Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

Introduction to Reversing DXE drivers

Bruno Pujos

February 9, 2016

Perspective

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

Subject

- Talk about reverse engineering firmware
- **U**nified **E**xtended **F**irmware **I**nterface (UEFI)
- **D**river e**X**ecution **E**nvironmenent
- Everything here concern Intel x86 architecture

Interest

- What's going on my computer ?
- Developing your own
- Security
- For all this reading the documentation is **not** enough

Wait firmware ?

Agenda

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

1 Firmware and Flash

Firmware

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- Firmware is a **software**
- It is stored on non-volatile memory (ROM, flash, . . .)
- Low-level control program for the device
- Pretty much a firmware in everything
- It is also the ”first” code running at boot time

(Really) Basic computer

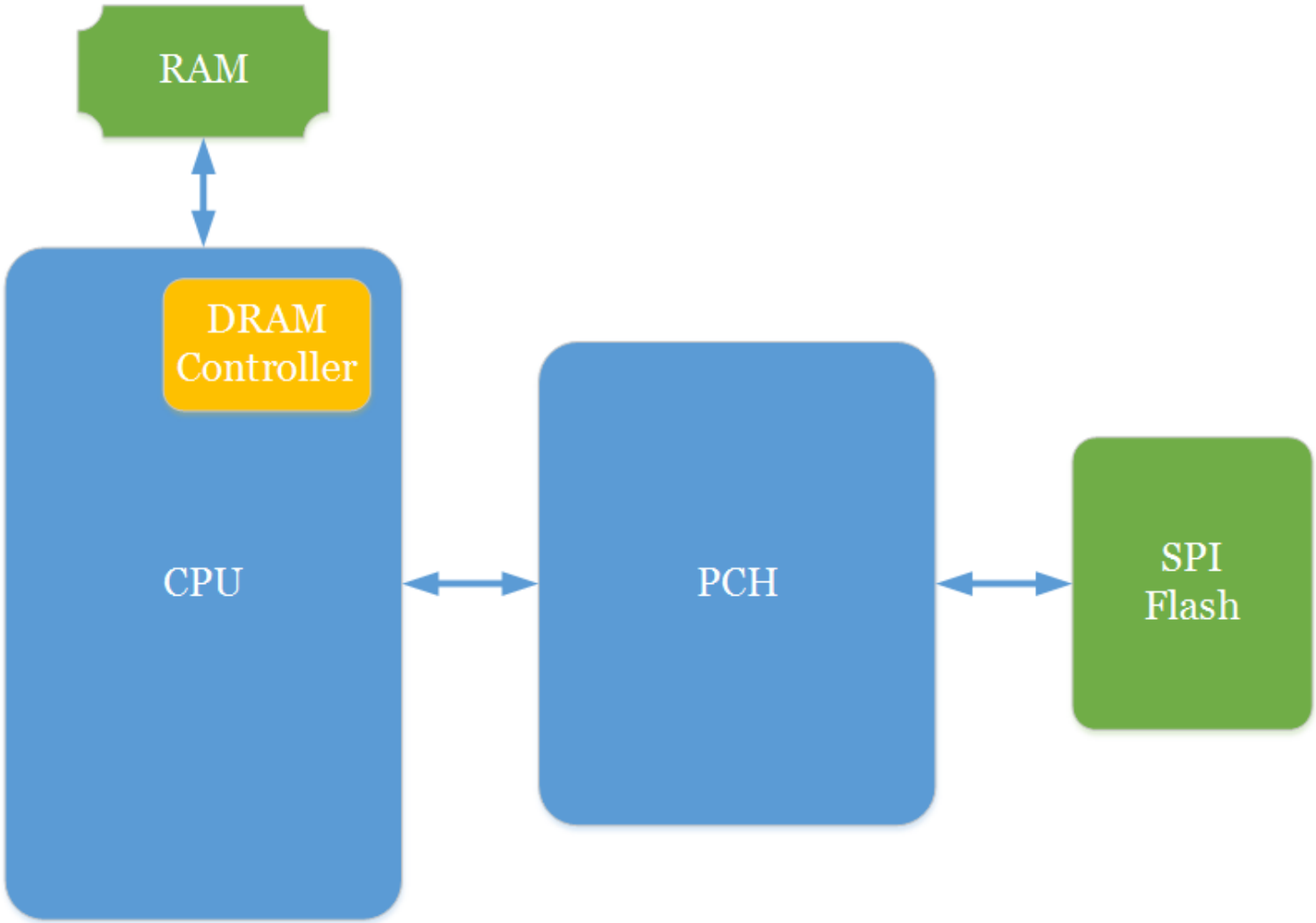
Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion



SPI Flash

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- **S**erial **P**eripheral **I**nterface
- SPI is not a ”real” standard. . .
- Store our firmware and other stuff
- In theory could be a LPC Flash
- What is inside it ?

