MSVBVM60.DLL. It provides various features like anti-debugger, anti-IDA Debugger, Anti-Ollydbg, Anti-VirtualBox, Anti-VM and Anti-Anubis as shown in figure 1.

Figure 1: RDG Tejon Crypter

4. ANALYSIS

4.1. Static analysis

The static analysis of the obfuscated executable was performed to gather initial knowledge about it. Following steps were performed as part of static analysis.

4.1.1 Basic Static Analysis

a) Anti-Malware Scanning

The obfuscated executable was checked with various anti-malware scanning frameworks like VirusTotal, Jotti and

Eureka which resulted in flagging of the executable as malicious by most of the anti-malware engines(30 out of 47 on VirusTotal).
One conclusion that can be drawn out of it is that an executable flagged as malicious need not be really performing some malicious activity. The anti-malware engines may be reporting it as malicious because of its obfuscated nature and the various anti-analysis techniques implemented.

b) Strings
Strings program was used to identify all the strings present in the executable. Ollydbg and IDA-Pro can also be used for this purpose. Some of the important strings identified were, ZwQuerySystemInformation, ZwSystemDebugControl, NtQuerySystemInformation, GetVersionExA, CreateFileA, DeviceIoControl, IsdebuggerPresent, ProductVersion and ProductName. An experienced analyst can guess from these strings what kind of techniques are used by the crypter.

c) PEID
PEID is a tool which can be used to identify the packer used to obfuscate a particular executable. The opening of the executable with PEID returns Microsoft Visual Basic 5.0 / 6.0 [Overlay]. It tells that the executable has been compiled with Microsoft VC 6.0 and linked in Overlay mode.
PEID is not always successful in detecting the packers and thus, should not be the sole reason for deciding whether a binary is packed or not.

d) Dependency Walker
The dependency walker can be used to identify all the DLLs to be loaded by the executable. It also displays a list of functions imported and exported by the binary. The main DLL for 'notepad.protected.exe' is MSVBVM60.dll and the other DLLs are kernel32.dll, user32.dll, advapi32.dll, gdi32.dll, ole32.dll, msvcrt.dll and ntdll.dll.
The dependency walker shows that the the binary imports a lot of functions. It is generally assumed that the obfuscated or packed binaries would have less number of imports. Less imports may indicate that binary is packed. But, large number of imports shown by dependency walker need not imply that the binary is not obfuscated/packed.

e) PEView - PE File Format
PEView shows that the time-stamp for creation of the executable 'notepad.protected.exe' is 03/02/2009. It shows that the obfuscated executable has only .data section.

4.1.2 Advanced Static Analysis

a) IDA PRO
IDA-Pro is a disassembler which can be used to understand the malicious code without actually running it. For the executable under study, IDA-Pro shows only the .data section.
On opening the executable with the trial version of the tool, an error message "file structure error while loading" is received which results in only partial loading of the executable. It is noticeable that if the licensed version of IDA-Pro is used or only some of the features are used to obfuscate the executable, it loads properly without any such error.
The graph mode of IDA-Pro is not of much use as it just displays the two main functions of the executable which are DllFunctionCall and ThunRTMain. It doesn't provide any further information and thus, it can be said that the graphical view is not available.

4.**2. Dynamic Analysis**

4.2.1. Basic Dynamic Analysis

a) Process Monitor

Process monitor is used to capture the activities performed by an executable on files and registry. The following important activities by ' notepad.protected.exe' were observed.

