Exploring the impact of a hard drive backdoor

Jonas Zaddach
<zaddach@eurecom.fr>

Outline

- Introduction
- Firmware reverse engineering
- Backdoor injection
- Remote access
- Discussion
- Conclusion

About myself

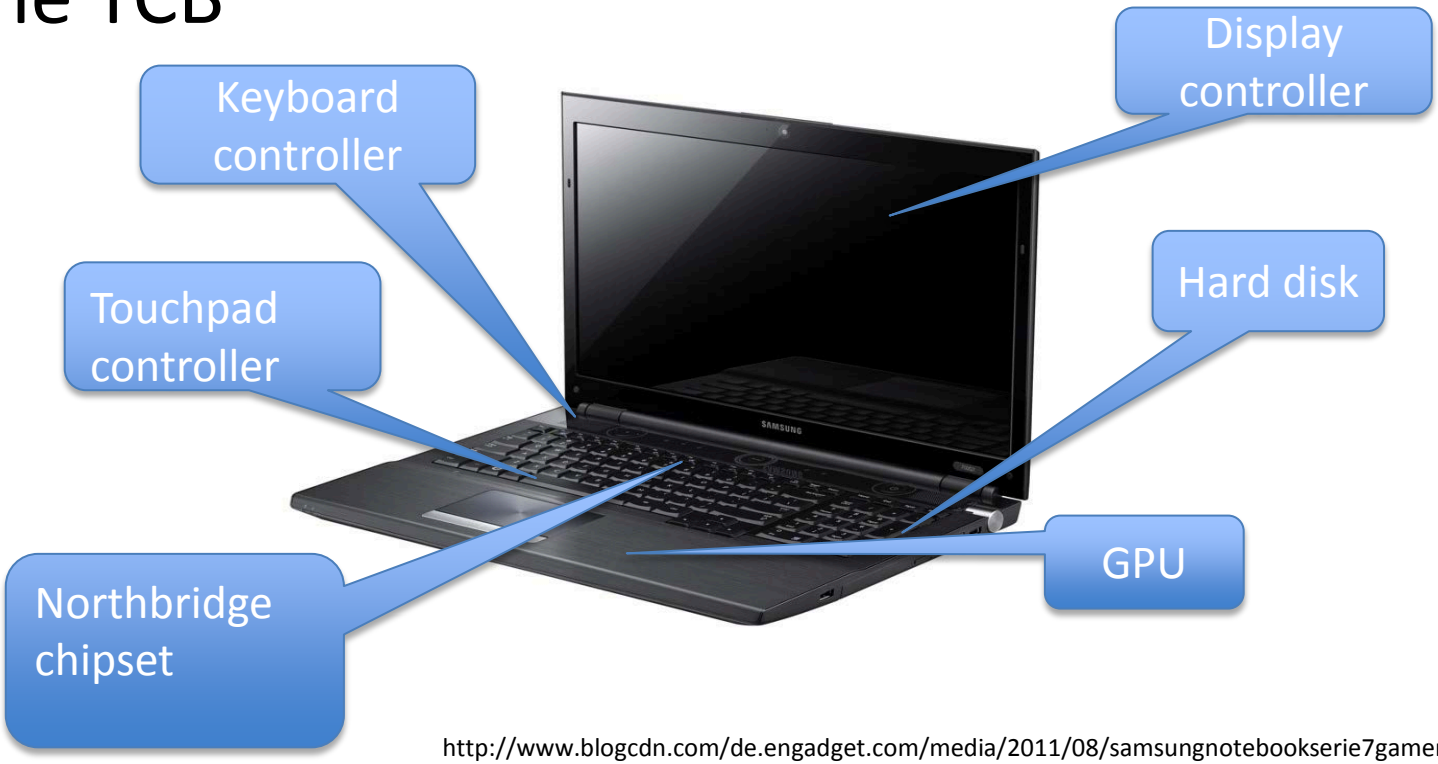
- PhD Candidate on the topic of Embedded Firmwares' Security at EURECOM
- My website (Publications, etc)
- Current work
  - Avatar – Firmware emulation
  - Firmware survey project

Acknowledgements

- Thanks to my Advisor Davide Balzarotti and co-advisor Aurélien Francillon for enabling me to do research!
- Thanks to Travis Goodspeed for getting me started on hacking this HDD
- Thanks to all the authors (Anil, Travis, Moitrayee, Davide, Aurélien, Erik, Ioannis) of our paper for this great research

Motivation

- A computer of computers: All code is part of the TCB



http://www.blogcdn.com/de.engadget.com/media/2011/08/samsungnotebookserie7gamer2.jpg

Motivation

- Why a firmware attack?
  - Firmware infections are very hard to find and even harder to remove
- Why the hard drive?
  - Almost all persistent information is stored on hard drives
- How can such a backdoor be accessed?
  - Shown in this presentation

Goals

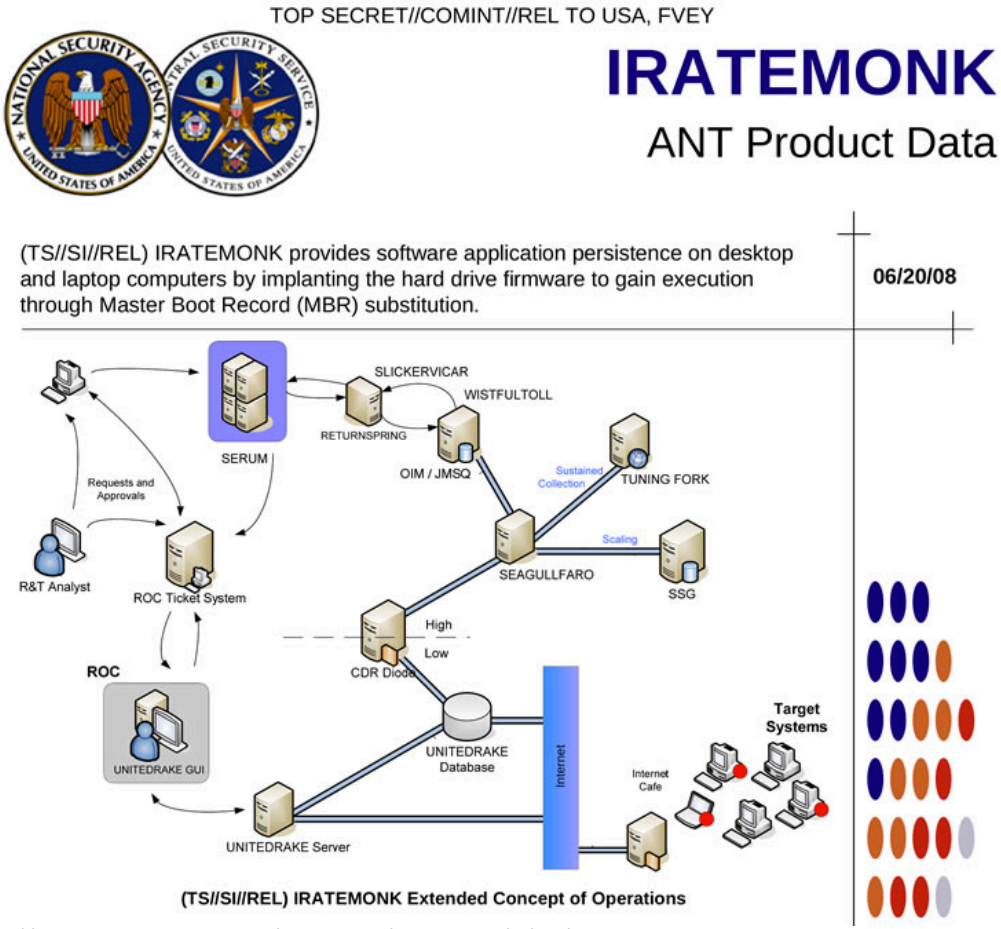
- Compromise the firmware of a COTS disk
- Design a backdoor to exfiltrate data
- Evaluate performance and impact
- Discuss countermeasures

