The Computer Forensics Challenge and Anti-Forensics Techniques

HackInTheBox – Kuala Lumpur - Malaysia

Domingo Montanaro

<conferences@montanaro.org>

Rodrigo Rubira Branco

<rodrigo@kernelhacking.com>

Kuala Lumpur, August 06, 2007

Agenda

Defeating forensics analysis

- **Subverting clones/imaging processes**
- **Backdoors/Rootkits/Whatever**
- **Etc ;D**

Data Remanence -> Magnetic Media

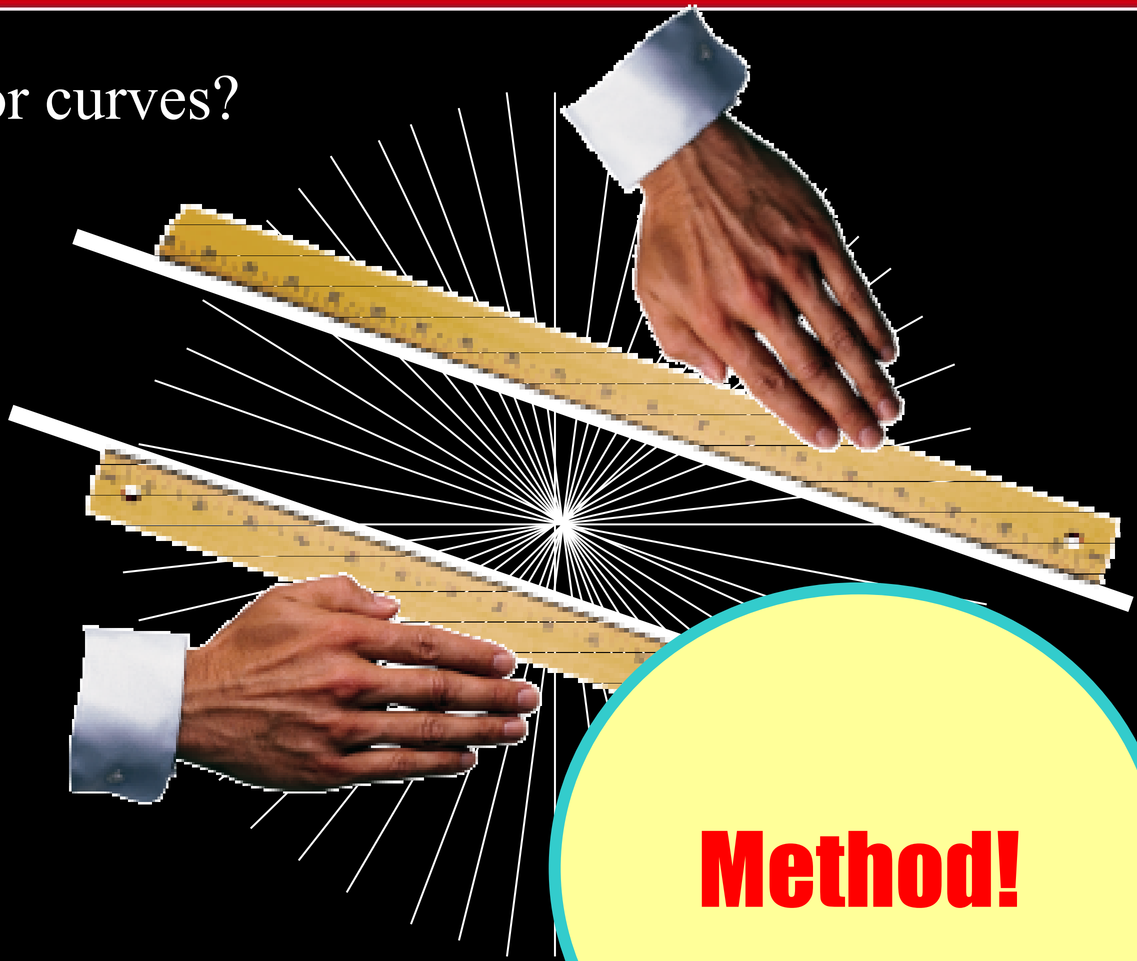
- **From erased data (covering some filesystems)**
- **From overwritten data**
- **From destroyed media**

Being prepared to the incident

- Turn off or keep turned on the hw? It **Depends**
- RAM Clone ? **Always**
  - ➢ Using the SO or hw specialized with DMA support?
- Take the HD out or clone? **Clone**
- Physical Manipulation of evidences? For Sure – Special equipment
- Hard Locks ? You kidding me, right?

Methodology

Straight Lines or curves?



Method!

Methodology

Forensics analysis require deep information technology knowledge

Just a few examples that can simply modify the “guilty-non guilty” boolean variable:

- **ADS**
- **MD5**
- **Simple image stego**
- **Slack Space**
- **Hiding data inside the "visible" filesystem**
- **Rootkits - Subverting the first step - Imaging**

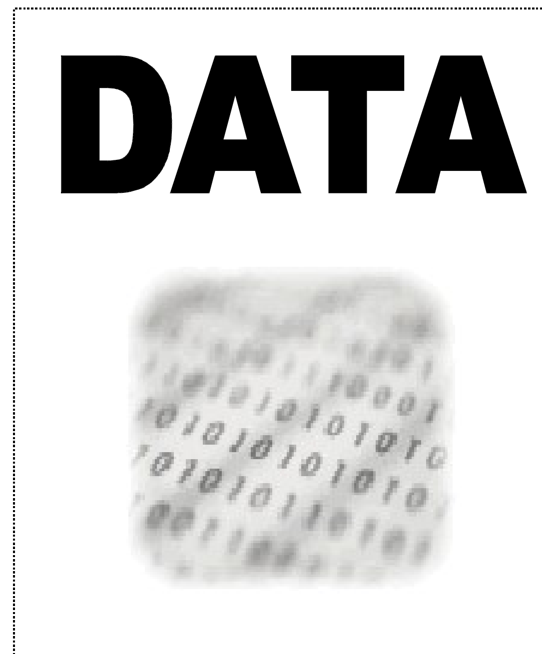
Aligning knowledge – the very beginning

Simple file deletion on FAT filesystem

1

2

3



First Step

Fat entry deleted

This indicates that the area blocks occupied by that file are now free

Second Step

The file’s registry on the directory’s entry is modified

First char is changed (Ex: E5 Hex [Fat32])

Third Step? No! :(

DATA

Data is still there

Data blocks are still avaliable for recovering until other aplication write in the same clusters

How the recovery process works

Index damaged and Directory entry ok -> Easy recover by parsing directory information and some items from the Index (example: format on Windows machines) – Remembering that NTFS stores a copy of it’s MFT in the middle of the unit

No Index and no Directory -> Should be easy by header/footer search and grabbing the middle contents, but some fragmentation issues could lead to get “currupted” files, which consist in “garbage” in the middle of a true “mailbox” file.

Tool to perform recovery on header/footer (and also expected size) search: foremost

Oops: It’s almost impossible to see tools in the wild that perform structured file analysis, which are totally necessary to recover files by it’s internals characteristics (file format).

For file formats, www.wotsit.org

Fact: Only 1 kb of garbage in a contiguous file of 10MB can lead to non recovery of this file if no file format comparison is made

Magnetic Level

Causes:

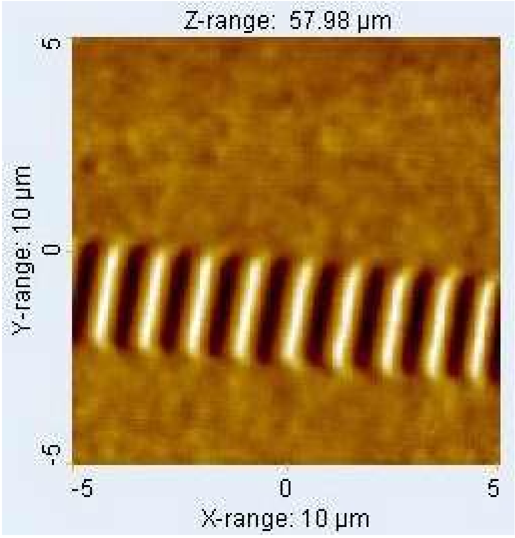
- **Data overlapping:**

- **Changing OS and FileSystem**
- **Wipe tools**

Magnetic Level

Method:

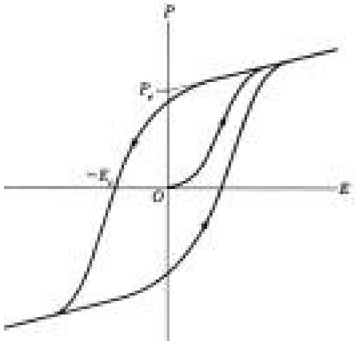
- **STM (Scanning Tunneling Microscopy)**
- **SPM (Scanning Probe Microscopy)**
- **MFM (Magnetic Force Microscopy) ->**
- **AFM (Atomic Force Microscopy)**