- It scans various directories till its own directory.
- RegOpenKey on HKLM\System\CurrentControlSet\Control\SafeBoot\Option which was not found.
- RegQueryValue on HKLM\SOFTWARE\Policies\Microsoft\Windows\Safer\CodeIdentifiers\TransparentEnabled with DATA=1 to check if the software restriction policies are active.
- RegQueryValue on HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\GRE_Initialize\DisableMetaFiles which was not found.
- RegQueryValue for HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs.
- Checked for HKLM\Software\Microsoft\Windows NT\CurrentVersion\Diagnostics
- RegSetValue for HKLM\SOFTWARE\Microsoft\Cryptography\RNG\Seed
- Detects procmon - HKLM\System\CurrentControlSet\Services\PROCMON23\INSTANCES
- RegQueryValue for HKCU\Control Panel\Desktop\MultiUILanguageId
- RegQueryValue for HKLM\SOFTWARE\Microsoft\CTF\SystemShared\CUAS
- RegQueryValue for HKCU\Keyboard Layout\Toggle\Language Hotkey
- RegQueryValue for HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\IMM\Ime File
- RegOpenKey HKCU\Software\Policies\Microsoft\ControlPanel\International\Calendars\TwoDigitYearMax
- RegOpenKey for HKCU\Control Panel\International\Calendars\TwoDigitYearMax
- Checked for \Device\Harddisk0\DR0 using DeviceIoControl

  CreateFile \Device\Harddisk0\DR0 SUCCESS Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened

  DeviceIoControl \Device\Harddisk0\DR0 SUCCESS Control: SMART_GET_VERSION

  DeviceIoControl \Device\Harddisk0\DR0 SUCCESS Control: SMART_SEND_DRIVE_COMMAND

  DeviceIoControl \Device\Harddisk0\DR0 SUCCESS Control: SMART_RCV_DRIVE_DATA

  CloseFile \Device\Harddisk0\DR0 SUCCESS
- RegQueryValue for HKLM\System\CurrentControlSet\Services\Disk\Enum\0 for which value was:

  IDE\DiskQEMU_HARDDISK_____1.0_____\4d513030303020312020202020202020202020202020
- RegQueryValue for HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ProductId
- Search for .HLP under system32 and Windows\help
- Exits

4.2.2 Advanced Dynamic Analysis

Advanced dynamic analysis was performed using the debugging tool 'ollydbg' to analyse the run time behavior of the executable in a controlled way on Windows XP SP3 x86 virtual environment. The findings have been discussed in the results section.

5. RESULTS

5.1. Anti-Debugging

Anti-debugging techniques are used to detect the presence of debugger. There have been much research on various techniques for most of which reference can be found in the papers[1][2][3][4][5][6]. These techniques can be API based, Flag based, Timing based, Exception Based or Breakpoint detection based. The techniques found to be

implemented by the RDG Tejon Crypter are:

a) IsDebuggerPresent

This is the most common technique to detect the presence of a debugger. IsDebuggerPresent function is a part of Win32 Debugging API and is exported by kernel32.dll. It checks if the BeingDebugged flag present in the Process Execution Block(PEB) is set. If set, it means that the process is being debugged and the API returns 1 else there is no debugger present and thus, 0 is returned. Below is the assembly code checking for the BeingDebugged flag in PEB through TIB.

```
Address     Hex dump          Command                                  kernel32.IsDebuggerPresent(void)
7C81F424    64:A1 1800000     MOV EAX, DWORD PTR FS:[18]
7C81F42A    8B40 30           MOV EAX,DWORD PTR DS:[EAX+30]
7C81F42D    0FB640 02         MOVZX EAX,BYTE PTR DS:[EAX+2]
7C81F431    C3                RETN
```

Defense: This debugger detection technique can be bypassed by:

i) Modifying the assembly code so that it returns 0 in EAX.

ii) Patching BeingDebugged Flag to set its value to 0.

iii) Using Olly advanced plugin that provides an option to set the BeingDebugged Flag to 0.