Similar work

- Similar hacking was presented by sprite_tm (Jeroen Domburg) at OHM2013
  - Different HDD brand
  - Using JTAG debugging
  - More information here: http://spritesmods.com/?art=hddhack

Similar work

- But we were both not the first to try this idea …



http://upload.wikimedia.org/wikipedia/commons/1/1a/NSA_IRATEMONK.jpg

Historical development

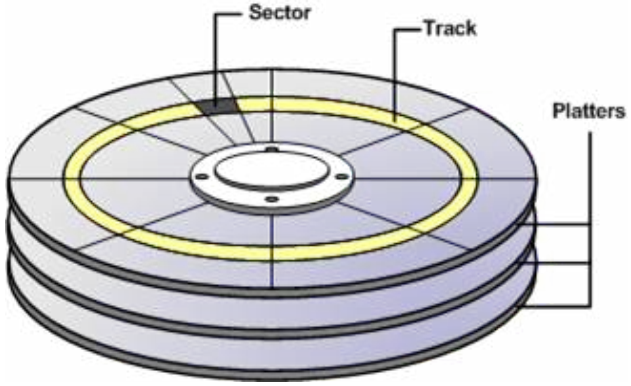
- IBM 350: Announced in 1956

http://www-03.ibm.com/ibm/history/exhibits/storage/images/PH0350A.jpg

Introduction of IDE drives

- Integrated Disk Electronics simplifies HDD attachment
  - Disk controller steers motors and analog data stream coding
  - PC speaks to drive over AT attachment protocol

http://www.harddriverecovery.org/blog/wp-content/uploads/2011/06/seagatedrive.jpg



http://www.escotal.com/Images/computer/hardrivegeometry.jpg

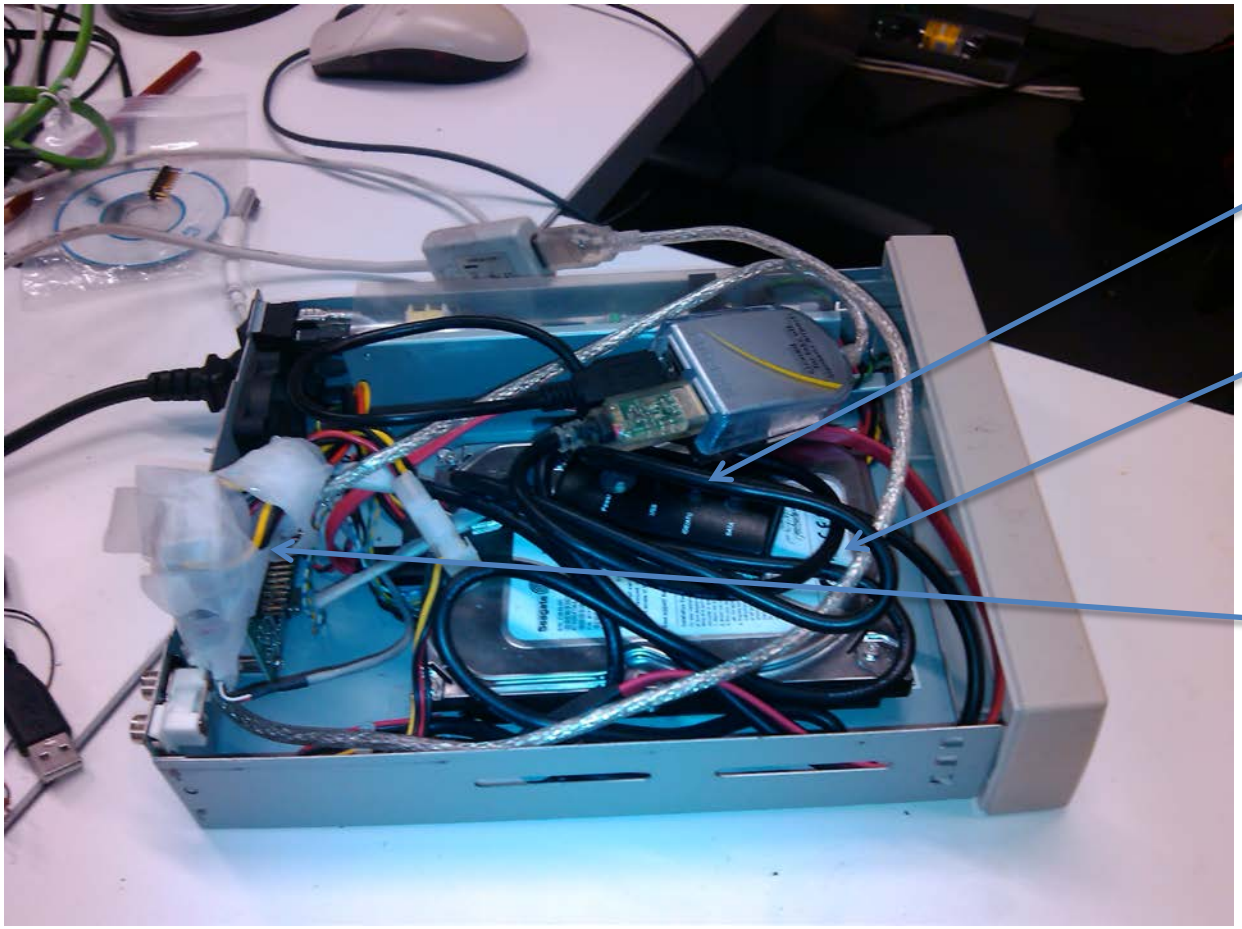
Typical HDD firmware

- Runs on a microprocessor (ARM, MIPS, …)
- Can be reprogrammed
- Is stored in flash and on disk
- Has several tasks
  - Decode ATA protocol
  - Translate Logical Block Addressing (LBA) to disk geometry (Cylinder Head Sector – CHS)
  - Coordinate other electronics (Motors, data stream decoding)