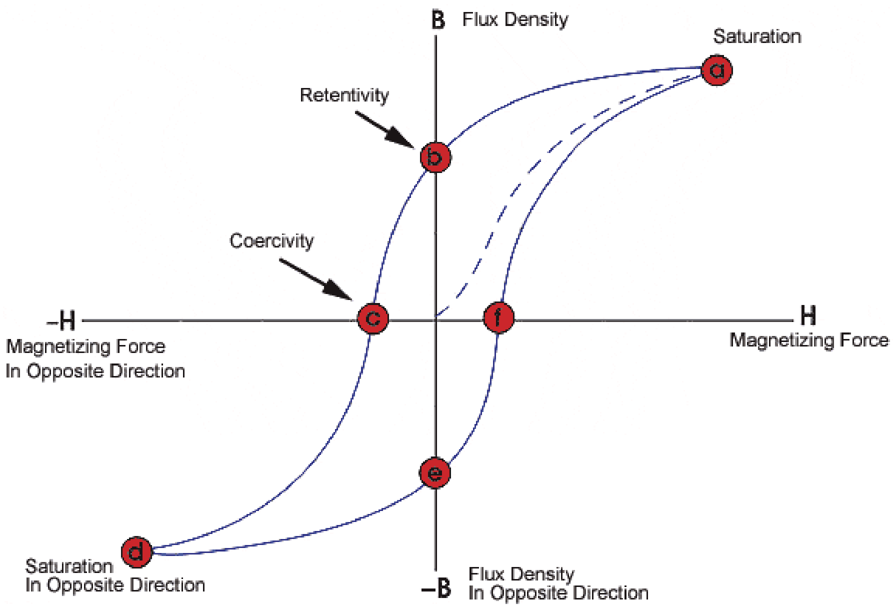
From: LFF – IF - USP

Why? HYSTERESIS

Study: The Hysteresis Loop and Magnetic Properties



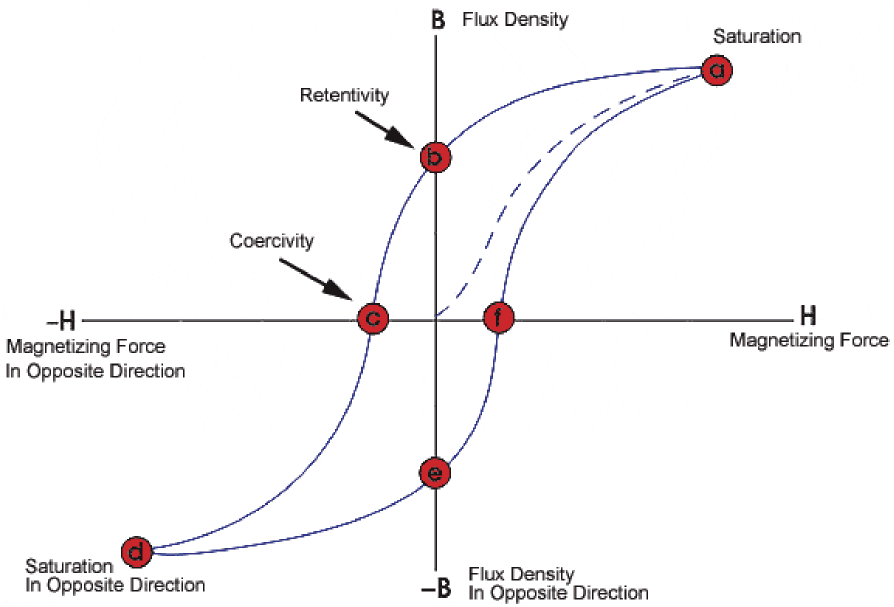
Magnetic Level



From Iowa’s State University Center for Nondestrutive Evaluation NDT (Non Destrutive Testing)

The loop is generated by measuring the magnetic flux of a ferromagnetic material while the magnetizing force is changed. A ferromagnetic material that has never been previously magnetized or has been thoroughly demagnetized will follow the dashed line as **H** is increased. As the line demonstrates, the greater the amount of current applied **(H+)**, the stronger the magnetic field in the component **(B+)**. At point "a" almost all of the magnetic domains are aligned and an additional increase in the magnetizing force will produce very little increase in magnetic flux. The material has reached the point of magnetic saturation. When **H** is reduced to zero, the curve will move from point "a" to point "b." At this point, it can be seen that some magnetic flux remains in the material even though the magnetizing force is zero. This is referred to as the point of retentivity on the graph and indicates the remanence or level of residual magnetism in the material. (Some of the magnetic domains remain aligned but some have lost their alignment.) As the magnetizing force is reversed, the curve moves to point "c", where the flux has been reduced to zero. This is called the point of coercivity on the curve. (The reversed magnetizing force has flipped enough of the domains so that the net flux within the material is zero.) The force required to remove the residual magnetism from the material is called the coercive force or coercivity of the material. As the magnetizing force is increased in the negative direction, the material will again become magnetically saturated but in the opposite direction (point "d"). Reducing **H** to zero brings the curve to point "e." It will have a level of residual magnetism equal to that achieved in the other direction. Increasing **H** back in the positive direction will return **B** to zero. Notice that the curve did not return to the origin of the graph because some force is required to remove the residual magnetism. The curve will take a different path from point "f" back to the saturation point where it with complete the loop.

Magnetic Level



In other words:

Hd's Heads are only prepared to read and write 0 or 1.

When one bit is 0 and it changes to 1, the head will "read/feel" 1 at the read time, but what is stored in the media is (for example) analogic 0,78 value

bit 1 original

Changed to 0

HD's heads will read 0

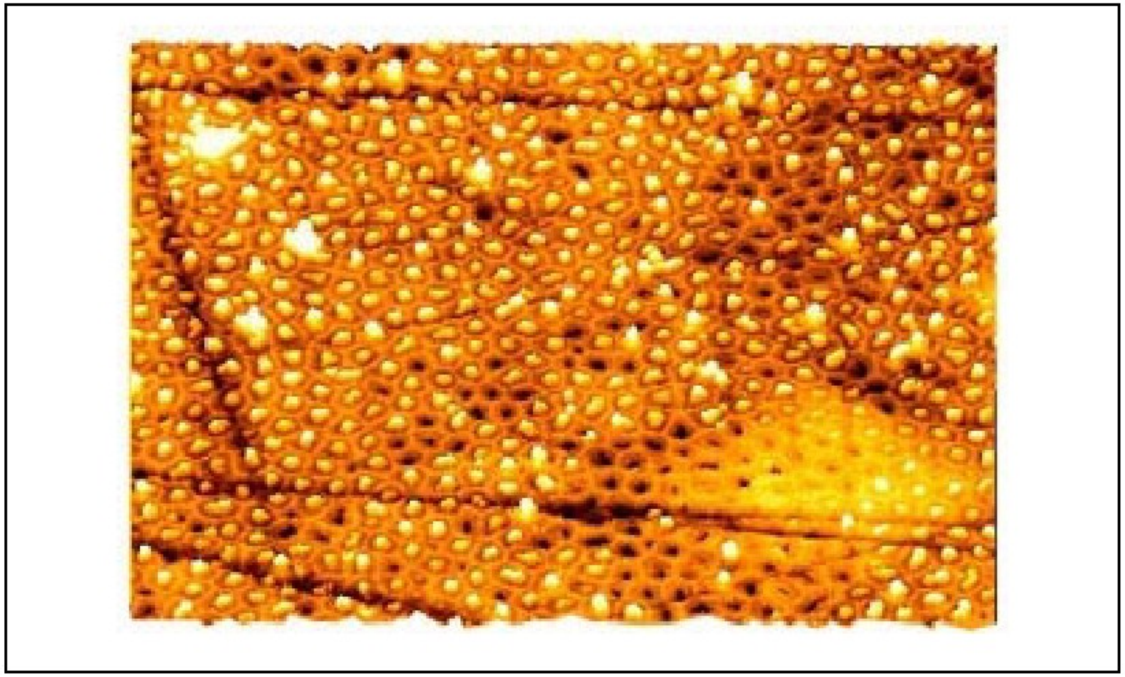
Electronic Microscopes (such as confocal blue laser scaning) it is possible to notice other "states" – rudimentar 0,12 for example

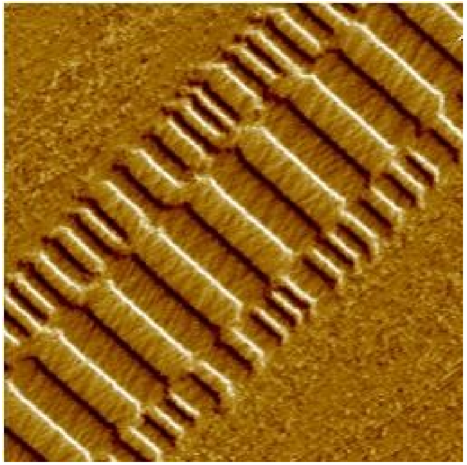
Magnetic Level

- **Possible because Information is digital, but it’s supporting technology is analogic**

