FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Reverzní analýza UEFI modulů PEI a DXE
Student:	Bc. Luigino Camastra
Vedoucí:	Ing. Josef Kokeš
Studijní program:	Informatika
Studijní obor:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	Do konce letního semestru 2020/21

Pokyny pro vypracování

1) Seznamte se s technologií UEFI. Popište její základní myšlenky, principy, možnosti.
2) Popište architekturu UEFI, jeho komponenty a vztahy mezi nimi.
3) Zaměřte se na moduly PEI a DXE. Vysvětlete jejich účel v rámci bootovacího procesu.
4) Popište metody získání analyzovatelného obrazu obou modulů a demonstrujte je prakticky na existujícím hardware.
5) Technikou reverzního inženýrství prozkoumejte hlavní PEI modul.
6) Dále prozkoumejte DXE modul. Zaměřte se na jeho spuštění z PEI a jeho hlavní funkčnosti.
7) Diskutujte zjištěné výsledky.

Seznam odborné literatury

Dodá vedoucí práce.

prof. Ing. Róbert Lórencz, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 30. ledna 2020

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práca

Reverzná analýza UEFI modulov PEI a DXE

Bc. Luigino Camastra

Katedra informační bezpečnosti
Vedúci práce: Ing. Josef Kokeš

19. mája 2020

Poďakovanie

V prvom rade by som sa chcel podakovat veducemu prace panovi Ing. Josefovi Kokesovi za poskytnuté rady a odbornu pomoc pri vypracovani tejto diplomovej prace. Velke podakovanie patri mojej rodine a kolegom z prace (hlavne kolegovia Jan Rubín a Adolf Středa) za velku podporu.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 19. mája 2020 . . . . . .

Odkaz na túto prácu

Camastra, Luigino. *Reverzná analýza UEFI modulov PEI a DXE*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2020.

Abstrakt

Táto diplomová práca sa venuje reverznej analýze UEFI modulov. Popisuje UEFI fázy a ich podstatu pri bootovaní operačného systému. Druhá časť sa zaoberá spôsobmi, ako získať UEFI firmware image, a nástrojom k získaniu hlavných DXE a PEI modulov. Následne opisuje hlavnú funkciu získaných modulov. Na záver poukazuje na to, či program zodpovedá dokumentácií a taktiež na bezpečnostné programátorské zvyky.

Klíčová slova Unified Extensible Firmware Interface, UEFI, PEI fáza, DXE fáza, SPI Flash, Reverzné inžinierstvo, Firmware

Abstract

This thesis describes reverse engineering of UEFI modules. UEFI phases and their main responsibilities while booting an operating system are introduced. In the second part of this work, options for dumping the UEFI firmware image and which tools are used for acquiring PEI and DXE modules are described. Furthermore, main functionalities of the obtained modules are reported. Finally, the UEFI modules are compared with the documentation and checked whether they adhere to security conventions.

Keywords Unified Extensible Firmware Interface, UEFI, PEI phase, DXE phase, SPI Flash, Reverse engineering, Firmware

Obsah

Zoznam obrázkov

Úvod

Počítač je zariadenie na spracovanie dát, ktorý sa stal neodmysliteľnou súčasťou nášho každodenného života. Pre bežného používateľa je však jeho spustenie neznámym procesom. Pri každom zapnutí počítača zahájime proces, ktorý umožní bezpečne a dôveryhodne spustiť operačný systém. Proces štartovania je vykonaný technológiou, ktorá začína pred spustením operačného systému.

V tejto diplomovej práci sa budeme venovať práve tejto technológií, konkrétne UEFI – Unified Extensible Firmware Interface. UEFI predstavuje softvérové rozhranie najnižšej úrovne medzi operačným systémom a firmwarom hardvérových komponentov. Nachádza sa vo flash pamäti v základnej doske počítača. Na rozdiel od staršieho BIOSu podporuje väčšie pevné disky, má rýchlejšie doby bootovania, viac bezpečnostných prvkov a lepšiu grafiku. Disponuje funkciou Secure Boot, ktorá umožňuje štart systému s použitím certifikovaných softvérových komponentov.

Procesy, ktoré odkrývajú fungovanie systému ma vždy zaujímali a práve preto som si vybral túto tému.

Cieľom mojej diplomovej práce je prostredníctvom nástrojov reverzného inžinierstva analyzovať hlavné UEFI moduly v PEI a DXE fázy. Po vykonaní reverznej analýzy poukážeme na jej hlavnú funkčnosť počas bootovania operačného systému, bezpečnostné programátorské zvyky a taktiež na to, či program zodpovedá dokumentácií.

Práca je rozdelená do 7 kapitol. V prvej kapitole sa budeme venovať úvodu do UEFI a predstavíme si jednotlivé skupiny patriace do UEFI komunity. V druhej kapitole definujeme UEFI špecifikácie. Následne si opíšeme základnú UEFI architektúru a predstavíme si niektoré architektonické rozhrania. Štvrtá kapitola sa zaoberá získaniu firmwaru z SPI flash. Výsledok reverznej analýzy hlavných UEFI modulov v PEI a DXE fáze budeme opisovať v piatej a šiestej kapitole. V poslednej 7 kapitole budeme diskutovať o výsledkoch reverznej analýzy hlavných UEFI modulov.

Kapitola 1

Úvod do UEFI

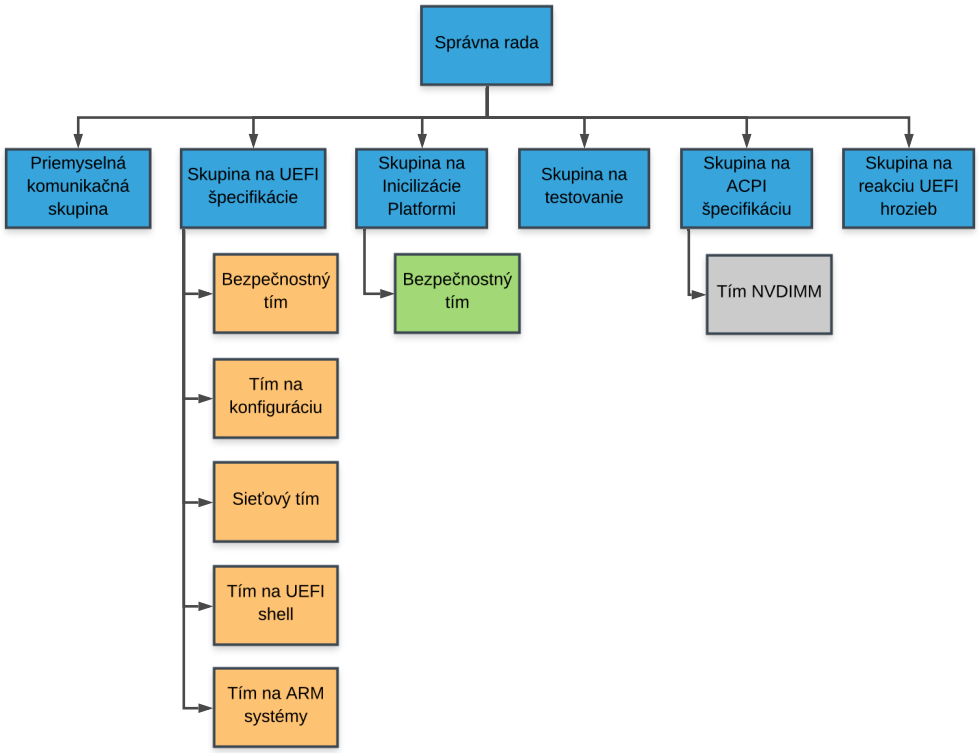
V polovici roku 1990 Intel začal vyvíjať 64-bitové procesory z rodiny Itanium (IA-64). Počas vývoja narazil na problém súvisiaci s limitujúcou BIOS (Basic Input Output System) technológiou. Bolo to v období, keď procesory bežali na 16-bitový kód tzv. „Real Mode“ a počítač bol architekturálne obmedzený na 1 MB RAM . Hlavnou funkciou BIOSu bolo spustiť primárny test časti základnej dosky, pamäťového systému a pripojených komponentov, nájsť a inicializovať operačný systém. V poslednom rade taktiež spustiť tzv. bootovací system – proces zavádzania operačného systému pri zapnutí počítača. Tieto hlavné funkcie sa výrazne nezmenili a ostali dodnes [1].

Procesor z rodiny Itanium nemal dobré ohlasy z dôvodu nevhodnej počítačovej firmware architektúry. To prinútilo firmu Intel vyvinúť nový framework podporujúci procesory z rodiny Itanium, tzv. Extensible Firmware Interface (EFI). Intel publikoval svoje EFI špecifikácie a v roku 2005 sa spojilo viacero technologických firiem, aby vytvorili tzv. UEFI Forum. Tak vznikla UEFI komunita. Jej hlavnou úlohou je modernizovať bootovací proces operačného systému. UEFI je teda štandard, ktorý opisuje EFI firmware architektúru a špecifikáciu – Unified Extensible Firmware Interface (UEFI). V súčasnosti predstavenstvu UEFI predsedá firma DELL[1].

UEFI Forum bol založený vo Washingtone a delí sa na niekoľko skupín zobrazených na obr. 1.1. UEFI členovia podporujúci vývoj sú napríklad AMD, AMI, Apple, Dell, HP, IBM, Insyde, Intel, Lenovo, Microsoft a Phoenix[2]. Následne si opíšeme jednotlivé skupiny, ktoré patria pod UEFI Forum.

1.1 Priemyselná komunikačná skupina

Skratka skupiny je ICWG (Industry Communications work group). Hlavnou úlohou priemyselnej komunikačnej skupiny (ICWG) je dostať UEFI špecifikácie do povedomia priemyselného sektoru cez rôzne vzdelávacie, promočné a marketingové programy. Spolupracuje s „Original design manufacturer“ (ODM) a



Obr. 1.1: Rozdelenie UEFI Forum

„Original equipment manufacturer“ (OEM) komunitou, s priemyselnými partnermi a médiami. ICWG je zodpovedná za každý úspech, pokrok a zviditeľnenie UEFI Forum. Vyvíja marketingovo technický komunikačný kanál prostredníctvom ktorého sa vydávajú vedecké články, tlačové správy a prezentácie [3].

1.2 Skupina na UEFI špecifikácie

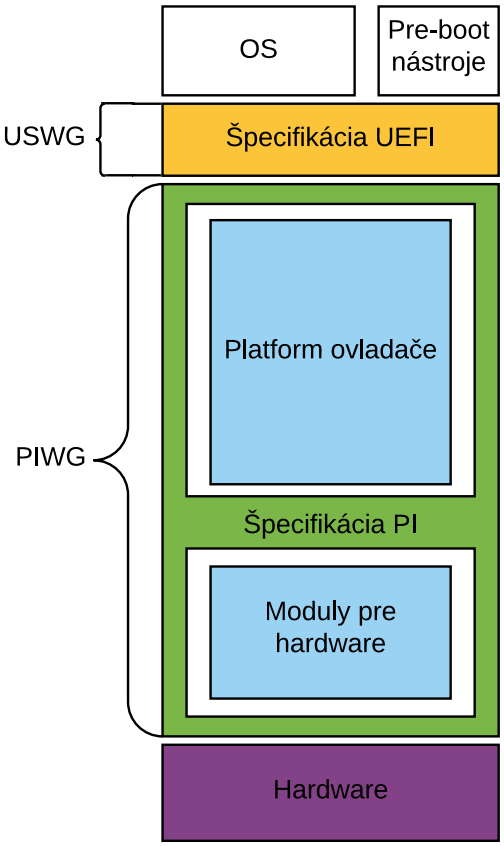
Skratka skupiny je USWG (UEFI specification work group). Úlohou USWG skupiny je manažovať a rozvíjať definíciu UEFI špecifikácie. Cieľom UEFI špecifikácie je definovať rozhranie, dátové štruktúry a konvencie používané pri prenose kontroly z firmware platformy na operačný systém. UEFI poskytuje obmedzený počet bežiacich služieb. Okrem podpory UEFI špecifikácie, skupina pracuje taktiež na návode „Driver Writer's Guide”, v ktorom opisuje postup, ako vyvíjať produkty spolupracujúce s UEFI špecifikáciami. Na obr. 1.2 je náhľad na UEFI prostredie. To sa skladá z dvoch častí:

- „Platform Initialization work group“ (PIWG) je zodpovedná za časť „Platform Initialization“ (PI).

- „UEFI specification work group“ (USWG) je zodpovedná za UEFI časť[3].

Do skupiny sú zaradené jednotlivé tímy ako:

- Bezpečnostný tím, zodpovedný za všetky bezpečnostné prvky. Jeho úlohou je skonštruovať bezpečnostnú architektúru v UEFI špecifikácií.
- Tím na konfiguráciu, zodpovedný za všetky konfiguračné prvky a za vytvorenie infraštruktúry UEFI konfigurácie známe ako HII (Human Interface Infrastructure), ktorá taktiež patrí do UEFI špecifikácie.
- Sieťový tím, zodpovedný za sieťové prvky. Vykonáva všetky aktualizácie sieťových prvkov v UEFI špecifikácií, obzvlášť zahrnutie sieťovej infraštruktúry IPv6.
- Tím na UEFI shell, zodpovedný za príkazové prvky shell, za vytvorenie špecifikácie UEFI shell a udržiavanie a vyvíjanie špecifikácie UEFI shell.
- Tím na ARM systémy definuje UEFI špecifikácie pre ARM systémy[3].



Obr. 1.2: Náhľad do UEFI PI

1.3 Skupina na Inicializácie Platformy

Skratka skupiny je PIWG (Platform Initialization work group). PIWG skupina je zodpovedná za preinicializačnú fázu „Pre-EFI Initialization“ (PEI) a spúšťanie ovládačov špecifikácie „Drvier Execution Enviroment“ (DXE). Autorom PEI a DXE špecifikácií je Intel. Existujú aj predom dohodnuté špecifikácie odhlasované väčšou skupinou členov UEFI Fórum. Implementácia týchto špecifikácií slúži na inicializovanie počítača a tiež ako rozhranie pre ďalšiu vrstvu UEFI špecifikácie. Implementácia PEI a DXE fázy si však nevyžaduje súhlas od UEFI Fórum. Na obr. 1.2 môžeme vidieť, akú časť má PIWG na UEFI prostredí [3].

V skupine sa takiež nachádza Bezpečnostný tím, ktorý je zodpovedný za všetky bezpečnostné prvky. Jeho úlohou je skonštruovať bezpečnostnú architektúru pre špecifikácie PEI a DXE [3].

1.4 Skupina na testovanie

Skratka skupiny je UTWG (UEFI Testing work group).Skupina je zodpovedná za:

- Odsúhlasenie certifikovaných testovacích súprav pre „Unified Extensible Firmware Interface“ (UEFI špecifikácia)
- Odsúhlasenie certifikovaných tzv. „open-source“ testovacích súprav pre špecifikáciu „Unified Extensible Firmware Interface“ (UEFI).
- Odsúhlasenie certifikovaných testovacích súprav pre „Advanced Configuration and Power Interface” špecifikáciu (ACPI)[3]

1.5 Skupina na ACPI špecifikáciu

Skratka skupiny je ASWG (ACPI Specification working group).

ASWG skupina zodpovedá za špecifikáciu „Advanced Configuration and Power Interface“ (ACPI). Úlohou špecifikácie je definovať flexibilné a rozšíriteľné rozhranie na konfiguráciu systému, správu napájania a RAS (Reliability, Availability, Supportability) funkcie. RAS funkcie sú užitočné pre systémy vo všetkých segmentov na trhu, cez mobilné zariadenia až po enterprise servre[3]. ACPI obsahuje statické tabuľky, ktoré slúžia pre operačný systém na konfiguráciu počas bootovacieho procesu. ACPI cez operačný systém pomáha kontrolovať spotrebu elektrickej energie v jednotlivých zariadeniach v systéme[4].

1.6 Skupina riešiaca UEFI hrozby

Skratka skupiny je USRT (UEFI Security Response Team).

Skupina riešiaca UEFI hrozby je zodpovedná za komunikáciu medzi bezpečnostnými výskumníkmi alebo tými, ktorí obojavili bezpečnostnú zraniteľnosť, a UEFI komunitou. USRT taktiež určuje rozsah bezpečnostnej zraniteľnosti a pomáha pri koordináci opráv[5].

Kapitola 2

UEFI špecifikácia

UEFI špecifikácia definuje rozhranie medzi operačným systémom a platformou firmwara . Rozhranie pozostáva z tabuliek obsahujúce dáta, ktoré súvisia s platformou firmwara a boot procesom. Obsahuje taktiež služby a servisné funkcie určené pre operačný systém a jeho zavádzač. Všetky tieto funkcie predstavujú štandart, ktorý opisuje moderný boot operačného systému a aplikácií pred jeho štartom[1].

