March 3rd 2014 – OSSIR/JSSI 2014

Paper first presented at ACSAC 2013
Awarded Best Student Paper Award

Implementation and Implications of a Stealth Hard-Drive Backdoor

Jonas Zaddach
Davide Balzarotti
Aurélien Francillon
Moitrayee Gupta

Anil Kurmus
Erik-Oliver Blass
Travis Goodspeed
Ioannis Koltsidas

EURECOM
Sophia Antipolis

IBM Research

Aurélien Francillon

- **Maitre de conférences a EURECOM**
  Depuis Sept. 2011

- **Eurecom**
  Petite Grande école d'ingénieurs a Sophia Antipolis

  Membre de l'institut mines-télécom
- **2/3 d'étudiants étrangers**
- **Nouveau diplôme cette année**
  Diplôme d'ingénieur de spécialisation (CTI)

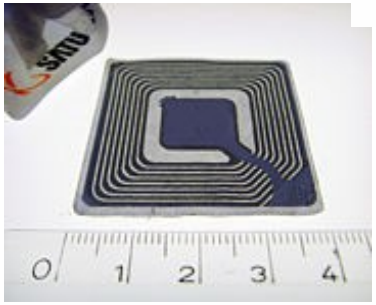
Embedded systems

SmartCards

Connected devices

Sensors

RFID



Industrial systems

Your computer is made of many “computers”

Keyboard controller

Network card

CPU microcode

Hard-drive

Northbridge chipset

GPU

All of them are considered to be in our TCB.
What could happen when one is compromised?

Goals

- It's about threat models!
  - Do we care about hardware compromises?
  - Is it practical, feasible?
  -
- Steps:
  - Reverse engineered the firmware of a disk
    - a Common Off The Shelf (COTS) SATA disk
  - Designed a backdoor
    - covertly extracts data from a host/network (i.e., exfiltrates data)
  - Evaluated its impact and performance
  - Discussed countermeasures

Simple example: HDD rootkit for persistence

- Malware compromises OS
- Updates HDD firmware with malicious firmware
- Disk is formatted, OS “re-installed”
- But malicious HDD firmware remains!
- Malware compromises OS again

Problems

- How to craft a malicious firmware update?
  - Reverse engineering existing firmware
- Can we create a more stealthy payload?
  - Not making modifications to the host

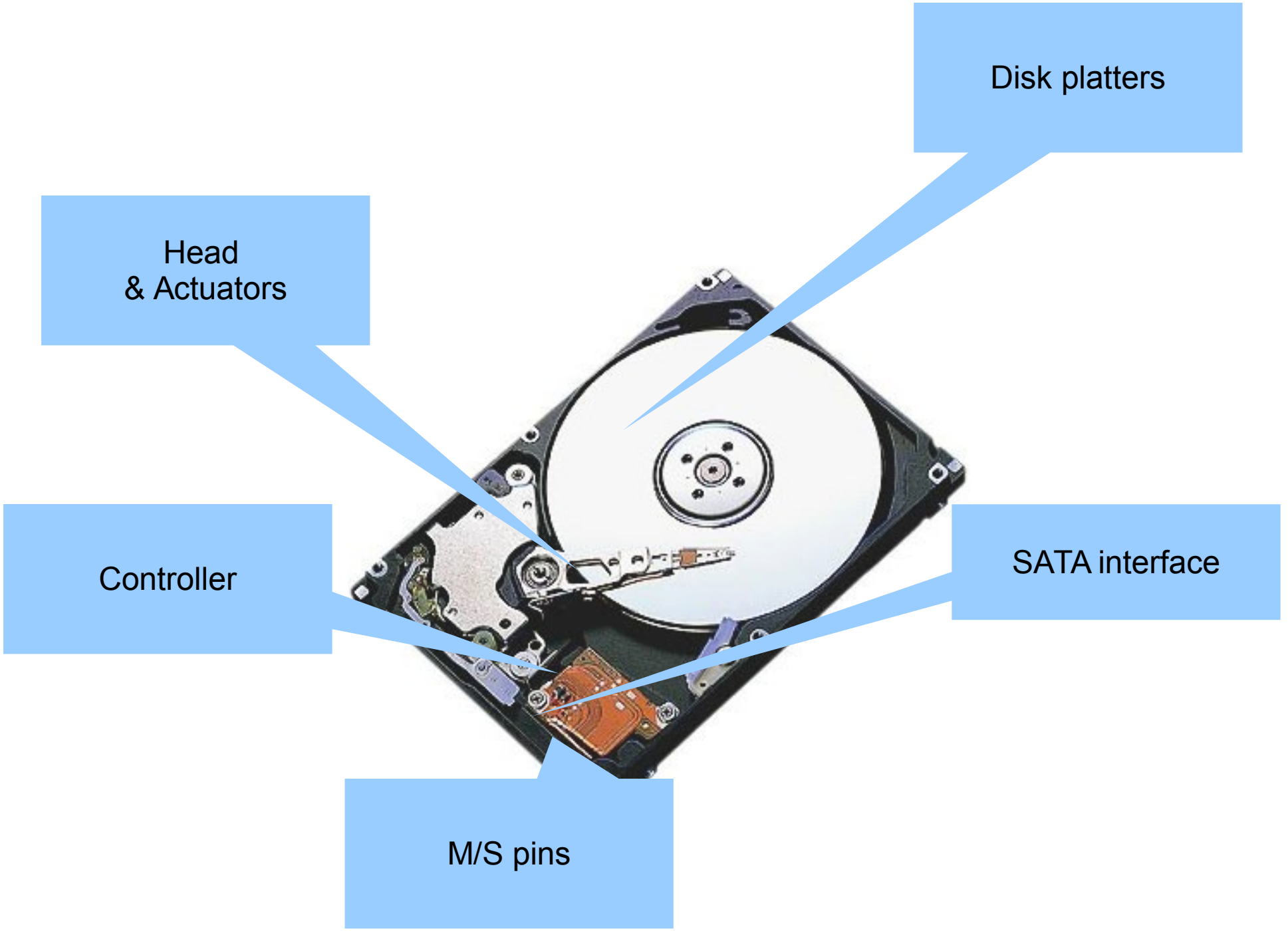
Do it for real:

- otherwise we would not learn much!