Pictures taken from methods in the previous slides

FIGURE 1:
AN ATOMIC FORCE IMAGE OF MAGNETIC RECORDING MEDIA SHOWING THE SUSPENDED MAGNETIC PARTICLES (used courtesy of Park Scientific Instruments, http://shell7.ba.best.com/~wwwpark/appnotes)



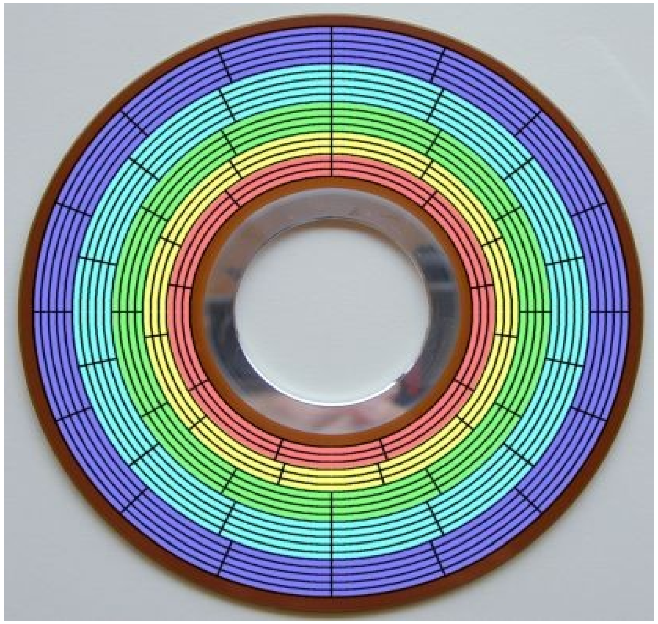


Residuals of overwritten information on the side of magnetic disk tracks. Reproduced with permission of VEECO

Magnetic Level

- **And How about 1-Step wipe? Good enough. Why?**

Simply to understand. Hard drives are coming with tons of storage space and it's “physical size” is always the same (most of the times same number of platters/heads then the previous model). The platters and heads are almost the same scheme and the storage size is increasing each time more. So, various techniques to increase speed/storage capabilities imply on reducing data recovering from electronic microscopy, such as Zoned Bit Recording



Graphic from PcGuide.com

As far as the track is from the center, it supports more sectors, increasing the space for storage but drastically reducing magnetic data recovery

Damaged Hard Drives

Causes:

- **Accidents**

- Accidental Falls

- Destroying on purpose

Damaged Hard Drives

Method:

- Platters removal
- Special liquid for clearing the platters
- Low level reading of platters by generics heads that have pre-configured vectors of reading

False positive about Defects

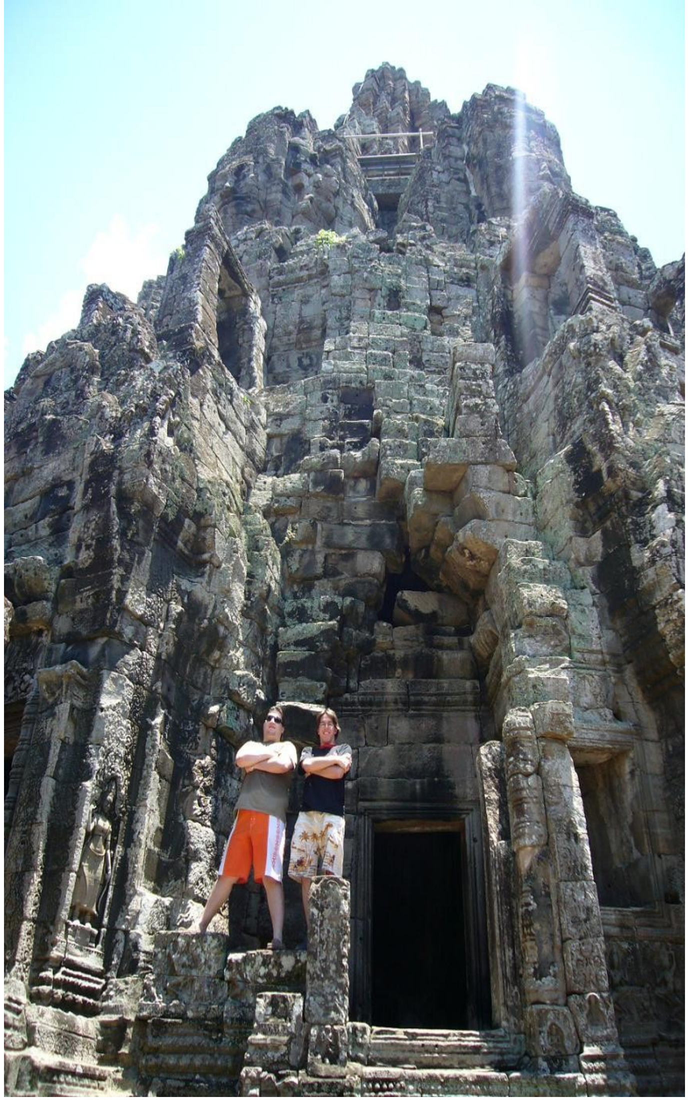
Most of data recovery softwares work trough BIOS (int 13h) or the OS to access disk clusters

- 1 Cluster normally consists in 1 header, 512 bytes and ECC byte
- When Recovery Software tries to get a cluster from the HD, if it comes with a ECC bad checksum, it will assume that this specific cluster is a “bad cluster”
- One not-that-hard-to-code backdoor can simply forge this ECC bad checksum (error types “UNC” – Uncorrectable data - or AMNF – Address Mark Not Found) statically or dynamic to keep it’s code on the media hard-to-find.
- So, to achieve reading of these sectors, some ATA commands that ignore ECC need to be issued to recover byte-a-byte rather then sector-per-sector as most OS and BIOS do.

Acknowledges – The trip is finishing :(

- Filipe Balestra and Nicolas Waisman for helping in the Immunity Debugger Stuff
- HITB crew (mainly to XWings) for the nice time and patience here in Malaysia
- Your time in this talk!

Expecting again a Brazilian Woman? Haha, gotcha! ->



Thanks!

Questions?

Thank you :D

There's where
we come from ;)

Rodrigo Rubira Branco
<rodrigo@kernelhacking.com>
Domingo Montanaro
<conferences@montanaro.org>

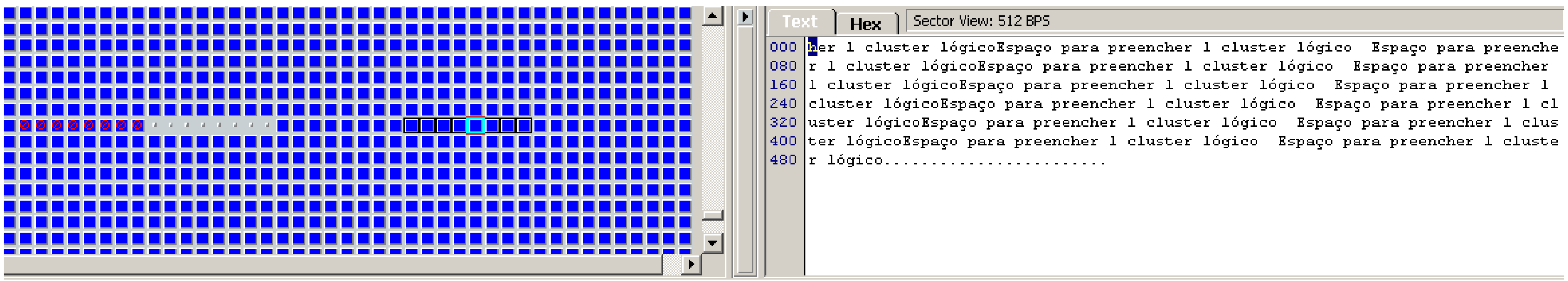
Slack Space

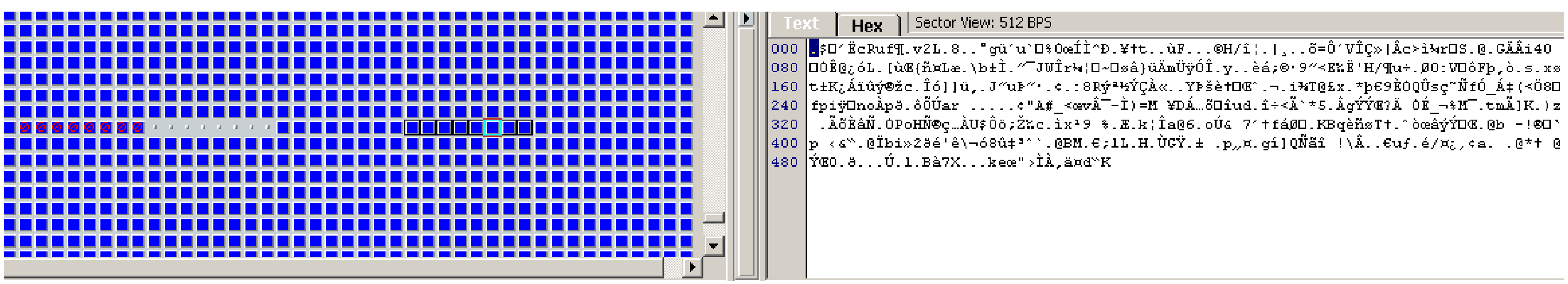
Non-addressable space in the MFT than can be written by specfic tools (RAW)

