

---

# An Amazing Journey into the depth of my Hard Drive

Jonas Zaddach

---

# About myself

---

- Master Thesis on security of Amazon EC2 machines
- PhD Candidate on the topic of Embedded Firmwares' Security at [EURECOM](#) [1]
- [My website](#) [2] (Publications, etc)
- Email: [zaddach@eurecom.fr](mailto:zaddach@eurecom.fr)



---

# Acknowledgements

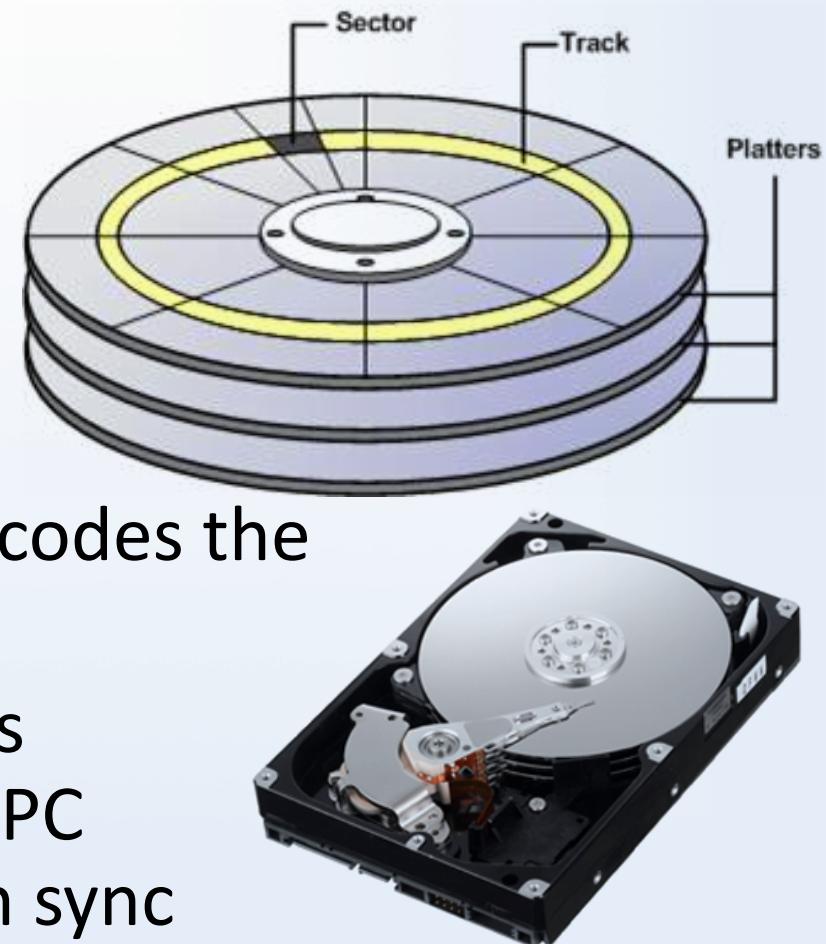
---

- Thanks to my Advisor Davide Balzarotti and “Co-Advisor” Aurélien Francillon for enabling me to do this research!
  - Thanks to Travis Goodspeed for getting me started
  - [Similar hacking](#) has been done by sprite\_tm (Jeroen Domburg) on a different HDD brand [3]
  - [A description](#) of a sophisticated data exfiltration backdoor based on compromised HDDs [5]
-

# What is a hard drive? – Physical view

---

- A bunch of magnetized disks that store binary information
  - The heads move over those disks
  - A DSP or custom chip decodes the analog signal
  - A microprocessor handles communication with the PC and keeps components in sync
- 



