

Development of UEFI modules & Debugging them without NDA

Alx14 & Mars

Moscow, August 25-26, 2022

About Us

ALX14

- □ Independent Researcher
- □ UEFI, FemtoCells, IOS Researcher
- □ https://t.me/alx14
- □ https://twitter.com/alx_pwn

Mars

- Independent Investigator
- Penetration testing of WEB-applications,
 UEFI Researcher
- □ https://t.me/lostinffuf









Agenda



What is it? UEFI	Libreboot	OpenOCD
Type of Solutions	TianoCore EDKII	Intel System Studio
With Proprietary Blobs	LinuxBoot, NERF, HEADS?	Hardware for debug UEFI (Free)
Full Opensource	□ JTAG	Intel Galileo with OpenOCD
Coreboot	Types of Debugging UEFI	Hardware for debug UEFI (NDA)

- Intel JTAGs
- □ Intel ITP-XDP
- Intro to Closed Chassis
 - Debugging
- □ Intel® DCI
- □ Intel® CCA And USB Debug

- Variants of debugging
- What JTAG on x86 can be useful for?
- □ Setup environment for EDKII
- Build EDKII for Quark Platform
- □ FLASH Update using CH341a

- Connect SPI to CH341a
- □ Connect UART to Galileo
- Test UEFI Shell
- UEFI DEBUGGER OVERVIEW
- Host & Target Debug Setup
- Debugging UEFI Firmware using Intel® UDK w/ GDB





0

111.

What is it



- □ (Unified) Extensible Firmware Interface
- Modern industrial standard for x86 firmware
- □ Initially developed by Intel as BIOS replacement for IA64
- Performs HW initialization required to start an OS
- Modular and feature rich, uses well defined and known formats
- Mostly written in C, much easier to develop as legacy BIOS

Various Options for Intel Platforms



With Proprietary Blobs Full Open source

With conf binary components)

EDK II + Intel® Firmware
 Support Package (FSP)
 EDK II + binary modules
 Coreboot

□ Uses pre-compiled PEIM/DXE

With Proprietary Blobs

Intel® FSP is a binary distribution of Intel's silicon initialization code. The resources necessary to implement Intel silicon code are not publicly available.

Intel's FSP Strategy

 Distribute binaries of our proprietary silicon code to the public
 Enable this binary to plug into arbitrary firmware designs (coreboot, TianoCore, etc.)

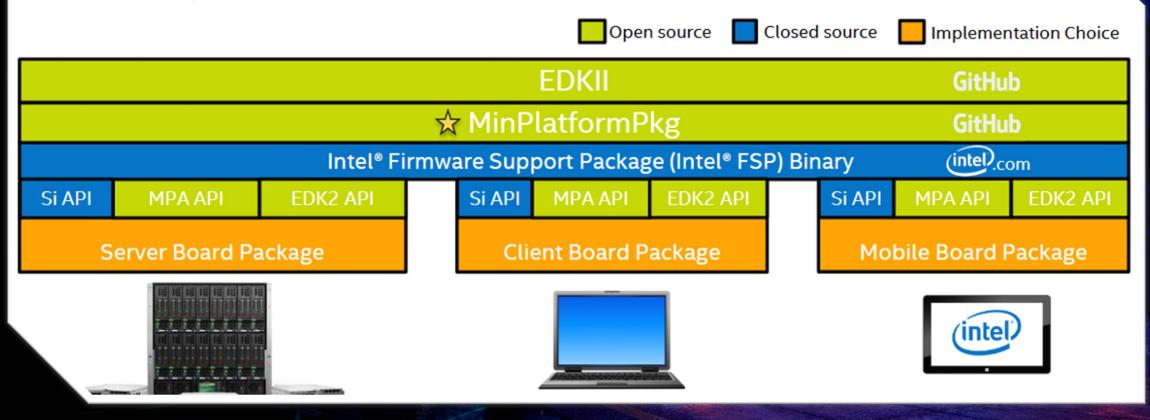
1726

2022

With Proprietary Blobs



Intel Open Platform Firmware Stack



Full Opensource







Open source bootable firmware development project for a variety of architectures. The philosophy is to do the bare minimum to keep the equipment running and then hand over control to another program called the payload.