First part:
Reverse engineering and modifying an HDD

Backdooring the firmware

- Can be done easily by the manufacturer
  - What about any moderately funded attackers?
  - Without physical access?
- Requires:
  - Reverse engineering
  - Developing a payload
  - Packaging a firmware update


Disk platters
Head
& Actuators
Controller
SATA interface
M/S pins

Diagnostic menu

- Available over serial port (M/S pins)
- 100s of diagnostic commands
- Commands available to dump RAM:

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]

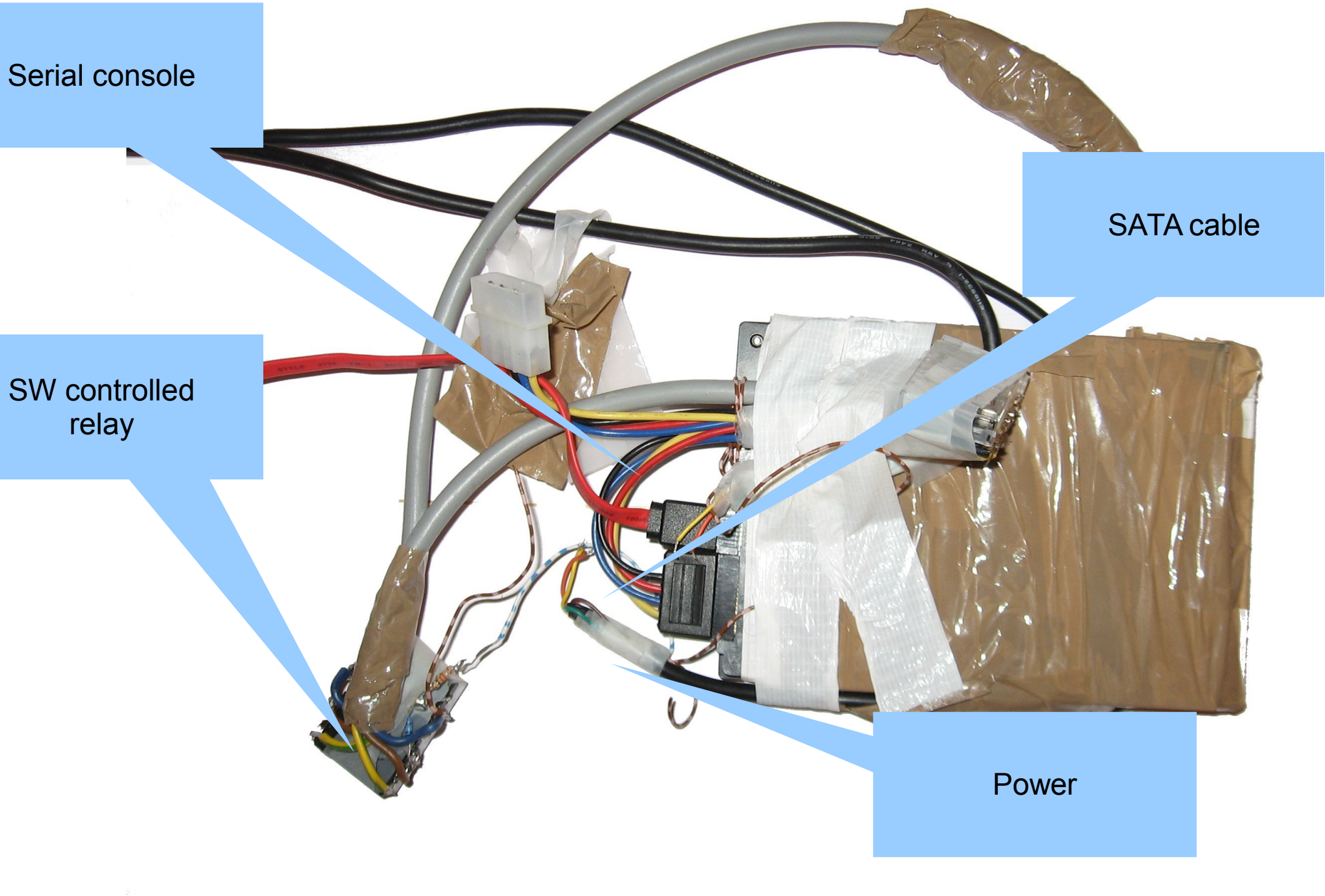
All Levels '=': Rev 0011.0002, Flash, **Poke Memory Byte**, =[AddrHi],[AddrLo],[Data],[Opts]
Online ^C: Rev 0011.0000, Flash, **Firmware Reset**