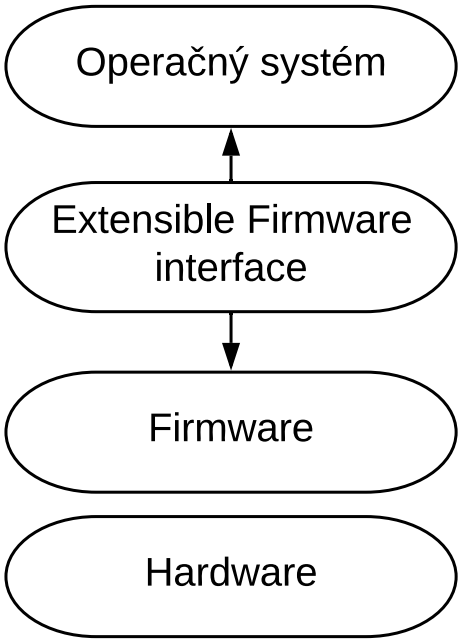
UEFI špecifikácie definujú aj rôzne protokoly na prístup k rôznym hardvérom a bootovacím zariadeniam v systéme. Medzi UEFI špecifikácie patrí aj generický model ovládača, ktorý sa vie prispôsobiť na všetky rôzne zbernice alebo zariadenia. Model ovládača je navrhnutý tak, aby podporoval spúšťanie modulárneho kódu pred bootovacím prostredím. Tieto ovladače môžu manažovať alebo kontrolovať hardverové zbernice alebo zariadenia v platforme[1].

Pod UEFI prostredím môžeme spúšťať 3 typy entít:

- Aplikačné: Niektoré príklady bežných EFI aplikácií zahrňujú EFI shell, príkazy EFI shellu, flash nástroje a diagnostické nástroje. EFI aplikácia môže volať aj inú EFI aplikáciu[1].

- Zavádzač operačného systému: Je to špeciálny typ EFI aplikácie na načítanie operačného systému. Poskytuje inicializačné rutiny, kým zavádzač nevytvorí dostatočné prostredie, aby operačný systém mohol prevziať kontrolu. Keď operačný systém prevezme kontrolu, jadro EFI uvoľní všetky nepotrebné boot služby a ovládače[1].

- Ovládače: Rozdiel medzi EFI ovládačom a EFI aplikáciou je schopnosť EFI ovládača bežať v pamäti, kým EFI ovládač nevráti žiadnu chybovú hlášku. EFI ovládač môže načítať jadro EFI firmwaru, boot manažér alebo ďalšie EFI aplikácie[1].



Obr. 2.1: EFI rozhranie medzi operačným systémom a firmware platformou

2.1 Platform Initialization

Platform Initialization (PI) je špecifikácia publikovaná skupinou UEFI Forum, ktorá opisuje vnútorné rozhranie medzi časťami firmware platformy. To umožňuje lepšiu interoperabilitu medzi komponentami firwmaru z rôznych zdrojov.[6].

PI má rovnaké základy, štruktúry a hlavné služby ako EFI. Architektúra bola postupne vytvorená z operačného systému až do reset vektoru. Reset vector je adresa alebo pointer, aby procesor vedel z ktorej adresy má vykonávať inštrukcie[7]. Jednotlivé fázy boli rozdelené nasledovne: reset vektor, prebudenie systému s inicializáciou a prístupom k operačnej pamäti RAM, inicializácia permanentnej operačnej pamäti RAM, výber bootovacieho zariadenia a beh prostredia. PI tiež podporuje aj niekoľko ďalších bootovacich módov ako tzv. „Compatibility Support Module“ (CSM). Tento modul umožňuje podporu staršieho bootovacieho procesu definovaného v roku 1980 [6].

2.2 UEFI vs Platform Initialization

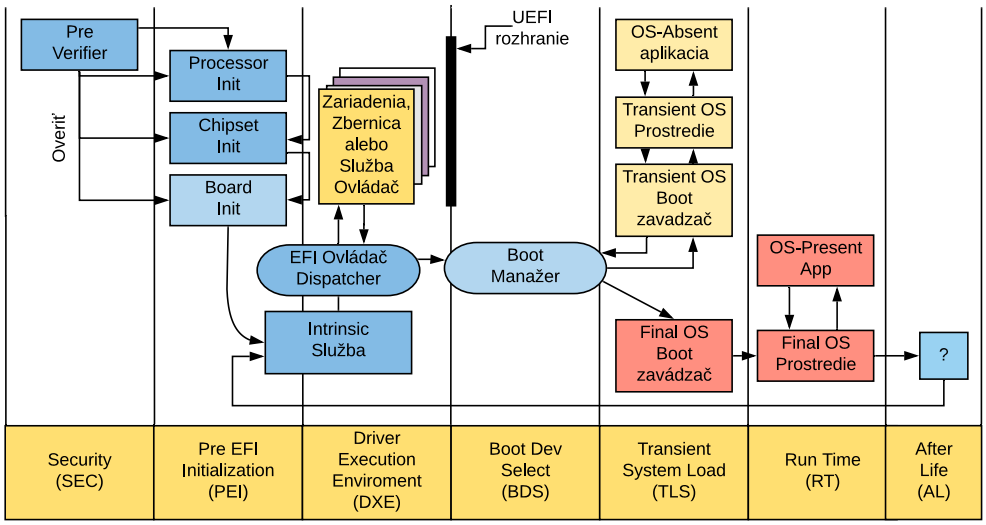
UEFI je čisto rozhranie, ktoré nepopisuje, ako je firmware platforma postavená alebo ako je UEFI zaradené do PI. UEFI je technológia, ktorá umožňuje spotrebiteľom (zavádzač operačného systému, inštalátory, ROM adaptéry z boot zariadení, pre-OS diagnostiky, utility) vytvoriť spustiteľné prostredie

pre OS. UEFI popisuje bootovanie alebo odovzdávanie kontroly nasledujúcej vrstve tzv. zavádzaču operačného systému. UEFI ponúka zaujímavé schopnosti. Môže vytvoriť limitujúce spustiteľné prostredie, v ktorom sa môžu spúšťať aplikácie namiesto načítavania operačných systémov. Vytvorenie spustiteľného prostredia nie je však hlavným cieľom UEFI[8].

PI špecifikácia je z väčšej časti nepriehľadná pre zariadenia boot pre-OS, operačné systémy a ich zavádzače, pretože pokrýva veľa softvérových aspektov, ktoré sú irelevantné pre spotrebiteľov. PI opisuje fázy od reset vectoru až do úspešného predania kontroly kompatibilného prostredia UEFI[8].

2.3 EFI Boot prechod

EFI Boot prechod je zlúčenie Platform Initilization (PI) špecifikácie a UEFI špecifikacie. Na obr. 2.2 môžeme vidieť celkový EFI Boot prechod od „SEC“ fázy až po fázu „After Life“. V tejto podkapitole si opíšeme úlohy EFI Boot fáz až na posledné tri fázy. Tieto fázy „Transient System Load“, „Run Time“, „After Life“ nie sú pre nás zaujímavé.



Obr. 2.2: EFI Boot prechod

2.3.1 SEC

Skratka SEC je „Security“. V tejto fázy nie je inicializovaná žiadna pamäť, je to začiatočný stav. Pre zásobník a haldu tu nie je dostatok pamäti.

SEC fáza je zodpovedná za prípravu na zavolanie preinicializačného prostredia (PEI) a za vytvorenie dočasného pamäťového priestoru v operačnej pamäti RAM. Preto Intel ešte vytvoril systém na vytvorenie dočasného pamäťového priestoru. Vyhne sa tak drahému inicializovaniu vlastnej pamäti

„Static Random Access Memory“ (SRAM). K vytvoreniu tohto pamäťového priestoru „cache-as-RAM“ (CAR) využil cache pamäť procesora, ktorá slúži na uloženie zásobníku a haldy. Dočasná pamäť je dostatočné veľká na to, aby sa na nej vykonala PEI fáza. SEC fáza sa spúšťa z pamäti firmwaru, využíva sa tak metóda spúšťania programu priamo z dlhodobej pamäti bez kopírovania do pamäti RAM. Táto metóda sa nazýva „execute-in-place“ (XIP)[6].

Cieľom SEC fázy je vytvoriť stav podľa UEFI PI špecifikácie. Zahrnuté je v tom spúšťanie jedného vlákna, vytvorenie volacieho zásobníku (call-stack) s minimálnou veľkosťou a snaha predať hlavnému PEI modulu (PEI core) zoznam nenulových dátových štruktúr[6].

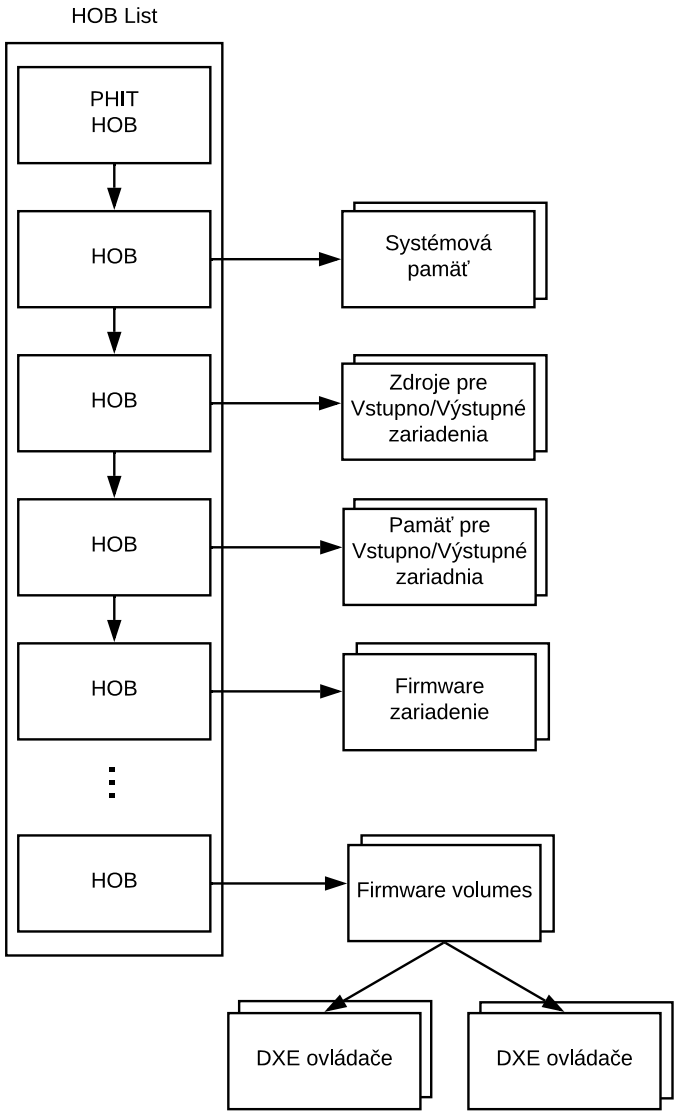
2.3.2 PEI

Pre-EFI Initialization (PEI) je fáza, v ktorej sa spúšťa časť UEFI PI infraštruktúry. PEI fáza prijme kontrolu od SEC fázy spustenej z pamäti firmwera. „PEI core“ je hlavný modul, ktorý dostáva kontrolu od SEC fázy. PEI core očakáva od SEC fázy pamäťové miesto na uloženie zásobníku, lokáciu „Boot firmware volume“ (BFV) alebo tzv. „Firmware volume“ (FV) . Vo BFV sa nachádzajú ďalšie PEI moduly s hlavným PEI core modulom. PEI moduly môžu byť dodané rôznymi výrobcami procesorov, chipset predajcami alebo výrobcami matičných dosiek. PEI modul vie predať svoje schopnosti ostatým PEI modulom pomocou PEIM-to-PEIM rozhrania (PPI). Implementačne je PEI core zmenšená verzia spustiteľného súboru (PE/COFF), v ktorom sa nachádza zafixovaná relokačná tabuľka pre spúšťanie vo firmwari [6].

PI PEI fáza má vykonať minimálnu prácu na objavenie permanentnej pamäti, ktorá je potrebná na odovzdanie kontroly nasledujúcej spúšťacej fáze. Permanentná pamäť je pamäť inicializovaná v PEI fáze, ktorá nemôže byť premiestnená na iný adresný priestor kvôli vykonávaniu ďalšej fázy. Posledná akcia PEI fázy je vyvolať PPI odkázaný na „Driver Execution Environment Initial Program Loader“ (DXE IPL). Tým DXE IPL nájde DXE core súbor nachádzajúci sa v firmware volume. Načíta a odovzdá kontrolu DXE core súboru s alokovaným priestorom v pamäti tzv. „Hand-Off Blocks“ (HOB) listom. Ak by sa hýbalo pamäťovým priestorom vytvoreným PEI fázou počas „Driver Execution Environment“ (DXE) fázy, tak operačný systém to označí ako poškodený pamäťový priestor a vyvolá chybu [6]. V piatej kapitole sa budeme podrobnejšie venovať PEI fáze a PEI core modulu.

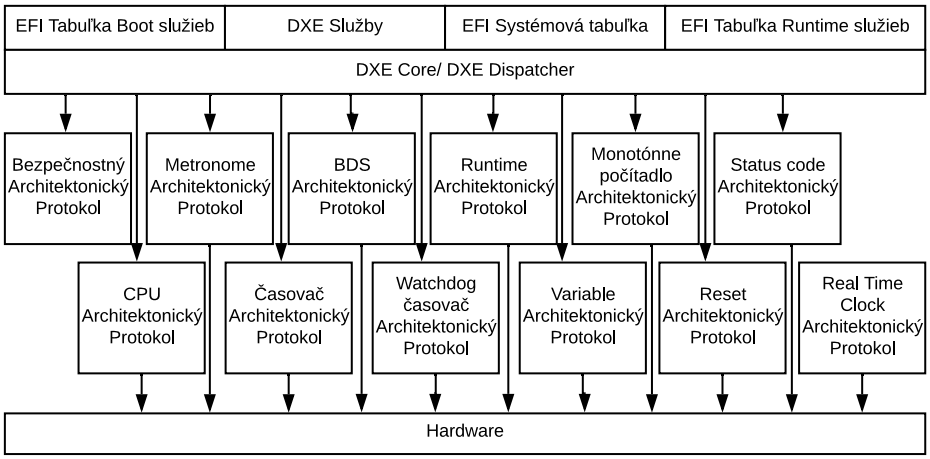
2.3.3 DXE

„Driver Execution Environment“ (DXE) je nasledujúca spustiteľná fáza, ktorá príjíma kontrolu od PEI fázy. Parametre DXE core súboru sú ukazovateľom na zmeňovaný „Hand-Off Blocks“ (HOB) list[6]. Ten je znázornený na obr. 2.3.



Obr. 2.3: List HOB štruktúr

„DXE core“ je súbor, ktorý inicializuje a poskytuje systémovú tabuľku, tzv. „UEFI System Table“, a sériu pamäťových služieb. Tento súbor bol skompilovaný tak, aby bol kompatibilný pre rôzne mikroarchitektúry. DXE core potrebuje byť detailne informovaný o konkrétnej platforme, o tzv. „interrupt managment“, udržiavaní času a o UEFI premenných. O toto sa postarajú DXE ovládače vo firmware volume. DXE ovládače sú taktiež spustiteľné súbory PE-/COFF, ale narozdiel od PEI modulov môžu byť dekomprimované a spúšťané z operačnej pamäti RAM. To je nevyhnutné kvôli ich výkonom a schopnosťiam predvádzať komplexnejšie algoritmické operácie[6].



Obr. 2.4: DXE rozhranie

Na obrázku 2.4 je opísaný vzťah medzi systémovou UEFI tabuľkou a architektonickými protokolmi DXE, ktoré poskytujú UEFI služby špecifické od danej platformy. Napríklad služba UEFI „SetVariable()“ prislúcha k architektúre „Variable Architectural Protocol“. Architektonické protokoly alebo inak tzv. „Architectural protocols“ (AP) musia vedieť spolupracovať s programovou vrstvou, ktorá povoľuje OS interagovať s hardwarovou vrstvou tzv. „Hardware abstraction layer“ (HAL) . Umožňuje to zmenu na príslušnej AP v prípade, že sa zmení daná platforma[6].

V DXE fáze sa vykonávajú operácie ako inicializácia vstupno/výstupných zberníc, napríklad tzv. „Peripheral Component interconnect“ (PCI) a vytvorenie tzv. „System Managment BIOS“ (SMBIOS) tabuľky [6].

Na rozdiel od PEI modulov, kde sa rozhrania odkazujú medzi sebou pomocou PPI pri DXE ovládačoch si vystavujú medzi sebou GUID identifikátory korenspondujúce API funkcie, rovnako ako pri UEFI protokolov. API funkcie alebo dáta sú pomenované pomocou indentifikátora GUID[6].

DXE je taktiež zodpovedný za stav stroja počas spúšťania a jeho predanie do ďalšej fázy. DXE bol rozšírený iba pre výrobcov matičnej dosky a nepovoľuje spúšťanie modulov tretích strán z inej ROM pamäte. Iba kryptografické podpisy modulov nachádzajúce sa vo fimware volume vyprodukované výrobcom matičnej dosky budú spustené alebo vystavené. Fimware volume moduly sa nachádzajú väčšinou pri aktualizácií operačného systému[6].