- NTFS uses logical cluster of 4kb
- Files less than 4kb use 4kb (outside MFT)
- Tools can build a own MFT and address directly on the disk its own blocks to use as a container for the backdoor (and can mark it as bad block to the filesystem, so it would not be overwritten)
- Combining this to crypto/steganographic technics should make the forensics job much harder (and most of times when it’s well done, efforts will be lost)

Update: Tool: Slacker from the Metasploit project

Slack Space


Text
Hex
Sector View: 512 BPS
000 her 1 cluster lógicoEspaço para preencher 1 cluster lógico Espaço para preenche
080 r 1 cluster lógicoEspaço para preencher 1 cluster lógico Espaço para preencher
160 1 cluster lógicoEspaço para preencher 1 cluster lógico Espaço para preencher 1
240 cluster lógicoEspaço para preencher 1 cluster lógico Espaço para preencher 1 cl
320 uster lógicoEspaço para preencher 1 cluster lógico Espaço para preencher 1 clus
400 ter lógicoEspaço para preencher 1 cluster lógico Espaço para preencher 1 cluste
480 r lógico.......................


Text
Hex
Sector View: 512 BPS
000
080
160
240
320
400
480

Slack Space

```
19B751940  75 73 74 65 72 20 6C F3  67 69 63 6F 45 73 70 61  uster lógicoEspa
19B751950  E7 6F 20 70 61 72 61 20  70 72 65 65 6E 63 68 65  ço para preenche
19B751960  72 20 31 20 63 6C 75 73  74 65 72 20 6C F3 67 69  r 1 cluster lógi
19B751970  63 6F 0D 0A 45 73 70 61  E7 6F 20 70 61 72 61 20  co..Espaço para 
19B751980  70 72 65 65 6E 63 68 65  72 20 31 20 63 6C 75 73  preencher 1 clus
19B751990  74 65 72 20 6C F3 67 69  63 6F 45 73 70 61 E7 6F  ter lógicoEspaço
19B7519A0  20 70 61 72 61 20 70 72  65 65 6E 63 68 65 72 20   para preencher 
19B7519B0  31 20 63 6C 75 73 74 65  72 20 6C F3 67 69 63 6F  1 cluster lógico
19B7519C0  0D 0A 45 73 70 61 E7 6F  20 70 61 72 61 20 70 72  ..Espaço para pr
19B7519D0  65 65 6E 63 68 65 72 20  31 20 63 6C 75 73 74 65  eencher 1 cluste
19B7519E0  72 20 6C F3 67 69 63 6F  00 00 00 00 00 00 00 00  r lógico........
19B7519F0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
19B751A00  10 24 8F 92 CB 63 52 75  66 B6 11 76 32 4C 01 38  .$▌´ËcRuf¶.v2L.8
19B751A10  18 01 B0 67 FC B4 75 60  8D 25 D2 9C CD CC 5E D0  ..°gü´u`▌%Ò▌ÍÌ^Ð
19B751A20  06 A5 86 74 06 12 F9 46  1B 02 17 A9 48 2F EE A6  .¥▌t..ùF...©H/î¦
19B751A30  10 7C B8 05 18 F5 3D D4  B4 56 CE C7 BB 2A 2A 2A  .|¸..õ=Ô´VÎÇ»***
19B751A40  2A 54 45 58 54 4F 20 45  53 43 4F 4E 44 49 44 4F  *TEXTO ESCONDIDO   ->Hidden Data
19B751A50  2A 2A 2A 40 BF F3 4C 03  5B F9 8C 7B F1 A4 4C E6  ***@¿óL.[ù▌{ñ¤Læ
19B751A60  08 5C 62 B1 CC 07 94 AF  4A 57 CE 72 BC A6 9D 7E  .\b±Ì.▌¯JWÎr¼¦▌~
19B751A70  7F F8 E2 7D FC C4 6D DC  FF D3 CE 14 79 1E 18 E8  ▌øâ}üÄmÜÿÓÎ.y..è
19B751A80  E1 3B A9 B7 39 94 3C 45  89 CB 27 48 2F B6 75 F7  á;©·9▌<E▌Ë'H/¶u÷
19B751A90  02 D8 4F 3A 56 8D F4 46  FE 2C F2 0C 73 1A 78 F8  .ØO:V▌ôFþ,ò.s.xø
19B751AA0  74 B1 4B BF C1 EF FB FD  AE 9E 63 07 CE F3 5D 5D  t±K¿Áïûý®▌c.Îó]]
19B751AB0  FC 2C 0E 4A 94 75 DE 94  B7 0B A2 12 3A 38 52 FD  ü,.J▌uÞ▌·.¢.:8Rý
19B751AC0  AA BD DD C7 C0 AB 1D 07  59 DE 9A E8 86 81 8C 88  ª½ÝÇÀ«..YÞ▌è▌▌▌▌
19B751AD0  1C AC 04 69 BE 54 40 A3  78 0B 2A FE 80 39 C8 D2  .¬.i¾T@£x.*þ▌9ÈÒ

Sector 13482636 of 14329917    Offset: 19B751981    = 114
```

Use of redundant/Zero/Align spaces

Executables (ELF, Win32PE, etc) when compiled, depending on the compiler, most of the times need to have some space for alignment between soubroutines.

Not a new idea in the IT field, since it's used by virii coders (injecting malware instructions into space used for alignment)

```
4AD051A5: C3 RETN ; end of subroutine
4AD051A6: 90 NOP ;
4AD051A7: 90 NOP ;
4AD051A8: 90 NOP ;
4AD051A9: 90 NOP ;
4AD051AA: 55 PUSH EBP ; begin of next subroutine
```

Alignment that can be used to store data
Can be 0x90, 0xCC or signature-based like GCC

On a 2GB "system" filesystem, it's possible to store nearly 1 MB on a "Second Filesystem" inside the "system" filesystem, only using alignment spaces (including DLLs) – Need to remember that relative (short) JMPs are needed to return in the program normal flow.

Going even deeper

So, every filetype has it's possibilities of storing "evil" data, not regarding compression formats.

Harmful to think on all this knowledge about hiding information (stego) in files to come in a toolkit.

Scenario:

LibStego – Supports data hiding on several file formats, applying the parsing tons of these formats from wotsit.org

Supporting: 3 modes of operation

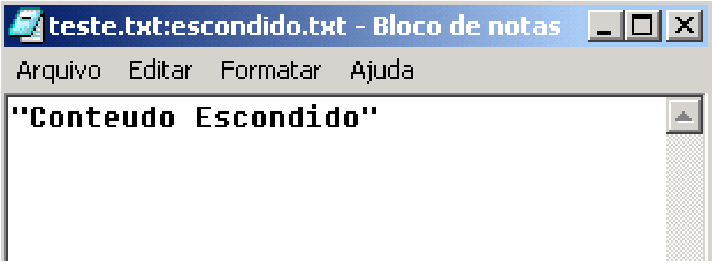
1) Growing up files – Ex: comments on graphic files (as showed before)

2) Use redundant space on Multimedia formats (GIF, JPEG, AVI, MOV, etc), OLE formats (doc, xls, ppt, etc – not talking about compression here too) and others (DWG, CDR, etc)

3) Use alignment space on executable files (PE, ELF, etc)

ADS – Alternate Data Streams

```
C:\ads>echo "Conteudo Normal" > teste.txt

C:\ads>echo "Conteudo Escondido" > teste.txt:escondido.txt

C:\ads>dir /a
Pasta de C:\ads

22/11/2004  00:59       <DIR>
22/11/2004  00:59       <DIR>          ..
22/11/2004  00:59                  20 teste.txt
               1 arquivo(s)            20 bytes
               2 pasta(s)  1.696.808.960 bytes disponíveis

C:\ads>type teste.txt
"Conteudo Normal"

C:\ads>notepad teste.txt:escondido.txt
```