b) NtGlobalFlags

NtGlobalFlag is a DWORD value inside the process PEB at offset 0x68 in x86 system and at 0xBC in x64 system. When a process is being debugged, the flags FLG_HEAP_ENABLE_TAIL_CHECK(0x10), FLG_HEAP_ENABLE_FREE_CHECK(0x20), and FLG_HEAP_VALIDATE_PARAMETERS(0x40) are set for the process and thus, NtGlobalFlag contains 0x70. Therefore, the value of NtGlobalFlag is retrieved and compared with 0x70 to detect the presence of a debugger. Below is the assembly code to fetch the value of NtGlobalFlags.

```
Address         Hex dump          Command
7C90FF1B  /$  64:A1 1800000       MOV EAX,DWORD PTR FS:[18]
;ntdll.RtlGetNtGlobalFlags(guessed void)
7C90FF21  |.  8B40 30             MOV EAX,DWORD PTR DS:[EAX+30]
7C90FF24  |.  8B40 68             MOV EAX,DWORD PTR DS:[EAX+68]
7C90FF27  \.  C3                  RETN
```

Defense: The countermeasures are:

i) Patch the PEB.NtGlobalFlag and PEB.HeapProcess flags to the values as if the process is not being debugged.

ii) Olly Advanced plugin has the option to set these flags.

c) NtQueryInformationProcess

The NtQueryInformationProcess() function is exported by ntdll.dll which can be used to retrieve information about a specific process. The syntax for the function (as per the MSDN) is as follows:

```
NTSTATUS WINAPI NtQueryInformationProcess(
  _In_    HANDLE ProcessHandle,
  _In_    PROCESSINFOCLASS ProcessInformationClass,
  _Out_    PVOID ProcessInformation,
  _In_    ULONG ProcessInformationLength,
```

```
    _Out_opt_  PULONG ReturnLength
);
```

One of the values that can be passed for *ProcessInformationClass* is Processdebugport(0x07). A non-zero return value indicates that the process is run under the control of a debugger. This technique has been used to detect the presence of debugger. Below is the assembly code that sets up the stack and calls NtQueryInformationProcess.

```
push 4 ;ProcessInformationLength
push 7 ;ProcessDebugPort
push -1 ;GetCurrentProcess()
call NtQueryInformationProcess
```

Defense: The ways to defeat this detection technique are:

i) Internally, this function queries for the non-zero state of the EPROCESS->DebugPort field. This value can be set to zero to hide the debugger.

ii) The assembly code can be modified to return the appropriate value.

d) API Redirection

To hide the the position of the functions called and to make it difficult for reverse engineering, it calls a stub that performs a JMP in which address is referenced from the import. Below is an example of such call.

```
CALL <JMP.&MSVBVM60.#100>
```

e) Exceptions

The crypter adds a web of exceptions in the executable, some of which are non-continuable exceptions. If run on a virtual machine, the process exits. But, if it is run under a debugger, it doesn't exit and throws exception. Adding exception introduces confusion making it harder to understand and reverse.

f) Registry Checks

i) The process performs RegQueryValue for the registry:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit_DLLs

For any application getting executed, Windows checks this key, loads the DLL in the newly executed application, and calls the entry point of the registered DLL. This causes the code in the DLL to run under the context of the application. The AppInit DLLs are loaded by using the LoadLibrary() function during the DLL_PROCESS_ATTACH process of User32.dll. It is possible that the process checks if any DLL is getting injected.

ii) The process checks for the presence of following registry key.

HKLM\Software\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\notepad.protected.exe

This registry key is intended to specify the name of a debugger, based on which one can debug an application when it starts. For example, to debug notepad.protected.exe when it starts, one simply has to go to this registry key and create a subkey called notepad.protected.exe. Under this subkey, a string value can be added specifying the name of the debugger under which the application would be launched.[12][13]

The process queried this registry key to check if its name exists as a subkey with some debugger specified. But, it didn't find its name "notepad.protected.exe" here. This registry key(with the specific process name) can also be used to specify the "GlobalFlag" string value which sets the flags for the particular executable.

iii) The process also checked the registry key:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\AeDebug

This key is used to specify the postmortem debugger which is automatically invoked by the system in case an application stops responding. Checking this key doesn't clarify whether the process is running under debugger. But, it can tell which debugger is set as a postmortem debugger that would be launched in case of some access violation.