SPI Flash

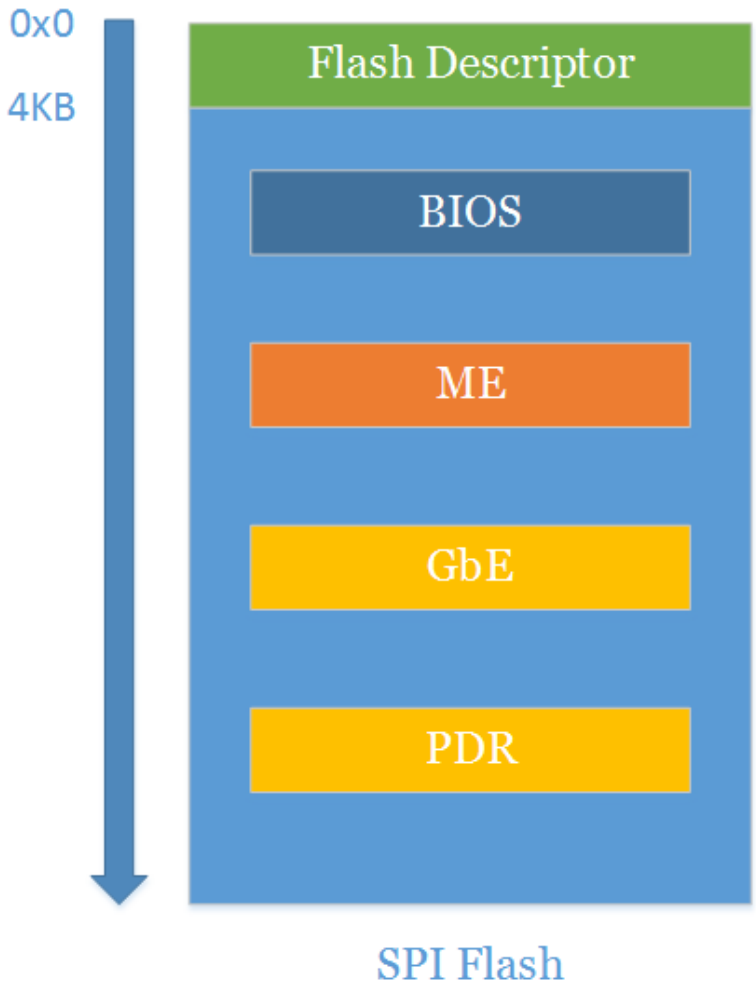
Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion



Getting the SPI Flash content

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

Hardware

- Not "simple" (at least for me)
- Full access
- Read and Write

Software

- Simple but limited access
- Never seen a flash limited for reading but possible
- Accessible through the PCH using MMIO registers (FADDR, FDATA and HSFC)
- Several tools allow you to dump the SPI Flash

Demo!

Demo!

Chipsec[1]

- Open-source tool developed by Intel
- Written in Python
- Works on Windows, Linux and UEFI shell
- Give a good abstraction on hardware
- Driver not signed
- Not complete and some bugs

Getting the SPI Flash content

```
python chipsec_util.py spi dump <FILE.BIN>
```

[1] https://github.com/chipsec/chipsec

Agenda

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

2 UEFI

UEFI

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

UEFI 101

- **U**nified **E**xtended **F**irmware **I**nterface
- Specification for firmware development since 2005
- Successor of EFI (1998)
- ”BIOS firmware following the UEFI specification” (or just say UEFI)

Goal

- Compatible with legacy
- Abstraction from the hardware and the implementation
- Modular: allowing code reuse
- Community effort
- . . .

We got our BIOS what now ?

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

What is in our BIOS ? Where is the code ?