Posledná DXE komponenta je BDS (Boot Device Selection) spájajúca BIOS alebo spúšťanie UEFI, ktoré podporuje ROM adaptéry alebo disky z tretích strán. BDS má ako jediný poslednú možnosť uzamknúť zdroje systémovej matičnej dosky. Do toho patrí uzamknutie SMRAM alebo flash pamäte[6].

Počas DXE fázy sa registrujú aj komponenty pre operačné módy. Za operačné módy máme namysli tzv. „System Management Mode“ (SMM) a „Plat-

form Management Interrupt“ (PMI) / Machine Check Architecture (MCA) na procesorov z rodiny Itanium. Oba operačné módy sú pomocou DXE ovládača inicializované a načítané do pamäti ako tzv. „System Management RAM“ (SMRAM) alebo do iného rezervovaného pamäťového miesta pre SMM a PMI[6].

DXE poskytuje ovládačom zdieľané dátové tabuľky a ďalšie služby, ako ACPI, SMBIOS, tzv. „Itanium System Abstraction Layer“ (SAL) a tzv. „System Table“ (SST)

Prechod z BIOSu alebo z ďalšieho proprietárneho bootovacieho riešenia do UEFI a PI predstavuje veľké ťažkosti pre výrobcov. Ale po vývojovom úsilí a prechode budú UEFI a PI ponúkať ďalšie budúce technologické inovácie[6].

2.3.4 BDS

Boot Device Selection (BDS) Architektural protokol sa spúšťa počas BDS fázy. BDS Architektural protokol je nájdený počas DXE fázy a je spustený ak sa splnia nasledujúce podmienky:

- Ak všetky DXE Architekturálne protokoly sú registrované v databáze handlou.
- DXE Dispatcher nemá žiadne DXE ovládače, ktoré potrebuje načítať alebo spustiť. Tento stav nastane ak sa spracujú všetky prioritné súbory z firmware volume a všetky DXE ovládače sú spustené a načítané[8].

BDS Architectural Protokol načíta a lokalizuje rôzne aplikácie spúšťané v pre-boot prostredí. Aplikáciami myslíme napríklad OS boot zavádzač alebo rozšírené služby spustené pred finálnym načítaním operačným systémom. Medzi rozšírené služby v pre-boot prostredí patria inštalácia konfigurácie, rozšírené diagnostiky, flash update, „Driver Execution Environment“ (OEM) služby alebo samotný boot kód OS[8].

Dodávatelia ako „Independent BIOS vendor“ (IBV), „Original equipment manufacturer“ (OEM) a „Independent Sofware vendor“ (ISV) si môžu vybrať, či BDS implementujú podľa základnej referencie alebo či budú implementovať všetko odznovu [8].

Kapitola 3

Základná UEFI architektúra

Unified Extensible Firmware (UEFI) je rozhranie opisujúce programovateľné rozhranie pre platformu. Za platformu myslíme matičnú dosku, chipset, procesor (CPU) a ďalšie komponenty. UEFI povoľuje agentov pred spustením operačného systému alebo tzv. „pre-OS“. Pre-OS agenti môžu byť OS zavádzače, diagnostické nástroje a ďalšie aplikácie potrebné pre systém. Spolupracujú s ďalšími aplikáciami vrátane UEFI ovládačov. UEFI teda reprezentuje rozhranie, ktoré interaguje s ovládačmi a aplikáciami. V tejto kapitole opíšeme niektoré architektonické aspekty rozhrania. Medzi architektonické aspekty patria objekty a rozhranie, opísané v UEFI špecifikáci[8].

V nasledujúcich riadkoch si vysvetlíme kľúčové UEFI objekty. Tie môžeme rozdeliť do nasledujúcich bodov:

- Objekty spravované UEFI firmwarom – objekty slúžia na manažovanie stavu systému vrátanie I/O zariadení, pamäti a eventov.
- UEFI System table – primárna dátová štruktúra s informačnými dátami a volanými funkciami pre systémové rozhranie.
- Databáza handlov a protokoly – na určenie, aké rozhrania sú už registrované.
- UEFI image – označenie spustiteľného súboru v UEFI spustiteľnom prostredi.
- Eventy – signály pre software na vykonávanie definovanej aktivity [8].

Niekoľko rôznych typov objektov môžeme spravovať cez UEFI služby. Niektoré UEFI ovládače môžu pristupovať k tzv. „enviroment variables“. Niekedy si UEFI ovládače vyžadujú použiť monotónne počítadlo, časovač alebo realtime hodiny[8].

3.1 UEFI System Table

UEFI System Table je najdôležitejšia dátová štruktúra v UEFI. Ukazovateľ na System Table je odovzdaný každej zavolanej UEFI aplikácií alebo UEFI ovládaču. Pomocou ukazovateľa na System Table je UEFI aplikácia alebo UEFI ovládač schopný mať prístup k konfiguračným dátam a bohatej kolekcii UEFI služieb. UEFI služby obsahujú nasledujúce:

- UEFI Boot služby
- UEFI Runtime služby
- Služby protokolov

UEFI Boot služby a UEFI Runtime služby sú tabuľky prístupné cez tzv. „UEFI Boot Services Table“ a „UEFI Runtime Services Table“. Obidve tabuľky sú položky v dátovej štruktúre „UEFI System Table“. V UEFI System Table je počet a typ služieb fixovaný inak pre každú verziu UEFI špecifikácie. UEFI Boot služba a UEFI Runtime služba sú definované v UEFI 2.8 špecifikáci[9].

„Protocol services“ je skupina súvisiacich funkcií a dát pomenovaných tzv. „Globally Unique Identifier“ (GUID), 16 bytov unikátny identifikátor. Protocol services poskytujú softwarovú abstrakciu pre sieťové, diskové a konzolové zariadenia. Môžu byť tiež použité ako ďalšie služby na rozšírenie ďalších služieb dostupné pre danú platformu. Protokoly sú mechanizmy na rozšírenie funkcionality UEFI firmwaru[9].

3.2 Databáza handlou

Databáza handlou obsahuje skupiny protokolov a objektov. Protokol je dátová štruktúra označená identifikátorom GUID. Dátová štruktúra môže obsahovať dátové polia, služby, oboje, alebo nemusí obsahovať nič[8].

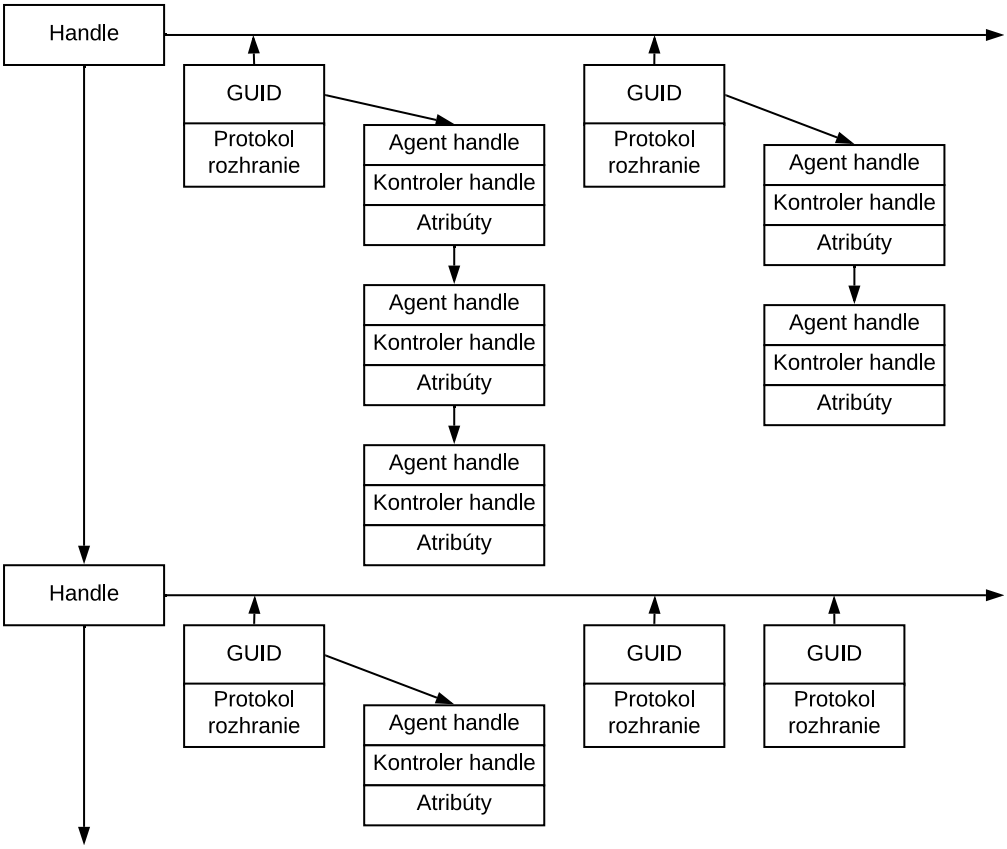
Na začiatku je databáza handlou prázdna. Počas inicializácie platformy UEFI ovládač a UEFI aplikácie vytvoria handlu, k handle pripoja jeden alebo viac protokolov. Informácie v databáze handlou sú globálne a prístupné pre všetky spustiteľné UEFI image[8].

Databáza handlou je list UEFI handlou a zároveň aj centrálne úložisko pre objekty, ktoré má na starosti UEFI firmware. Každý UEFI handle je identifikovaný unikátnym handle číslom spravovaným systémovým firmwarom. Handle číslo môžeme chápať ako databázový kľuč k vstupu do databázy hadnlou. Každý vstup do databázy hadnlou je kolekcia jedného alebo viacerých protokolov. Typy protokolov pomenovaných pomocou GUID identifikátora, pripojenych k UEFI handlu určuje typ handla [9].

UEFI handla môže reprezentovať komponenty ako:

- Spustiteľný súbor ako UEFI ovládač alebo UEFI aplikácia

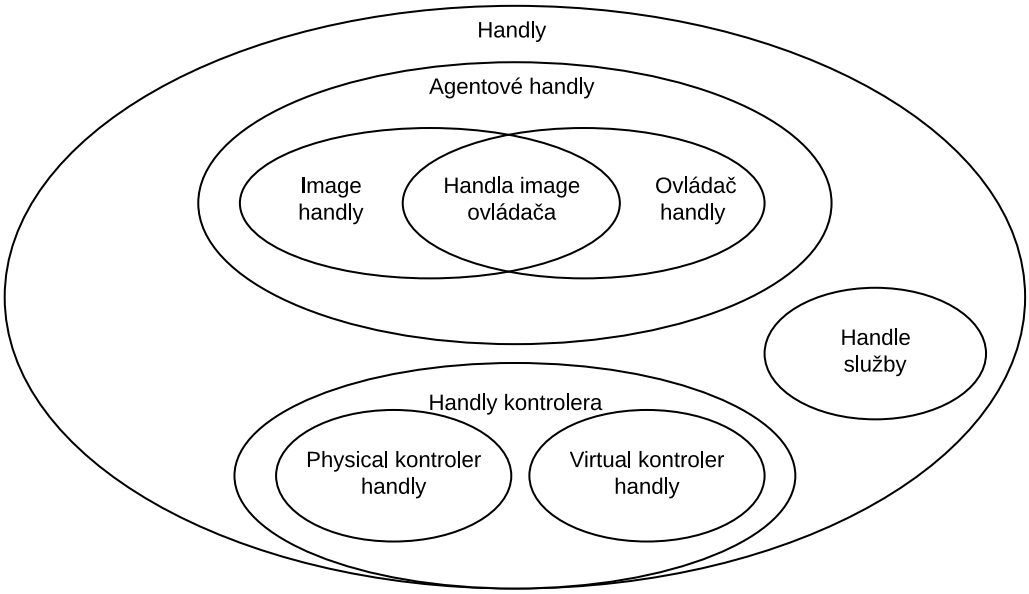
- Zariadenia ako sieťové karty a disková partícia
- UEFI služby prístupné k ovládačom ako EFI Decompress a EBC (EFI Byte Code) Interpreter



Obr. 3.1: Databáza handlou

Na obr. 3.1 vidíme časť databázy handlou a protokolov. Ku každému protokolu prislúcha aj list objektov. Databáza handlou používa tento list UEFI handlou, aby vedela určiť, ktorí agenti využívojú ktoré protokoly. Je to veľmi dôležitá informácia pre prácu s UEFI ovládačmi. To im umožňuje byť bezpečne načítané, naštartované, zastavené a odstránené bez žiadnych konfliktov so zdrojmi[9].

Na obr. 3.2 sú znázornené typy handlou, ktoré sa môžu nachádzať v databáze handlou. Zároveň je tam znázornený vzťah medzi rôznymi typmi handlou. Existuje len jedna databáza handlou, v ktorej sú vytvorené všetky handle. Služby pracujúce s databázou handlou nerozoznávajú typy handlou. Typy handlou sú rozoznávané podľa toho aké typy protokolov k nim patria[9].



Obr. 3.2: Typy handlou

3.3 Protokoly

UEFI je z veľkej časti postavený na protokoloch. Protokoly slúžia na to, aby dva odlišné moduly alebo ovládače vedeli medzi sebou komunikovať.

Ovládače vytvárajú protokoly pozostávajúce z dvoch častí. Telo protokolu je štruktúra v štýle jazyku C a označujeme ho ako rozhranie. Rozhranie obsahuje príslušné ukazovatele na funkciu a dátové štruktúry[8].

Ku každému protokolu prislúcha daný GUID. GUID slúži ako meno pre protokol. Treba poznamenať, že GUID nepatrí do príslušnej dátovej štruktúry.

```
#define EFI_COM_NAME_PRO_GUID \
{0x6a7a5cff, 0xe8d9, 0x4f70,
{0xba, 0xda, 0x75, 0xab, 0x30, 0x25, 0xce, 0x14}}

typedef struct _EFI_COM_NAME_PRO_GUID EFI_COM_NAME_PRO_GUID;

struct _EFI_COM_NAME_PRO_GUID {
EFI_COM_NAME_GET_DRIVER_NAME      GetDriverName;
EFI_COM_NAME_GET_CONTROLLER_NAME  GetControllerName;
CHAR8 *SupportedLanguages;
};
```

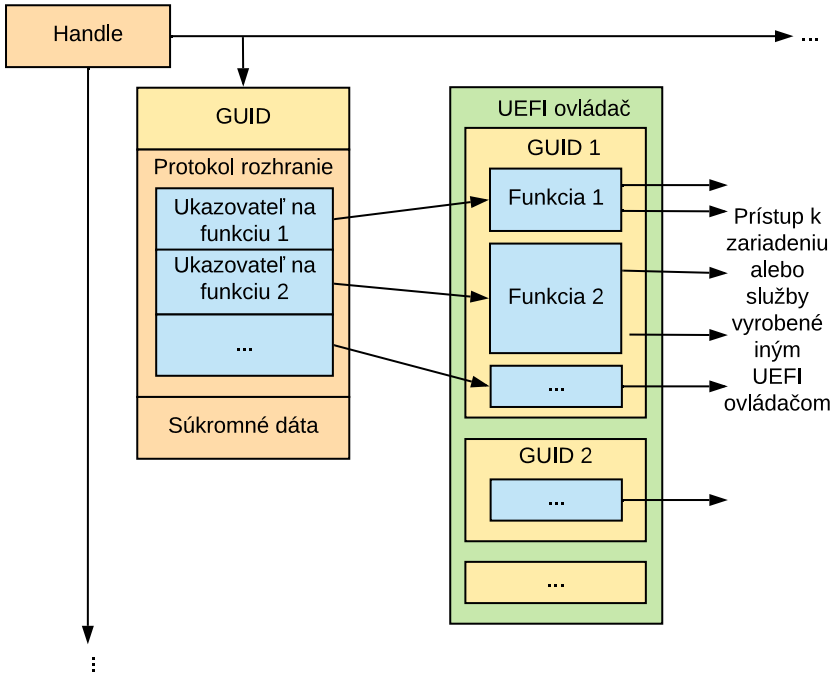
Listing 3.1: Príklad Component Named Protocol

V úseku 3.1 je príklad jedného protokolu „Component Named Protocol“. Môžeme si všimnúť počet funkcií a dátových elementov v štruktúre. V našom prípade dve funkcie a jedna dátová premena[9].

Protokoly sa zhromažďujú do jednej databázy. Databáza nie je navrhnutá ako „flat“, umožňuje protokolom zgrupovať sa. Každá grupa je označené ako handle. Handle je dátový typ odkazujúci na danú grupu. Táto databáza je známa tiež ako handle databáza. Handle sú alokované dynamicky. V danom systéme protokoly nemusia byť unikátne, ale musia mať unikátnu handlu. Inak povedané k jednej handle nemôže prislúchať viacero protokolov majúci rovnaký identifikátor GUID[9].