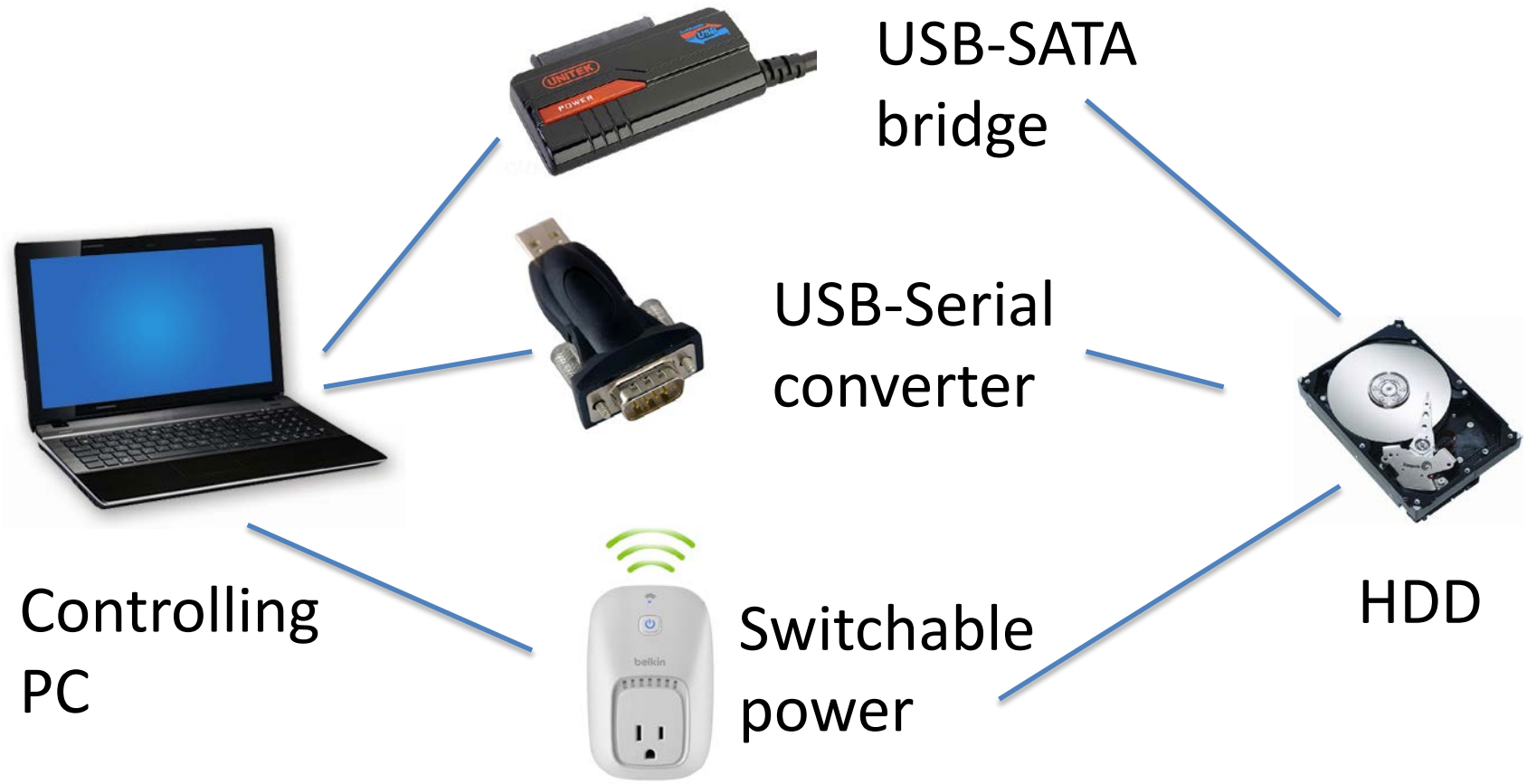
Outline

- Introduction
- Firmware reverse engineering
- Backdoor injection
- Remote access
- Discussion
- Conclusion

Second Prototype



Experimental setup



Accessing the firmware

- Firmware update files are in proprietary format
  - not straightforward to reverse
- JTAG on the PCB seems to be disabled
  - OpenOCD cannot read memory
- Serial port on master-slave jumpers shows diagnostic menu

Diagnostic firmware menu

- Diagnostic menu is accessed by pressing CTRL-Z in the serial terminal[1]

```
Online ESC: Rev 0011.0000, Flash,   Abort Looping Command or Batch File
Online '?': Rev 0011.0000, Flash,   Display Diagnostic Buffer Information
Online ^Z: Rev 0011.0000, Flash,   Enable ASCII Diagnostic Serial Port Mode
All Levels '+': Rev 0012.0000, Flash,   Peek Memory Byte,
   +[AddrHi],[AddrLo],[NotUsed],[NumBytes]
All Levels '-': Rev 0012.0000, Flash,   Peek Memory Word, -
   [AddrHi],[AddrLo],[NotUsed],[NumBytes]
All Levels '=': Rev 0011.0002, Flash,   Poke Memory Byte,
   =[AddrHi],[AddrLo],[Data],[Opts]
Online ^C: Rev 0011.0000, Flash,   Firmware Reset
```

[1] http://forum.hddguru.com/viewtopic.php?t=11926&start=

Dumping the firmware

- Python script can extract firmware
  - Accessing invalid addresses crashes firmware
  - Neighborly thanks to Travis Goodspeed for dumping the firmware
- Code execution not possible
  - Code is write-protected, cannot insert hook into execution flow