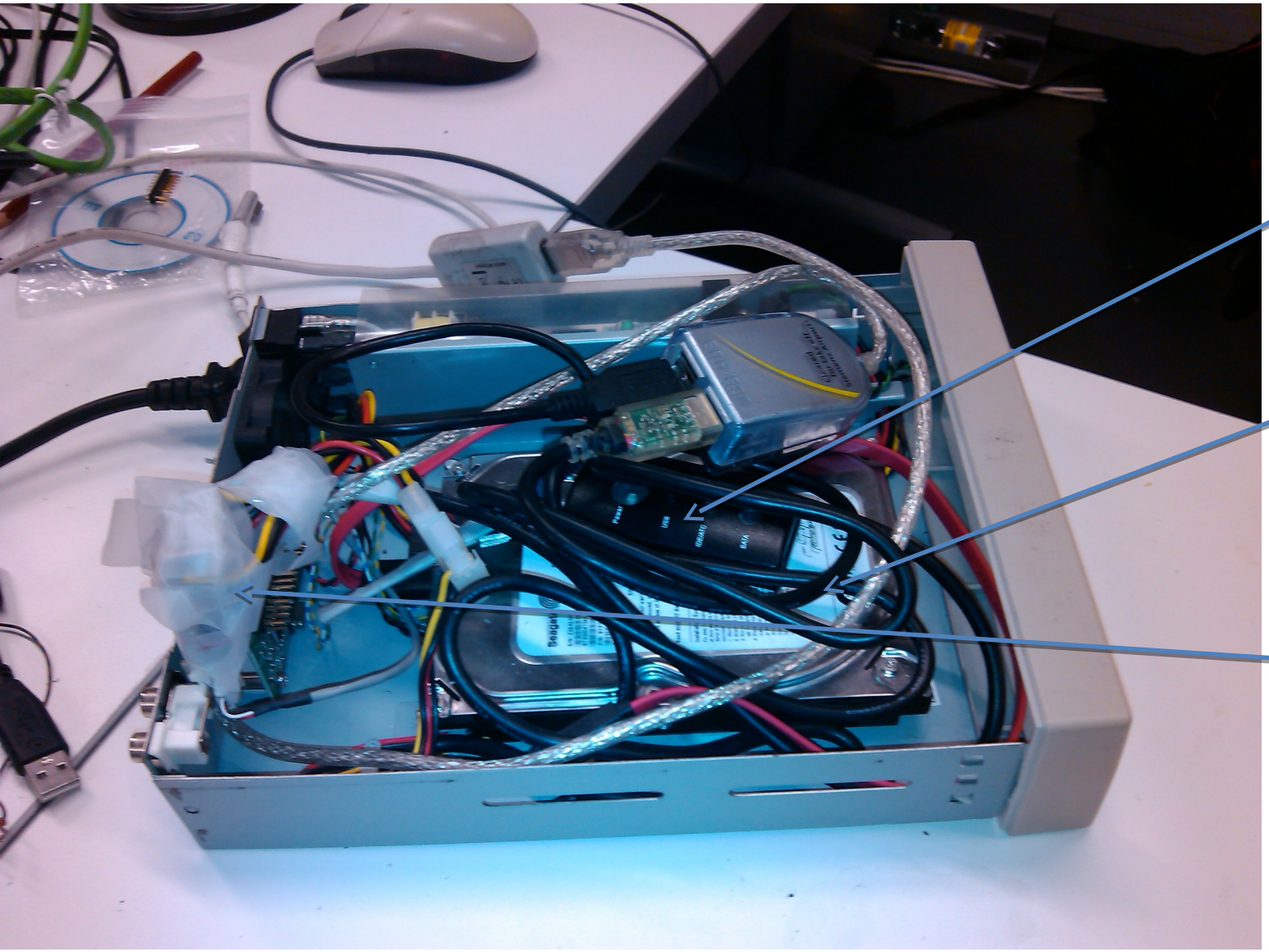
Reverse engineering approach

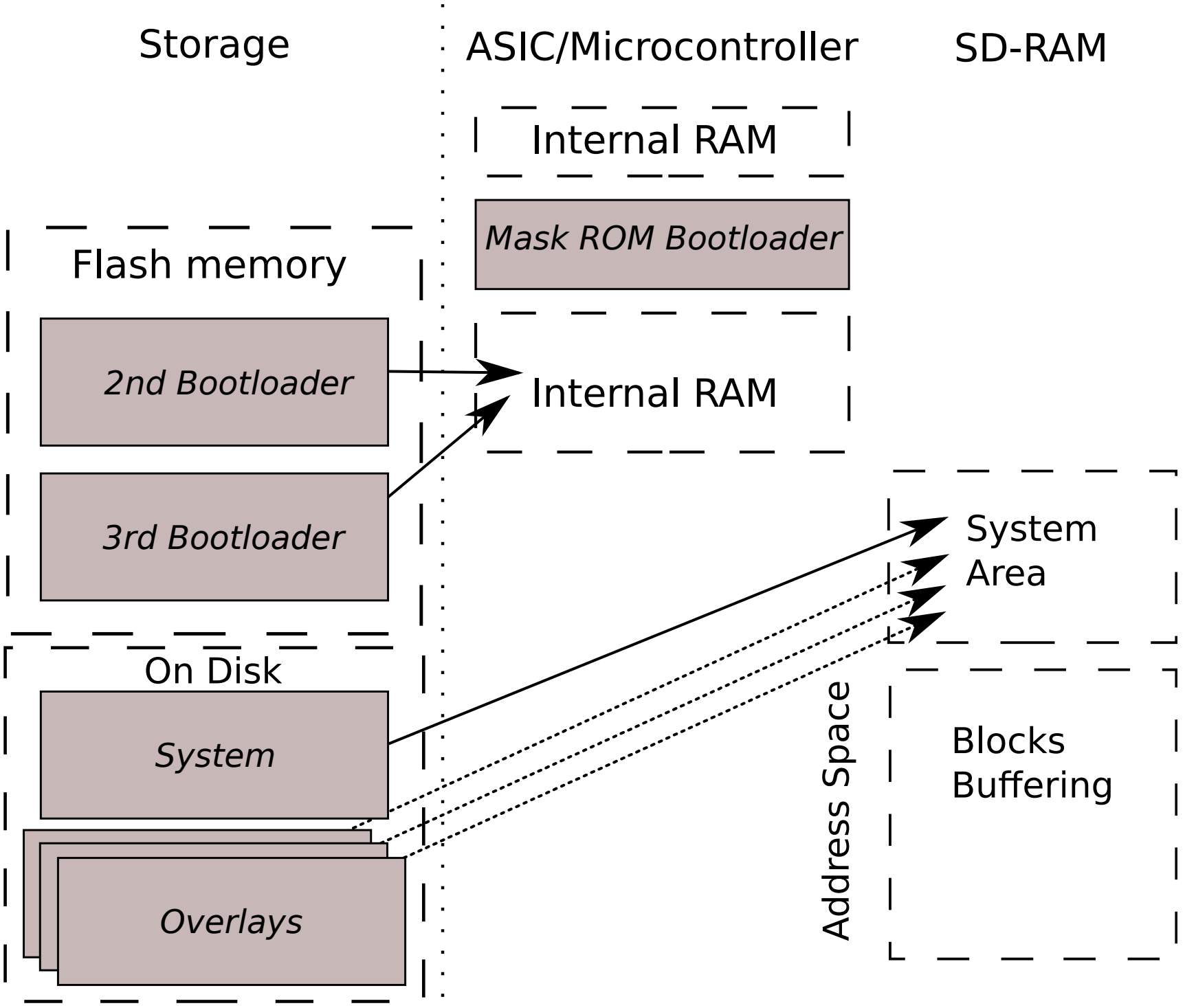
- Use serial diagnostic menu
  - PEEK/POKE primitives
  - Memory dump
- Understand firmware
  - But it's very large and obscure...
  - We need a way to debug the running firmware
  - To hook the backdoor in the original code