Intel Management Engine (ME) firmware — we disable and effectively neutralize this blob by removing most of its code and setting flags to disable the ME coprocessor at boot time. Completely transfers control of the payload, with no part remaining resident in the system or even available for callback. Uses a very minimal interface to the payload and otherwise imposes no standards on the ecosystem.





Known briefly as GNU Libreboot - is a free software project which aims to replace the proprietary firmware on most free, lightweight computers and provides documentation aimed at ordinary users.

It is installed as a coreboot distribution without proprietary binary code. □ It is not a direct fork of coreboot; instead, it is a parallel effort that works closely together and rebase from time to time, created by changing the current code base in a similar way to the Linux-Libre project, which does a cleanup of the Linux kernel of proprietary components.







Tianocore EDK II is the reference implementation of UEFI by Intel. EDK is the abbreviation for EFI Development Kit and is developed by the TianoCore community. TianoCore EDK II is the defacto standard generic UEFI services implementation.

* For Intel CPUs need intel FSP



OFFZONE



LinuxBoot, NERF, HEADS? What's What?

LinuxBoot, NERF, HEADS? What's What?





LinuxBoot

is the project that replaces specific firmware functionality with a Linux kernel. LinuxBoot is agnostic to what initramfs is used with the kernel.

NERF

is LinuxBoot with u-root as the initramfs. u-root contains boot policy tools in Go (e.g. PXE booting, booting via GRUB config) among standard busyboxlike utilities rewritten in Go.

HEADS

is a secure runtime that can be used as the initramfs for LinuxBoot. Take a look at the repository on GitHub. See osresearch.net for more documentation on HEADS.



V_` [/_` / _]

____ | _____ [_____







JTAG

□ Joint Test Action Group IEEE 1149.1

□ Is an industry standard for verifying designs and testing printed circuit boards after manufacture. JTAG implements standards for on-chip instrumentation in electronic design automation (EDA) as a complementary tool to digital simulation.

IEEE Standard 1149.1

https://standards.ieee.org/findstds/standard/1149.1-2013.html









111.

Types of Debugging UEFI



Over OpenOCD (Full opensource)

target system.

Over NDA tools aka intel System Studio

Emulation

Open On-Chip Debugger is open-source software that interfaces with a hardware debugger's JTAG port. OpenOCD provides debugging and in-system programming for embedded target devices. OpenOCD provides the ability to flash NAND and NOR FLASH memory devices that are attached to the processor on the

□ Fully proprietary Opensource Cost 2000\$ An EFI DXE binary Working with special emulator based on intel debuggers Unicorn Non - NDA Has NDA

Hardware for debug UEFI (freeware version)



Intel Galileo Gen2 board secondary market price is 20\$ open hardware schematics and open source reference code for UEFI & EDK II (no NDA required)

FTDI or other JTAG board

- □ I am use Olimex ARM-USB-OCD-H (50\$)
- □ FTDI 2232H for JTAG more chipper 10\$

 That's a minimum of \$30 to start
 It allows even students to start working with UEFI

OFFZONE

Intel Galileo with OpenOCD

Recommended setup for debugging

Host System

□ USB 2.0 male-male A-B cable

JTAG Probe

ARM-JTAG-20-10 Adapter

□ JTAG Port

□ Intel® Galileo Board

Serial Cable to view boot process

Power Supply





F F ONE 2022

Hardware for debug UEFI (NDA version)

Signing NDA

Buying Intel System Studio - 2000\$ or 30 day trial

Buying Compatible Board for Debug (minnow board) 400\$

□ Buying intel Bluebox – 3000\$

- Intel ITP-XDP on ebay 370-1000\$
- Intel SVT not found on public

□ That's a minimum of \$2,700 to start

2022

Intel variants of JTAGs



Intel In-Target Probe eXtended Debug Port (ITP-XDP)

Intel Direct Connect Interface (DCI aka debug over USB 3.0):
 USB3 Hosting DCI (USB Debug cable)
 BSSB Hosting DCI (Intel SVT Closed Chassis Adapter)