Bootloader Prompt

- CTRL-C reboots and displays bootloader

```
ASCII Diag mode

F3 T>
Spinning Down

Spin Down Complete
Elapsed Time 6.012 secs
Delaying 5000 msec

Jumping to Power On Reset
SEA-3 Yeti Boot ROM 2.0  (12/06/2007)
Copyright Seagate  2007
```

```
Boot Cmds:
 DS
 AP <addr>
 WT <data>
 RD
 GO
 TE
 BR <divisor>
 BT
 WW
?
 RET
>
```

Inject code

- Bootloader menu commands allow code execution
  - AP sets address pointer
  - WR writes byte to address pointer
  - RD reads byte from address pointer
  - GO executes code at address pointer
- *Getc* and *putc* functions are known from disassembly
- With some trial and error a self-developed tiny GDB stub (2.6k) can be injected

GDB Stub

- Uses a serial interface and a simple text-based protocol
  - 6 primitives are enough to give debugging support with software breakpoints:
    Read memory, write memory, read registers, write registers, continue and get signal
- GDB's stub implementation is not for ARM and too big (for my purpose)

Reconaissance

- Gather information on processor
  - CPUID → Arm966
  - Debug unit → Disabled
  - Caches → No caching
- Reconstruct memory map
  - Some memory regions are known from the FW dump
  - IO region is known from disassembling serial port driver
- Dump flash memory contents

Memory Map

Memory Range	Type
0x00000000 – 0x00008000	Code SRAM
0x00100000 – 0x00120000	ROM
0x00200000 – 0x00400000	Code DRAM
0x04000000 – 0x04004000	Data SRAM
0x06000000 – 0x07000000	Data DRAM
0x40000000 – 0x50000000	IO

Overview of the boot process

- ROM bootloader
  - Loads next stage from flash
- Flash bootloader
  - “Stripped-down” firmware
  - Enables DRAM and sets up memory protection
  - Loads main FW from disk
- Main firmware
  - Handles normal disk operation
- Overlays
  - Loaded by main FW, e.g., for the diagnostic menu

Keeping control

- Software debugging is fragile
  - Overwriting exception vectors removes debugger handler
  - Memory write protection prevents setting breakpoints
  - Memory layout changes necessitate moving debugger stub
- No external debugging interrupt
  - Emulated with breakpoint in serial receive interrupt

Analysis woes

- Analyzing the firmware turned out to be quite hard …
  - Almost no strings
  - No known APIs
  - Software debugger cannot set watchpoints
    - Data tracing is hard
  - Firmware excessively uses of global variables
  - Lots of function pointer tables

Understanding the OS

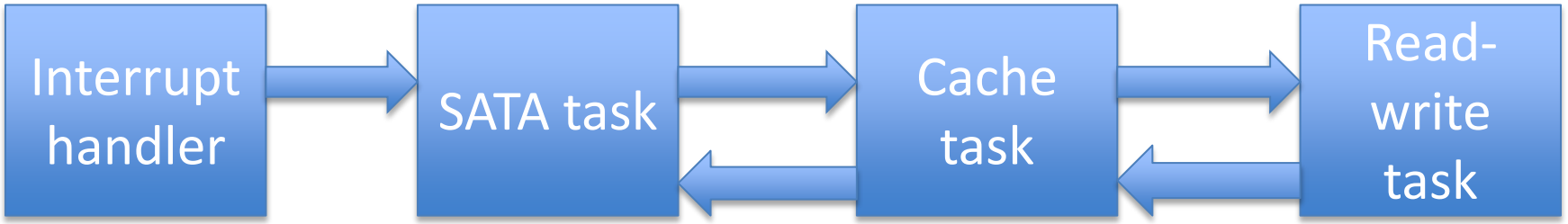
- Custom real-time OS
- Simple scheduler
  - Fixed number of tasks
  - Event-based
    - Tasks are woken depending on accepted events mask
  - Preemptive
    - Tasks are changed after interrupts
  - Cooperative
    - Task yields when generating an event

Reversing ATA command handling

- Experiment setting
  - HDD connected through USB-SATA bridge
  - Bridge controlled by Python libusb script
  - Cypress bridge chip has special mode for sending raw ATA commands :)
  - (Also Linux kernel does not like devices that do not respect SATA timeouts)

Tasks involved in reading

- ATA read command received by HDD
- Tasks process command by passing events
  - Execution traces can now be recorded with AVATAR



Outline

- Introduction
- Firmware reverse engineering
- Backdoor injection
- Remote access
- Discussion
- Conclusion

Hooking the backdoor

- Data can be modified anywhere between reception and R/W task
  - This backdoor hooks between cache and read/write task
- Checksums protect data integrity per block
  - 16-bit checksum
  - 32-bit checksum
  - Checksumming code is contained in firmware …

Simple solution

- First hook is in write path and scans block for magic commands
  - If a command is detected, LBA to read is stored in memory
- Second hook is in read path and checks if
  - Backdoor has stored LBA to read
  - Read LBA is a trigger LBA