Na obr. 3.3 je znázornený jeden protokol, vytvorený UEFI ovládačom, prislúchajúci k handli v databáze handlou. Protokol pozostáva z identifikátora GUID a z rozhrania protokolu[9].

UEFI ovládač poskytujúci rozhranie protokolu udržiava aj privátny dátový element v štruktúre. Protokol rozhranie obsahuje ukazovatele na funkcie protokolu. Protokol funkcie sa nachádzajú v UEFI ovládači. UEFI ovládač môže obsahovať jeden alebo viac protokolov, to závisí na zložitosti ovládača[9].



Obr. 3.3: Konštrukcia protokolu

3.4 UEFI images

UEFI image môžu byť rozličného typu, ale všetky UEFI image obsahujú „Portable Executable/The Common Object File Format“ (PE/COFF) hlavičku. PE/COFF hlavička definuje spustiteľný formát spustiteľného kódu. PE/COFF image je známa hlavička definovaná v „Microsoft Portable Executable and

Common Object File Format“ špecifikáci. Spustiteľný kód môže mať formát pre IA32, X64, „Intel processor family“ (IPF), „Efi Byte code“ (EBC). Hlavička súboru definuje typ procesoru a typ imagu[8]. UEFI image je rozdelený do troch typov:

- UEFI aplikácia
- UEFI boot služby ovladače
- UEFI bežiace ovladače

UEFI image je načítaný a relokovaný do pamäte pomocou boot služby `LoadImage()`. Existuje niekoľko pamäťových miest kde sa UEFI image môžu nachádzať:

- Rozšírené ROM pamäte na „Peripheral Component Interconnect“ (PCI) kartách
- Systémové ROM pamäte alebo systémové flash pamäte
- Medialne zariadenie ako hardisk, floppy, CD-ROM, DVD alebo USB
- LAN servery

UEFI image nie sú skompilované ani linkované na špecifickej adrese v pamäti. Miesto toho obsahujú relokačné tabuľky. Umožňuje to UEFI imagu nachádzať sa kdekoľvek v systémovej pamäti[9]. Boot služba `LoadImage()` urobí nasledovné:

- Alokuje pamäť pre UEFI image, ktorý sa má načítať
- Automaticky aplikuje upravenú relokačnú tabuľku na image
- Vytvára nový image handle v databáze handlou, ktorý inštaluje aj inštanciu `EFI_LOADED_IMAGE_PROTOCOL`

Inštancia `EFI_LOADED_IMAGE_PROTOCOL` obsahuje informáciu o načítanom UEFI imagu. Kvôli tejto informácií publikovanej v databáze handlou je UEFI image dostupný pre všetky UEFI komponenty[9].

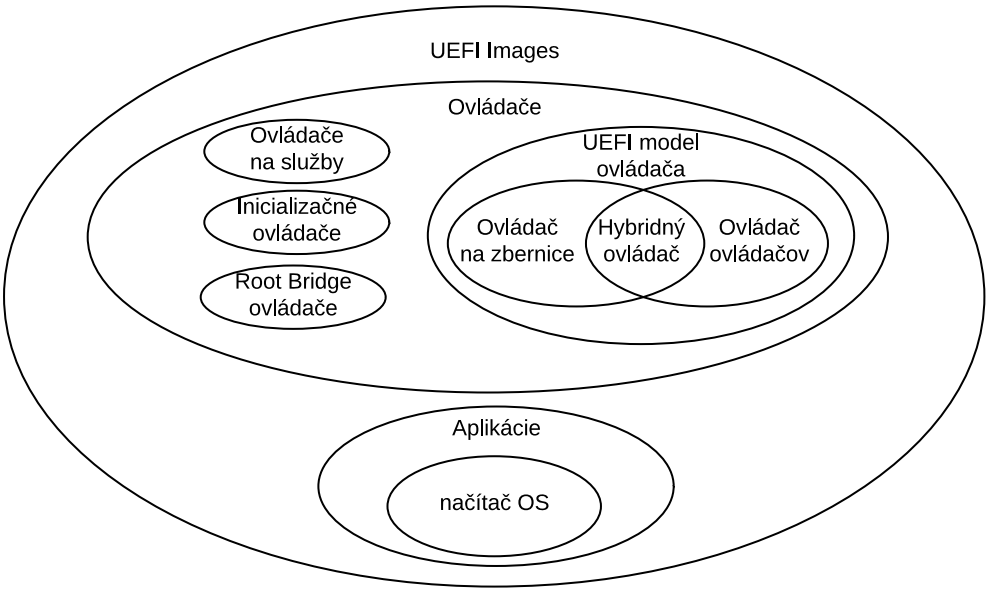
Po načítaní UEFI imagu pomocou `LoadImage()` môže byť image spustený zavolaním funkcie `StartImage()`. V hlavičke UEFI imagu sa nachádza adresa entry pointu aby funkcia `StartImage()` vedela na akej adrese má zavolať main funkciu načítaného UEFI imagu. Argumenty zavolanej main funkcie sú vždy:

- Handle image na spúšťaný UEFI image
- Ukazovateľ na „UEFI System table“

Handle image a ukazovateľ na UEFI System table umožňuje UEFI imagu:

- Prístup k všetkým UEFI službám, ktoré sú prístupné v UEFI platforme.
- Získať informácie, z kadiaľ bol UEFI image načítaný a kde sa UEFI image nachádza v pamäti.

Na obr. 3.4 znázorňujeme rôzne typy UEFI imagu a vzťahy medzi nimi[9]. Rôzne typy UEFI imagu sú opísané v dokumentacií „EDK II Driver Writer's Guide for UEFI 2.3.1"[9]



Obr. 3.4: Typy UEFI imagov

3.5 Eventy

Eventy sú ďalšie typy objektov manažované cez UEFI službu. Eventy poskytujú synchrónne a asynchrónne spätné volanie ak nastane konkrétna udalosť. Tieto eventy môžu byť vytvorené alebo odstránené a taktiež môžu byť v signalizovanom stave alebo iba v čakajúcom stave[9]. UEFI image môžu vykonať nasledujúce operácie:

- Vytvoriť event
- Odstrániť event
- Informovať sa, či je event v signalizovanom stave
- Čakať, aby event bol v signalizovanom stave[9]

Rôzne typy UEFI eventov sú opísané v dokumentacií „EDK II Driver Writer's Guide for UEFI 2.3.1"[9]

Kapitola 4

Získanie imagu UEFI firmwara z SPI flash pamäte

V tejto kapitole si opíšeme rôzne prístupy k získaniu imagu UEFI firmwara z SPI flash pamäte, samotný zisk imagu UEFI firmwara, metódu spracovania, parsovania a získania užitočných informácií zo získaného imagu.

Prvý krok je získať UEFI firmware image z SPI flash pamäte. V našom prípade, kedže máme fyzický prístup k hardwaru, existujú dva prístupy k získaniu UEFI firmware imagu:

- Softwarový prístup
- Hardwarový prístup[10]

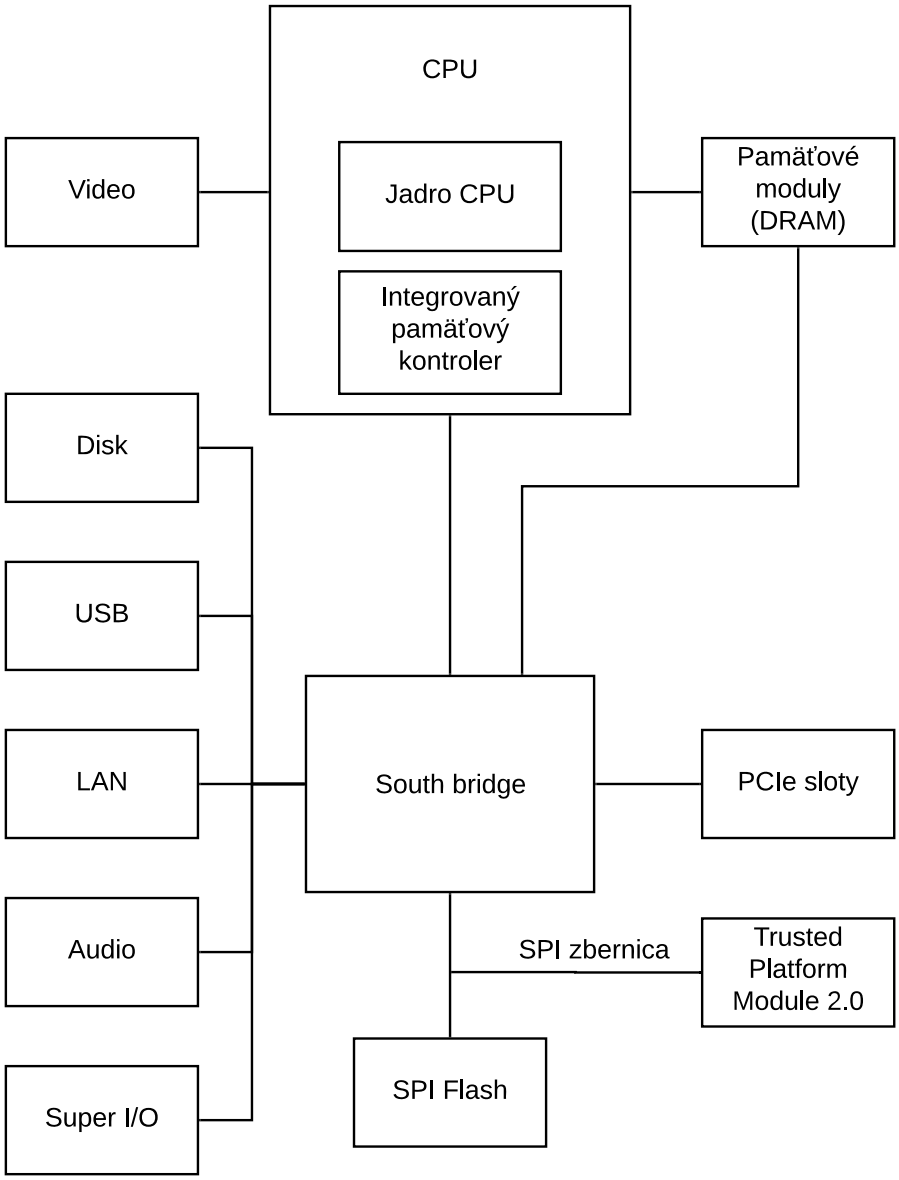
Na obr. 4.1 môžeme vidieť, kde sa UEFI firmware nachádza na moderných platformách. Obr. 4.1 demoštruje architektúru typického počítačového chipsetu.

Na obr. 4.1 sa nachádzajú dva hlavné komponenty v chipsete:

- Procesor (CPU)
- South bridge

South bridge poskytuje pripojenie medzi kontrolermi periferných zariadení a procesorom. V moderných systémoch založených na architektúre Intel x86 a architektúre Intel x64 sa UEFI firmware nachádza v „Serial Peripheral Interface“ (SPI) flash pamäti. SPI flash je fyzicky pripojený na South bridge, ako je vidieť na obr. 4.1 [10].

Na získanie firmware imagu uloženom v SPI flash pamäti potrebujeme vedieť prečítať obsah SPI flash pamäte. Ako je uvedené vyšie, získanie firmware imagu môžeme dosiahnuť dvoma spôsobmi, softwarovým alebo hardwarovým. Na získanie firmwaru softwarovým spôsobom musíme vedieť komunikovať so



Obr. 4.1: Diagram moderného Intel chipsetu

SPI kontrolerom pomocou softwaru bežiacom na procesore. Na získanie firmware imagu hardwarovým spôsobom musíme fyzicky pripojiť špecialne zariadenie na SPI flash pamäť nachádzajúcej sa v základnej doske. To špeciálne zariadenie sa nazýva SPI programátor, pomocou ktorého dokážeme prečítať obsah SPI flash pamäte[10].

4.1 Softwérové získanie firmwaru

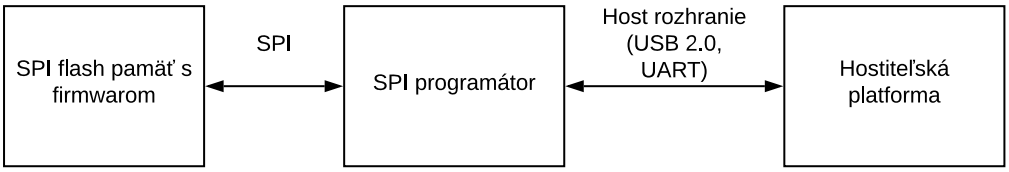
Na získanie UEFI firmwaru z cieľového hardwaru pomocou softwaru potrebujeme prečítať obsah SPI flash pamäte z operačného systému. Obsah SPI flash pamäte získame cez registre v „PCI configuration space“.To je blok registrov, ktoré špecifikujú konfiguráciu zariadenia napojeného na zbernicu „Peripheral Component Interconnect“ (PCI). Tieto registre sú namapované v hlavnej pamäti a môžeme do nich zapisovať alebo z nich čítať pomocou regularných operacií ako čítanie a zápis do pamäti[10].

Treba poznamenať, že lokácie SPI registrov na komunikáciu s SPI kontrolerom sú rozdielne na základe typu chipsetu. Ako nájdem lokáciu SPI v pamäti sa môžeme dozvedieť napríklad v literatúre „Rootkit Bootkits-Reversing Modern Malware and Next Generations Threats“[10]. Lokácia pamäťového miesta, kde sa nachádzajú registre v „PCI configuration space“, sú namapované v privilegovanom pamäťovom priestore. Prístup k privilegovanému pamäťovému priestoru môžeme dosiahnuť len pomocou programu bežiacom v tzv. kernel móde[10].

Napríklad pomocou nástroja „Chipsec“ bežiaceho v kernel móde docielime prístup k „PCI configuration space“, ktorý umožňuje prečítať UEFI firmware z SPI flash pamäte v základnej doske.

4.2 Hardwarové získanie firmwaru

Na získanie UEFI firmwaru z cieľového hardwaru pomocou hardwaru sa musí fyzicky pripojiť zariadenie na SPI flash pamäť a priamo prečítať jej obsah. Je to dôveryhodnejší prístup ako softwarový prístup. S hardwarovým prístupom dokážeme získať aj ostatné firmwari uložené v SPI flash ako „Intel Management Engine“ (ME) a „Gigabit Ethernet“ (GBE), ktoré nemusia byť prístupné pri softwarom prístupe[11].



Obr. 4.2: Nastavenie, ako získať image z SPI flash pamäte

Na prečítanie obsahu zo SPI flash pamäti podľa obr. 4.2 potrebujeme SPI programátor, ktorý fyzicky pripojíme na SPI flash pamäť. SPI programátor treba pripojiť na hostiteľskú platformu cez USB alebo „Universal asynchronous receiver-transmitter“ (UART) rozhranie, ktoré obdrží UEFI firmware image. V SPI programátorovi beží konkrétny software na prečítanie dát z SPI flash pamäti a na prevod prečítaných dát do hostiteľskej platformy alebo

do uživateľského počítača. Konkrétny software na prečítanie dát môže byť buď poskytnutý so SPI programátorom alebo voľne dostupný software ako, napríklad „Flashrom“[10].

4.3 Získanie UEFI firmwaru imagu

UEFI firmware image sme získali softwarovo pomocou programu „Chipsec“. „Chipsec“ je framework na analýzu bezpečnosti na úrovni platformy ako hardware zariadení, systémový firmware, nízkoúrovňové ochranné mechanizmy a konfigurácie rôznych komponentov platformy.

Chipsec software obsahuje set modulov vrátane jednoduchých testov na ochranu hardwaru, testy na správnu konfiguráciu, testy na chyby vo firmwari a platform komponenty, bezpečnostné hodnotenie a fuzzing nástroje pre rôzne zariadenia platforiem a rozhrania, nástroje na získanie firmwaru [12].

UEFI firmware image sme získali zo stroja s konfiguráciami:

- Značka stroja: DELL Latitude E6420
- OS: Microsoft Windows 10
- CPU: Intel Core i7 2620M @ 2.70GHz Sandy Bridge
- RAM: 4 GB Single Channel DDR3 @ 665 MHz
- SSD disk: 180 GB

Na obr. 4.3 vidíme, ako cez príkazový riadok spúšťame skript s parametrami `python chipsec_util.py spi dump rom.bin`. Spustením skriptu získame celý image zo SPI flash pamäti, kde sa získaný image zapíše do súboru `rom.bin`. Po získaní imagu potrebujeme program, ktorý dokáže parsovať UEFI image. V našom prípade sme si na parsovanie UEFI imagu vybrali program „UEFITool“.

UEFITool je cross-platform C++/Qt program na parsovanie, vyťahovanie a modifikovanie UEFI a BIOS firmware imagov. Podporuje aj parsovanie BIOS imagu začínajúc od flash descriptoru alebo od binárnych súborov obsahujúcich UEFI firmware volumes[13].