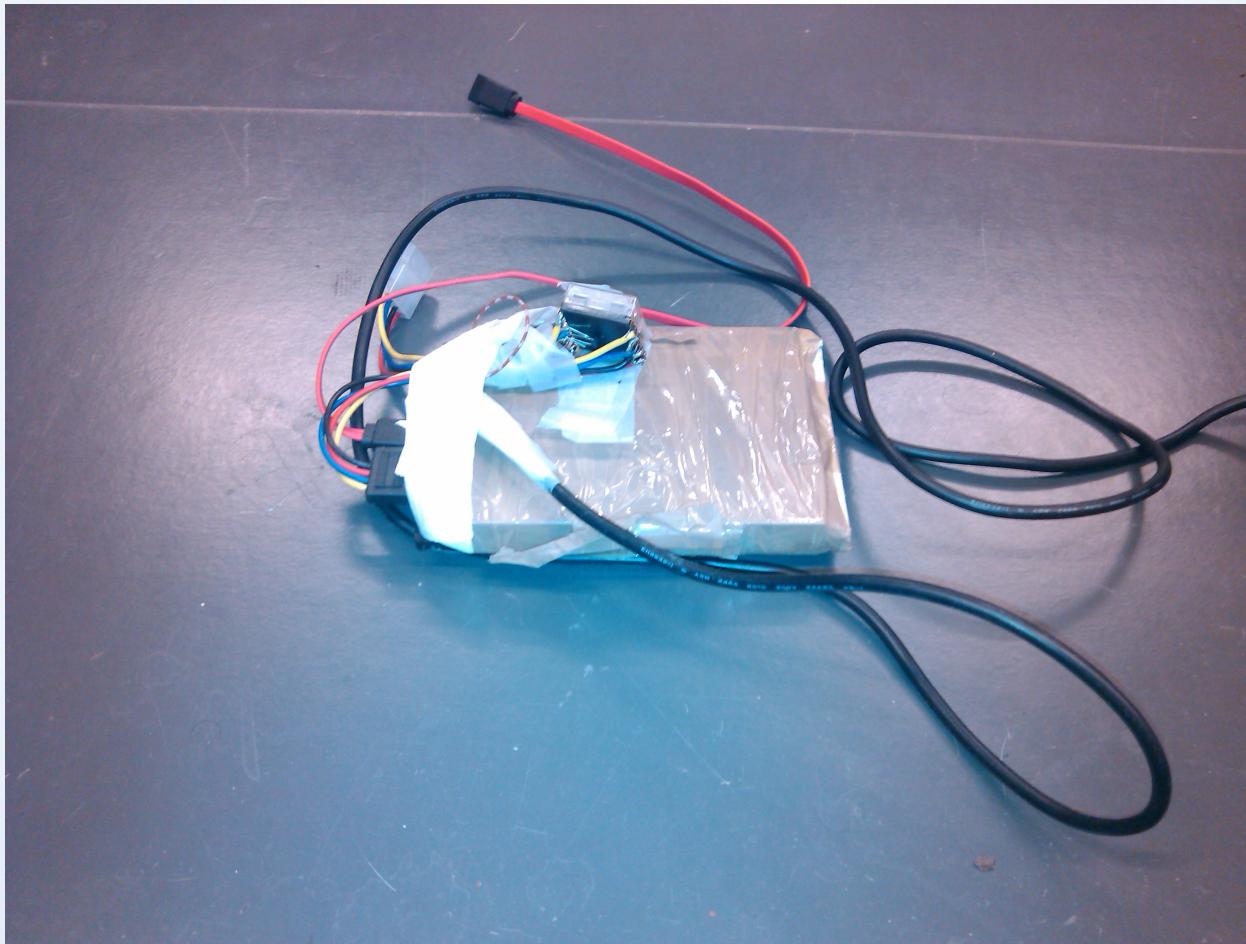
# What is a hard drive? – Logical view

---

- Bytes are grouped into blocks (typically 512 bytes), which are addressed by a block number (LBA)
  - The computer can (among other) read and write blocks
  - Lots of care is taken that written blocks do not change (error correction, etc)
-