  → Replace LBA to read with the one from the backdoor

Interfacing the backdoor

ATA cmd: Read LBA 0x4567

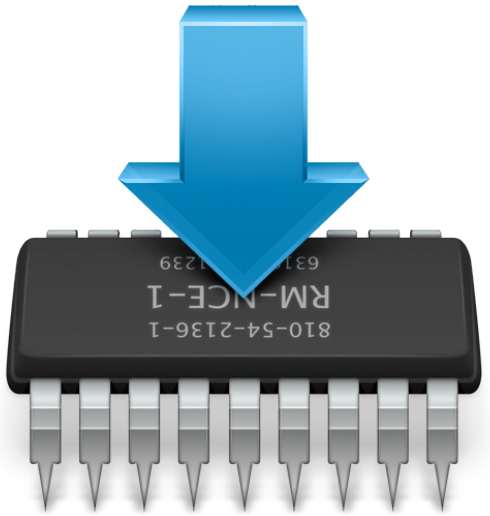
Content of LBA 0x1234

Backdoor copies sector at LBA 0x1234 to 0x4567

Backdoor detects command

Making the backdoor permanent

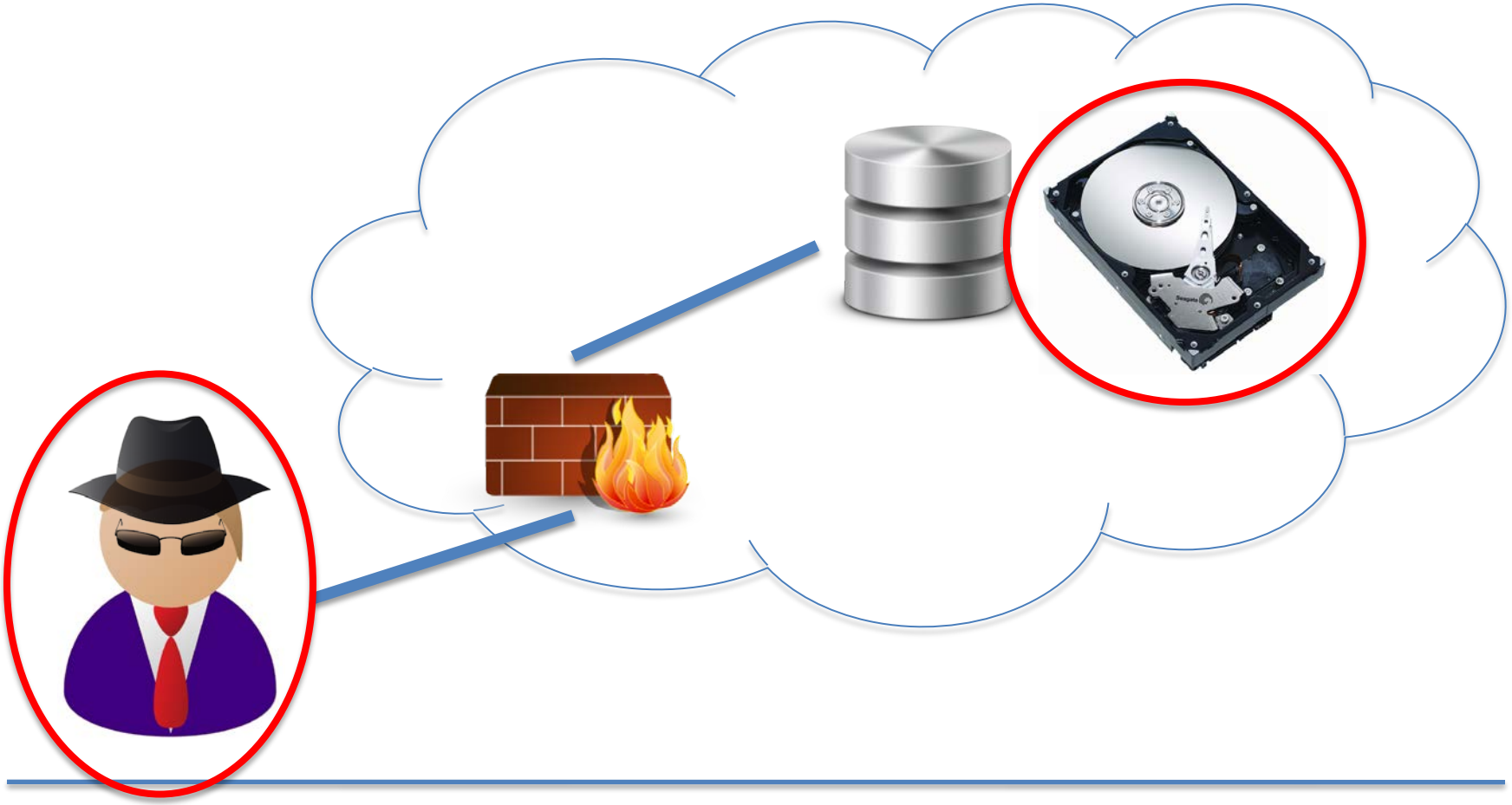
- Firmware update file format reverse-engineered
- HDParm or custom driver could send firmware update command
- Once installed, a malicious FW can refuse firmware updates



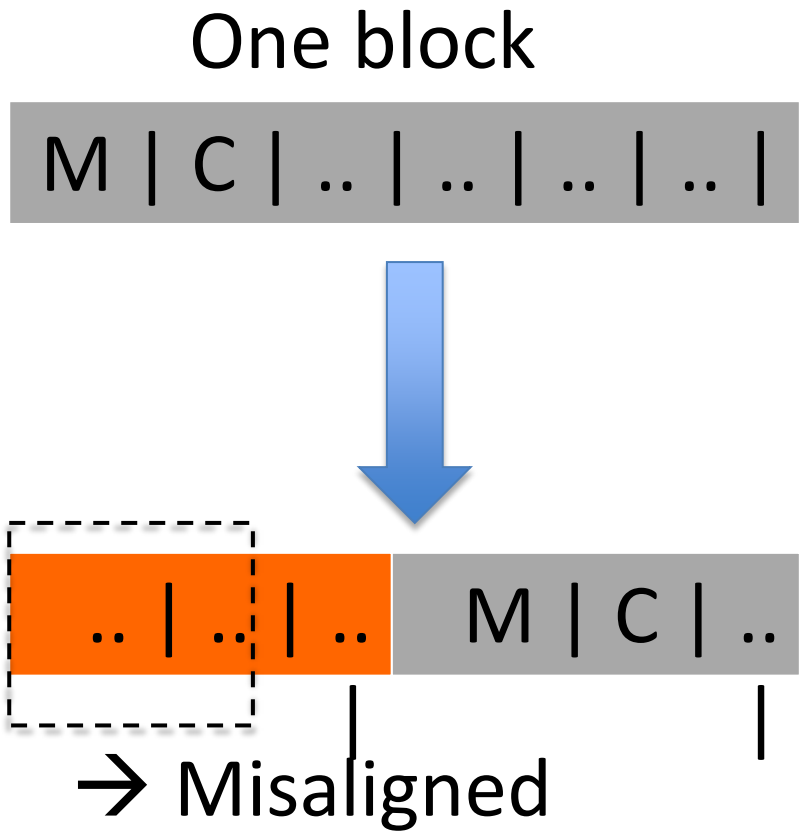
Outline

- Introduction
- Firmware reverse engineering
- Backdoor injection
- Remote access
- Conclusion

