RSA CONFERENCE 2014
FEBRUARY 24 – 28 | MOSCONE CENTER | SAN FRANCISCO

Anti-Stealth Techniques: Heuristically Detecting x64 Bootkits

SESSION ID: HTA-T07A

Lars Haukli

Security Researcher
Blue Coat
@zutle

The Realization

- Bootsectors only way of compromising x64 [1]
- Cross view detection effective for quite some time
- Enter Rovnix [2,3]
  - Polymorphism instead of faked contents
  - Rendered cross view useless
- Eureka! Found another way
  - Potentially detect all known x64 kernel-mode rootkits to date!

Common Denominators

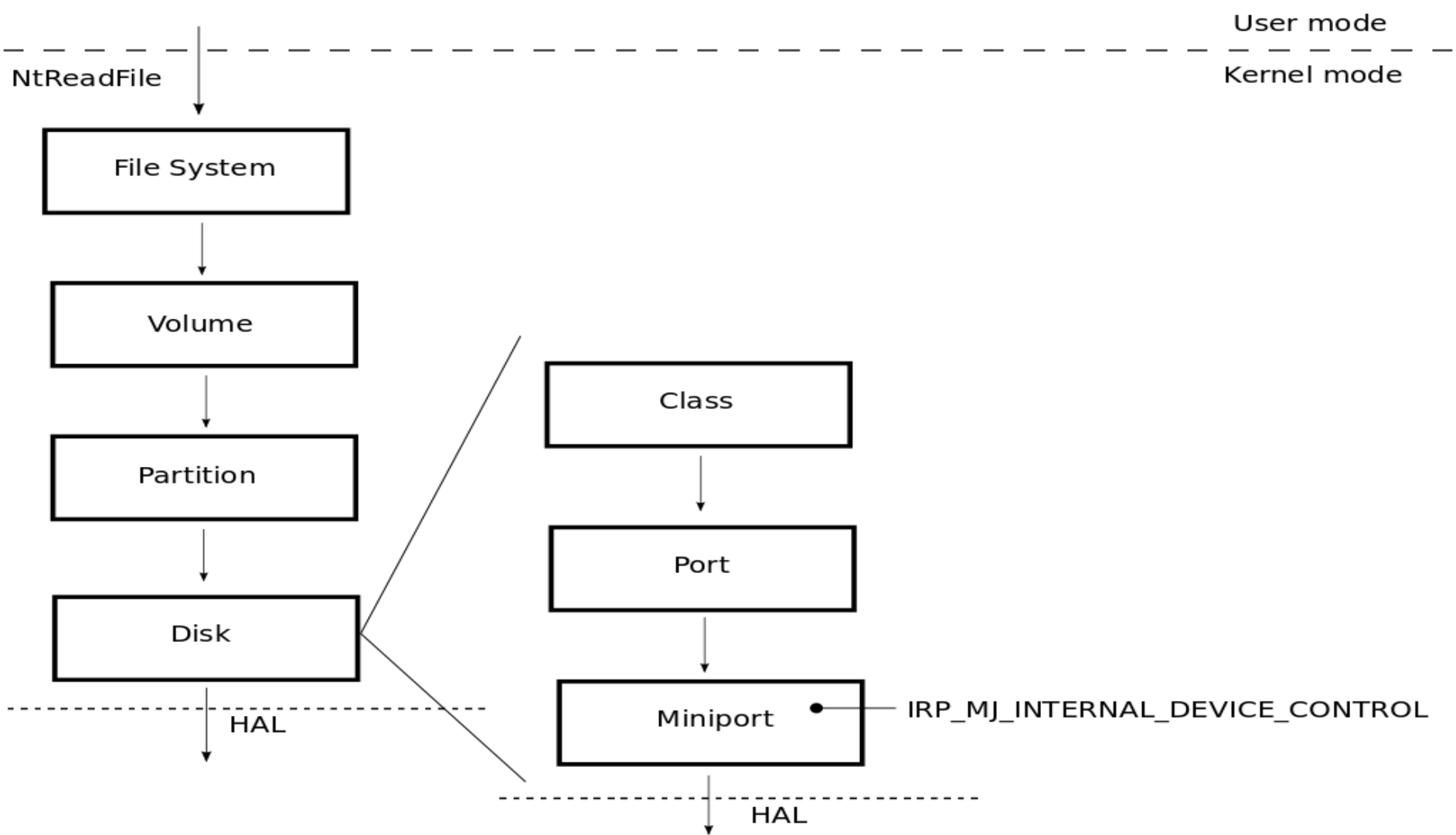
- Aim is to load an unsigned kernel mode driver
  - Security model prevents this
- BIOS boot sequence is insecure
  - Still used (backwards compatibility)
- Boot sequence manipulated
  - Run code before security features kick in