Flash descriptor je dátová štruktúra naprogramovaná v SPI flash čipu na všetkých Intel platformách. Obsahuje informácie ako napr. alokované miesta pre každý region v flash image, čítacie a zapisovacie práva pre každý region, rezervované miesto pre vendora na špecifické dáta, konfigurácia chipset parametrov a viac[11].

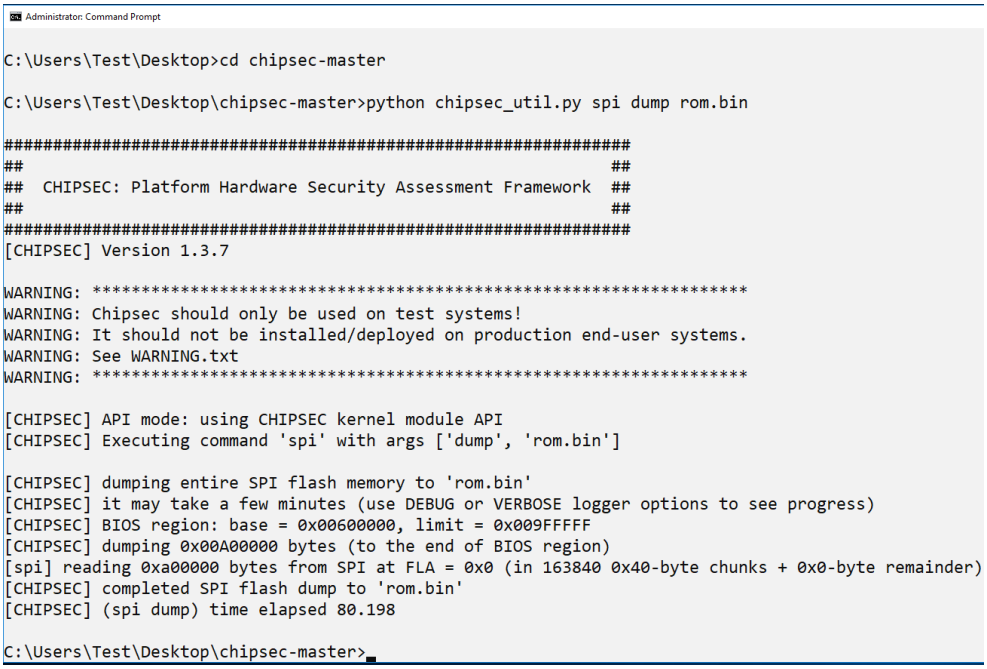
Na obr. 4.4 môžeme vidieť, ako sme získaný image zo SPI flash pamäti otvorili pomocou programu na parsovanie UEFITool. SPI flash image je rozdelený do 5 regiónov:

- Descriptor

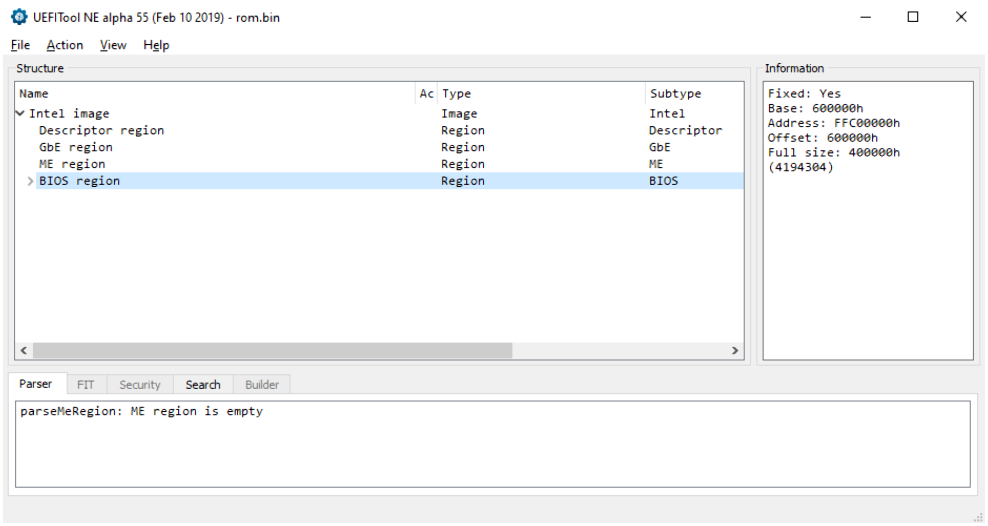
- BIOS (Basic Input Basic Output)
- ME (Intel Management Engine)
- GbE (Gigabit Ethernet)
- PDR (Platform Data Region) [11]

V BIOS regione sa nachádza UEFI firmware. Po rozliknutí BIOS regionu vidíme rozdelenie do firmware volumes ako na obr. 4.5.

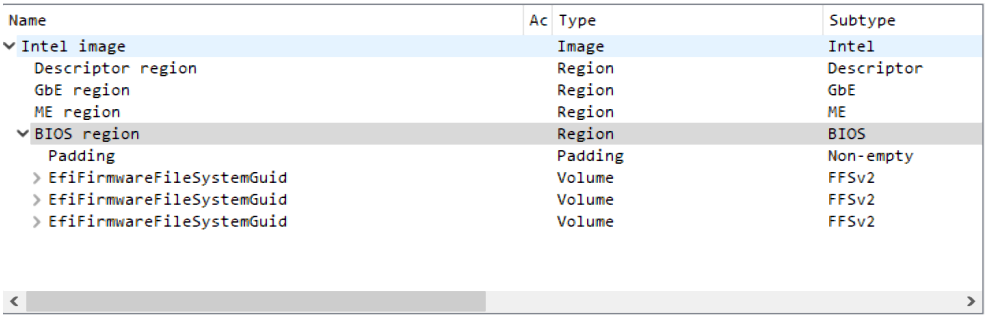
Vo firmware volumes sa nachádzajú dva hlavné moduly, a to „PEI core“ modul (obr. 4.6) a „DXE core“ modul (obr. 4.7). Tieto dva moduly sme extrahovali a následne zanalyzovali ich funkčnosť. V nasledujúcich kapitolách nájdeme kompletnú analýzu modulov PEI core a DXE core.



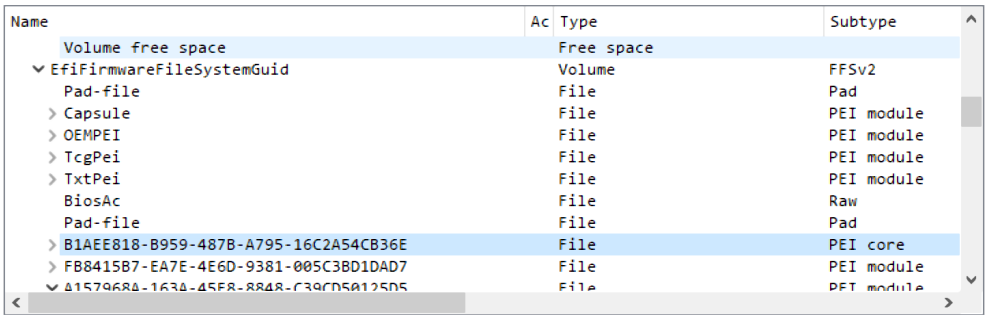
Obr. 4.3: Získanie UEFI imagu



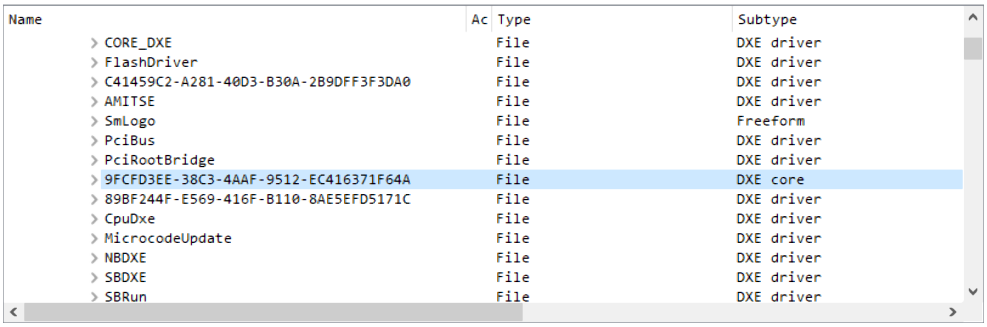
Obr. 4.4: UEFITool



Obr. 4.5: Znázornený BIOS region v UEFITool nástroji



Obr. 4.6: PEI core modul



Obr. 4.7: DXE core modul

Kapitola 5

Pre-EFI inicializácia PEI

Pre-EFI Inicializačná (PEI) fáza má dve primárne funkcie v boot UEFI spúšťaní:

- je zodpovedná za detekciu a zotavenie sa z poškodenia v úložnom priestore vo firmwari a poskytuje dôvod reštartu tzv. boot modu
- je zodpovedná za poskytnutie permanentného pamäťového miesta pre nasledujúcu DXE fázu

Pre-EFI inicializačná fáza poskytuje štandardizovanú metódu na načítanie a zavedenie inicializačných rutín na konfiguráciu procesora, chipsetu a systémovej matičnej dosky.

PEI fázu načíta tzv. Security (SEC) fáza. Primárny účel kódu operujúci v tejto fáze je inicializovať dostatočné prostredie pre DXE moduly. PEI fáza je tiež zodpovedná za určenie boot systém módu, inicializáciu a opis pamäťového miesta RAM a firmware volumes, ktoré obsahuje DXE core modul a architektonické DXE protokoly [8].

V nasledujúcej podkapitole sa budeme venovať terminológií použitej v tejto kapitole.

5.1 Terminológia

Permanetné pamäťové miesto

Permanentná pamäť je pamäť inicializovaná v PEI fáze, ktorá nemôže byť premiestnená na iný adresný priestor kvôli vykonávaniu nasledujúcej EFI boot fázy[14]. Permanentná pamäť sa inicializovala v systémovej pamäti RAM. HOB je štruktúra opisujúca permanentné pamäťové miesto pre ostatné PEI moduly a pre ovládače v DXE fáze[15].

Dočasné pamäťové miesto

PEI core požaduje od SEC fázy inicializovať v pamäti RAM dostatočné veľké pamäťové miesto pre PEI fázu, ktoré sa využije ako dátové úložisko

predtým ako sa inicializuje celá pamäť RAM. Toto miesto bude dočasné použité, predtým ako sa inicializuje permanentné pamäťové miesto v operačnej pamäti RAM[8].

Firmware Volume

Súborový systém UEFI firmware je rozdelený do niekoľkých volumes, súborov a sekcií. Firmware volume (FV) môžeme chápať ako vonkajší obal súborového systému firmwara, alebo ako partíciu na disku. Vo FV sa môže nachádzať množina firmware súborov a vo firmware súbore sa nachádzajú sekcie[16].

Boot Firmware Volume

V Boot Firmware Volume (BFV) sa nachádza PEI core modul a ostatné PEI moduly. Obsah v BFV má formát podľa súborového UEFI PI flash systému. PEI modul môže informovať PEI core ohľadne lokácií ostatných FV v systéme, čo umožňuje PEI coru nájsť ostatné PEI moduly vo FV. PEI core a PEI moduly sú označené unikátnym ID v súborovom UEFI PI flash systéme. Súborový UEFI PI flash systém poskytuje aj jeho obnovu ak nastane chyba typu zastavenie systému v nejakom bode alebo chyby pri zlej aktualizáci PEI modulov[8].

PEIM-to-PEIM rozhranie

PEI moduly komunikujú medzi sebou pomocou rozhrania PEIM-to-PEIM (PPI). PEI modul (PEIM) obsahuje štruktúru PEIM deskriptor, ktorý slúži na exportovanie rôznych služieb a dát. PEI deskriptor obsahuje flagy, ukazovateľa na GUID a ukazovateľa na dáta. Dáta môžu obsahovať list ukozovateľov na funkcie alebo na dáta. Ukazovateľ na funkciu zvyčajne odkazuje na PEIM-to-PEIM rozhranie cez ktoré PEI modul môže zavolať službu z iného PEI modulu. PEI modul lokalizuje PPI rozhranie pomocou funkcie `LocatePpi()`. Na exportovanie služby použije PEI modul funkcie `InstallPpi()` alebo `ReinstallPpi()`[17].

PPI databáza

PPI databáza je databáza na ukladanie dát identifikujúcich jedného alebo viacej PPI rozhraní [17].

Dependency expression

„Dependency expression“ (DEPEX) reprezentuje zakódovanú binárnu dátovú štruktúru, ktorá sa používa na určenie, aké PPI rozhrania treba mať načítané v pamäti. Ak sú všetky PPI rozhrania načítané v pamäti, PEI modul sa môže spustiť [6].

Hand-Off Block

„Hand-Off Block“ (HOB) je mechanizmus na odovzdanie systémových informacií z PEI fázy do DXE fázy v UEFI PI architektúre. HOB je jednoduchá dátová štruktúra v pamäti obsahujúca hlavičku a dátovú sekciu. V hlavičke sa nachádzajú informácie ako celková veľkost HOB štruktúry a formát dátovej sekcie. Pre všetky HOB štrutkúry je hlavička rovnaká. HOB sú sekvenčné alokované v pamäti a budú prístupné aj pre ostatné PEI moduly ak sa inicializovala permanentná pamäť. Postupnosť štruktúr v pamäti sa nazýva „HOB

List“. Prvý HOB v HOB liste musí byť štrutkúra tzv. „Phase Hand-off Information Table“ (PHIT) HOB, ktorá opisuje fyzickú pamäť používanú PEI fázou a boot mod objavený počas PEI fázy [8].

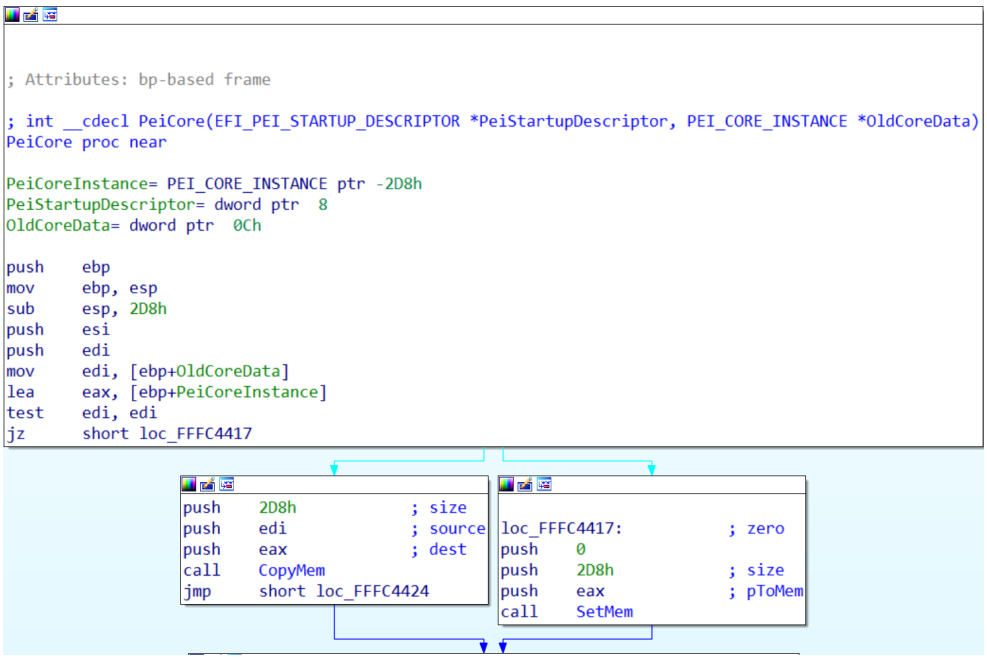
5.2 Analýza PEI Core modulu pomocou reverzného inžinierstva

V nasledujúcej podkapitole sa budeme venovať funkčnosti spustiteľného kódu hlavného PEI core modulu, ktorý bol získaný z SPI Flash pamäte. Pomocou analýzy spustiteľného kódu pomocou reverzného inžinierstva a čítanie existujúcih dokumentáci sme popísali jej funkčnosť.

Na začiatku PEI core musí prijať ukazateľa na FV a ukazateľa na zásobník. Pomocou integrovanej read-only FV a súborového systému je PEI core schopný vyhľadať všetky ostatné PEI moduly. PEIM je vlastne „execute-in-place“ (XIP) spustiteľný súbor vo FV. PEIM je uložený ako „firmware súbor“, ktorý opisuje typ súboru v našom prípade „PEIM“. Firmware súbor okrem spustiteľného PEI súboru obsahuje v súborovom systéme aj DEPEX sekciu[6].