# First Prototype

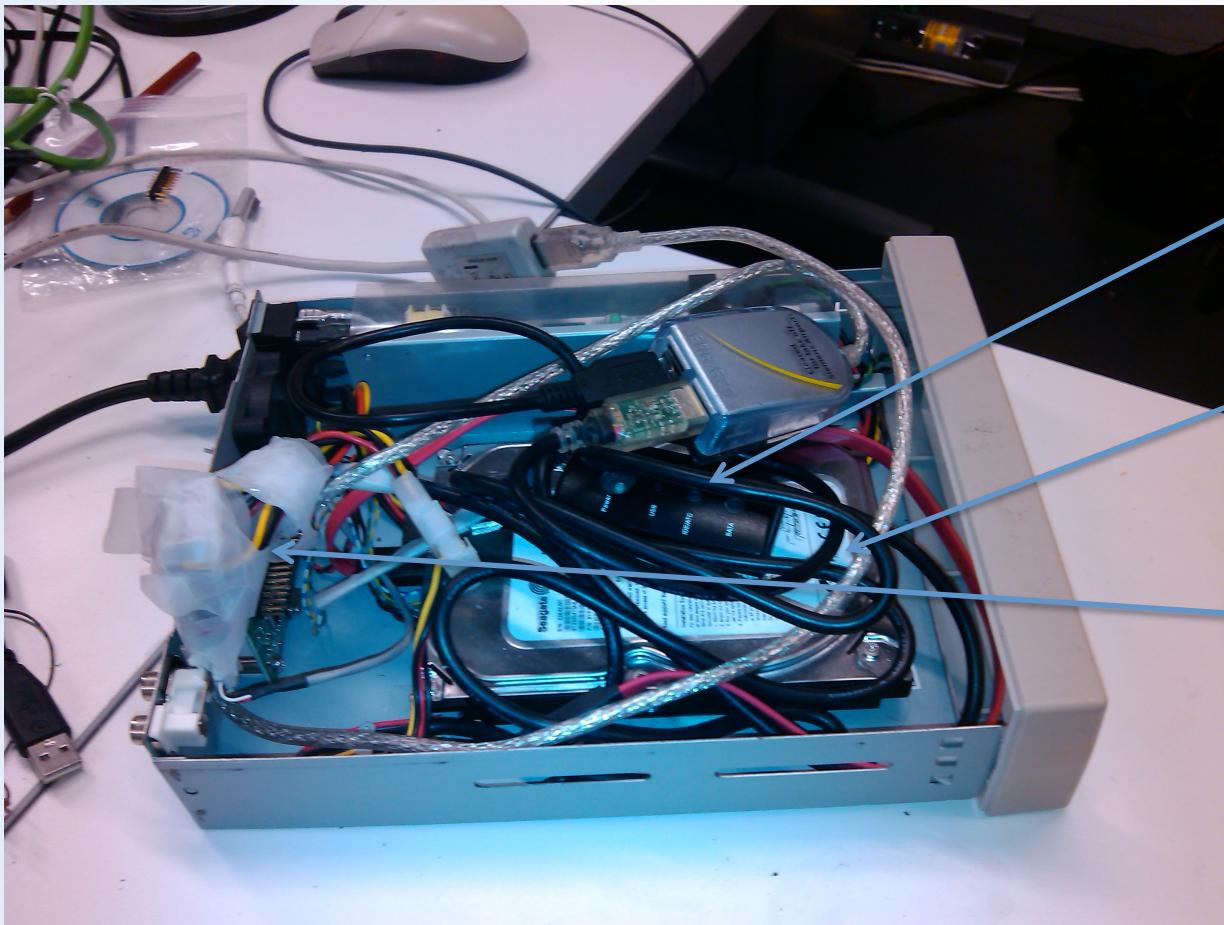
---



---

# Second Prototype

---



USB-SATA  
bridge

HDD

USB  
Controlled  
Switch

# Breaking in

---

- There is a JTAG port, but it seems to be disabled :(
  - Seagate drives have a diagnostic serial port accessible on the Master/Slave jumpers
  - This feature is known and documented in professional circles (e.g., [HDD recovery specialists \[4\]](#))
  - A text menu gives access to diagnostic functions
  - This feature is not specific to Seagate (also found a similar menu in WD and Samsung disks)
  - Type CTRL+Z on the serial console ...
-

# Diagnostic Firmware Menu

---

Online CR: Rev 0011.0000, Flash, Abort

Online ESC: Rev 0011.0000, Flash, Abort Looping Command  
or Batch File

Online ' ': Rev 0001.0000, Flash, Pause Output

Online '.': Rev 0011.0000, Flash, Display Active Status

Online '?': Rev 0011.0000, Flash, Display Diagnostic  
Buffer Information

Online '`': Rev 0012.0001, Flash, Display Read/write  
Statistics

Online '\$': Rev 0012.0002, Flash, Display Read/write  
Statistics By Zone

Online '{': Rev 0011.0000, Flash, Toggle EIB-Specific  
R/W Tracing

---

# Diagnostic Firmware Menu (2)

---

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

---

# Dumping the firmware

---

- Hmm, we got peek and poke, that's cool
  - With a bit of trial and error, the firmware can be extracted (drive will crash if you use invalid address)
  - Neighborly thanks to Travis Goodspeed who dumped the firmware
  - But it gets even more interesting ... when you reboot the drive
-