Debugging the firmware

- No HW debug support
  - No JTAG
  - No HW breakpoints, watchpoints, …
- Write our own debugger stub
  - Software breakpoints (bkpt instr + data abort interrupts)
  - Communication with GDB over serial port
- Issues to keep the debugger in control
  - Several boot stages load new code
  - Debugger stub or breakpoints often overwritten
  - Many crashes and reboots …


Serial console
SATA cable
SW controlled relay
Power


USB-SATA
bridge
HDD
USB
Controlled
Switch


Storage
ASIC/Microcontroller
SD-RAM
Internal RAM
Mask ROM Bootloader
Flash memory
2nd Bootloader
Internal RAM
3rd Bootloader
System
Area
On Disk
System
Address Space
Blocks
Buffering
Overlays

Finding a hooking point for the backdoor

- Many technical difficulties...
  - Custom, event based OS
  - Large statically linked code, No symbols

- Data is transferred directly in the HDD's RAM via DMA
- Pointers are passed between different threads in the firmware, tracking them is difficult because the debugger does not allow data watchpoints

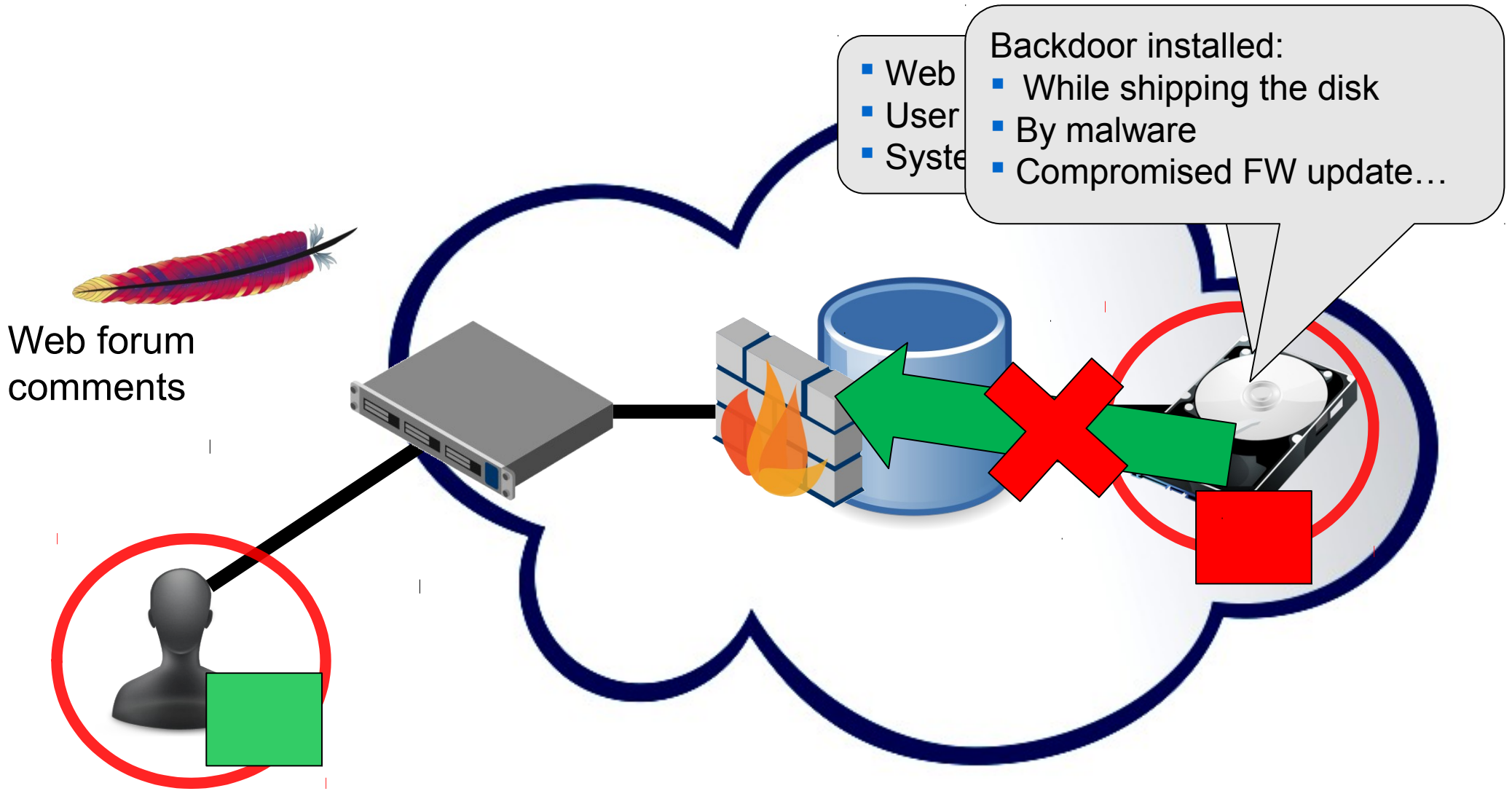
Backdoor Implementation

- Backdoor inserted in a firmware update
- Intercepts disk writes
- Can read blocks from disk (unstable*)
- No significant overhead (1%)

* Quality control is left as an exercise for 3 letters agencies

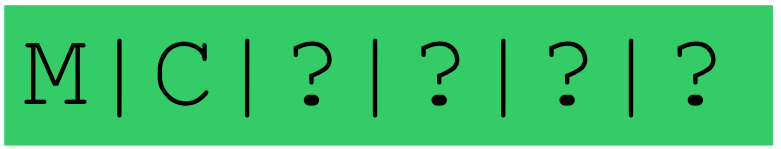
Second part:
A remote data exfiltration payload