Na obr. 5.1 môžeme vidieť začiatok hlavnej funkcie PEI core modulu. Vstupné argumenty deklarované štruktúrami sú:

```
EFI_PEI_STARTUP_DESCRIPTOR *PeiStartupDescriptor
PEI_CORE_INSTANCE *OldCoreData
```



Obr. 5.1: Začiatok funkcie PeiCore

V argumente `PeiStartupDescriptor`, ktorý bol predaný zo SEC fázy, sú informácie o:

- adrese „Boot Firmware volume“ (BFV) vo firmwari
- veľkosti inicializovanej „Cache-as-RAM“ (CAR) cache pamäti
- obsahuje pole PPI služieb poskytnuté SEC fázou

```
loc_FFFC4424:
add     esp, 0Ch
lea     eax, [ebp+PeiCoreInstance.PS]
push    0
push    eax
mov     [ebp+PeiCoreInstance.Signature], 'CieP'
mov     [ebp+PeiCoreInstance.PS], offset static_mPS ; "PEI SERV"
call    SetPeiServiceTablePToRegMM7
mov     esi, [ebp+PeiStartupDescriptor]
push    edi             ; OldCoreData
lea     eax, [ebp+PeiCoreInstance.PS]
push    esi             ; PeiStartupDescriptor
push    eax             ; PrivateData
call    InitHobMem
lea     eax, [ebp+PeiCoreInstance.PS]
push    edi             ; OldCoreData
push    eax             ; PeiServices
call    InitPpiServices
lea     eax, [ebp+PeiCoreInstance.PS]
push    edi             ; OldCoreData
push    eax             ; PeiServices
call    InitSecService
push    esi             ; PeiStartupDescriptor
lea     eax, [ebp+PeiCoreInstance.PS]
push    edi             ; OldCoreData
push    eax             ; PeiServices
call    InitDispatcherData
add     esp, 30h
test    edi, edi
jz      short loc_FFFC4490
```

Obr. 5.2: Päť inicializačných funkcií

`OldCoreData` je argument deklarovaný štruktúrou `PEI_CORE_INSTANCE`, kde sa uchovávajú všetky potrebné štruktúry ako:

- `EFI_PEI_SERVICE`
- `PEI_PPI_DATABASE`

- `PEI_CORE_DISPATCH_DATA`
- `EFI_PEI_HOB_POINTERS`
- `EFI_PHYSICAL_ADDRESS`
- `PEI_SECURITY_PPI`
- `EFI_PEI_SERVICES`

Uchováva aj ukazovateľ na zásobník, veľkosť zásobníka a ukazovateľ na cache pamäť „Cache-as-RAM“ (CAR). Hodnota argumentu `OldCoreData` pri prvom spúšťaní funkcie `PeiCore` je nulová. Nasleduje podmienka zachytená na obr. 5.1, ktorá zisťuje hodnotu argumentu `OldCoreData`. Ak je hodnota argumentu
`OldCoreData` nulová, tak sa inicializuje lokálna premenná `PeiCoreInstance`. V tomto prípade je lokálna premenná `PeiCoreInstace` uložená v dočasnom pamäťovom mieste. V prípade nenulovej hodnoty `OldCoreData` sa dáta prekopírujú do lokálnej premennej `PeiCoreInstance`. Po inicializácií premennej `PeiCoreInstance` nasledujú inicializačné rutiny, ako vidíme na obr. 5.2, ktoré sú opisané nasledovne.

Na obr. 5.2 môžeme vidieť 5 funkcií:

- `SetPeiServiceTablePToRegMM7`

  Argumenty:

  ```
  PEI_CORE_INSTANCE PeiCoreInstance
  ```

  Funkcia uloží ukazovateľ na pole `PeiCoreInstance.PS` do registru MM7 (MM7 je register veľkosti 64 bitov). Pole `PS` je deklarované štruktúrou `EFI_PEI_SERVICE`.

- `InitHobMem`

  Argumenty:

  ```
  EFI_PEI_SERVICES **PeiCoreInstance
  EFI_PEI_STARTUP_DESCRIPTOR *PeiStartupDescriptor,
  PEI_CORE_INSTANCE *OldCoreData
  ```

  Z argumentu `PeiStartupDescriptor` sa predajú informácie o CAR pamäti do premennej `PeiCoreInstance`. Jedna z úloh funkcie je inicializovať tzv. „Phase Handoff Information Table“ (PHIT) HOB , je to prvá štruktúra v HOB liste. PHIT opisuje boot mód a fyzickú pamäť používanú PEI fázou. Kopírujú sa dáta z `PeiCoreInstace.PS` do dočasného pamäťového miesta `PeiCoreInstance.ServiceTableShadow`.

- InitPpiServices

  Argumenty:

```
EIF_PEI_SERVICE **PeiCoreInstance
PEI_CORE_INSTANCE *OldCoreData
```

Vo funkcií sa inicializuje maximálny počet PPI rozhraní, v prípade analyzovaného modulu je to 127. Ak argument `OldCoreData` nie je nulový, tak inicializáciu maximálneho počtu PPI nevykoná.

- `InitSecService`

  Argumenty:

  ```
  EIF_PEI_SERVICE **PeiCoreInstance
  PEI_CORE_INSTANCE *OldCoreData
  ```

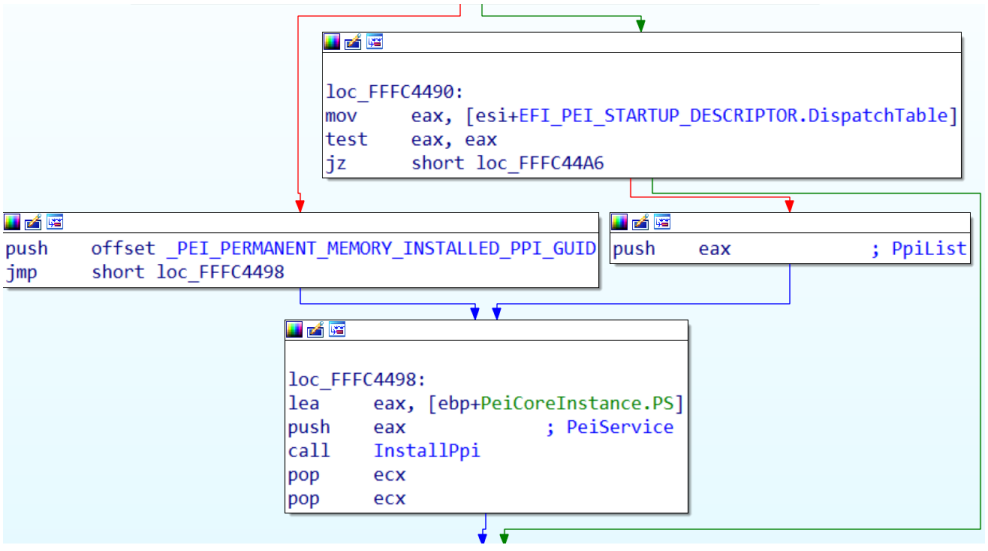
  Inicializuje sa PPI rozhranie `PEI_SECURITY_PPI`. V rozhraní sa nachádza služba `AuthenticationState`, ktorá slúži na autentizáciu súboru v pamäti.

- `InitDispatcherData`

  Argumenty:

  ```
  EFI_PEI_SERVICES **PeiCoreInstance
  PEI_CORE_INSTANCE *OldCoreData
  EFI_PEI_STARTUP_DESCRIPTOR *PeiStartupDescriptor
  ```

  Do `PeiCoreInstance` dočasného pamäťového miesta sa nakopírujú adresy BFV a aktuálny FV z `PeiStartupDescriptor`.



Obr. 5.3: Inštalácie PPI rozhraní alebo notifikácia

Po inicializačných rutinách sa zisťuje vytvorenie permanentného pamäťového miesta. Ak sa nevytvorilo permanentné pamäťové miesto, tak

```
.text:FFFC44A6
.text:FFFC44A6                 loc_FFFC44A6:
.text:FFFC44A6 2E4 lea     eax, [ebp+PeiCoreInstance.DispatchData.CurrentPeim]
.text:FFFC44AC 2E4 push    eax             ; DispatchData
.text:FFFC44AD 2E8 lea     eax, [ebp+PeiCoreInstance]
.text:FFFC44B3 2E8 push    eax             ; PeiCoreInstance
.text:FFFC44B4 2EC push    esi             ; PeiStartupDescriptor
.text:FFFC44B5 2F0 call    Dispatcher
.text:FFFC44BA 2F0 lea     eax, [ebp+OldCoreData]
.text:FFFC44BD 2F0 push    eax
.text:FFFC44BE 2F4 push    0
.text:FFFC44C0 2F8 push    0
.text:FFFC44C2 2FC lea     eax, [ebp+PeiCoreInstance.PS]
.text:FFFC44C8 2FC push    offset EFI_DXE_IPL_PPI_GUID_0xfffc4e4c
.text:FFFC44CD 300 push    eax
.text:FFFC44CE 304 call    LocatePpi
.text:FFFC44D3 304 push    dword ptr [ebp+PeiCoreInstance.HobList] ; HobList
.text:FFFC44D9 308 lea     eax, [ebp+PeiCoreInstance.PS]
.text:FFFC44DF 308 push    eax             ; PeiServices
.text:FFFC44E0 30C mov     eax, [ebp+OldCoreData]
.text:FFFC44E3 30C push    eax             ; This
.text:FFFC44E4 310 call    [eax+EFI_DXE_IPL_PPI.Entry]
.text:FFFC44E6 310 add     esp, 2Ch
.text:FFFC44E9 2E4 pop     edi
.text:FFFC44EA 2E0 mov     eax, 8000000Eh
.text:FFFC44EF 2E0 pop     esi
.text:FFFC44F0 2DC leave
.text:FFFC44F1 000 retn
.text:FFFC44F1                 PeiCore endp
.text:FFFC44F1
```

Obr. 5.4: Koniec funkcie PEI core

pred zavolaním hlavnej funkcie `Dispatcher` sa musí nainštalovať pole PPI rozhraní, ktoré bolo predané od SEC fázy. Ak sa vytvorilo permanentné pamäťové miesto a inicializovala sa hlavná pamäť RAM, tak PEI core inštaluje `_PEI_PERMANENT_MEMORY_INSTALLED_PPI_GUID` interface viď. obr. 5.3 aby notifikovalo ostatné PEI moduly o použití permanentného pamäťového miesta. Po inštalácií sa prejde na ďalšiu funkciu `Dispatcher` (obr. 5.4).

Dispatcher

Argumenty:

```
EFI_PEI_STARTUP_DESCRIPTOR *PeiStartupDescriptor
PEI_CORE_INSTANCE *PeiCoreInstance
PEI_CORE_DISPATCH_DATA *DispatchData
```

Argument `PeiStartupDescriptor` vo funkcií `Dispatcher` slúži na predanie dát pri zavolaní funkcie `PeiCore`. Tento prípad nastane ak konkretný PEI modul nájde a inicializuje permanentné pamäťové miesto. Argument `PeiCoreInstance` sa využíva na volanie služieb z `EFI_PEI_SERVICES` a predanie informácií o vytvorenom zásobníku v permanentnej pamäti. Z týchto troch argumentov sa najviac využíva argument `DispatchData`. Z argumentu `DispatchData` sa nachádzajú informácie typu:

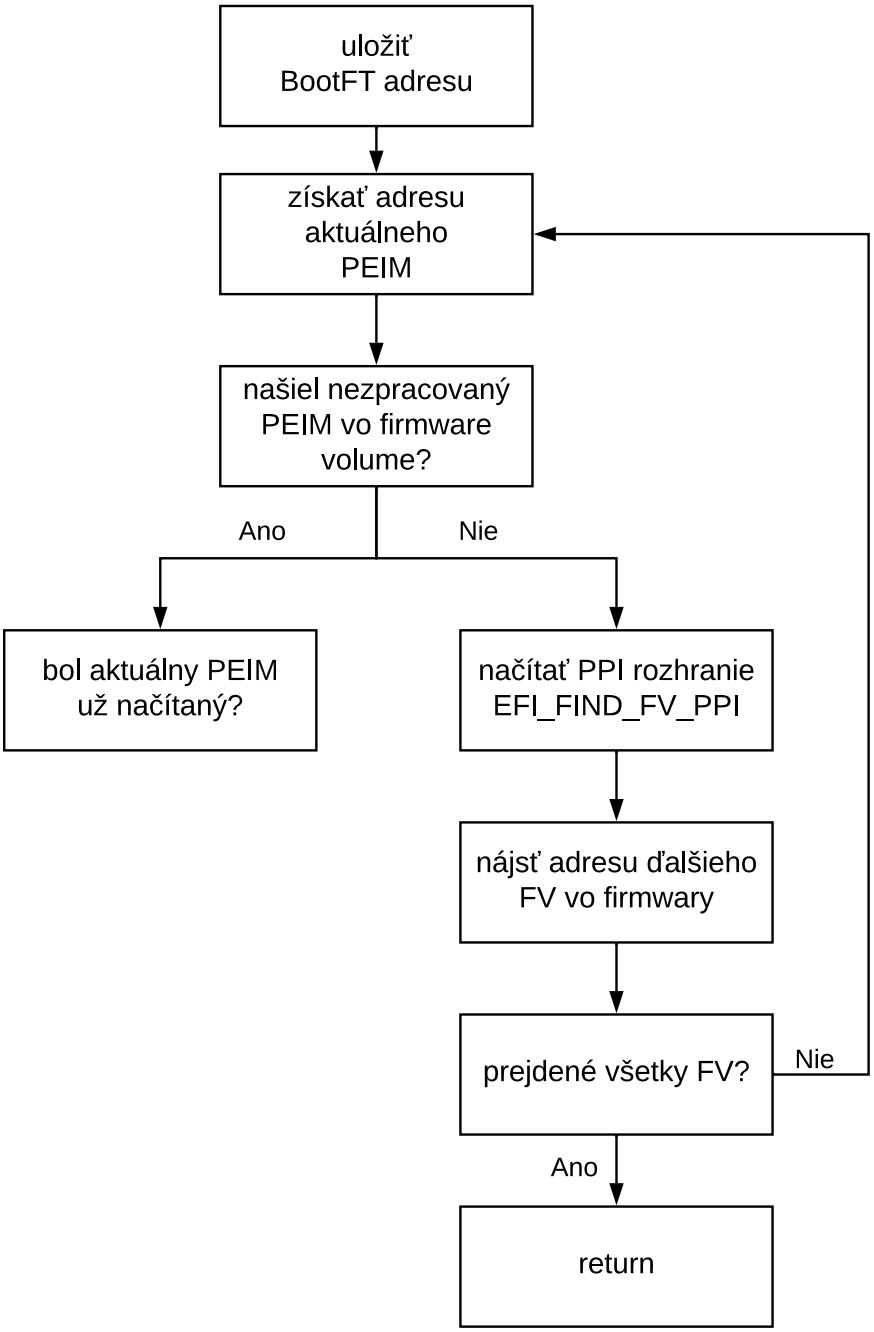
- adresa Boot firmware volume (BFV)
- adresa aktuálneho FV
- aké PEI moduly boli načítané
- adresa aktuálneho PEI modulu
- ukazovateľ na funkciu FindFv

Hlavnou úlohou Dispatchera je nájsť všetky PEI moduly vo FV a spustiť ich v správnom poradí. V Dispatchri sa vykoná aj prepnutie z dočasného pamäťového miesta na permanentné pamäťové miesto. Prepnutie nastane ak `Dispatcher` nájde konkrétny PEI modul určený na nájdenie a inicializáciu permanentného pamäťového miesta. Po prepnutí do permanentného pamamäťového miesta PEI core modul oznámi PEI modulom inicializáciu permanentného pamäťového miesta. Neskôr sa ukazovateľ na zásobník a dáta prenesené zo SEC fázy presunú do novej inicializovanej permanentnej pamäti. Na obr. 5.6 vidíme hlavný `Dispatcher` diagram znázorňujúci postup pri hľadaní a spúšťaní PEI modulov a prepnutie z dočasného pamäťového miesta na permanentné pamäťové miesto. Na začiatku `Dispatcher` uloží aktuálnu adresu BFV do lokálnej premennej `BootFVadd`, aby nevzniklo opätovné prehľadavanie BFV. Pomocou funkcie `PeiFindNextPeim` sa získa adresa nasledujúceho PEI modulu v konkrétnom FV. Môžu nastať dve situácie, ako je vidieť podľa obr. 5.5

- Na konkrétnom BFV sa nenachádza žiaden PEI modul
- Na konkrétnom BFV sa našiel PEI modul

Obr. 5.5 znázorňuje stav, keď BFV nenájde žiaden nezpracovaný PEI modul (ďalej len „modul“, ak nebude uvedné inak). V tomto prípade sa prejde do hľadania ďalších FV v firmwari. Na hľadanie FV vo firmwari, slúži funkcia `FindFV`, ktorá sa nachádza v PPI rozhraní `_EFI_PEI_FIND_FV_PPI`. Ak sa ukazovateľ na funkciu `FindFV` nenachádza v argumente `DispatchData`, musí sa načítať PPI rozhranie.To sa prevedie pomocou funkcie `LocatePpi`, ktorá nájde v PEI PPI databáze pomocou unikátneho GUID idetifikátora, v našom prípade {12481636-23A0-E544-BD8505BF3C7700AA}, spomínané PPI rozhranie
`_EFI_PEI_FIND_FV_PPI`.