Intel ITP-XDP





Figure

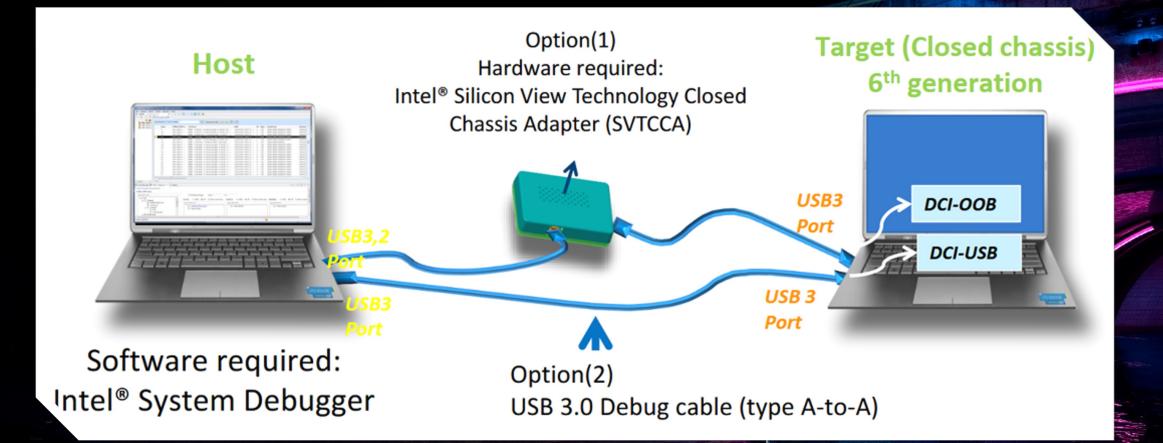
1: Cable connections between

ITP-XDP3 probe and the Sasby Island target



Intel Direct Connect Interface Overview



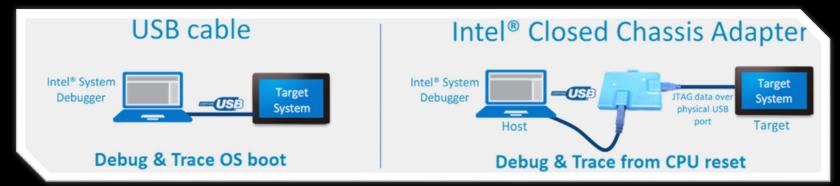


Intro to Closed Chassis Debugging



JTAG*-based system debug and system trace over low-cost USB* connections Intro to Closed Chassis Debugging

- Works on production hardware instead of expensive engineering samples
- Debug over a standard USB* connection instead of expensive JTAG probes
- Supports debugging Unified Extensible Firmware Interface (UEFI) platform firmware
- Developers no longer need to wait for a turn at a shared JTAG debugging station
- Design flexibility alleviates the requirement for an accessible hardware JTAG port



Intel® CCA And USB3.0 Debug Cable





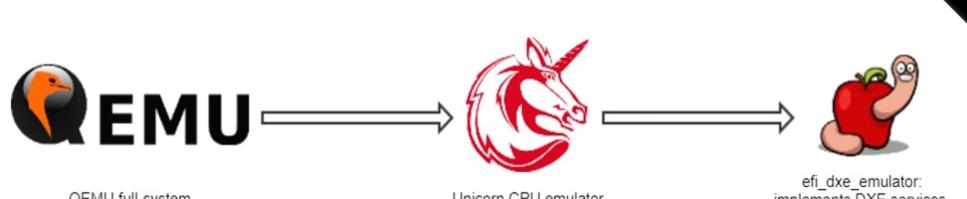


Intel SVTCCA can be purchased through the Intel® Design-In Tools Store

Intel® DCI DbC2/3 A-to-A Debug Cable

Debugging and emulation of UEFI





QEMU full-system emulator Unicorn CPU emulator

efi_dxe_emulator: implements DXE services, handle database, PE loader, heap, debugger, etc.