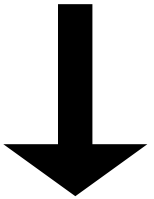
Exfiltration exemple: an online forum

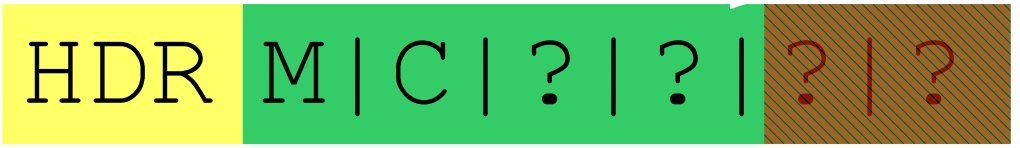


Initial idea

One block

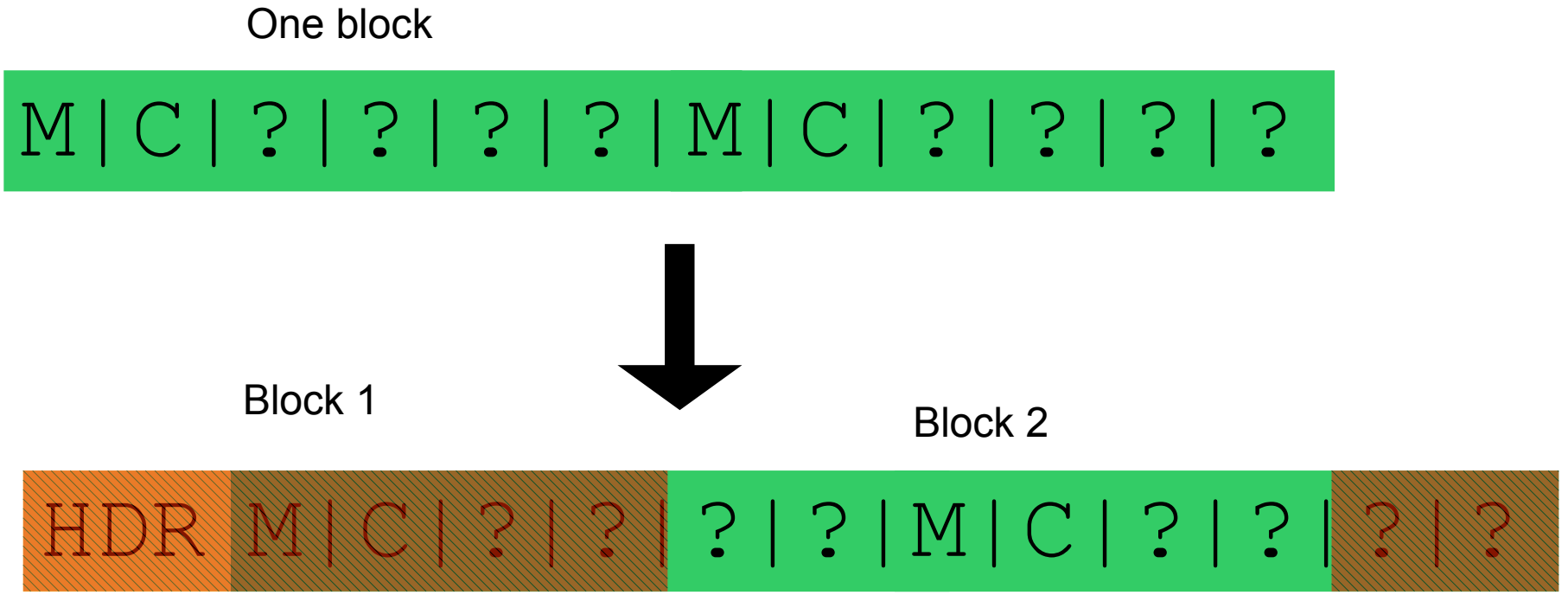






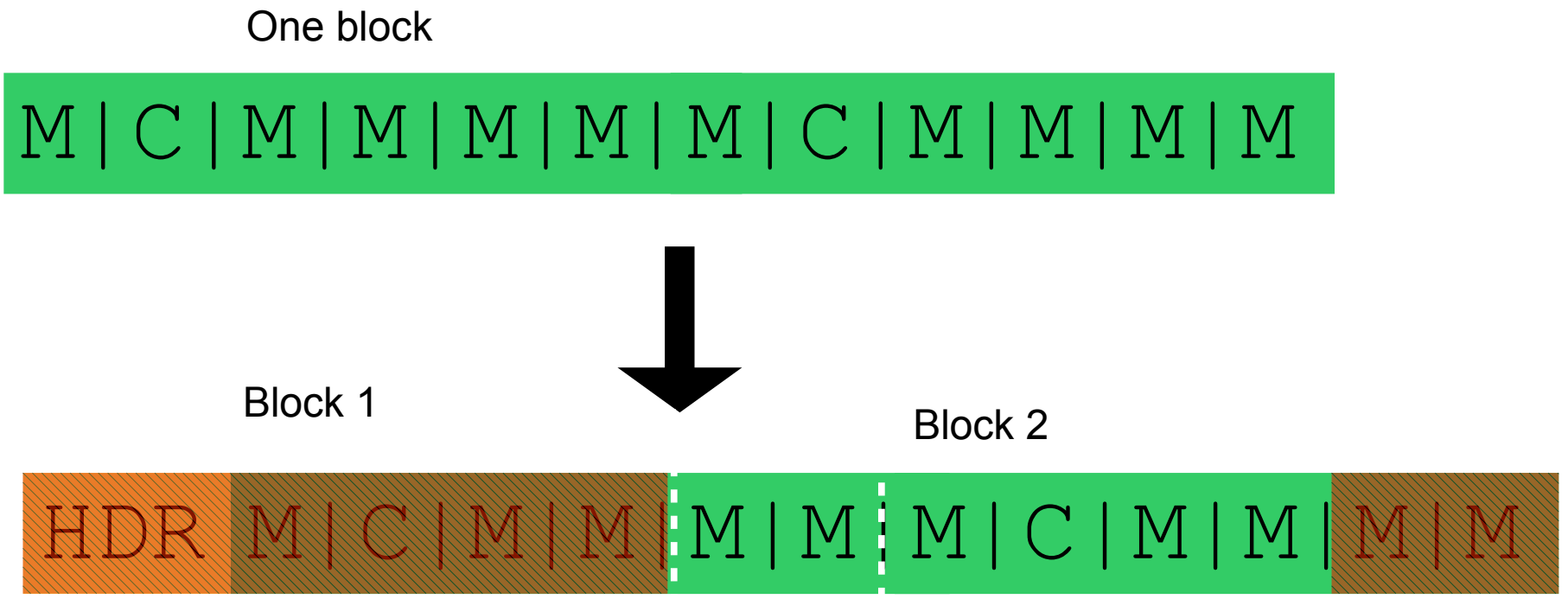
Alignment issue

Improvement

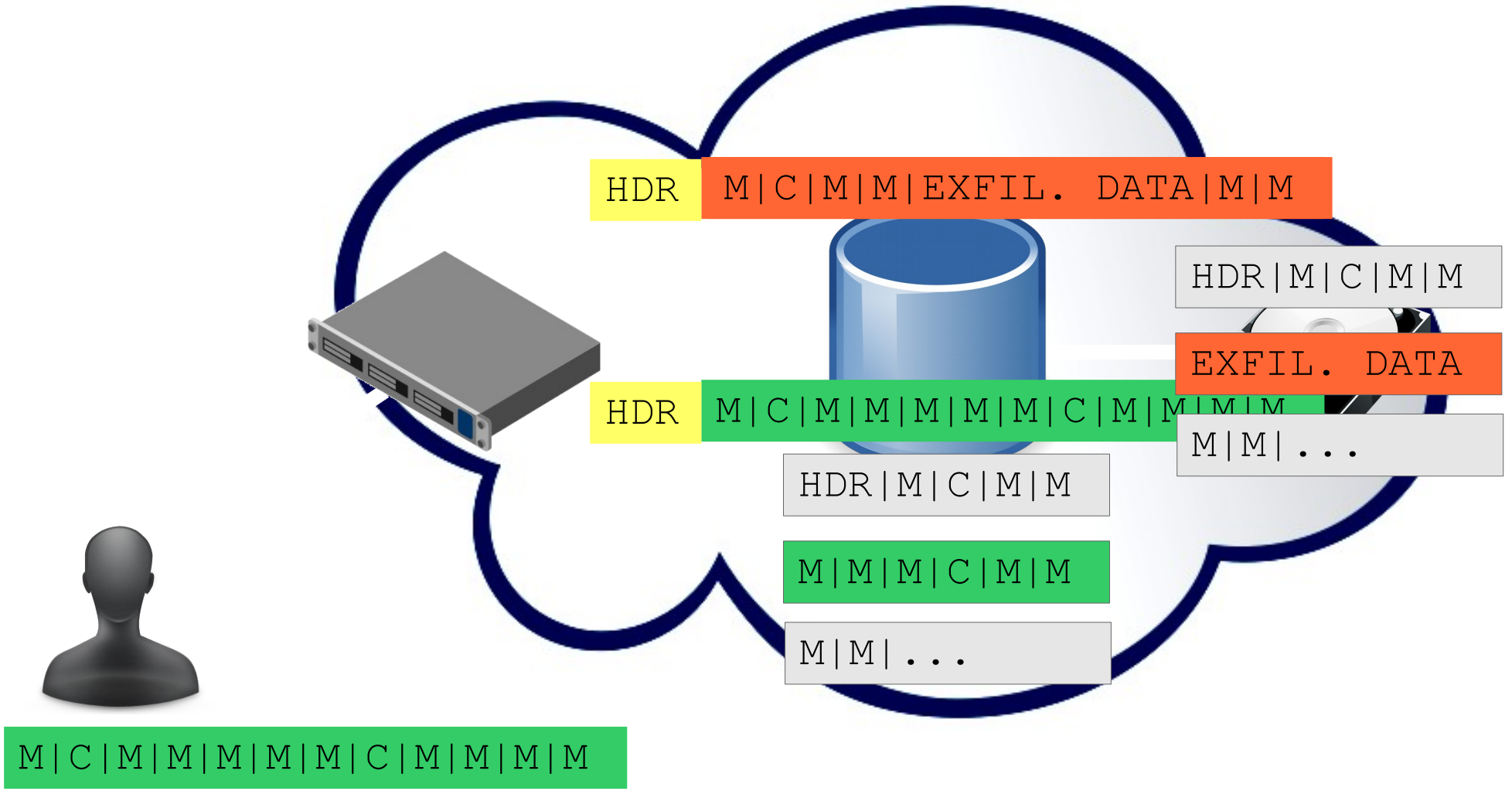


One full safely replaceable block

Exfiltration block format



Fast check + no false-positives


HDR
M|C|M|M|EXFIL. DATA|M|M
HDR|M|C|M|M
EXFIL. DATA
HDR
M|C|M|M|M|M|M|C|M|M|M|M
M|M|...
HDR|M|C|M|M
M|M|M|C|M|M
M|M|...
M|C|M|M|M|M|M|C|M|M|M|M

Other exfiltration challenges

- Format of exfiltrated data

  → Base64 encode sectors

- Caching

  → Wait, or create dummy traffic

Evaluation

Qemu
implementation

PHP-based forum

Exfiltrating a sensitive file

- Use HDD as remote block device
- Exfiltrate /etc/shadow in nine “queries”:
  - First retrieve partition table in MBR
  - Then superblock of ext3 partition
  - …
- Total time: < 1 minute

Countermeasures

- Encryption of data-at-rest
  - Works under some (uncommon) conditions
- Signed firmware updates
  - Helps, however:
    - Physical attacks
    - Manufacturer compromise
    - Vulnerabilities in code allowing run-time modifications
- Firmware integrity verification
  - e.g., use ROM code as root of trust
  - Secure boot
- Page-cache driven integrity checks

Conclusion

- RE-ed and compromised a COTS drive
  - 10 person-month effort
  - No significant performance overhead
- Data-exfiltration backdoor
  - No cooperation from host
  - Stealthy