Načítaná funkcia `FindFv` sa zavolá a funkcia vráti adresu FV. Následne sa vykonajú testy či vrátená adresa FV nie je adresa BFV alebo adresa aktuálneho FV. Pri vrátení adresy neprehľadaného FV sa prejde na prehľadávanie modulov vo FV. Ak sa nenájde už žiadna FV adresa alebo funkcia `FindFv` nevrátila žiadnu adresu FV, tak sa zistí, či boli načítané všetky moduly. Tento stav môže nastať v prípade, že neboli načítané všetky moduly



Obr. 5.5: Diagram hľadania firmware volume vo firmwari

kvôli DEPEX podmienke. Ta zisťuje, či daný modul sa môže načítať. Keď neboli načítané všetky moduly, prehľadávanie a načítanie modulov sa vykonáva znovu. Ak sa načítali všetky moduly a prehľadané FV, tak funkcia `Dispatcher`

skončí.

Na obr. 5.6 je diagram dispatchra, kde obdĺžniky vyznačené šedou farbou znázorňujú konkrétny stav nájdeného nezpracovaného modulu v BFV alebo FV. Následne sa zistí, či daný modul už bol načítaný. Nájdené moduly, ktoré boli načítané preskočíme, pretože sme ich už načítali. Pri nenačítanom module sa zisťuje DEPEX podmienka, či aktuálny modul môže byť načítaný. Ak sa DEPEX podmienka vyhodnotila ako pravda, tak sa pokračuje pri spracovaní modulu. Modul sa načíta do pamäti a vráti adresu vstupného bodu. Avšak ak sa DEPEX podmienka vyhodnotila ako nepravda, tak sa aktuálny modul nemôže načítať, musí sa preskočiť a prejsť na spracovanie ďalšieho aktuálneho modulu.

Pre načítaný modul v pamäti sa zavolá funkcia `AuthentacationState` z PPI rozhrania `EFI_PEI_SECURITY_PPI`, čo zabezpečí autentizáciu modulu. Ak sa modul neautentizoval správne, označí sa modul ako načítaný a spracovaný. Predtým ako, sa prejde do ďalšieho modulu zistí sa, či nebola vytvorená permanetná pamäť (pre tento prípad to platiť nebude).

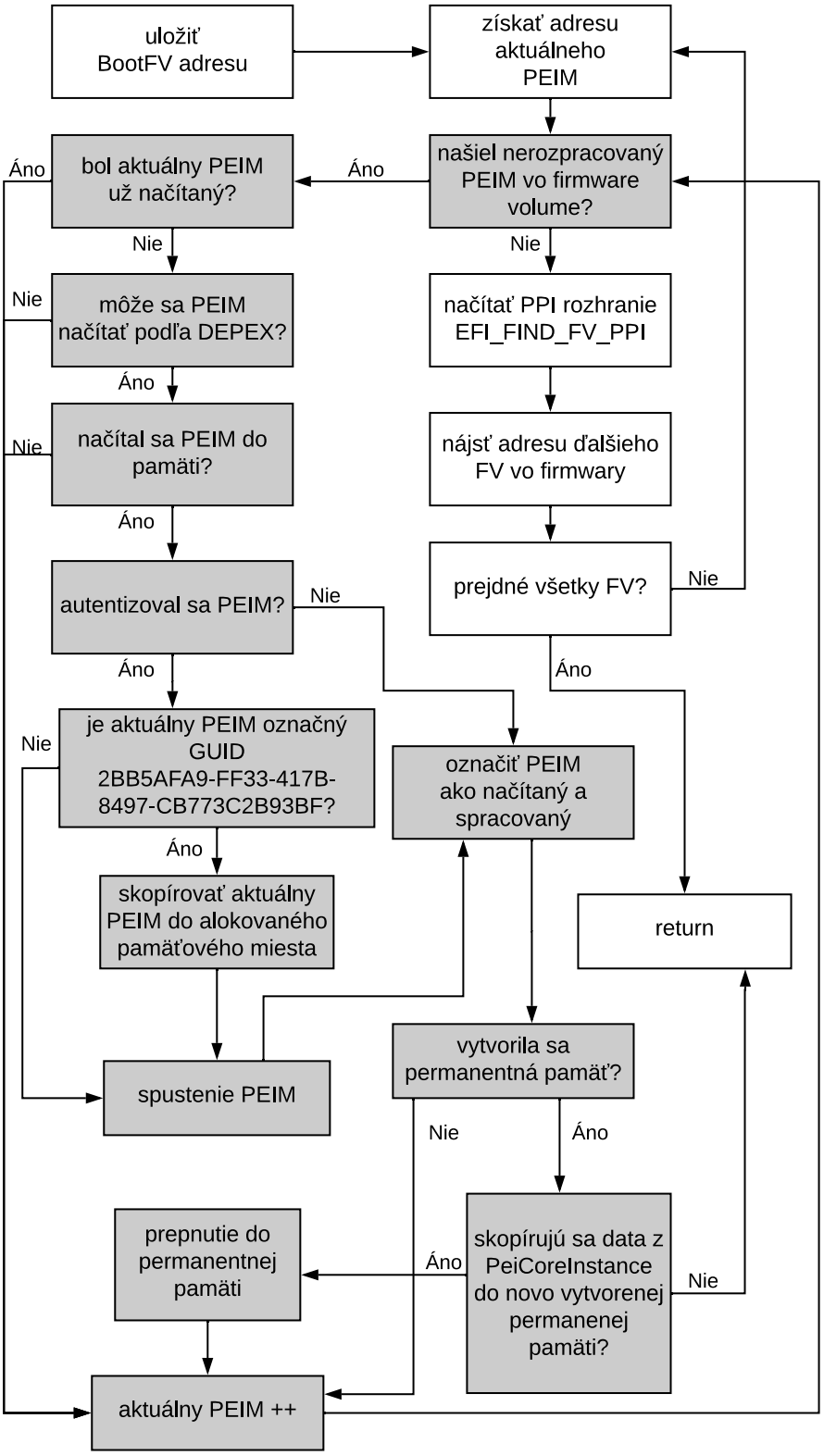
Ak sa modul autentizoval, zisťuje sa, či konkrétny modul nemá GUID 2BB5AFA9-FF33-417B-8497-CB773C2B93BF konkrétny GUID označuje modul s názvom „CPUPei“. To je modul, ktorý inicializuje permanentnú pamäť. Pre CPUPei modul sa alokuje pamäťové miesto v dočasnej pamäti, kde Dispatcher nakopíruje CPUPei modul. Neskôr nasleduje spustenie konkrétneho modulu zavolaním vstupného bodu. Po skončení vykonania modulu sa spustený modul označí ako načítaný a prejde sa na spracovanie ďalšieho modulu vo FV.

Špecialny prípad nastáva pri spúšťaní CPUPei modulu, ktorý inicializiju permanentnú pamäť. V CPUPei modulu sa zavolá funkcia `InstallPeiMemory`, ktorá funkcia inicializuje v operačnej pamäti RAM permanentnú pamäť. Potom sa do nej skopírujú dáta z `PeiCoreInstance`. Ak sa dáta úspešne skopírovali, dôjde k prepnutiu do permanentnej pamäti. To znamená zavolať funkciu `PeiCore` s novým zásobníkom a dátami z `PeiCoreInstance` v permanetnej pamäti. Po načítaní všetkých modulov sa funkcia `Dispatcher` skončí.

Na obr. 5.6 vidíme veľký `Dispatcher` diagram, ktorý je vytvorený zlúčením všetkých diagramov a pridaním zafarbených obdĺžnikov. Zafarbené obdĺžniky predstavujú stav, keď sa na konkrétnom BFV našiel modul.

PEI core modul spravil všetko preto, aby mohol odovzdať kontrolu DXE fáze, ako vidíme na obr. 5.4. Stačí lokalizovať PPI rozhranie `EFI_DXE_IPL_PPI` a zavolať funkciu `EFI_DXE_IPL_PPI.Entry` s argumentami:

- ukazovateľ na `EFI_PEI_SERVICES`
- ukazovať na vytvorený HOB List.



Obr. 5.6: Diagram dispatchra

Kapitola 6

Driver Execution Enviroment DXE

V „Driver Execution Enviroment“ (DXE) fázy prebieha najväčšia časť inicializácie systému. DXE fáza je zodpovedná za nájdenie a spúštanie DXE ovládačov v správnom poradí. DXE ovladače inicializujú procesor, chipset, platform komponenty a poskytujú softwarovú abstrakciu pre konzoly a boot zariadenia. Navazuje na PEI fázu, od ktorej získava stav systému cez list nezávislých dátových štruktúr tzv. „Hand-Off Blocks“ (HOB)[8].

DXE fáza pozostáva z niekoľkých komponentov:

- DXE core
- DXE dispatcher
- DXE ovladače[18]

DXE core poskytuje:

- Boot služby
- Runtime služby
- DXE služby

DXE komponenty spolupracujú na inicializáci platformy a poskytujú služby potrebné na bootovanie operačného systému, spoločne s „Boot Device Selection“ (BDS) fázy. DXE fáza je ukončená, ak sa operačný systém začína úspešne bootovať. To znamená keď, spúšťanie prechádza z DXE fázy do BSD fázy. Iba tzv. runtime služby poskytované DXE core a služby poskytované DXE ovládačmi sú povolené, aby pretrvali počas behu operačného systému[8].

6.1 Terminológia

Pool pamäť

Pool pamäť je prealokované pamäťové miesto s fixnou veľkosťou. Ak potrebujeme alokovať nové pamäťové miesto, zoberie sa požadovaná veľkosť od pamäťového poolu namiesto požiadania systému o pamäťové miesto [19].

Pamäťový deskriptor

Pamäťový deskriptor definuje systémovú pamäťovú mapu RAM a rozsah fyzickej pamäti rezervované firmwarom[20].

Apriori súbor

Apriori súbor je súbor označený GUID identifikátorom, ktorý uchováva zoznam UEFI ovládačov a aplikácií, ako majú byť načítané a spúšťané v správnom poradí.[8]

6.2 Analýza DXE Core modulu pomocou reverzného inžinierstva

V nasledujúcej podkapitole opisujeme funkčnosť spustiteľného kódu hlavného DXE core modulu. Taktiež opis funkčnosti DXE core modulu bol vytvorený pomocou reverzného inžinierstva spustiteľného kódu, štúdium existujúcih dokumentácií a čítanie jeho zdrojového kódu. DXE core je 64-bitový spustiteľný súbor navrhnutý tak, aby bol spustiteľný pri všetkých druhoch procesorov, chipsetov alebo s inými platform závislosťami. Schopnosť prenášania spustiteľného súboru je dosiahnutá začlenením týchto špecifikácií:

- Počiatočný stav DXE core záleží od HOB listu, nezáleží na žiadnej službe z predošlej fázy. Z čoho vyplýva, že predošlé fázy môžu byť uvoľnené (moduly z predošlích fáz nemusia byť v pamäti).
- DXE core neobsahuje žiadne pevné adresy a preto môže byť načítaný a spúštaný hocikde vo fyzickej pamäti.
- DXE core neobsahuje žiadne procesorové, chipsetové alebo platformové špecifikácie. DXE core je akási abstrakcia cez hardwarový systém až po architektonické rozhrania protokolov. DXE ovládače poskytujú architektonické rozhrania protokolov načítané DXE dispatcherom[8].

Najskôr musí DXE core prijať ukazovateľ na začiatok HOB listu, ktorý bol vytvorený PEI fázou.

Na obr. 6.1 vidíme začiatok hlavnej funkcie DXE core modulu. Vstupné argumenty sú:

```
void *Hob
```

Argument `Hob` je ukazovateľ na „Phase Handoff Information Table“ (PHIT) PHIT HOB štruktúru, čo je prvá štruktúra v HOB Liste. PHIT HOB štruktúra

```
; #16 __fastcall ModuleEntryPoint(void *HOB)
public _ModuleEntryPoint
_ModuleEntryPoint proc near

MemorybaseAddr= qword ptr -18h
HobStart= qword ptr  8
anonymous_0= dword ptr  10h
UserHandle= qword ptr  18h

mov     rax, rsp
mov     [rax+8], rcx
sub     rsp, 38h
mov     cs:pToHobStart, rcx
lea     r8, [rax+20h]   ; out_memLen
lea     rdx, [rax-18h]  ; out_MemoryBaseAddress
lea     rcx, [rax+8]    ; in_Hob
call    InitializeMemory
lea     rdx, EfiSystemTableTemplate ; EfiSystemTable
mov     ecx, 120        ; sizeOfSystemTable
call    AllocateRuntimeCopyPool
mov     cs:EfiSystemTable, rax
test    rax, rax
jnz     short loc_1800004CC
```

Obr. 6.1: Začiatok hlavnej DXE Core funkcie

opisuje permanentnú pamäť a aký typ systémového reštartu sa vykonal. Typ systémového reštartu je opísaný číslom tzv. „boot mode“. Na obr. 6.2 vidíme, ako je namapovaný HOB List v pamäti.

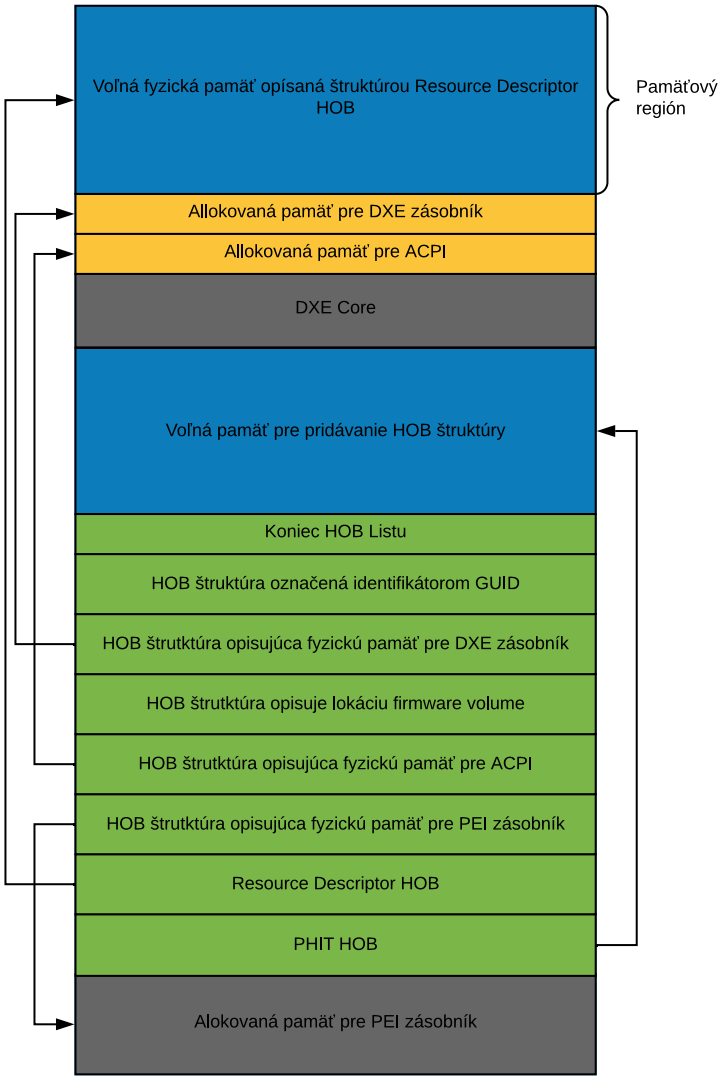
Následne pokračujú funkcie:

- `InitializeMemory`

  Argumenty:

  ```
  void **HobStart,
  EFI_PHYSICAL_ADDRESS *MemoryBaseAddress,
  UINT64 *MemoryLength
  ```

  `InitializeMemory` je zodpovedný za inicializáciu a mapovanie fyzickej pamäti podľa „Resource descriptor“ HOB štrukúry. Resource Descriptor HOB je štruktúra opisujúca nepremiestniteľné fyzické pamäťové regióny [15]. Najprv musí inicializovať tzv. „Pool“ pamäť, aby spúšťané UEFI ovládače alebo aplikácie mohli použiť funkcie na alokovanie pool pamäte.



Obr. 6.2: HOB List vo fyzickej pamäti

Po inicializácií Pool pamäti vyhľadá HOB Resource Descriptor štruktúru a vráti ukazovateľ na pamäťový región, určí jeho rozsah a pridá ho do pamäťového deskriptoru.