OFFZONE

efi_dxe_emulator components

efi_dxe_emulator components

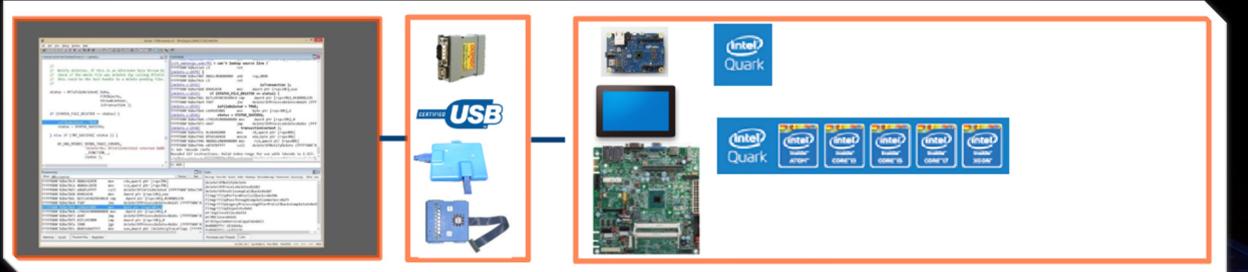


 efi_dxe_emulator - based on the famous Unicorn emulation engine, which is a fork of QEMU.

IQEMU is just a tool, Unicorn engine is a real framework, and as such it provides a rich set of APIs that can be used by a large number of programming languages through dedicated bindings. Unfortunately, the Unicorn engine itself is not sufficient to perform dynamic analysis of UEFI modules. Since Unicorn focuses only on emulation of the processor, it has no a priori knowledge of UEFI-related concepts such as boot services, runtime services, protocols or even the PE format. To address these shortcomings, the efi_dxe_emulator independently implements some of the most commonly used UEFI services.

Variants of debugging







Variants of debugging with intel UDK



🕲 eXDI 'exdi:clsid={F56FC1A6-3422-4320-A7F3-41EAEC2A367D}' - WinDbg:6.11.0001.4 🗕 🗖 📉	SoftDebugger Debug Console
File Edit View Debug Window Help	Intel(R) UEFI Development Kit Debugger Tool Version 1.5.0 Debugging through serial port (COM30:921600:None) Redirect Target output to TCP port (20715) Debug agent revision: 0.4
	Debug agent revision: 0.4
Command - eXDI 'exdi:clsid={F56FC1A6-3422-4320-A7F3-41EAEC2A367D}' - WinDbg:6.11.0001.4 🚬 📧	
Machine Name: Primary image base - 0x00000000 Loaded module list - 0x00000000 System Uptime: not available ffff856d7 cc int 3 0: kd>.load C:\Program Files (x86)\Intel\Intel(R) UEFI Development Kit Debugger Tool\UdkExtension.dll 0: kd>.aympath D:\WORK\GITSVN\BUILD\QUARK\DEBUG_VS2015X86\IA32\UEFICPUPKG\SECCORE\SECCORE\DEBUG\ Symbol search path is: D:\WORK\GITSVN\BUILD\QUARK\DEBUG_VS2015X86\IA32\UEFICPUPKG\SECCORE\SECCORE\DEBUG\ Expanded Symbol search path is: d:\work\gitsvn\build\quark\debug_vs2015x86\ia32\ueficpupkg\seccore\seccore\debug\ 0: kd>.reload /f SECCORE-0x0'FFFF6218 V	
0: kd>	
d:\work\gitsvn\edk2\sourceleveldebugpkg\library\pecoffextraactionlibdebug\pecoffextraactionl🖹 📧	×
<pre>AssWriteDr0 (Signature); AssWriteDr0 ((UINTN) ImageContext->PdbPointer); AssWriteDr2 ((UINTN) ImageContext); AssWriteDr3 (IO_PORT_BREAKPOINT_ADDRESS);</pre>	😕 Iocalhost:20715 - Tera Term VT 🚽 🗖 🗾
<pre>if (LoadImageMethod DEBUG_LOAD_IMAGE_METHOD_IO_NW_BREAKFOINT) { AsmWriteDr7 (0x20000480), AsmWriteCr4 (Cr4 BIT3),</pre>	File Edit Setup Control Window Help Variable MTRR[03]: Base=000000000000000000000000000000000000
<pre>// Do an IN from IO_FORT_BREAKFOINT_ADDRESS to generate a HW breakpoint until the port // returns a read value other than DEBUG_AGENT_IMAGE_WAIT // do {</pre>	Variable MTRR[03]: Base=000000000000000000 Mask=0000000000000000 Variable MTRR[04]: Base=000000000000000 Mask=00000000000000 Variable MTRR[05]: Base=0000000000000000 Mask=0000000000000 Variable MTRR[06]: Base=000000000000000000 Mask=000000000000000000000000000000000000
DebugAgentStatus - IGRead8 (IO PORT BREAKPOINT_ADDRESS);) while (DebugAgentStatus DEBUG_AGENT_IMAGE_WAIT);	MTRR Ranges UC: 000000000000000000000000000000000000
<pre>} else if (LoadImageMethod DEBUG_LOAD_IMAGE_METHOD_SOFT_INT3) { // // Generate a software break point. //</pre>	UC:000000000000000000000000000000000000
CpoBceakpoint () /	Debug Timer: FSB Clock = 200000000 Debug Timer: Divisor = 2
<pre>// // Restore Debug Register State only when Host didn't change it inside exception handler. ' E.g.: User halts the target and sets the HW breakpoint while target is in the above exception handler </pre>	Debug Timer: Frequency = 100000000 Debug Timer: InitialCount = 10000000 Send INIT break packet and try to connect the HOST (Intel(R) UDK Debugger Tool v 1.5)
Ln 150, Col 1 Sys 0:eXDI KD Proc 000:0 Thrd 000:0 ASM OVR CAPS NUM	HOST connection is successful? PDB = d:\work\gitsvn\Build\Quark\DEBUG_VS2015x86\IA32\UefiCpuPkg\SecCore\Sec Core\DEBUG\SecCore.pdb