g) NtQueryPerformanceCounter

As per MSDN, NtQueryPerformanceCounter (exported by ntdll.dll) returns the current value of a performance counter and, optionally, the frequency of the performance counter. Its syntax is:

```
NTSTATUS NtQueryPerformanceCounter(
  _Out_     PLARGE_INTEGER PerformanceCounter,
  _Out_opt_  PLARGE_INTEGER PerformanceFrequency
);
```

It is a timing based detection method which queries performance counters, the registers that store counts of hardware related activities within the processor. The query is made twice and difference is compared with a threshold. If it is below the threshold value, it implies that the process is running under a debugger.

```
[7380A4D5] CALL DWORD PTR DS:[<&ntdll.NtQueryPerformanceCounter>]
```

5.2. Anti-VM

The techniques to detect the virtual environments have been discussed widely on public forums and papers[7][8][9][10][11]. These detection methods can exploit the CPU bugs, model-specific registers, differences in timings, device emulation and other differences. As the virtual environment under consideration is KVM, many of the known attacks don't work in this scenario. Some of the techniques found to be implemented by the crypter are:

a) DeviceIoControl

DeviceIoControl is a Win32 API function which is used to perform IOCTL operations on the specified device driver. IOCTL operations through DeviceIoControl can be used to retrieve information like version number and the name related to a particular driver. The return value from DeviceIoControl is a Boolean value that indicates success or failure. The syntax is:

```
BOOL WINAPI DeviceIoControl(
  _In_        HANDLE hDevice,
  _In_        DWORD dwIoControlCode,
  _In_opt_     LPVOID lpInBuffer,
  _In_        DWORD nInBufferSize,
  _Out_opt_    LPVOID lpOutBuffer,
  _In_        DWORD nOutBufferSize,
  _Out_opt_    LPDWORD lpBytesReturned,
  _Inout_opt_  LPOVERLAPPED lpOverlapped
);
```

KVM uses Qemu to facilitate device emulation. Virtual machines are associated with specific device drivers and registry values that give away their nature. The device names may contain some substring that can lead to the detection of virtual environment. RDG Crypter uses this technique to detect virtual environment it is running in. Below is the call to the DeviceIoControl made by the process.

```
01105D5F call to DeviceIoControl
CPU Stack
Address     Value      ASCII     Comments
0012F1F0    7351D30A   ÓQs       ; /RETURN to MSVBVM60.7351D30A
0012F1F4    00000084   „         ; |hDevice = 00000084
0012F1F8    00074080   €@        ; |IoControlCode = 74080
0012F1FC    00000000             ; |InBuffer = NULL
```

```
0012F200    00000000            ; |InSize = 0
0012F204    0012F398    ˜ó      ; |OutBuffer = 0012F398 -> 00
0012F208    00000018            ; |OutSize = 24.
0012F20C    0012F3B0    °ó      ; |BytesReturned = 0012F3B0 -> 0
0012F210    00000000            ; \pOverlapped = NULL
```

It can also be verified from the results of Process Monitor Capture.

CreateFile \Device\Harddisk0\DR0 SUCCESS Desired Access: Generic Read/Write, Disposition: Open, Options: Synchronous IO Non-Alert, Non-Directory File, Attributes: n/a, ShareMode: Read, Write, AllocationSize: n/a, OpenResult: Opened
DeviceIoControl \Device\Harddisk0\DR0 SUCCESS Control: SMART_GET_VERSION
DeviceIoControl \Device\Harddisk0\DR0 SUCCESS Control: SMART_SEND_DRIVE_COMMAND
DeviceIoControl \Device\Harddisk0\DR0 SUCCESS Control: SMART_RCV_DRIVE_DATA
CloseFile \Device\Harddisk0\DR0 SUCCESS

b) Detection Based on HDD Model Name

This is a method to detect virtual environment based on their HDD model names. So, any process can search for some pattern or substring to detect the specific virtual environment.
For example:

**VIRTUAL* to detect "VIRTUAL HD"*
**VMWARE* to detect "VMWARE VIRTUAL IDE HARD DRIVE"*
**QEMU* to detect "QEMU HARDDISK"*

The process queries the registry value:

HKLM\System\CurrentControlSet\Services\Disk\Enum\0

which is:

IDE\DiskQEMU_HARDDISK_____1.0_____\4d513030303020312020202020202020202020202020

As it can clearly be seen, registry value contains “QEMU” substring and thus, would lead to detection of the virtual environment.

c) ProductId

Another technique used by RDG Tejon Crypter is based on the productID. It queries the following registry value.