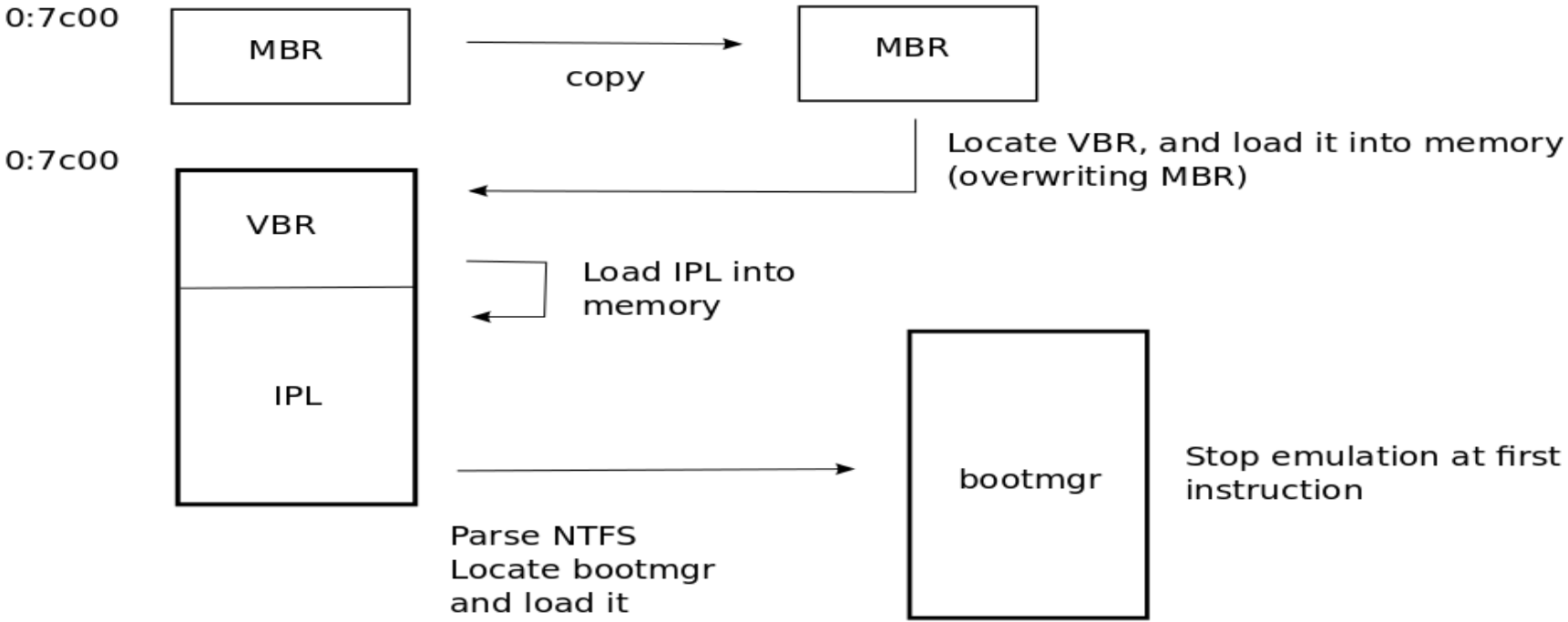
BIOS Boot Sequence

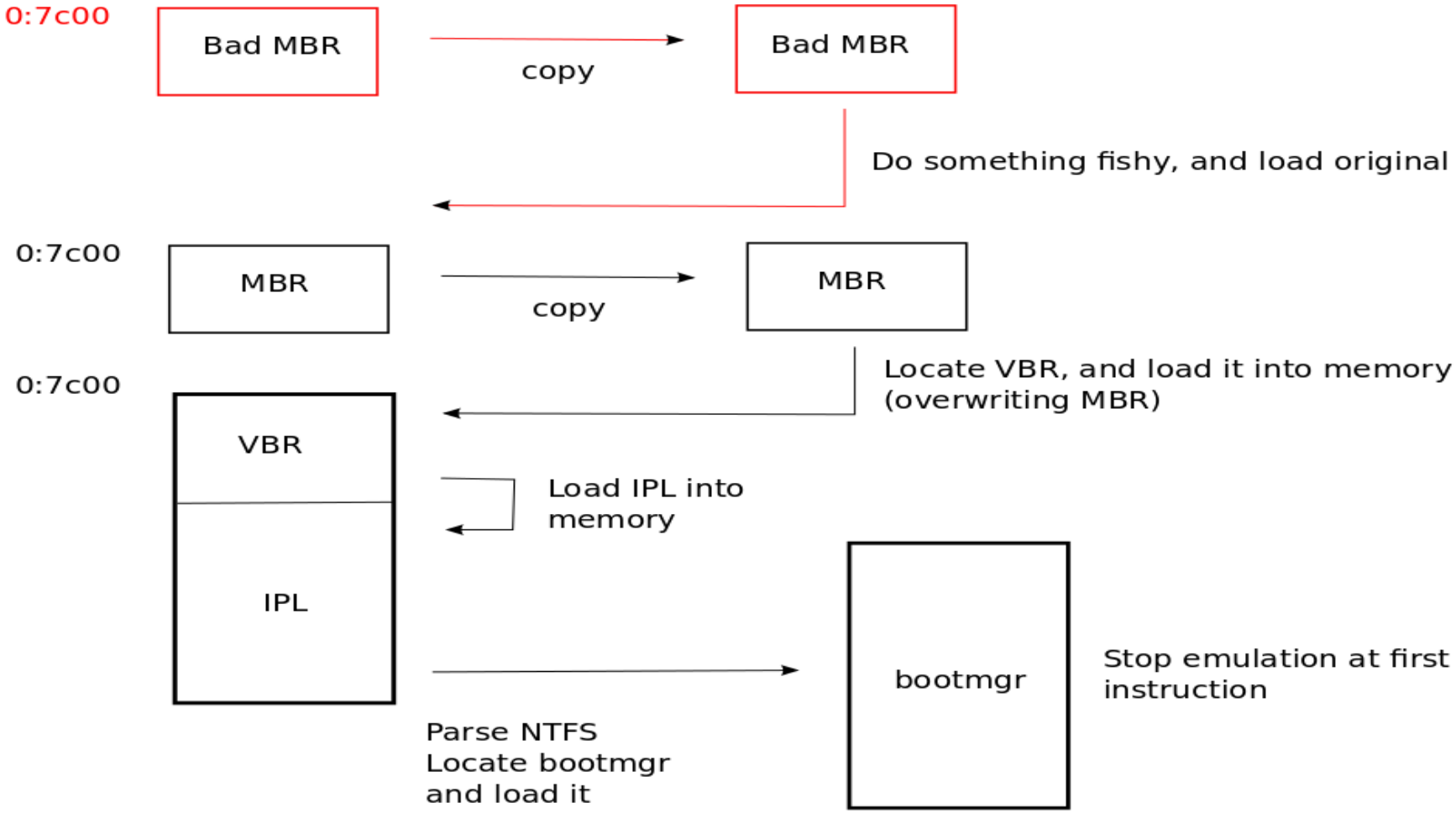
- 17 unsigned sectors on disk
  - MBR, VBR, bootstrap code (IPL)
- Responsible for locating and loading the “OS Loader”
  - bootmgr on Vista+
  - ntldr on XP
- Relies on BIOS interrupts for its heavy lifting
  - The most important one being int 13h for disk I/O