.

What JTAG on x86 can be useful for?



• Incident investigation (reading Firmware reading, rootkit detection)

2. For research proprietary modules and Exploit Dev (Secure Boot, Boot Guard, SMM)

5. Low-level debugging (UEFI DXE/PEI, drivers, hypervisor)

• Performance Analysis







and the second s

1

WILL HAVE

111.

11mm

111/1//

Setup EDKII on Linux

Build EDKII for Quark Platform



apt install -y git nasm iasl build-essential uuid-dev gdb gcc-4.9 g++-4.9 git clone https://github.com/tianocore/edk2.git -b vUDK2018 git clone https://github.com/tianocore/edk2-non-osi.git

export WORKSPACE=**\$PWD** export PACKAGES_PATH=**\$WORKSPACE**/edk2:**\$WORKSPACE**/edk2non-osi/Silicon/Intel export EDK_TOOLS_PATH=**\$WORKSPACE**/edk2/BaseTools cd **\$WORKSPACE**/edk2

git submodule update --init

make -C BaseTools

. edksetup.<mark>sh</mark> BaseTools

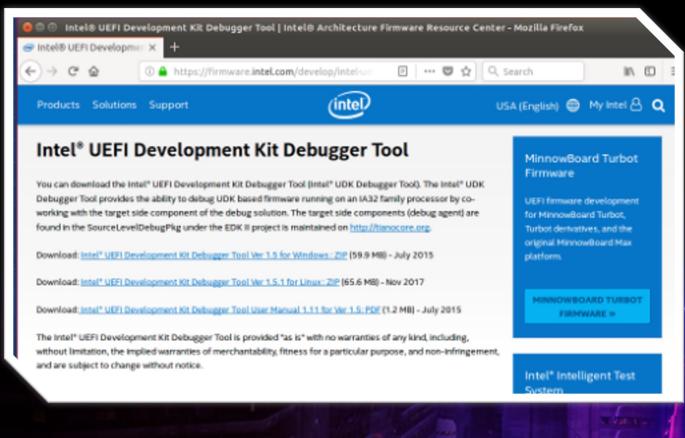
build -a IA32 -t GCC49 -p QuarkPlatformPkg/Quark.dsc -D SOURCE_DEBUG_ENABLE build -a IA32 -t GCC49 -p QuarkPlatformPkg/QuarkMin.dsc -D SOURCE_DEBUG_ENABLE

- Ubuntu 16.04
- GIT client
- GCC 4.9 compiler
- □ ASL compiler
- Python 2.7
- □ NASM