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ProductId

This technique can be used for successful detection as different commercial sandboxes use some specific known MS Windows Product IDs. For anubis, the product id to match is 76487-337-8429955-22614.

Defense: To bypass this detection technique, the product ID can be changed to some random value before execution of the application.

5.3. Noticeable Behavior

There are also some other function calls made by ‘notepad.protected.exe’ which are worth noticing:

a) Kernel32.GetVersionExA and ntdll.RtlGetVersion()
b) NtQueryDefaultUILanguage()
c) user32.getkeyboardlayout()
d) ntdll.RtlGetProductType
e) ntdll.RtlGetNativeSystemInformation

6. AUTHOR'S VIEWS

In addition to the techniques found to be used by RDG Crypter, some of the key observations are:

6.1 Error With Trial Version of IDA-Pro

The executable packed with the default features of RDG Crypter doesn't load properly with IDA-Pro and throws some file structure error. But, if we try to load it with the licensed version, it loads properly without any error. If we implement only the anti-anubis features of the Crypter, the executable loads properly even with the trial version of IDA-Pro.

So, it is possible that there is some bug in the implementation of some of the features of Crypter which causes this error. It also likely that the author, RDG MAX, has added such feature knowingly to make it more difficult for reverse engineers who rely on the trail version of IDA-Pro.

6.2 Anti-Debugging with Anti-Anubis Feature

Another key observation is that even on using the anti-anubis feature only, the crypter adds some anti-debugging techniques and exceptions. The purpose of this is probably to make it harder for the reverse engineers trying to study the techniques implemented. If it doesn't implement anti-debugging techniques, it would be comparatively easier to identify the anti-anubis techniques implemented by the Crypter. As it has so many checks, on detecting a debugger, it doesn't call some of the functions implemented for the detection of the virtual environment.

6.3 Web of Calls and Checks

A large number of checks have been implemented in the crypter. In addition, it exploits the jumps and exceptions to make it harder to study.

6.4 Qemu Detection

Qemu provides device emulation which is used by KVM and Anubis. The detection of virtual environment can be performed by querying the information about the emulated devices. As discussed above, RDG tejon Crypter uses DeviceIoControl and registry checks to search in the names or versions of the emulated devices. The device manager on Windows XP SP3 virtual machine(on KVM) looks like:

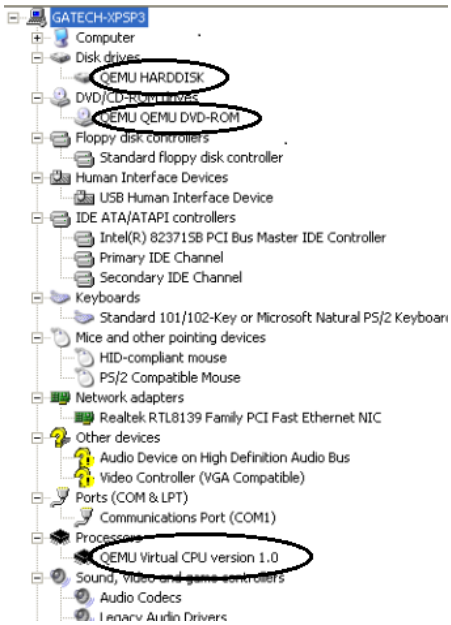

Figure 2: Device Manager

As shown in the figure 2, there are many options which can be checked to detect the virtual environment. In addition, there are some other registry keys which can also lead to the detection of the virtual environment.
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Enum\IDE\CdRomQEMU_QEMU_DVD-ROM_____1.0_____
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Enum\IDE\DiskQEMU_HARDDISK_____1.0_____

6.5 Exception Under Debugger
The executable doesn't run (opens notepad) on a virtual machine. But, if run under a debugger, it throws an exception and doesn't terminate.