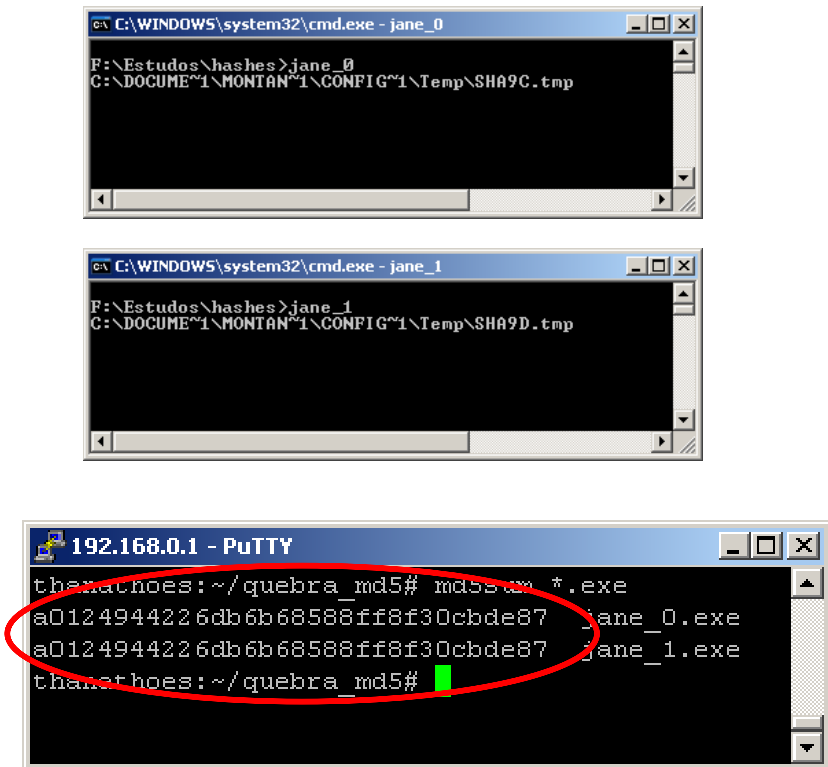
Hash Collision

```
black@bishop:~/quebra_md5$ ls
1.asc  1.bin  2.asc  2.bin  resultado.txt

black@bishop:~/quebra_md5$ cmp 1.bin 2.bin
1.bin 2.bin differ: char 20, line 1

black@bishop:~/quebra_md5$ md5sum 1.bin 2.bin
79054025255fb1a26e4bc422aef54eb4  1.bin
79054025255fb1a26e4bc422aef54eb4  2.bin
```

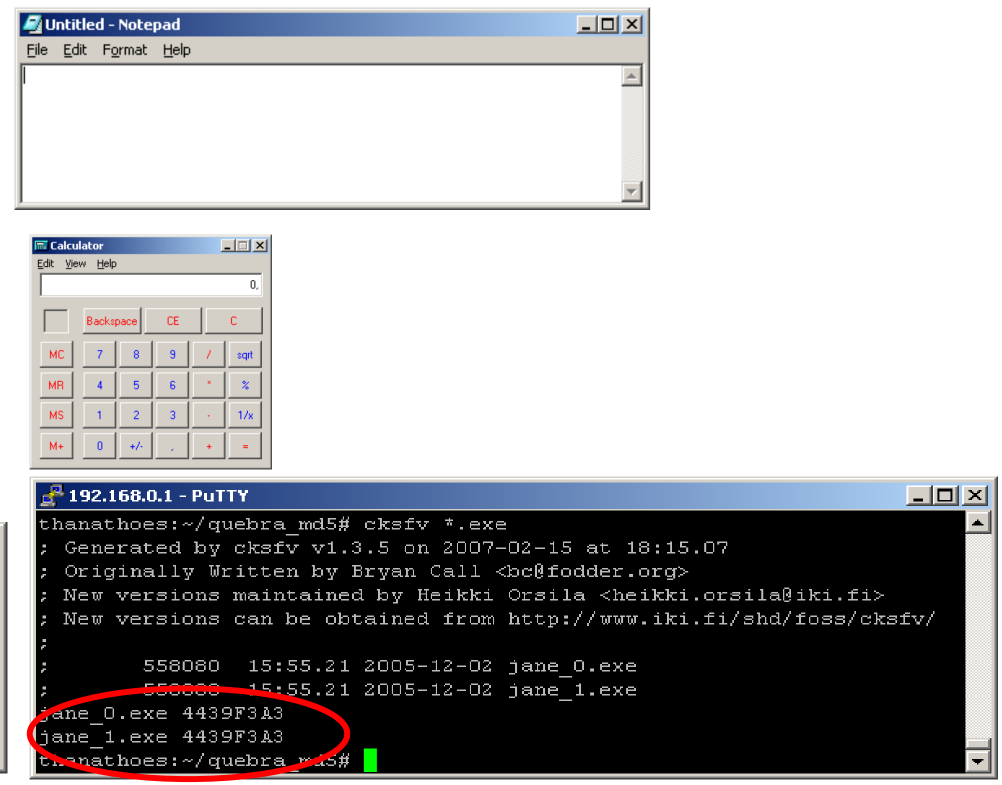
Hash collision

Not indicated to use only MD5 nowadays

From: Gerardo Richarte - CORE SDI
MD5 to be considered harmful today



Same MD5



Same CRC

Hash collision

Again, not good to use only MD5

http://www.doxpara.com/research/md5/confoo.pl

confoo $VERSION: Web Conflation Attack Using Colliding MD5 Vectors and Javascript
Author: Dan Kaminsky(dan\@doxpara.com)
Example: ./confoo www.lockheedmartin.com active.boeing.com/sitemap.cfm

Attack Vectors!

http://www.doxpara.com/stripwire-1.1.tar.gz

Stripwire emits two binary packages. They both contain an arbitrary payload, but the payload is encrypted with AES. Only one of the packages ("Fire") is decryptable and thus dangerous; the other ("Ice") shields its data behind AES. Both files share the same MD5 hash.

Simplistic Image Steganography

- Image files follow their layout standards, as of any other kind of file
- Each standard has it's own data hiding capabilities (GIF, BMP, TIFF, etc) – of course, not the original purpose

Ex: GIF89a

- Con: Not many tools to analyze file's layout, comparing it to a standard layout and a base of layout possibilities (out-of-range values in some fields)

And we are not even talking about the graphic part, which implies on techniques such as Color Reduction, LSB (Least Significant Bit) – noise, etc.

Dumbest stego method ;)

Nome	Tamanho	Tipo	Data de modificação
logo_h2hc	8 KB	Imagem no formato...	15/2/2006 18:44
trecho	585 KB	Winamp media file	15/2/2006 18:54

Two simple files

```
F:\Estudos\StegTest>copy logo_h2hc.gif /b + trecho.mp3
logo_h2hc.gif
trecho.mp3
        1 arquivo(s) copiado(s).

F:\Estudos\StegTest>
```

Simply copy command

Nome	Tamanho	Tipo	Data de modificação
logo_h2hc	592 KB	Imagem no formato...	16/2/2007 15:05
trecho	585 KB	Winamp media file	15/2/2006 18:54

The 2 files continue, but notice the size of "logo_h2hc.gif"

Opening the file on the standard Image Visualization app, it comes up what was expected

Dragging and dropping the same GIF file on a winamp's window, we have 37 seconds of sound.

Userland protections

We enjoined this picture from Julie Tinnes presentation on Windows HIPS evaluation with Slipfest

After kernel compromise, life is never the same

There are many techniques in the wild to subvert forensics analisys

In ring0 fights, it's all a mess. -> Let's protect the ring0!

First thing the we should do to analyze a compromised machine is to clone the RAM contents. Why? Because all binaries in the system can be cheated statically (binary itself modified) or dynamically (hooked in int80h).

So, what do we find in the RAM analysis? *Should be* Everything

Structures commonly searched in memory

EPROCESS and ETHREAD blocks (with references to the memory pages used by the process/threads)

Lists like PsActiveProcessList and waiting threads to be scheduled (used for cross-view detection)

Interfaces(Ex: Ethernet IP, MAC addr, GW, DNS servers)

Sockets and other objects used by running processes (with detailed information regarding endpoints, proto, etc)

Grabbing RAM contents

RAM clone

Windows

```
E:\bin\UnicodeRelease>.\dd.exe if=\\.\PhysicalMemory
of=E:\Ram_Clone.bin bs=512 conv=noerror
```