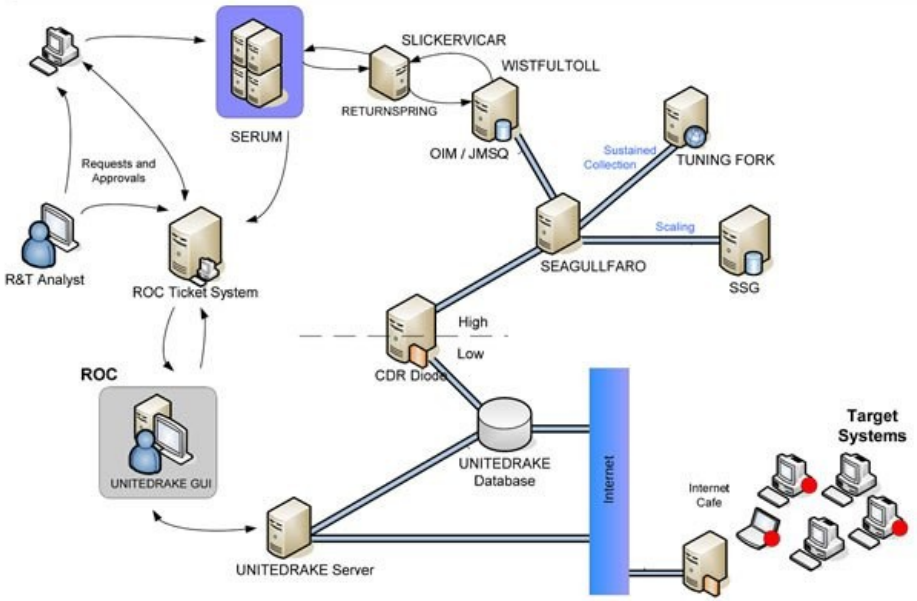
- So is this realistic ?

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

(TS//SI//REL) This technique supports systems without RAID hardware that boot from a variety of Western Digital, Seagate, Maxtor, and Samsung hard drives. The supported file systems are: FAT, NTFS, EXT3 and UFS.

(TS//SI//REL) Through remote access or interdiction, UNITEDRAKE, or STRAITBAZZARE are used in conjunction with SLICKERVICAR to upload the hard drive firmware onto the target machine to implant IRATEMONK and its payload (the implant installer). Once implanted, IRATEMONK's frequency of execution (dropping the payload) is configurable and will occur when the target machine powers on.