FileSystem

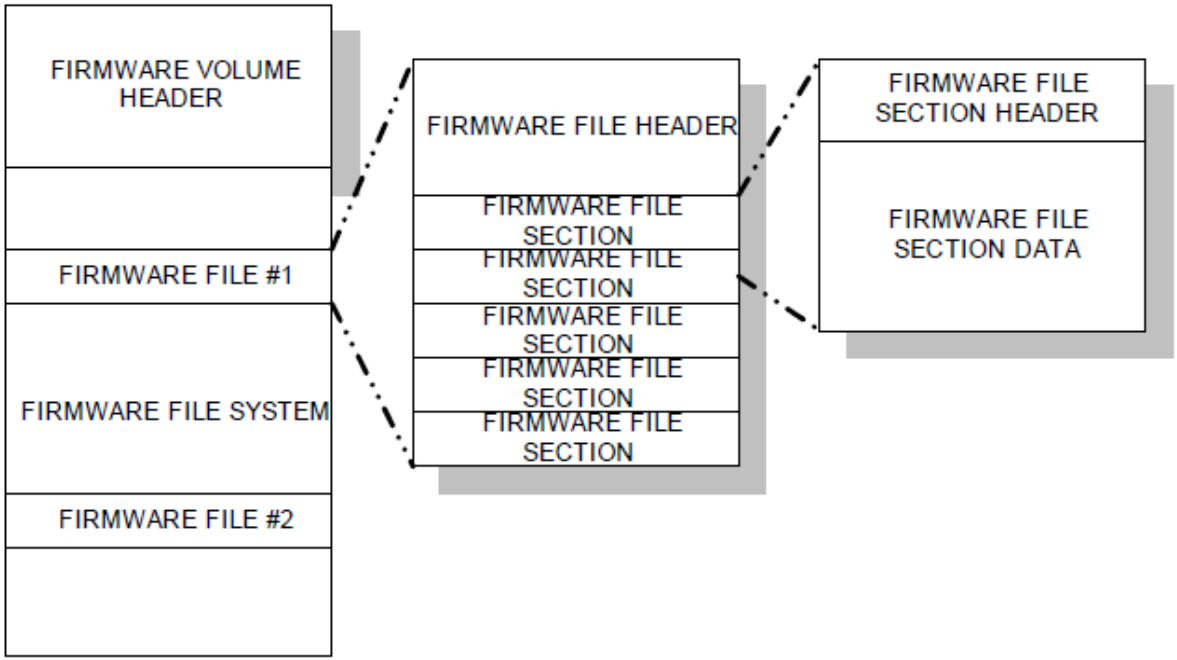
Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion



UEFI PI specification

FileSystem content

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- Different section containing different file
- Recursive FileSystem
- Most of the things are compressed

Demo!

Demo!

Tools

- Lots of tools for parsing the FileSystem
- Some constructor have addition to the specification
- `chipsec` does it but fail on certain things (in particular parsing of updates)
- Just grep -r 'uefi firmware parser tool' internet
- Using UEFIExtract[1]

[1] https://github.com/LongSoft/UEFITool

FileSystem Content

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- Several binary
- Different formats
- Data
- ...
- Lot of files
- Divided in sections

Boot step

1. Security (SEC) Phase
2. Pre-EFI Initialization (PEI) Phase
3. **Driver Execution Environment (DXE) Phase**
4. Boot Device Selection (BDS) Phase
5. Runtime (RT) Phase
6. Afterlife (AL) Phase

Agenda

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

3 DXE

Driver Execution Environment

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- Biggest part of a firmware
- Most of "user-input" will be handle here
- DXE phase is split in drivers
- DXE phase is generally concentrate in one section of the FileSystem

DXE driver

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

DXE drivers generality

- Drivers are executable PE32+ or TE (Tiano Executable)
- Native code or EFI Byte Code (never encounter EBC)
- different kind of ”drivers”:
  - EFI Application
  - Boot Service Driver
  - Runtime Driver

DXE driver goal

- Initialize hardware
- Hardware abstraction
- Management interface (GUI and so on)
- Network (PXE!)
- Boot paths
- . . .

DXE driver loading order

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

Loading order

- Pretty much arbitrary
- Dependencies between drivers
- Global order can be define
- Generally use DEPEX

DEPEX

- Store in a file at the same level than the driver (DXE or PEI)
- Really simple bytecode with 10 opcode
- Allow to precise protocol GUID necessary
- Can give you information about the dependencies
- Demo