6.6 Avoiding Qemu Detection[14]
There are some steps which can be performed to avoid Qemu/KVM detection or to reduce the chances of detection[citation].

i) Don't install guest additions
ii) Change MAC address
iii) Change BIOS information
iv) Change Hard Drive model
V) Replace 'QEMU' in all the emulated device names and registry keys/values with some random substring.

7. FUTURE WORK
This study covers only some of the techniques implemented by crypter. There is still much scope for further analysis of the techniques implemented.
i) A large number of techniques have been implemented by the crypter out of which only some have been discovered. Further study needs to be done to identify all other techniques.
ii) Some functions like GetTickCount(), NTQuerySystemInformation(), VMProtect(), NtQueryVirtualMemory() and GetDrivetype() have been identified but their use has not been observed yet. Further analysis needs to be done to confirm whether these are part of some anti-analysis technique or not.
iii) To bypass Qemu device emulation, 'QEMU' in all the device names and registry values was replaced. But, the executable was still able to detect the virtual environment. further study needs to be done to identity the other anti-anubis/anti-qemu techniques implemented by the crypter.

8. CONCLUSION
Software armoring is a great way to evade the code analysis and reverse engineering. It has some genuine uses in the software industry to protect the software code. But, malware authors have started to use these techniques to hide the malicious behaviour of their creations. RDG Tejon Crypter implements the anti-debugging techniques which are publicly available but makes it difficult to analyse by adding jumps, garbage code, exceptions and a large number of checks.
As KVM utilizes virtualization hardware extensions, many of the traditional detection methods don't work. But, Qemu device emulation provides the attack surface which is exploited by the crypter to detect KVM/Qemu/Anubis. The device names, registry values, BIOS information and MAC address are some of the techniques which can be used to detect KVM and Qemu.

ACKNOWLEDGMENTS
I would like to thank Paul Royal for the guidance, motivation and the support provided through this learning phase. I am grateful to him for giving me an opportunity to start my journey in the field of malware analysis.

REFERENCES

[1] Peter Ferrie. The 'Ultimate' Anti-Debugging Reference. May 2011.
[2] Tyler Shields. Anti-Debugging – A Developers View.Veracode Inc., USA.
[3] Mark V. Yason. The Art of Unpacking.
[4] Josh Jackson. Introduction Into Windows Anti-Debugging. September 2008.
[5] Nicolas Falliere Windows Anti-Debug Reference, Symantec, Sept 2007.
[6] Peter Ferrie. ANTI-UNPACKER TRICKS.
[7] Elias Bachaalany. Detect if your program is running inside a vm.
http://www.codeproject.com/system/vmdetect.asp.
[8] Tom Liston, Ed Skoudis. Thwarting Virtual Machine Detection. 2006
[9] Peter Ferrie. Attacks on Virtual Machine Emulators. Symantec Advanced Threat Research.
[10] VM and Sandbox Detections become more professional. Joe Security. August 2012.
[11] How does malware know the difference between the virtual world and the real world. SourceFire Vulnerability Research Team. October 2009.
[12] How to: Launch the Debugger Automatically. MSDN.
http://msdn.microsoft.com/en-us/library/a329t4ed%28v=vs.90%29.aspx
[13] Image File Execution Options. McAfee Labs. December 2008.
http://blogs.mcafee.com/mcafee-labs/image-file-execution-options
[14] Modifying KVM (qemu-kvm) settings for malware analysis. September 2012.
http://blog.prowling.nu/2012/09/modifying-kvm-qemu-kvm-settings-for.html