Status: Released / Deployed. Ready for Immediate Delivery

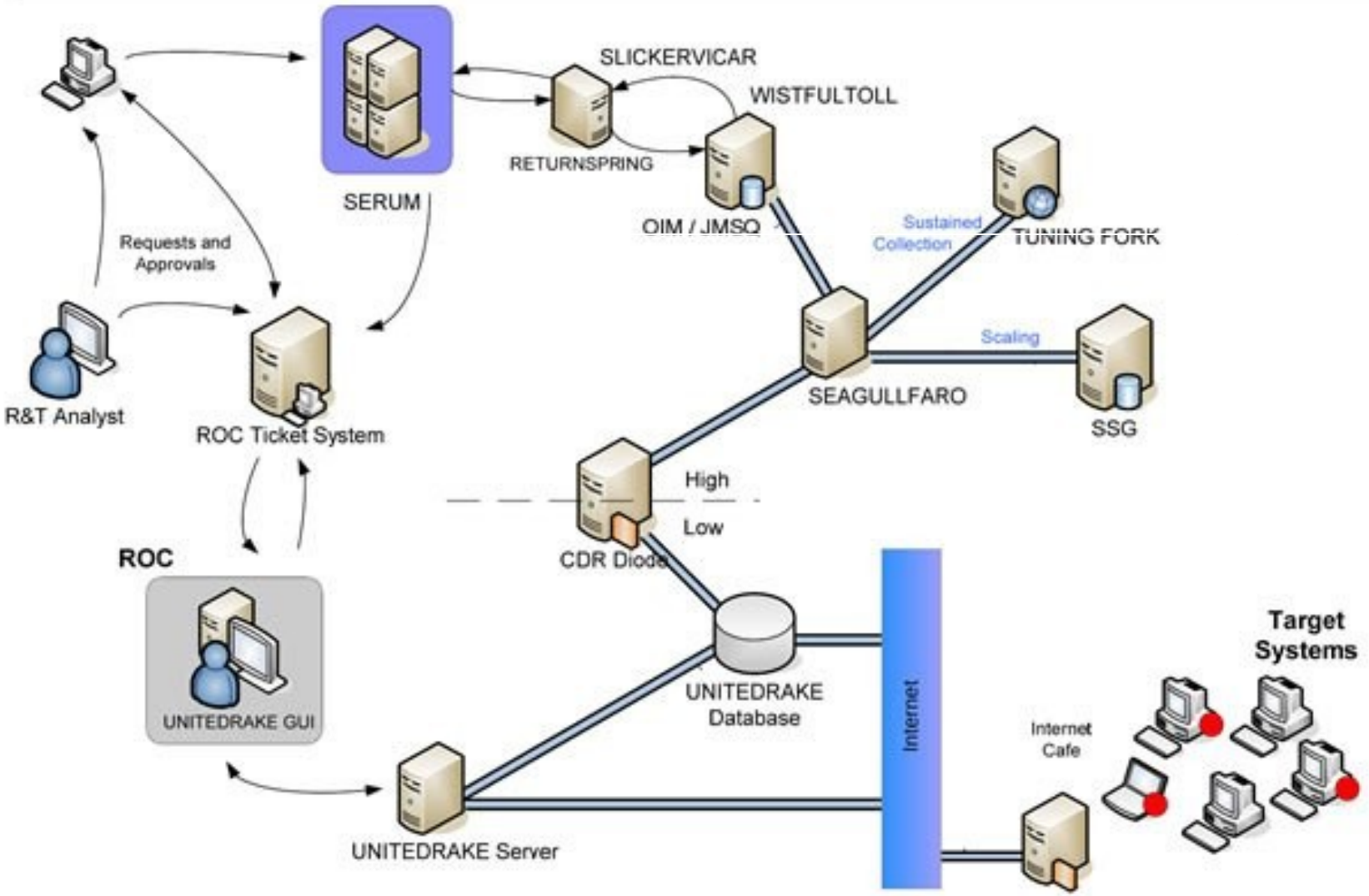
Unit Cost: $0

POC: [redacted], S32221, [redacted], [redacted]@nsa.ic.gov

Derived From: NSA/CSSM 1-52
Dated: 20070108
Declassify On: 20320108

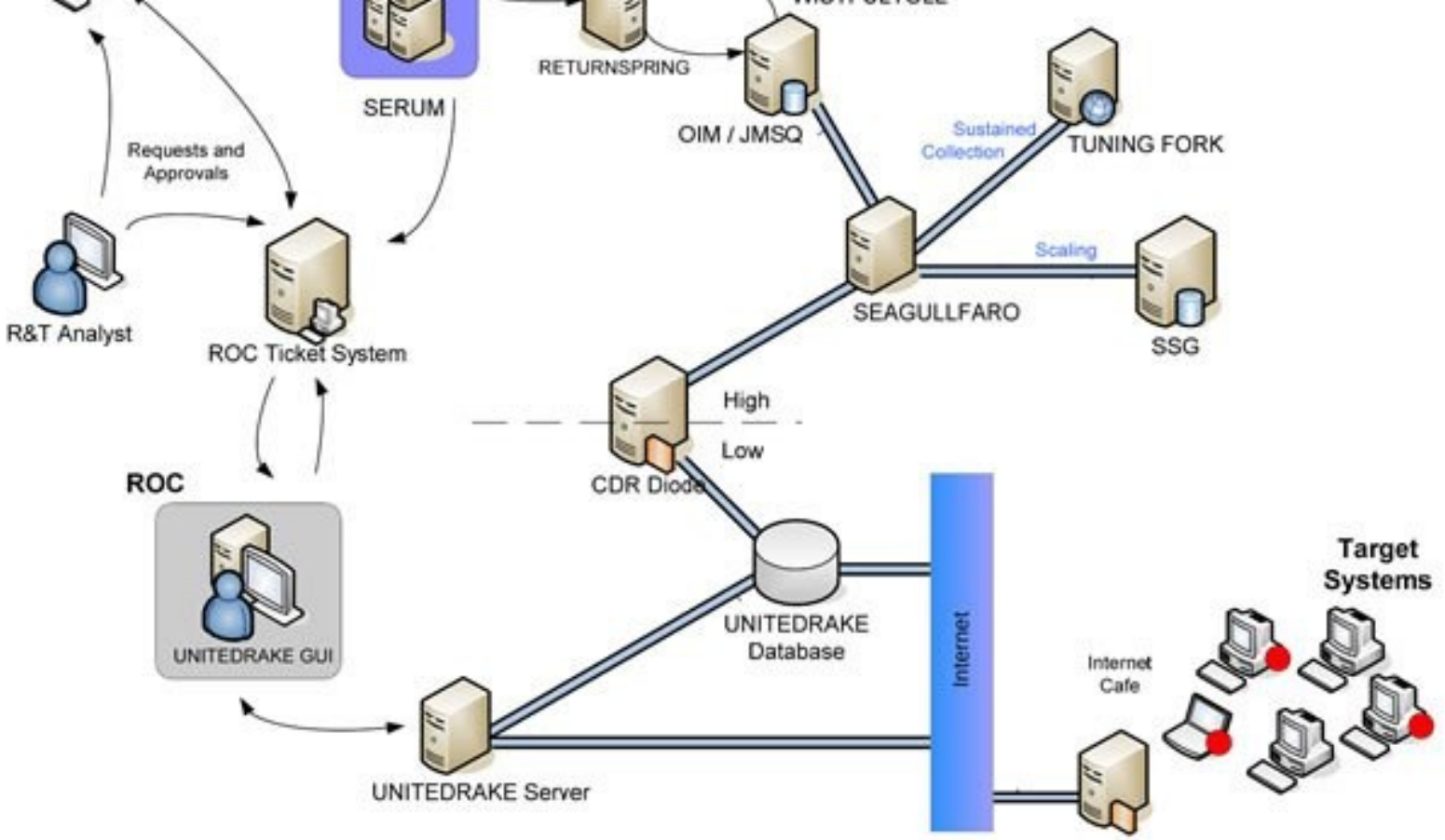
(TS//SI//REL) IRATEMONK provides software application persistence on desktop and laptop computers by implanting the hard drive firmware to gain execution through Master Boot Record (MBR) substitution.

06/20/08



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

(TS//SI//REL) This technique supports systems without RAID hardware that boot from a variety of Western Digital, Seagate, Maxtor, and Samsung hard drives. The



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

(TS//SI//REL) This technique supports systems without RAID hardware that boot from a variety of Western Digital, Seagate, Maxtor, and Samsung hard drives. The supported file systems are: FAT, NTFS, EXT3 and UFS.

(TS//SI//REL) Through remote access or interdiction, UNITEDRAKE, or STRAITBAZZARE are used in conjunction with SLICKERVICAR to upload the hard drive firmware onto the target machine to implant IRATEMONK and its payload (the implant installer). Once implanted, IRATEMONK's frequency of execution (dropping the payload) is configurable and will occur when the target machine powers on.

Status: Released / Deployed. Ready for Immediate Delivery

Unit Cost: $0

Lesson learnt

- We need to trust all those embedded devices…
  - But we can't!

- Performing security analysis of embedded systems is very challenging !
  - Very hard to analyze the disk
  - Static v/s Dynamic analysis

- We need to develop new methodologies and tools for dynamic security analysis of embedded systems

Future Work

- This work is also an excuse to
  - Identify challenges in performing security analysis of embedded systems
  - Develop new methodologies and tools for dynamic security analysis of embedded systems

Future Work

- A first result is already available:
  - *AVATAR: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares*

    Jonas Zaddach, Luca Bruno, Aurélien Francillon, Davide Balzarotti (NDSS'14)

    http://www.s3.eurecom.fr/tools/avatar/

Questions?

Kind thanks are due to those hard disks
who valiantly gave their lives
toward scientific investigation and research.