Services and Protocols

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

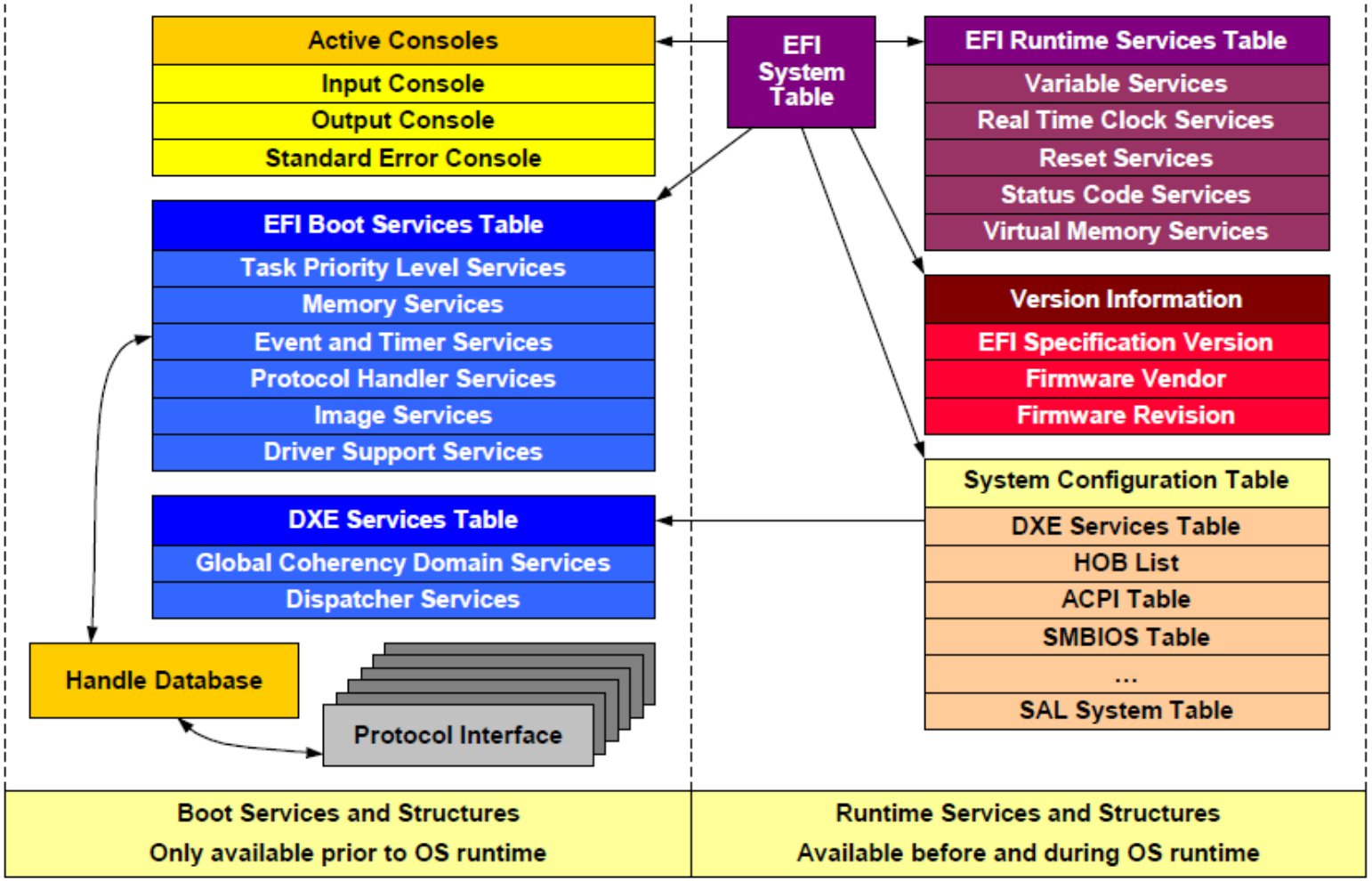
Services

- Driver need an API for basic things
- DXE Foundation MUST provide a limited set of services to the driver
- Allow timing, allocation, global variable, . . .
- And in particular declaration and request of protocols

Protocols

- Drivers need a way to communicate
- Protocols allow to propose services
- Can be anything from printing on the screen to sending network packet, or handling the USB.

EFI System Table



UEFI specification

Reversing first step

- EFI System Table and the services tables decoding
- Find any references to the services and in particular declaration and request of protocols
- Look for GUID
- With only that you can have a good idea of what the driver does
- IDA python plugin: `efi-utils`[1]

Demo!

[1] https://github.com/snare/ida-efiutils

Input and Output

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

MMIO

- Memory Map IO
- Typically a hard-coded address (but not always)
- Can give good information
- Use `chipsec` for finding the bases

IO port

- IO Port
- Can give good information
- Can be relative
- Some important one:

0x80	IO_POST
0xcf8	IO_PCI_CONFIG_ADDRESS
0xcfc	IO_PCI_CONFIG_DATA

Finding the documentation

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

Material Datasheet

- Processor
- PCH (or your equivalent)
- Graphics Card, network card, . . .

Code

- EDK, EDK2, UDK2014. . .
- Coreboot
- Driver implementation, . . .

Specification

- UEFI specification
- UEFI PI specification
- ACPI, PCI, TPM, . . .

Agenda

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

4 Conclusion

RE DXE driver 101

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- Nothing really hard
- Time consuming
- A lot of things are **not** documented (at least publicly)
- You don't need to read the specification for beginning
- If you are interested just go for it

Going further

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

- SMM
- SecureBoot
- MeasureBoot & TPM
- Vulnerability research
- ...

Questions ?

Introduction to Reversing DXE drivers

Firmware and Flash

UEFI

DXE

Conclusion

Questions ?