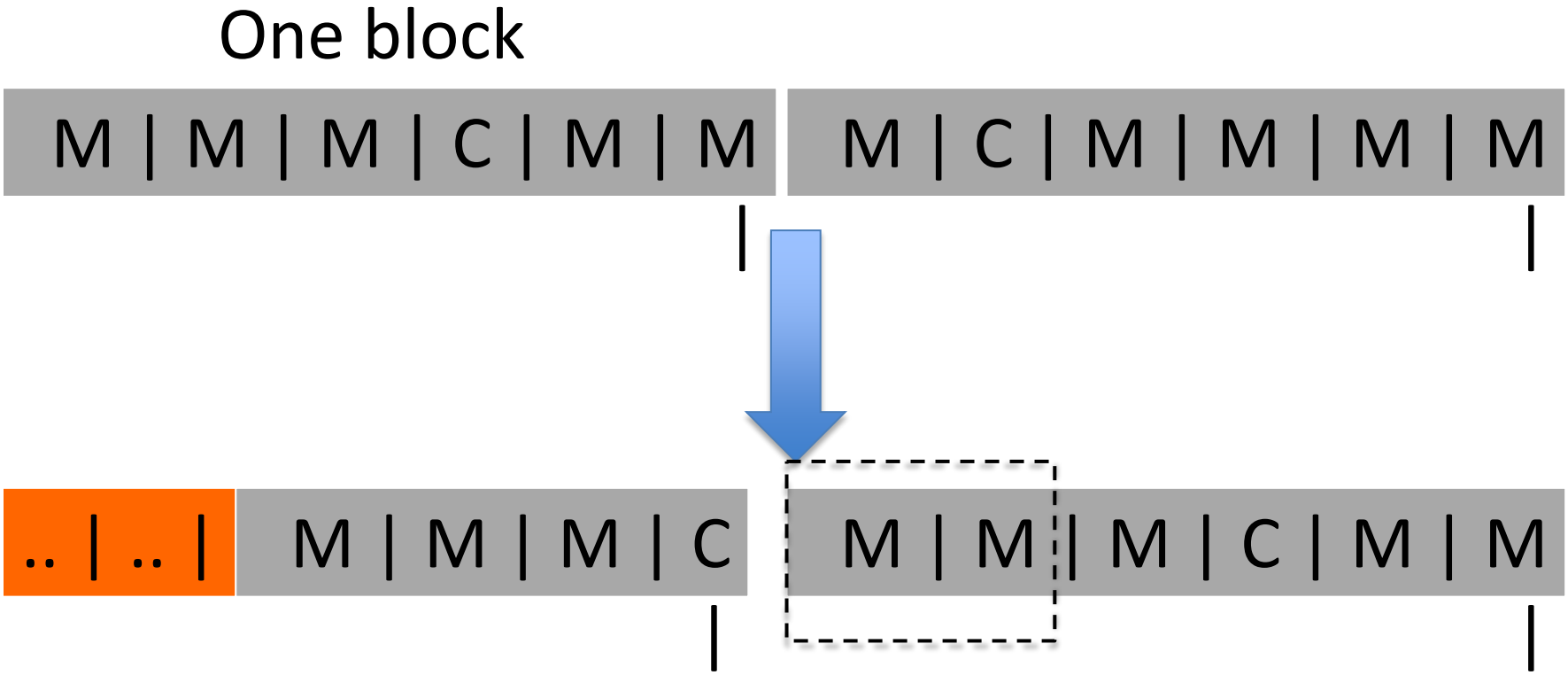
Scenario description



Handling misalignment



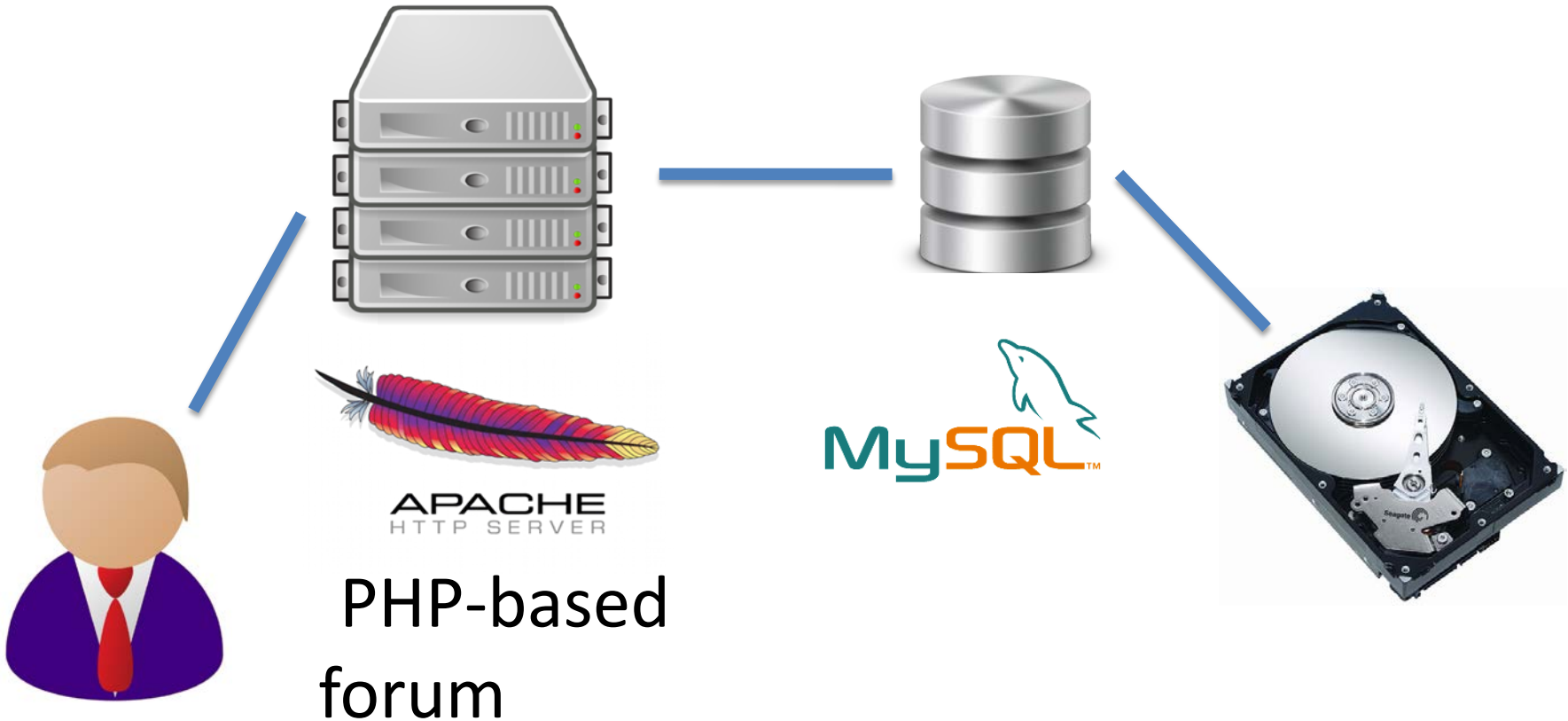
Handling misalignment



Exfiltration tweaks

- Make data robust
  - ASCII-Armor (base64)
- Caching
  - Wait
  - Create dummy traffic