🗖 GNU Debugger (GDB)

Host & Target Debug Setup

Download application: https:/firmware.intel.com-Develop-Tools





OFFZONE

Target source: SourceLevelDebugPkg at TianoCore.org

Nuances associated with developing



- incorrectly structured information in the documentation
- Don't forget git submodules update -unit
- Compilers to build for Quark platform should be 4.9
- Don't forget to install all the packages you need to build the firmware
- We did the entire compilation in the distribution Ubuntu 16.04, with Ubuntu 18.04 there are their own quirks and drawbacks, we started everything with the docker container
- Tip: It is better not to hurry, think about every command you entered
- □ You can use qemu with ovmf to test it, it is faster and easier (better in some cases)
- If you do not succeed in the build step, look carefully at the errors and try to figure out what went wrong. Maybe python is the wrong version, maybe nasm is not installed, maybe the compilers are the wrong version or make; maybe you don't have openssl missing or basically an error because of dependencies

FLASH Update using CH341a





Be sure to dump your firmware from Intel Galileo Flashing new fresh UEFI

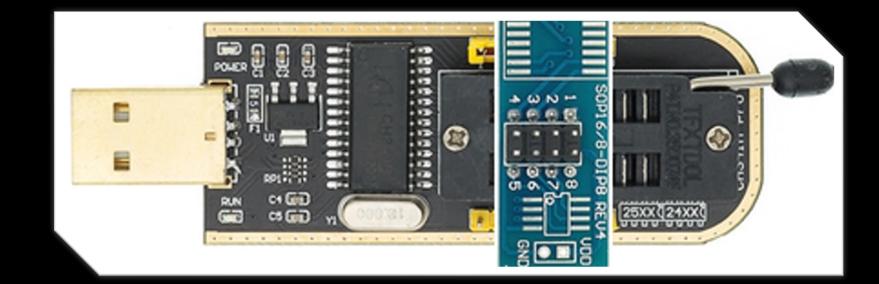
flashrom --programmer ch341a_spi -r galileo-orig.bin flashrom --programmer ch341a_spi –w QUARK.fd ~/YouWorkDir/Build/Quark/DEBUG_GCC49/FV/QUARK.f