Linux

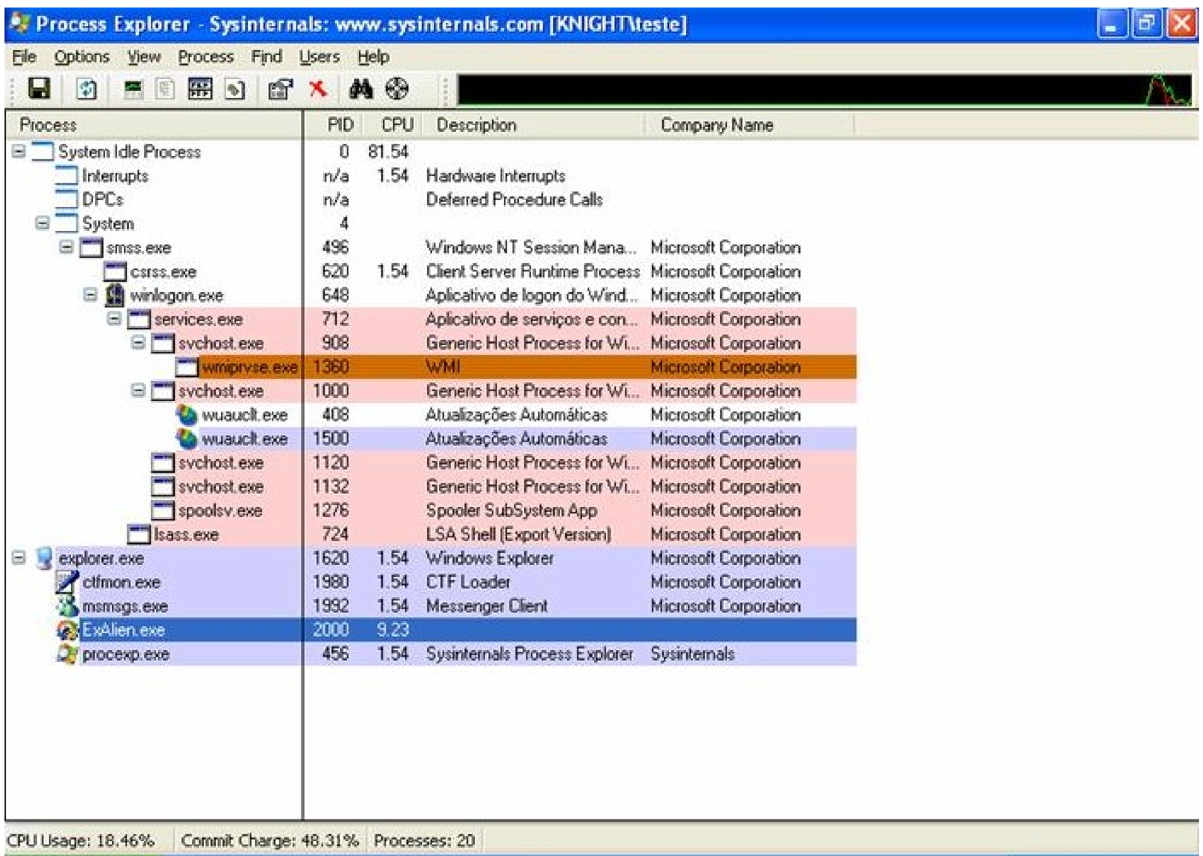
```
king:/mnt/sda1# ./dcfldd if=/dev/mem of=Ram_Clone.bin bs=512
conv=noerror
```

Trustable Method?

Windows Malware

Piece of cake: Malware running in user-space

(99% of trojan horses that attack brazilian users in Scam)

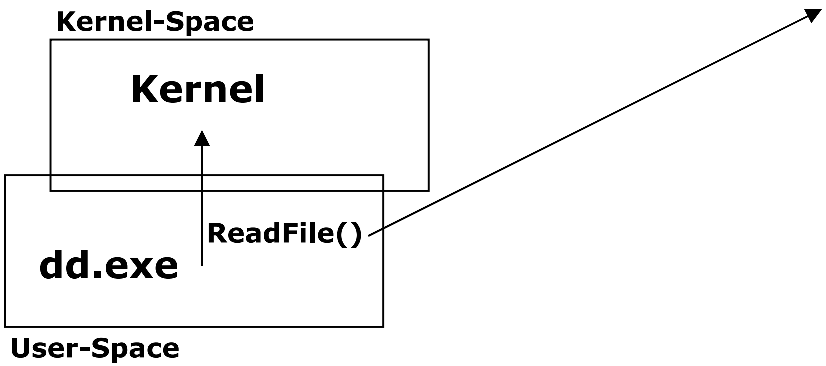


Windows Malware

Inject kernel modules to hide themselves

Examples:
- **Hacker Defender**
- **Suckit**
- **Adore**
- **Shadow Walker**

These rootkits use well known techniques (Ex: IAT hooking) to monitor/subvert user-space/kernel-space conversations.



Which File?

•\\.\PhysicalMemory

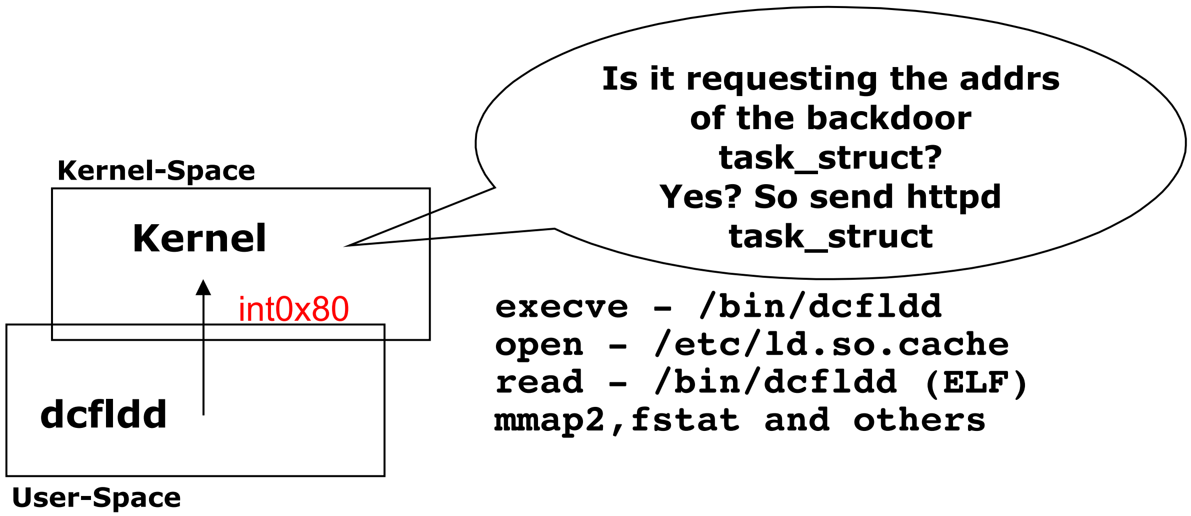
•\\.\PhysicalDrive0

Etc.

RAM Forensics – Linux Scenario

On Linux, to proceed with RAM analysis, tools like Fatkit are used (Static memory dump file analysis)

But at clone time, the destination image can be subverted if the machine is compromised with a custom rootkit



RAM Forensics

```
ssize_t h_read(int fd, void *buf, size_t count){
    unsigned int i;
    ssize_t ret;
    char *tmp;
    pid_t pid;

    If the fd (file descriptor) contains something
    that we are looking for (kmem or mem)

    return_address();
    At this point we could check the offset being
    required. If is our backdoor addr, send
    another task_struct
    ret=o_read(fd,buf,count);
    change_address();
    return ret;
    }
```

```
int return_address()
{
return our hacks to the
original state
}
```

```
int change_address()
{
put our hacks into
the kernel
}
```

Windows Malware

Let's say our scanner/detector/memory dumper/whatever resides in Kernel-Space and althout using ReadFile() uses ZwReadFile or ZwOpenKey or Zw*.**

Reliable?

- SST – System Service Table Hooking

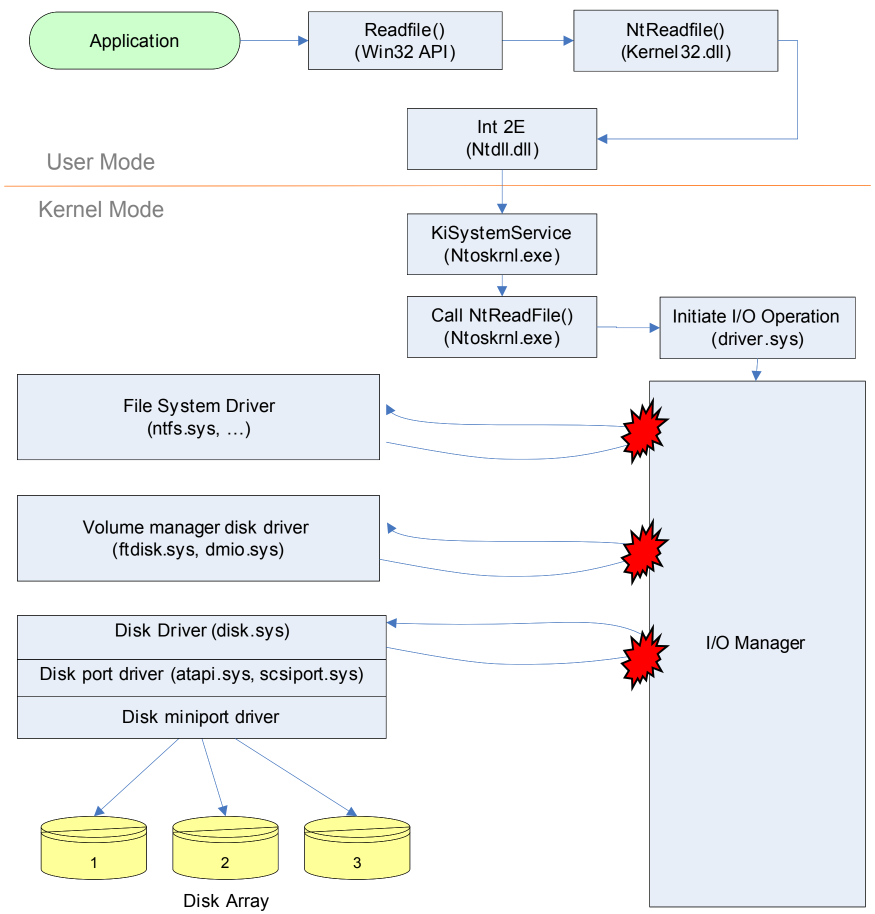
```
C:\>SDTrestore.exe
SDTrestore Version 0.2 Proof-of-Concept by SIG^2 G-TEC (www.security.org.sg)

KeServiceDescriptorTable                80559B80
KeServiceDecriptorTable.ServiceTable    804E2D20
KeServiceDescriptorTable.ServiceLimit   284

ZwClose                  19 --[hooked by unknown at FA881498]--
ZwCreateFile             25 --[hooked by unknown at FA881E16]--
ZwCreateKey              29 --[hooked by unknown at FA882266]--
ZwCreateThread           35 --[hooked by unknown at FA880F8E]--
ZwEnumerateKey           47 --[hooked by unknown at FA882360]--
ZwEnumerateValueKey      49 --[hooked by unknown at FA881EDE]--
ZwOpenFile               74 --[hooked by unknown at FA881D6C]--
ZwOpenKey                77 --[hooked by unknown at FA8822E2]--
ZwQueryDirectoryFile     91 --[hooked by unknown at FA881924]--
ZwQuerySystemInformation AD --[hooked by unknown at FA881A4A]--
ZwReadFile               B7 --[hooked by unknown at FA8810EE]--
ZwRequestWaitReplyPort   C8 --[hooked by unknown at FA881310]--
ZwSecureConnectPort      D2 --[hooked by unknown at FA8813EA]--
ZwWriteFile              112 --[hooked by unknown at FA881146]--

Number of Service Table entries hooked = 14
```