Experiment setting



PHP-based forum

Exfiltration of /etc/shadow

- HDD filesystem is “mounted” in Python
- Exfiltrate /etc/shadow in nine “queries”
  - Read MBR from block 0
  - Read superblock if ext3 partition
  - …
- Total time < 1 minute

Outline

- Introduction
- Firmware reverse engineering
- Backdoor injection
- Remote access
- Discussion
- Conclusion

Limitations

- Backdoor commands need to pass the block cache
  - In Linux, blocks are cached in memory and only evicted to the HDD when necessary
  - Limits maximum throughput
- In a RAID, HDD has only a partial view of the stored data
- Software encryption defeats the backdoor

Addressing limitations

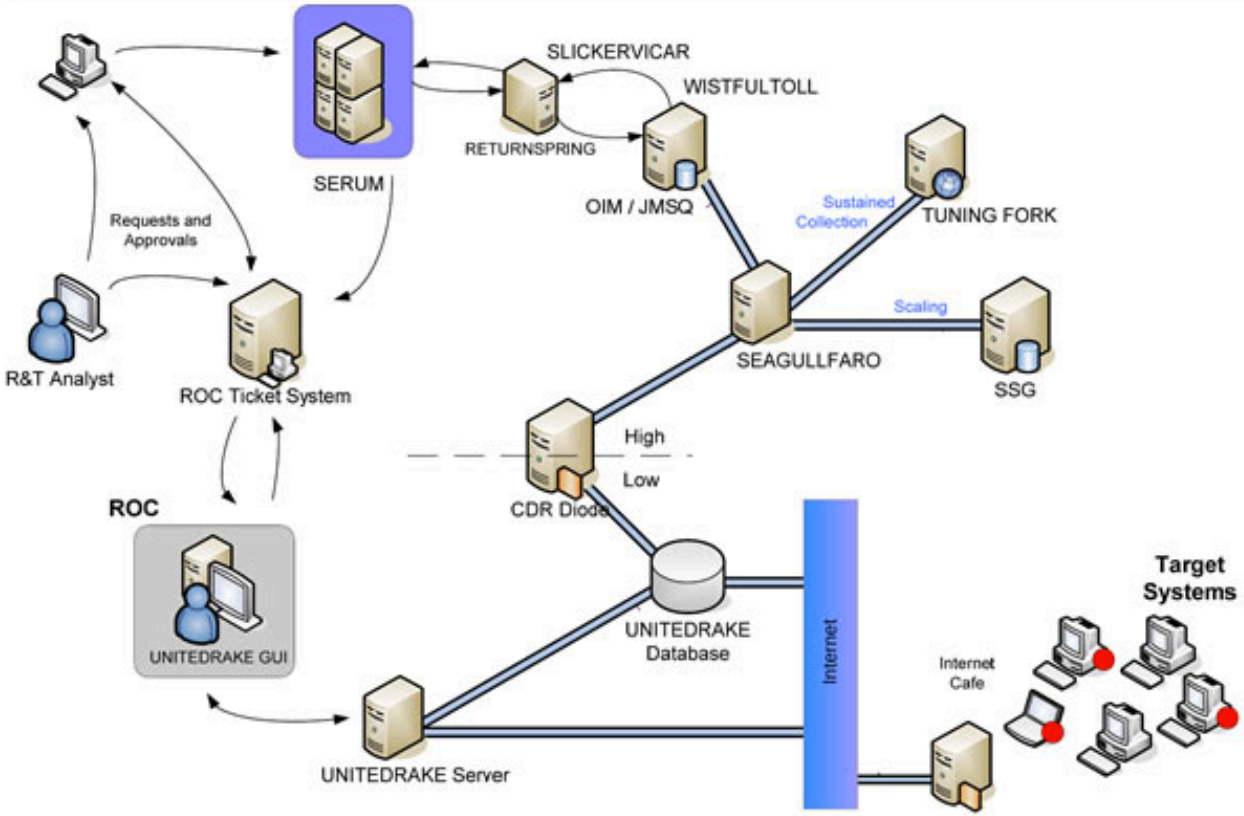
- Infect host code by
  - Injecting code into Master Boot Record
  - Detecting and infecting a boot loader (ntldr, Grub, …)
  - Detecting DLL loads and infecting DLLs
- Alleviates software encryption, low throughput
- Less stealthy

IRATEMONK

ANT Product Data

(TS//SI//REL) IRATEMONK provides software application persistence on desktop and laptop computers by implanting the hard drive firmware to gain execution through Master Boot Record (MBR) substitution.