# Bootloader Prompt

---

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

>

---

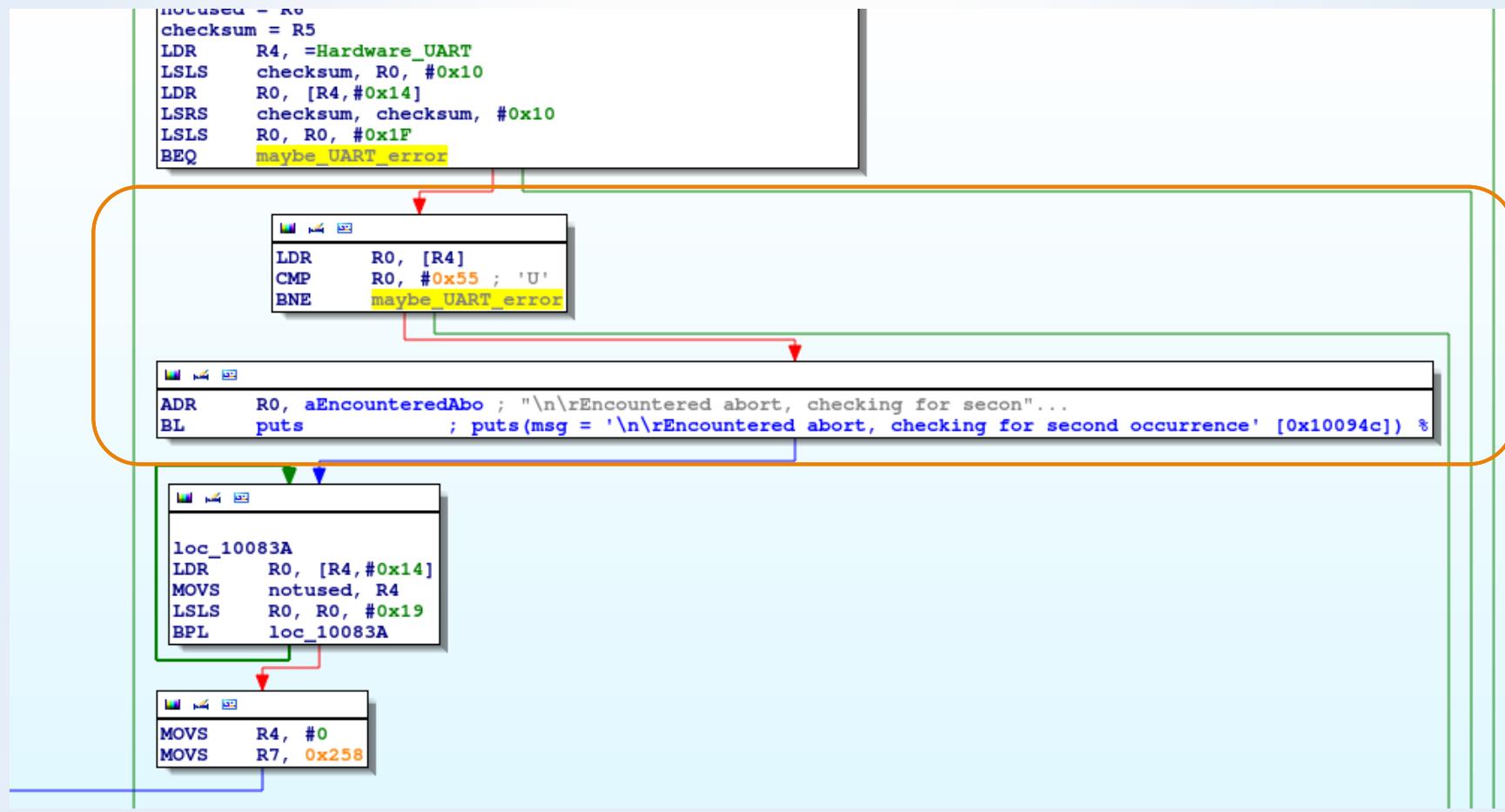
# Inject a debugger

---

- Now we have poke (AP + WT) and execute (AP + GO)!
- This allows us to load and execute code on the drive's ARM processor
- The addresses of the *getc* and *putc* functions are known from the firmware disassembly
- I developed a tiny GDB stub (2.6k) that communicates with my host over UART and allows me to debug code on the drive