Windows Malware

Ok, let's say we want to go deeper and grab a file directly from the HD: Then we use IoCallDriver() to talk directly with the HDD.

Reliable?

- IRP (I/O Request Packet) Hooking



Fonte: Rootkits – Advanced Malware

Darren Bilby

Keep it simple!

How about if our memory grabber just sets up a pointer to offset 0x00 of RAM memory and copies to another var till it reaches the end of memory? (Regardless of race conditions to kernel memory)

Reliable?

WatchPoints in memory pages (DR0 to DR3)

When our backdoor offset is hit by the "inspector" it will generate a #DB (Debug Exception) which we can work on it

Securely? Grabbing the RAM contents

Some hardwares attempt to get the RAM contents

These type of solutions rely on the DMA method of accessing the RAM and then acting on it (CoPolit) or dumping it (Tribble)

- Tribble – Takes a snapshot (dump)of the RAM

http://www.digital-evidence.org

- CoPilot – Audits the system integrity by looking at the RAM Contents

www.komoku.com/pubs/USENIX-copilot.pdf

- Other Firewire (IEEE 1394) Methods – Michael Becher, Maximillian Dornseif, Christian N. Klein @ Core05 CanSecWest

Reliable method?

Joanna Rutkowska showed on BlackHat DC 2007 a technic using MMIO that could lead the attacker to block and trick a DMA access from a PCI card.

The Kernel War

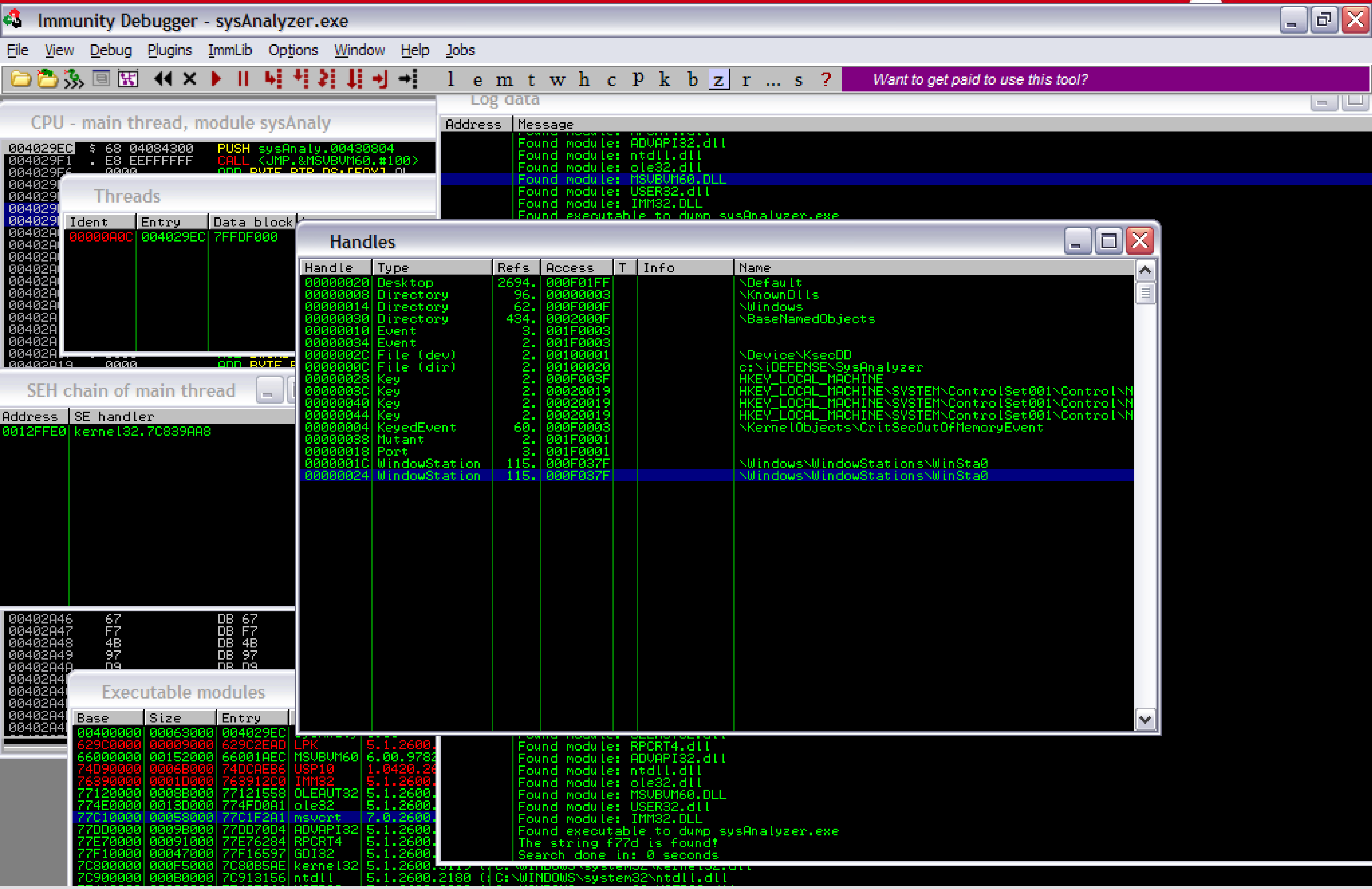
- As Montanaro showed until now in the presentation, if the attacker compromised the machine and have access to the kernel, a lot of problems will appear:
  - We can signature detect the forensics tool:
    - Multiple (continuous) memory reads
    - Multiple (continuous) disk reads
  - Even deeper:
    - Binary program signature (like antiviruses use to detect a virus)
    - Program behaviour (what the program does? how they does that?)

Looking for patterns

- **We have used the excelent Immunity Debugger with a simple python script to search a binary file for patterns:**

```
allmodules = imm.getAllModules()

for key in allmodules.keys():

    imm.Log("Found module: %s" %key)

usekey = ""

for key in allmodules.keys():

    if key.count(".exe"):

        imm.Log("Found executable to dump %s" %key)

        usekey = key

        break

module_to_dump = allmodules[key]

base = module_to_dump.getCodebase()

size = module_to_dump.getCodesize()

codememory = imm.readMemory(base,size)

hex_codememory = codememory.encode('hex-codec')

<Here you put your magic ;) like if you want to recognize sequences of bytes, strings unmodified between versions, etc>
```

Looking for patterns



Looking for patterns

- **The program behaviour is a really easy way to identify a forensic tool:**
  - Regular reads to some directories (like configuration files, libraries and others)
  - Start read position in a memory dump (some systems first try to discover a backdoor manipulating the system, opening the memory devices, some others just try to load a kernel module to verify kernel violations, etc)

Detecting forensics tool

- We can hook system loading interfaces to easily spot a new program been runned, and them analyse the program and compare to a signature base:
  - ld.so, init_module, lsm, load_binary, do_execve, do_fork, ...
- But, how about other tools?