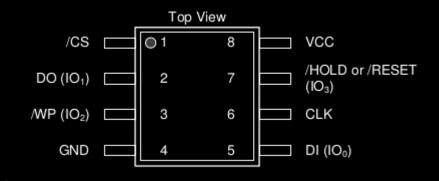
Connect SPI to CH341a



Motherboard ISP Connector Pin out

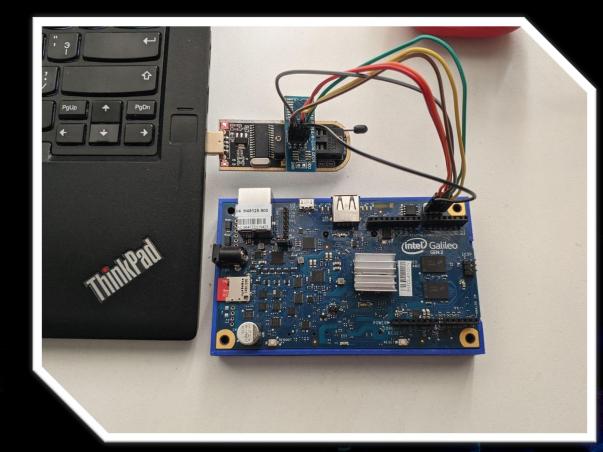






Connect







UART and JTAG to Galileo

SPI to CH341a

Test UEFI Shell

COM5 - Tera Term VT

File Edit Setup Control Window Help arlyPlatformInit for PlatType=0x08

Quark EDKII Stage 1 Boot Image 0 ***** Quark EDKII Stage 2 Image 9xFFD09090:0x801E0000 ***** Quark EDKII Payload Image 9xFFC00000:0x80100000 ***** -----(XX arly Platform Thermal Sensor Init ARNING: Ioh MAC IB:0, D:20, F:6] NO HW ADDR CONFIGURED!!! ARNING: Ioh MAC IB:0, D:20, F:7] NO HW ADDR CONFIGURED!!! egValue = 00000000 sFirstBoot = 1 . EnableFastBoot= 1. oot with Full cfg T stall PPI: 7408D748-FC8C-4EE6-9288-C4BEC092A4±0 RC Entry RC McFuseStat 0x00000429 RC Fuse : fus_dun_ecc_dis. RC dram_width 0 RC rank_enables 1 ddr_speed Ø RC flags: SCRAMBLE_EN RC flags: SCRANBLE_EN RC density=1 tCL=6 tRAS=37500 tWTR=10000 tRRD=10000 tFAW=40000 nstallEfiMemory. ound 0×60000 bytes at 0×0 . ound 0×60000 bytes at 0×60000 . ound $0 \times 7CF0000$ bytes at $0 \times FE00000$. ound 0×10000 bytes at $0 \times FF00000$. ound 0×10000 bytes at $0 \times FF00000$. ound bilobolydes wie onin 8000 0xCCB0000, MemoryLength 0x3140000 mullain Base Address : 0xFDF0000 ound Microcode ADDR:SIZE 0xFFF200574:8x2000 muMain Base Address : 0xFDF0000 ound Microcode ADDR:SIZE 0xFFF20574:0x2000 aveConfig. 10 IoApicBase = FE01C000 emoryInit Complete. arly PCIe controller initialization latform Errats After MRC arlyPlatformConfigGpioExpanders () egister PPI Notify: F894643D-C449-42DI-8EA8-85BDD8C65BDE ROGRESS CODE: U003020003 I0 emp Heap : BaseAddress=0x8007C000 Length=0x4000 otal temporary memory: 32768 bytes. temporary memory baap used for MobList: 4440 bytes. temporary memory heap occupied by memory pages: 0 bytes. ld Stack size 16384. New stack size 131072 tack Nob: BaseAddress=0x80000 Length=0x4000 eap Offset = 0x733R8000 Stack Offset = 0x733B0000 PDB = /home/edk2/Build/Quark/DEBUG_GCC49/IA32/MdeModulePkg/Core/Pei/PeiMain/DEBUG/PeiCore.dll oading PEIM at 0x0000FDDD150 EntryPoint=0x40000 PDB240 PeiCore.efi einstall PPI: 8C8CE578-803D-4FIC-9935-896185C32DD3 einstall PPI: 8C8CE578-803D-4FIC-9935-896185C32DD3 einstall PPI: 9573C07A-3DCE+4CA-BDF-1E9689F2349A einstall PPI: 9573C07A-3DCE+4CA-BDF-1E9689F2349A einstall PPI: 9580ABFE-5979-4914-972P-6DEF28C278A6 einstall PPI: 805652B4-6D33-4DCE-89DB-83DF9766FCCA nstall PPI: 80464D-C449-42DI-88A-88DDB8C65BDE offy: PPI Guit: F894643D-C449-42DI-88A-88DDB8C65BDE NE% Settings NE% Settings NE% Settings NE% Settings NE% Settings Status = Success **FRR Settings** R Default Type: 00000000000000000

FF ONE 2022

🔟 COM5 - Tera Term VT

- 0

File Edit Setup Control Window Help

Fs1:\EFI\exploits> ls Directory of: fs1:\EFI\exploits

05/04/22 03:15a <DIR> 4,09 05/04/22 03:15a <DIR> 16,33 05/04/22 04:00a 88,00 1 File(s) 88,064 bytes 2 Dir(s)

4,096 . 16,384 .. 88,064 GalileoPwn.efi oytes

s1:\EFI\exploits> GalileoPwn.efi

Inable to locate SMM access protocol: 0x8000000e SMM access 2 protocol is at 0xe40f550 Available SMRAM regions: * 0x0fe00000:0x0fffffff Buffer for SMM communicate call is allocated at 0xfdde010 Inable to locate SMM base protocol: 0x8000000e SMM communication protocol is at 0xf05ecdc Communicate() returned status 0x00000000, data size is 0x1000 SmmHandler() was executed, exploitation success! Press any key to quit...







0

IIIII

111.

/IIMINS

at the last

DEMO