# Accelerating the stub loading



# Reconnaissance

---

- Get ARM Coprocessor registers → CPUID, Memory protection settings, cache settings, etc.
  - Drive still crashes when an invalid address is accessed → Reconstruct the memory map
    - Some regions are already known from the memory dump in the diagnostic menu
    - IO region is known from the serial port
-

# Memory Map

---

Memory Range	Type
0x00000000 – 0x00008000	Code SRAM
0x00100000 – 0x00120000	ROM
0x00200000 – 0x00400000	Code DRAM
0x04000000 – 0x04004000	Data SRAM
0x40000000 – 0x50000000	IO
0x60000000 – 0x70000000	Data DRAM

---

# Dumping the Flash

---

- Identify the flash read function in IDA
  - Break execution at beginning of Flash read function
  - Modify the parameters to read the part of the Flash that is interesting
  - Dump the memory where the Flash data was read to with the GDB “dump binary ... ” command
-

# Following the execution

---

- Keeping control is challenging
  - After loading the firmware from Flash to DRAM, this memory range is marked as read-only, breaking SW breakpoints → Overwrite the write-protect instruction with NOP
  - Loading of the OS overwrites exception vector table, removing our debug exception handler  
→ Watch flash loads and block the one writing to address 0x0
-

# Following the execution (2)

---

- OS uses the whole code SRAM where the GDB stub resides → Move GDB stub to free DRAM memory after DRAM has been initialized and before SRAM is overwritten
  - Execution “escapes” the debugger → Put a breakpoint in the UART interrupt handler, so that a CTRL+C will trigger the breakpoint
-

# Dissecting the realtime OS

---

- The OS is custom kernel
    - Fixed number of tasks
    - Preemptive
    - Event-based: Each task has an accepted events mask, tasks can wait for a specific event or yield with generating an event to other tasks
-

# Tasks in the bootloader FW

---

- Interrupt handler: Handles all hardware interrupts
  - Read/write task: Handles accesses to the magnetic platters
  - SATA task: Parses SATA requests and sends responses
  - Diagnostic task?
  - Load main firmware task?
-

## But wait ...

---

- Why did you say “Bootloader FW”?
  - Well, actually the firmware from Flash serves only to load the actual firmware from the disk
  - A very small unpacker stub then distributes this firmware in memory and runs it
  - The main firmware is based on the same OS as the bootloader firmware
-

# Tasks in the main FW

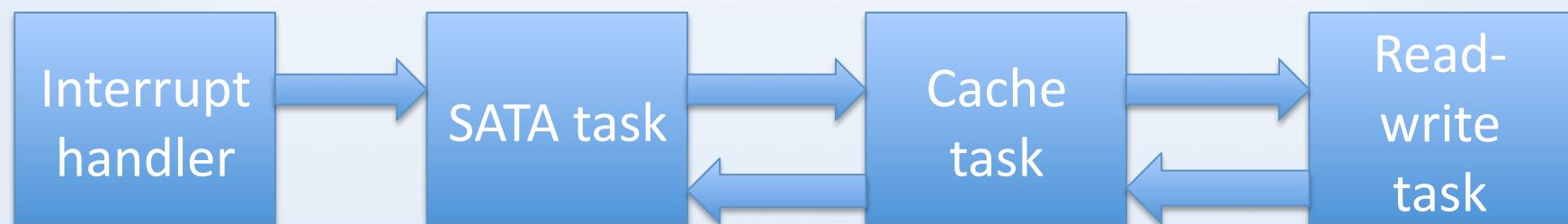
---

- Interrupt handler
  - Read/write task
  - Diagnostic task
  - SATA task
  - Cache manager task
  - ??? task
  - Power management task
-

# Data flow for a SATA request

---

- Initial SATA packet written to memory by HW
- All data is kept in global variables, dataflow is hard to trace ...