Fighting against Forensics tools – The old school

- A lot of different talks about different ways to hide information from a Forensics tool – our approach is not to try to hide it, but discover a forensic tool running in the system (if someone is analysing the system, is because they already know something is wrong)

Old school quick tour

- **Shadow Walker talk at Blackhat by Sherri Sparks and Jamie Butler showed the idea of use TLB desyncronization to hide your rootkit**
- **Basicly it uses:**
  - Page fault handling patches
  - Pages are marked as non-present, and the page-fault system will verify if the instruction pointer is pointing to the faulted address (cr2) to differentiate between a read/write and one execution
  - The page fault system marks this pages as non-pageable to differentiate between 'protected' pages and the common ones (in Linux if you are just using kernel pages don't need to care about that)

Old school quick tour

- **There are a lot of problems with this approach against a Forensic analyst (skilled one) – as spotted by the authors of this idea:**
  - It's easy to detect IDT modifications and for sure to check the page faulting mechanics
  - Non present pages in non paged memory range are really not normal

Old school quick tour

- Another approach is to hide your patches to the kernel using the debugger registers (we covered a lot about how to do that in our presentation about kernel integrity protection in the VNSecurity Conference)
- The problem is it can also be verified just using the segmentation support existent in the platform to bypass breakpoint hit or (also easy) just patching the debugging interrupt handling by yourself and trying to modify the debug registers (it will generate and exception if someone have set the general detection flag in dr7)

Anti-forensics hide rootkit

- **If you need to use disk (to transfer things to the machine and don't want to use syscall proxying-like systems) you can do that in many different ways (pointed by Montanaro) and also:**
  - Transfer your data to system memory
  - Force it to be loaded in a high virtual memory, and causes a page-out of this data (you also need to patch the paging system)
  - If it is a big machine you can use kmap to remap your addresses from ZONE_HIGH to ZONE_NORMAL when you need to manipulate it (read/write)
  - A simple crypting routine using a session key is enough (do you remember we are protecting the system against a memory dump) – We don't care about rootkit detection itself

What is needed in an anti-forensic rootkit?

- It must detect a forensic analysis and react to it (maybe removing all the evidences, including itself)
- In some way it must be 'pattern free', so it cannot be detected by common ways (to detect it will be needed a lot of knowledge from the analyst, and it is almost impossible to detect if you don't know the rootkit itself)
- Maybe the Virtualized Rootkit is dead, but what about use another hardware resource in rootkits?

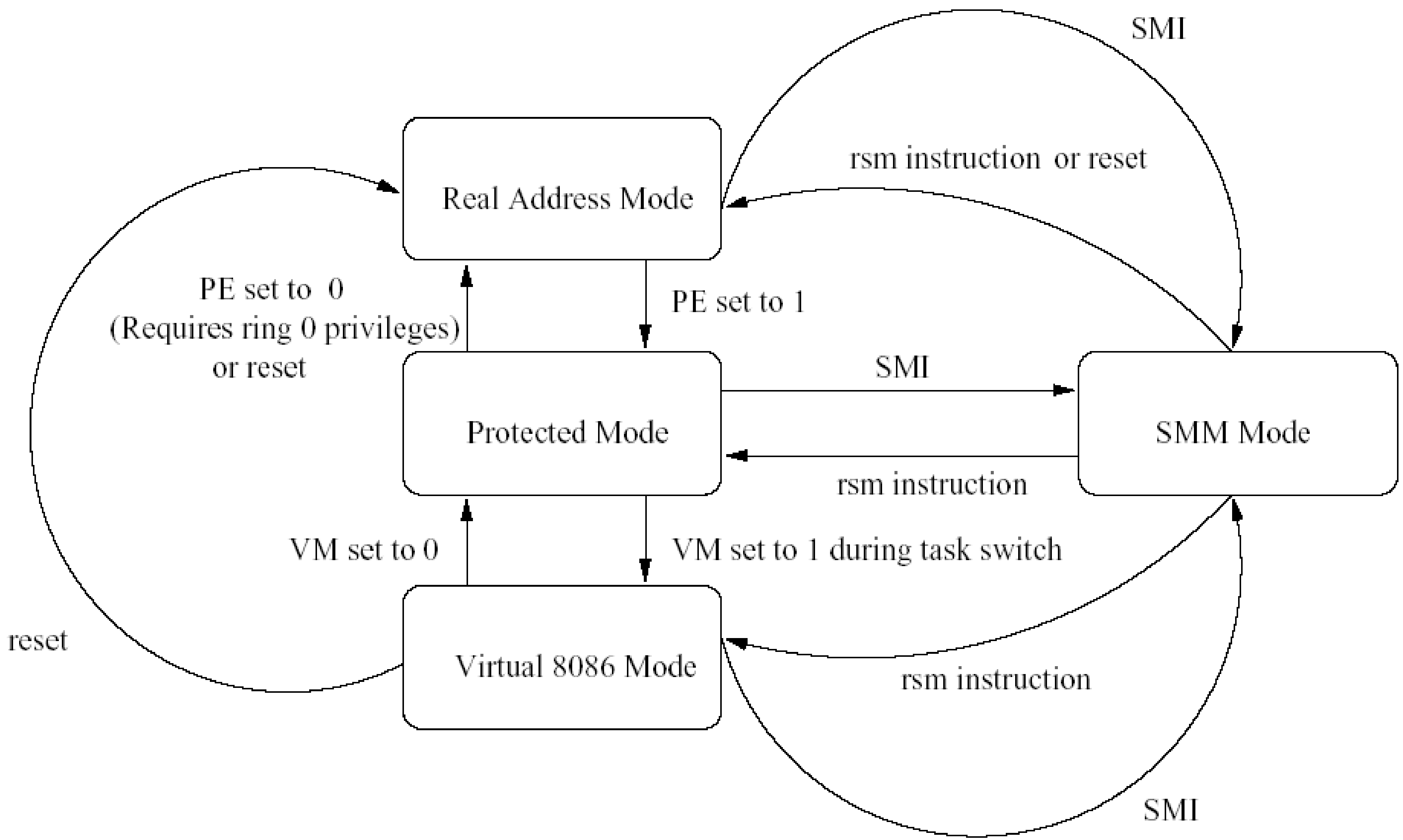
How? SMM!

SMM – System Management Mode

The Intel System Management Mode (SMM) is typically used to execute specific routines for power management. After entering SMM, various parts of a system can be shut down or disabled to minimize power consumption. SMM operates independently of other system software, and can be used for other purposes too.

From the Intel386™ Product Overview – intel.com

SMM and Anti-Forensics?



SMM and Anti-Forensics?

- Duflot paper released a way to turn off BSD protections using SMM
- A better approach can be done using SMM, just changing the privilege level of a common task to RING 0
- The segment-descriptor cache registers are stored in reserved fields of the saved state map and can be manipulated inside the SMM handler
- We can just change the saved EIP to point to our task and also the privilege level, forcing the system to return to our task, with full memory access
- Since the SMRAM is protected by the hardware itself, it is really difficult to detect this kind of rootkit

Descriptor Cache

- From the Intel Manual: “Every segment register has a “visible” part and a “hidden” part. (The hidden part is sometimes referred to as a “descriptor cache” or a “shadow register.”) When a segment selector is loaded into the visible part of a segment register, the processor also loads the hidden part of the segment register with the base address, segment limit, and access control information from the segment descriptor pointed to by the segment selector. “
- RPL – Request Privilege Level
- CPL – Current Privilege Level
- DPL – Descriptor Privilege Level

Descriptor Cache

- **In the saved state map (inside SMM):**
- TSS Descriptor Cache (12-bytes) - Offset: 7FA4
- IDT Descriptor Cache (12-bytes) - Offset: 7F98
- GDT Descriptor Cache (12-bytes) - Offset: 7F8C
- LDT Descriptor Cache (12-bytes) - Offset: 7F80
- GS Descriptor Cache (12-bytes) - Offset: 7F74
- FS Descriptor Cache (12-bytes) - Offset: 7F68
- DS Descriptor Cache (12-bytes) - Offset: 7F5C
- SS Descriptor Cache (12-bytes) - Offset: 7F50
- CS Descriptor Cache (12-bytes) - Offset: 7F44
- ES Descriptor Cache (12-bytes) - Offset: 7F38

SMM Relocation

- SMM has the ability to relocate its protected memory space. The SMBASE slot in the state save map may be modified. This value is read during the RSM instruction. When SMM is next entered, the SMRAM is located at this new address - in the saved state map offset 7EF8
  - Some problems to perform CS adjustments
- It can be used to avoid SMM memory dumping for analysis

Generating #SMI's

- We explained really deeply why the system will generate #SMI in Xcon this year
- Now, we can just instrument our kernel (in any portion of it, so turning really difficult to detect) an I/O operation to a shared address between devices (as Duflot spotted in his paper, 0xA0000h) sounds good
- This idea can be used together with a BIOS rootkit, to configure an SMI handler, lock the SMM (relocating the SMRAM) and then transfering control back to normal boot system – if someday the system triggers a SMI, it will install the backdoor, bypassing all kind of boot protections