Emulating the Boot Sequence

- Use custom BIOS
  - Setup IVT (interrupt handlers)
- Load MBR into emulator memory at 0:7c00 and start emulation
- Stop emulation at first instruction of bootmgr
  - bootmgr is the first signed component in the boot sequence
- int 13h handler must use anti-rootkit techniques
  - Bypass any low level hooks to avoid reading faked contents


User mode
Kernel mode
NtReadFile
File System
Volume
Partition
Disk
HAL
Class
Port
Miniport
IRP_MJ_INTERNAL_DEVICE_CONTROL
HAL


0:7c00
MBR
copy
MBR
Locate VBR, and load it into memory
(overwriting MBR)
0:7c00
VBR
IPL
Load IPL into
memory
bootmgr
Stop emulation at first
instruction
Parse NTFS
Locate bootmgr
and load it


0:7c00
Bad MBR
copy
Bad MBR
Do something fishy, and load original
0:7c00
MBR
copy
MBR
0:7c00
Locate VBR, and load it into memory
(overwriting MBR)
VBR
Load IPL into
memory
IPL
bootmgr
Stop emulation at first
instruction
Parse NTFS
Locate bootmgr
and load it

Heuristic Detection

- Anomalies in boot sectors
  - Executing 0:7c00 more than twice
  - Hooking int 13h
  - bootmgr patched in memory
  - Placement of VBR on disk

Disabling Bootkits

- MBR case
  - Stop emulation at second execution of 0:7c00
- VBR/IPL case
  - Stop emulation at first instruction of bootmgr
- Retrieve original contents from emulator memory
- Replace modified parts with original
  - Breaks the rootkit's load chain
  - Reboot system, voila!

Conclusion

- Anomalies in boot sectors are detectable by emulation
- Emulation makes it trivial to retrieve original contents
  - Counters polymorphism
  - Counters encryption
- Break the rootkit's load chain to defeat it
- UEFI instead of BIOS
  - Signed firmware is secure
  - Artifacts from 1970s are not

BLUE COAT

Thank you for your attention!

Special thanks to my co-researcher Leif Arne Søderholm

Lars Haukli

Security Researcher, Blue Coat
lars.haukli@bluecoat.com
@zutle

BLUE COAT

Appendix

References

[1] “Securing the Windows 8 Boot Process”, Microsoft Technet, http://technet.microsoft.com/en-us/windows/dn168167.aspx

[2] “Hasta La Vista, Bootkit: Exploiting the VBR”, Aleksandr Matrosov, Eugene Rodionov, David Harley

http://www.welivesecurity.com/2011/08/23/hasta-la-vista-bootkit-exploiting-the-vbr/

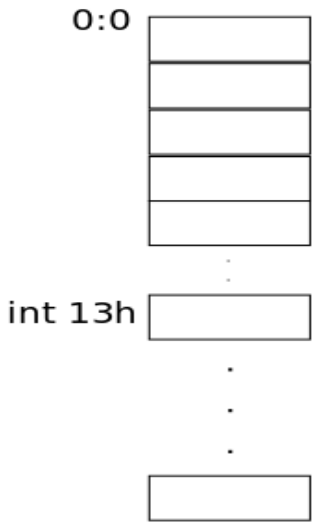
References

[3] Rovnix bootkit framework updated, Aleksandr Matrosov,

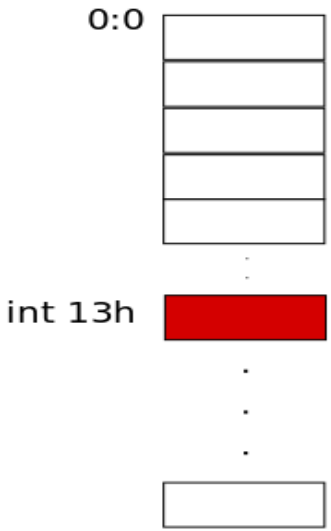
http://www.welivesecurity.com/2012/07/13/rovnix-bootkit-framework-updated/