# Tapping into the data flow

---

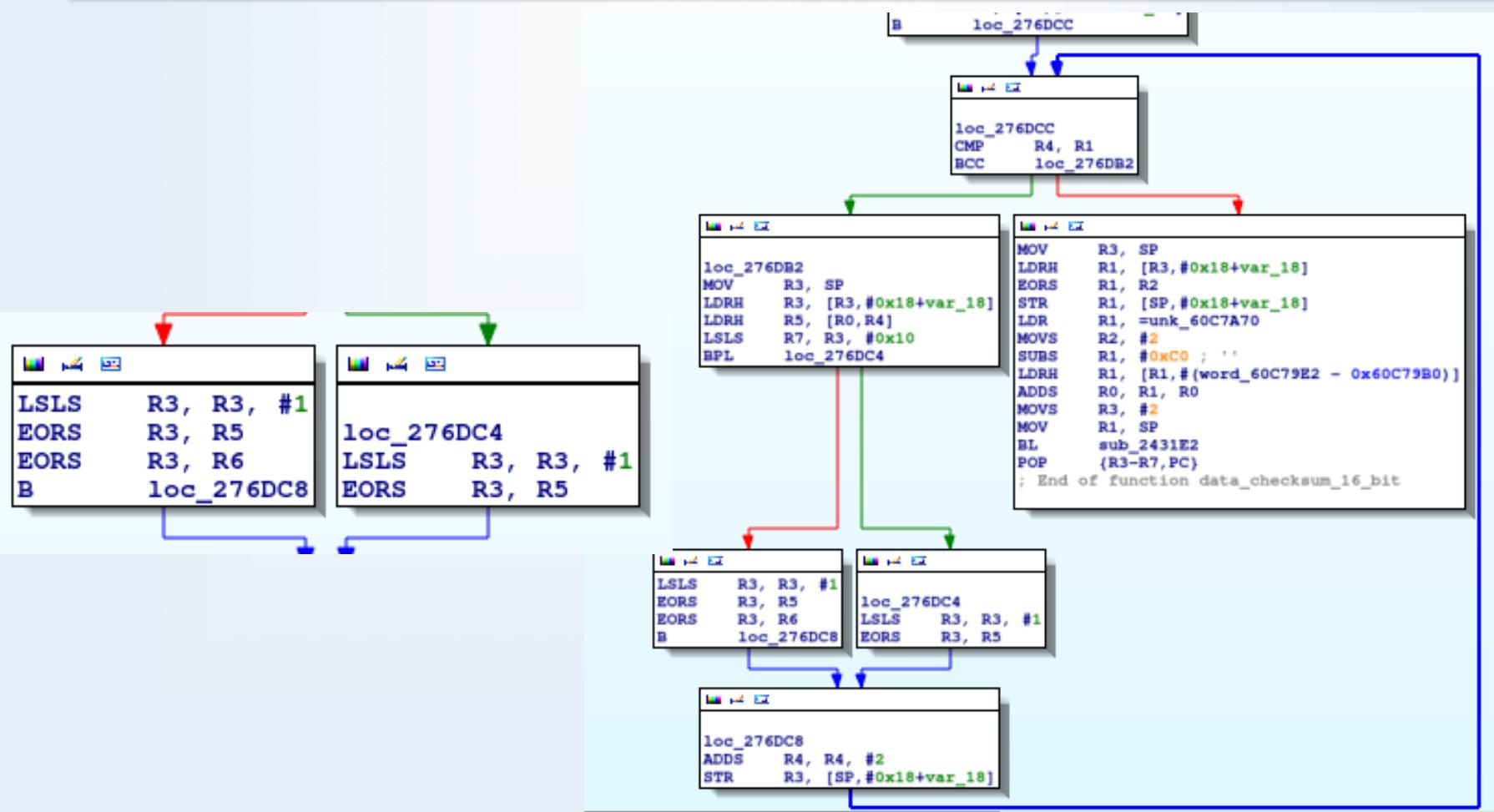
- Basically you can change data anywhere on its way to the R/W task
  - I chose to intercept the flow in the cache task
    - This is where I first found a data structure pointing to the packet
    - Modifying the packet data would give us a full-blown backdoor ...
-