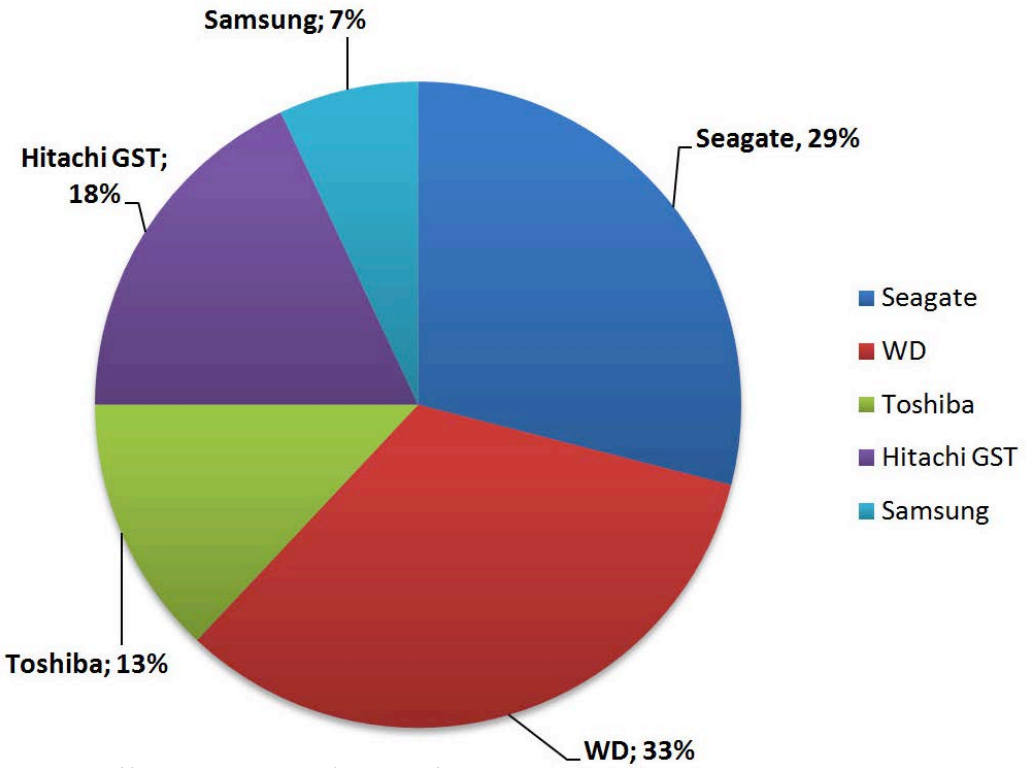
06/20/08



(TS//SI//REL) IRATEMONK Extended Concept of Operations

Impact

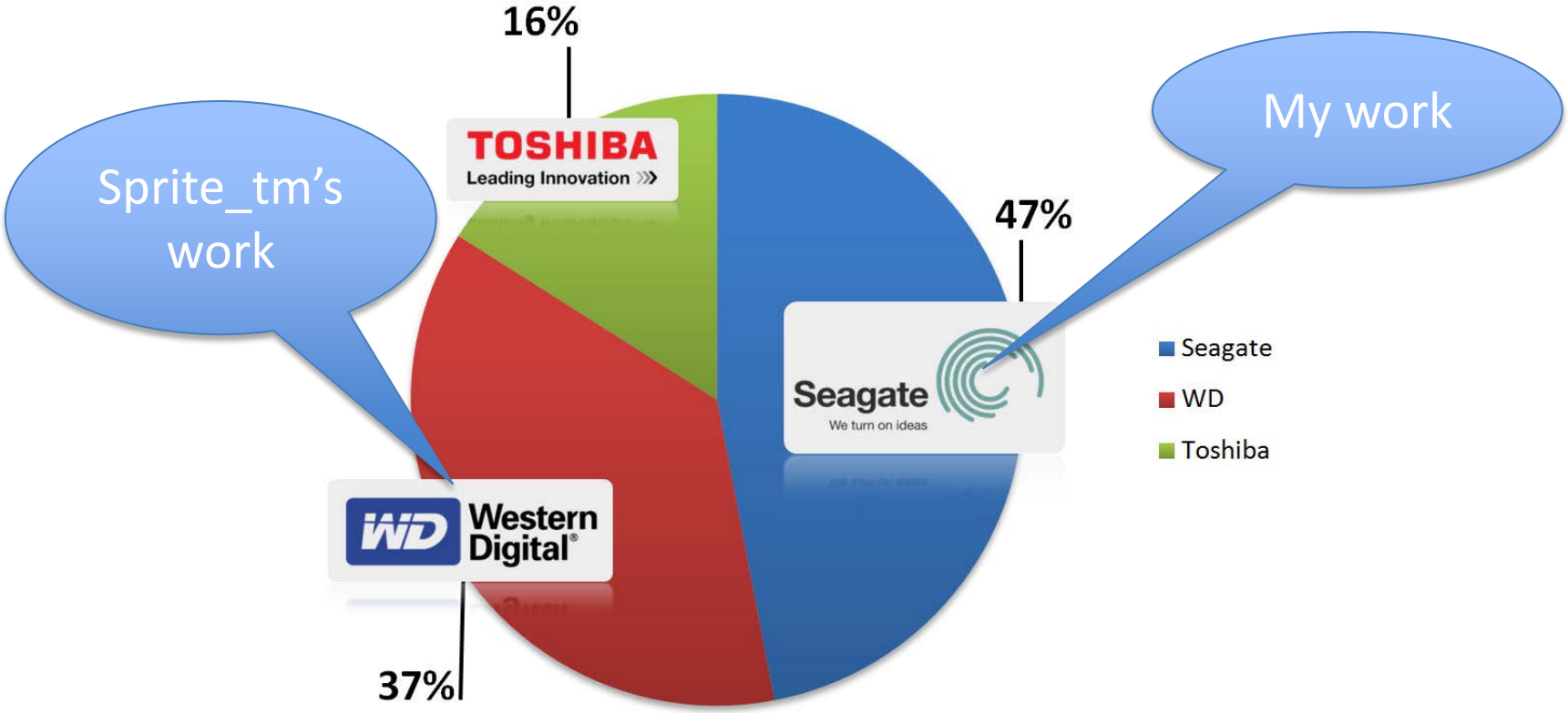
HDD vendors market share Q3 2011



http://news.techeye.net/business/hdd-business-to-become-mexican-standoff

Impact

HDD vendors market share 2012 (projected)



http://news.techeye.net/business/hdd-business-to-become-mexican-standoff

Specific countermeasures

- Backdoor detection
  - Host level: Sporadically read blocks from HDD after write and verify integrity
  - Network level: Detect backdoor commands in network packets
- Data hiding
  - Software HDD encryption
- System integrity
  - Verify that operating system has not been tampered with

General countermeasures

- Firmware integrity
  - Sign firmware
  - Start from a root of trust (e.g., ROM bootloader)

  →Does not help against code injection/ROP

  →Difficult to realize with plugin model
- Remote attestation
  - Prove that firmware has not been modified

General countermeasures

- Better firmware analysis tools
  - Static (binary) analysis
  - Dynamic analysis
  - Emulation
- Establish minimum security standards
  - E.g., scanner for “worst practices”

Outline

- Introduction
- Firmware reverse engineering
- Backdoor injection
- Remote access
- Discussion
- Conclusion

Conclusion

- Presented a firmware backdoor attack
  - Which is able to exfiltrate data
  - No modifications to PC code necessary
- Attack is almost impossible to detect
  - Backdoor command needs to be observed or known
- Make sure no one tampers your HDD!
  - Supply chain
  - Root access to PC

Questions?

Reversing the firmware file format

- Reverse the update function
- Find flash dump and memory dumps in firmware update file
- File is organized in sections
  - First stage bootloader
  - Flash image
  - Main firmware
  - Overlays
  - Servo controller 8051 code :)

Reversing the firmware file format

- Each section can again contain chunks
  - Flash data chunk
  - Memory chunk
- I will clean the script on the flight back and post it on my website