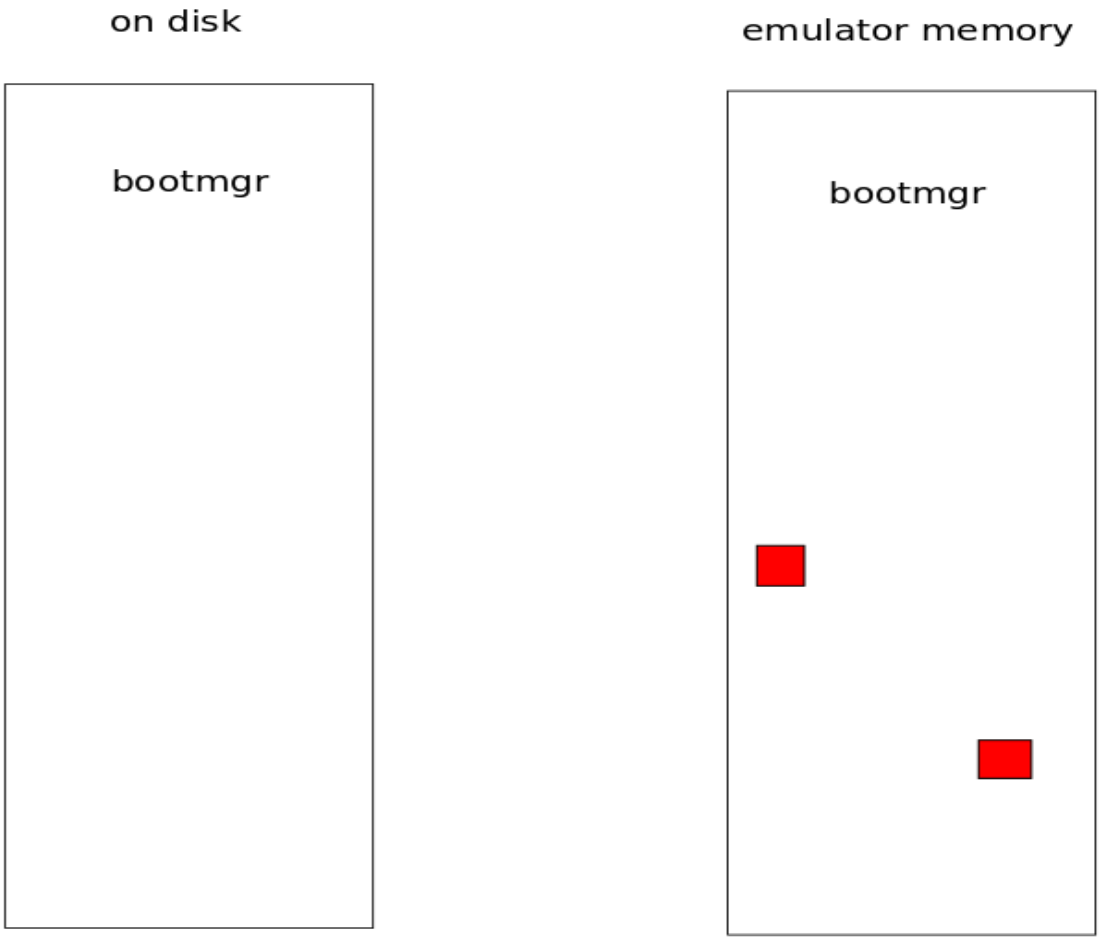
Additional illustrations

Emulator memory before emulation



Emulator memory after emulation




on disk
bootmgr
emulator memory
bootmgr