# Checksums

---

- Unfortunately, the drive raises an error and crashes when data is modified
  - Observing the data in memory closely shows that each 512-byte packet is followed by 8 additional bytes
    - One 16-bit checksum
    - One 32-bit checksum
  - After trying to figure the algorithm out for 2 days, I found it in the code ...
-

# Checksums (2)



# Roll your own backdoor

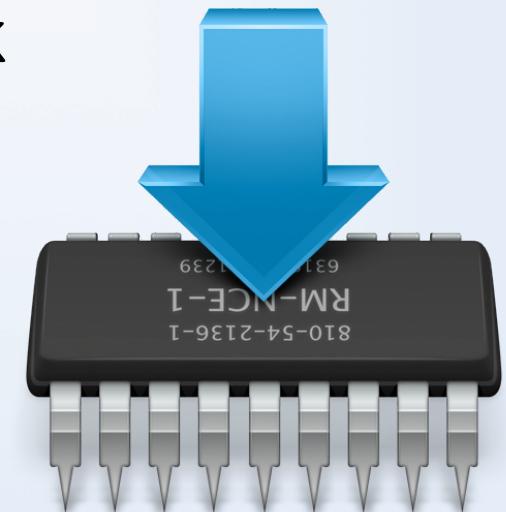
---

- Now we have all components for a backdoor
    - Wait for a magic packet (written to LBA x) that tells you which packet (at LBA y) to exfiltrate
    - Read that packet from LBA y, fix the checksums, and write it to LBA x
    - When LBA x is read again, a copy of the data at LBA y is retrieved
-

# Distribute your FW

---

- Currently Seagate firmwares are updated through a DOS utility
- Hdparm also has a firmware update functionality, but it did not work for my drive
- The DOS utility could easily be embedded into the system start to flash the HDD once the computer is rebooted



# Detection

---

- A modified FW is almost impossible to detect (except if you trigger the malicious behaviour)
  - A modified FW can pretend to do a firmware upgrade while not doing one to protect itself
  - Once written to the Flash, the firmware can burn a fuse of the Flash chip and make it read-only
  - Only secure detection is through extraction and comparison
-

# Countermeasures

---

- Sign that FW and only accept signed FWs!
- Do not allow code injection in the bootloader
- Does not help against bugs that allow code injection ...
- Do not leave anybody with root privileges near your hard drive



---

# Further reading

---

- [The paper](#) which describes an advanced remote exfiltration backdoor based on this firmware backdoor [5]
  - [Another paper](#) that describes a technique to make debugging of embedded firmwares easier [6]
  - Sprite\_tms [writeup](#) of his HDD hacking [3]
-

# Questions

---



---

# References

---

- [1] <http://www.eurecom.fr>
  - [2] <http://www.s3.eurecom.fr/~zaddach/index.html>
  - [3] <http://spritesmods.com/?art=hddhack&page=1>
  - [4] <http://forum.hddguru.com>
  - [5] [http://www.s3.eurecom.fr/docs/acsac13\\_zaddach.pdf](http://www.s3.eurecom.fr/docs/acsac13_zaddach.pdf)
  - [6] <http://www.s3.eurecom.fr/tools/avatar/>
-

# GDB Stub

---

- GDB can connect to targets using a serial interface and a simple protocol
  - There is a stub implementation in the source code tree, but not for ARM and it's bloated (for my purpose)
  - 6 primitives are enough to give debugging support with software breakpoints:  
Read bytes, write bytes, read registers, write registers, continue and get signal
-

# Diagnostic Overlays

---

- The firmware supports overlays, which is a means for OEMs to include custom functionality
  - Overlays can hook into control flow and add functionality
  - An overlay for diagnostics is provided with the original firmware
  - The overlay is loaded once its functionality is needed
-

# Reversing the firmware file format

---

- Try to find flash dump and memory dumps in a firmware update file → Bingo!
  - File is organized in sections, each section containing
    - First stage bootloader
    - Flash image
    - Main firmware
    - Overlays
  - If you are interested, write me for my hackish script
-