- `AllocateRuntimeCopyPool`

  Argumenty:

  ```
  UINTN AllocationSize,
  void *Buffer
  ```

  Ako vidíme na obr. 6.1, `AllocateRuntimeCopyPool` je zodpovedná za inicializáciu hlavnej systémovej tabuľky `EFI_SYSTEM_TABLE` a runtime

službieb `EFI_RUNTIME_SERVICES`. `EFI_SYSTEM_TABLE` je najdôležitejšia dátová štruktúra v UEFI. Ukazovateľ na `EFI_SYSTEM_TABLE` je argumentom každého spúšťaného ovládača alebo UEFI aplikácie. Z tejto dátovej štruktúry má spustiteľný UEFI image prístup ku konfigurácií systému a všetkým UEFI službám ako:

- Boot UEFI služby
- Runtime UEFI služby
- Služby protokolu[8]

`AllocateRuntimeCopyPool` alokuje pool veľkosti 120 bajtov, kde sa do alokovaného poolu prekopíruje globálna premenná `EfiSystemTableTemplate` deklarovaná štruktúrou `EFI_SYSTEM_TABLE`. Po skončení funkcia vráti ukazovateľ na alokovaný pool a uloží ho do globálnej premennej `EfiSystemTable` podľa obr. 6.1.

Na obr. 6.3 môžeme vidieť opäť volanie funkcie `AllocateRuntimeCopyPool`, ktorá alokuje pool veľkosti 136 bytov, kde sa opäť prekopíruje globálna premenná `EfiRuntimeServicesTableTemplate` deklarovaná štruktúrou `EFI_RUNTIME_SERVICES`. Vrátený ukazateľ na alokovaný pool sa uloží do globálnej premennej `EfiSystemTable.RuntimeServices`.

- `InitializeImageServices`

  Argumenty:

  ```
  void *HobStart
  ```

  `InitializeImageServices` hľadá v HOB liste načítaný DXE core image a získa informácie o mieste DXE core imagu v pamäti. Pomocou získaných informácií `InitializeImageServices` vytvorí tzv. `ImageHandle` na DXE Core a prída vytvorený `ImageHandle` do databázy handlou viď. obr. 6.4 . K vytvorenej DXE core `ImageHandle` inštaluje protokol `EFI_LOADED_IMAGE_PROTOCOL`.

  `EFI_LOADED_IMAGE_PROTOCOL` je protokol opisujúci lokáciu načítaného imagu v pamäti, typ alokovanej pamäti pre image a odkiaľ bol image načítaný. Na konci funkcie sa pridá aj protokol `PE32_IMAGE_PROTOCOL_GUID` do databázy handlou viď. obr. 6.4, ktorý slúži na načítanie/uvoľnenie UEFI imagu do/z pamäti a zároveň spúšťanie načítaných UEFI image v pamäti[21].

- `InitializeGcdServices`

  Argumenty:

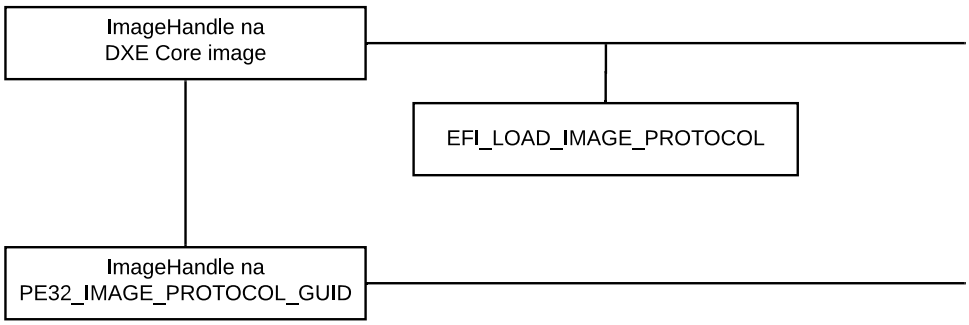
  ```
  void **HobStart,
  EFI_PHYSICAL_ADDRESS MemoryBaseAddress,
  UINT64 MemoryLength
  ```

`InitializeGcdServices` inicializuje „Global Coherency Domain“ (GCD) službu a mapuje pamäťové miesta v HOB liste. GCD služba sa používa na spravovanie pamäti a spravovanie zdrojov pre vstupno/výstupné zariadenia[17].

Najprv sa inicializujú dve mapy, jedna pre pamäť a druhá pre mapy vstupno/výstupných zariadení. Potom musí prehľadať všetky špecifické HOB štruktúry tzv. „Resource Descriptor HOB“. Po prehľadaní „Resource Descriptor HOB“ štruktúr si namapuje tieto fyzické pamäťové

```
loc_1800004CC:          ; EfiSystemTable
lea     rdx, EfiRuntimeServicesTableTemplate
mov     ecx, 136        ; sizeOfSystemTable
call    AllocateRuntimeCopyPool
mov     rcx, cs:EfiSystemTable
mov     cs:EfiRuntimeServices, rax
mov     [rcx+EFI_SYSTEM_TABLE.RuntimeServices], rax
mov     rcx, [rsp+38h+HobStart] ; HobStart
call    InitializeImageServices
mov     r8, [rsp+58h]   ; size
mov     rdx, [rsp+20h]  ; MemoryBaseAddress
lea     rcx, [rsp+38h+HobStart] ; Hob
call    InitializeGcdServices
mov     r11, [rsp+38h+HobStart]
mov     rdx, cs:EfiDxeServices ; pointer
lea     rcx, EFI_DXE_SERVICES_TABLE_GUID_0x18000bab0 ; guid
mov     cs:pToHobStart, r11
call    InstallConfigurationTable
mov     rdx, [rsp+38h+HobStart] ; pointer
lea     rcx, TCG_EFI_HOB_LIST_GUID_0x18000baa0 ; guid
call    InstallConfigurationTable
lea     rdx, dword_18000B690 ; pointer
lea     rcx, EFI_MEMORY_TYPE_INFORMATION_GUID_0x18000bda0 ; guid
call    InstallConfigurationTable
```

Obr. 6.3: Volanie iniciliazačných funkcií a aktualizácia konfiguračnej tabuľky



Obr. 6.4: Aktuálny stav databázy handlou

regióny do inicializovaných máp.

- `InstallConfigurationTable`

  Argumenty:

  ```
  EFI_GUID *Guid
  VOID *Table
  ```

  `InstallConfigurationTable` pridáva, odstraňuje alebo updatuje elementy v konfiguračnej tabuľke `ConfigurationTable`. Každý element v tabuľke je popísaný štruktúrou `EFI_CONFIGURATION_TABLE`. Tato štruktúra obsahuje identifikátor GUID a ukazovateľ na tabuľku. GUID sa používa na identifikáciu ukazovateľa tabuľky alebo ukazovateľa na miesto v pamäti. Na obr. 6.3 vidíme volanie funkcie `InstallConfigurationTable` tri krát:

  - Pridáva GUID `EFI_DXE_SERVICES_TABLE_GUID` do konfiguračnej tabuľky a k nemu prislúcha ukazovateľ na štruktúru `EFI_DXE_SERVICES`.
  - Pridáva GUID `TCG_EFI_HOB_LIST_GUID` do konfiguračnej tabuľky a k nemu prislúcha ukazovateľ na začiatok HOB listu.
  - Pridáva GUID `EFI_MEMORY_TYPE_INFORMATION_GUID` do konfiguračnej tabuľky a k nemu prislúcha ukazovateľ na konkrétnu HOB štruktúru, ktorá opisuje typ pamäťového miesta a počet pamäťových stránok.

- `GetProtocolInterface`

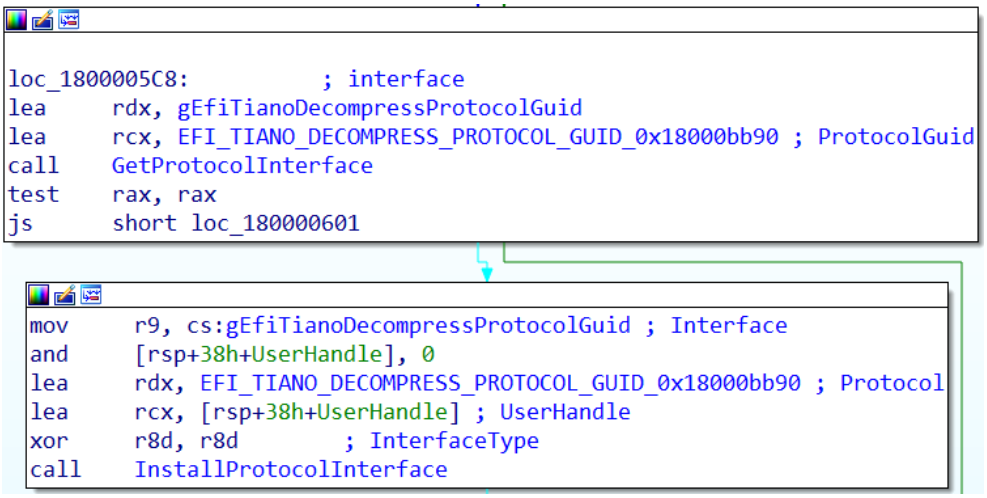
  Argumenty:

  ```
  EFI_GUID *ProtocolGuid,
  VOID **Interface
  ```

  `GetProtocolInterface` hľadá v HOB liste špecifické rozhranie protokolu podľa identifikátora GUID. Ak funkcia nájde napríklad indentifikátor GUID `EFI_TIANO_DECOMPRESS_PROTOCOL_GUID` v HOB liste, vráti do argumentu `gEfiTianoDecompressProtocolGuid` ukazovateľ na protokol rozhranie (obr. 6.5). V DXE core sa funkcia `GetProtocolInterface` volá niekoľko krát. V našom prípade hľadá v HOB liste tieto GUID identifikátory:

  - `EFI_STATUS_CODE_RUNTIME_PROTOCOL_GUID`
  - `EFI_DECOMPRESS_PROTOCOL_GUID`
  - `EFI_TIANO_DECOMPRESS_PROTOCOL_GUID`
  - `EFI_CUSTOMIZED_DECOMPRESS_PROTOCOL_GUID`
  - `EFI_PEI_FLUSH_INSTRUCTION_CACHE_GUID`

- EFI_PEI_PE_COFF_LOADER_GUID
- EFI_PEI_TRANSFER_CONTROL_GUID



Obr. 6.5: GetProtocolInterface

- `ReportStatusCode`

  Argumenty:

  `EFI_STATUS_CODE_VALUE Value`

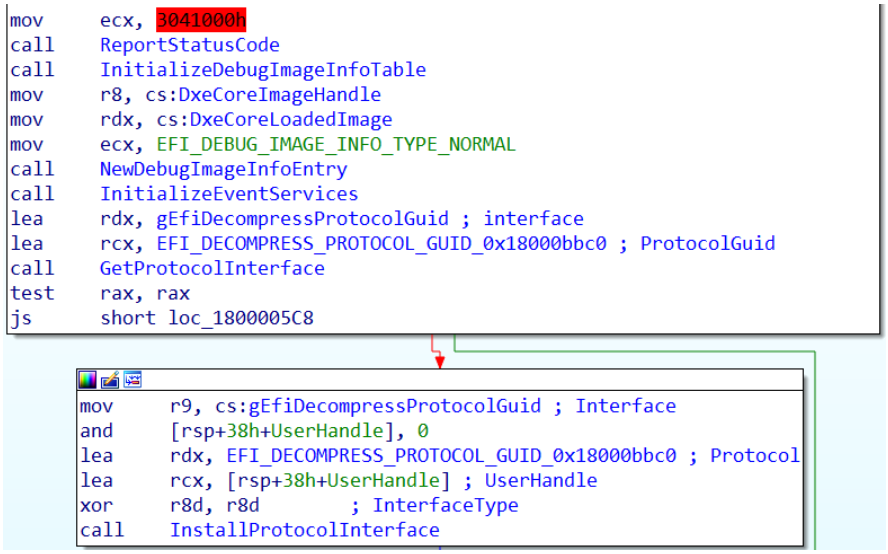
  `ReportStatusCode` reportuje aktuálny stav spúšťania aktuálneho softwarového modulu. Informácia o aktuálnom stave sa určuje pomocou číselných hodnôt, kde každá hodnota vyjadruje aktuálny stav modulu. Číselné hodnoty sú uvedené v dokumentacií [21]. V tomto prípade, ako vidíme na obr. 6.6, je hodnota `3041000h`. Ak by sme si hodnotu preložili podľa dokumentácie [21], výjde nám nasledovne:

  - `03000000h EFI_SOFTWARE`
  - `40000h EFI_SOFTWARE_DXE_CORE`
  - `1000h EFI_SUBCLASS_SPECIFIC`
  - `0h EFI_SW_DXE_CORE_PC_ENTRY_POINT`

  Z vypísaných hodnôt sme zistili, že sa spúšťa hlavná funkcia DXE Core.

- `InitializeDebugImageInfoTable`

  Vytvára a inicializuje tabuľku `DebugImageInfo`. `DebugImageInfo` sa využíva pri debugovaní a na získanie informácií načítaného imagu v pamäti[21]. Globálna premenná `DebugImageInfo` je deklarovaná štruktúrou `EFI_SYSTEM_TABLE_POINTER`. Do `DebugImageInfo.EfiSystemTableBase`



Obr. 6.6: ReportStatusCode, inicializácia debug informácií a eventov

sa prekopíruje fyzická adresa systémovej EFI tabuľky, aby sa pri debugovaní vedelo, kde vo fyzickej pamäti sa nachádza systémová EFI tabuľka. Vo funkcií `InitializeDebugImageInfoTable` sa do konfiguračnej tabuľky (`EFI_SYSTEM_TABLE.ConfigurationTable`) pridáva GUID identifikátor `EFI_DEBUG_IMAGE_INFO_TABLE_GUID` a k nemu ukazovateľ na globalnu premennú `DebugInfoTableHeader` deklarovaný štruktúrou `EFI_DEBUG_IMAGE_INFO_TABLE_HEADER`. Ten uchováva všetky potrebné informácie o imagu.

- `NewDebugImageInfoEntry`

  Argumenty:

  ```
  UINT32 ImageInfoType
  EFI_LOADED_IMAGE_PROTOCOL *LoadedImage
  EFI_HANDLE ImageHandle
  ```

  `NewDebugImageInfoEntry` zapisuje informácie o imagu do globálnej tabuľky `DebugInfoTableHeader`. Ako vidíme na obr. 6.6, do tabuľky `DebugInfoTableHeader` sa zapisuje `ImageHandle` na DXE core (`DxeCoreImageHandle`) a informácie o imagu (`DxeCoreLoadedImage`).

- `InitializeEventServices`

  `InitializeEventServices` inicializuje globálny list eventov `EventQueue` a inicializuje tzv. „Timer event“. Timer event je typ signálu, ktorý sa presúva z čakacieho stavu do signalizovacieho stavu po ubehnutí časového intervalu[8].

- `InstallProtocolInterface`

  Argumenty:

  ```
  EFI_HANDLE *Handle
  EFI_GUID *Protocol
  EFI_INTERFACE_TYPE InterfaceType
  void *Interface
  ```

  `InstallProtocolInterface` je služba, ktorá pridáva protokolové rozhranie do existujúceho handla v databáze handlou alebo vytvára nový handle v databáze handlou prislúchajúci k protokolovému rozhraniu[9]. Ak sa funkcii `GetProtocolInterface` podarí nájsť v HOB liste protokoly uvedné nižšie, tak sa pomocou služby `InstallProtocolInterface` pridajú do databáze handlou.

  - `EFI_DECOMPRESS_PROTOCOL_GUID`
  - `EFI_TIANO_DECOMPRESS_PROTOCOL_GUID`
  - `EFI_CUSTOMIZED_DECOMPRESS_PROTOCOL_GUID`

- `NotifyOnArchProtocolInstallation`

  `NotifyOnArchProtocolInstallation` registruje udalosti do jednotlivých architektonických protokolov. Architektonické protokoly sa nachádzajú v globálnom poli `ArchProtocols` deklarované štruktúrou `ARCHITECTURAL_PROTOCOL_ENTRY`. Ak sa inštaluje jeden z protokolov v `ArchProtocols` sa vyšle signál.

- `FwVolBlockDriverInit`

  Argumenty:

  ```
  EFI_HANDLE ImageHandle
  EFI_SYSTEM_TABLE *SystemTable
  ```

  `FwVolBlockDriverInit` v HOB Liste hľadá HOB štruktúru typu `EFI_HOB_TYPE_FV`. `EFI_HOB_TYPE_FV` je typ štruktúry, ktorá opisuje lokáciu firmware volume. Ak sa nájde tento typ štruktúry, tak vráti ukazovateľ na miesto v pamäti kde sa nachádza `EFI_HOB_TYPE_FV` HOB štruktúra. K ukazovateľovi sa namapuje štruktúra `EFI_FW_VOL_BLOCK_DEVICE`. Táto štruktúra opisuje firmware volume handle, adresu na firmware volume a atribúty firmware volume. K vytvorenej handle na konkrétny firmware volume v databáze handlou, ako na obr. 6.7, sa pridajú rozhrania protokolov:

  - `EFI_DEVICE_PATH_PROTOCOL_GUID`
  - `EFI_FIRMWARE_VOLUME_BLOCK_PROTOCOL_GUID`
  - `EFI_FIRMWARE_VOLUME_DISPATCH_PROTOCOL_GUID`

- `FwVolDriverInit`

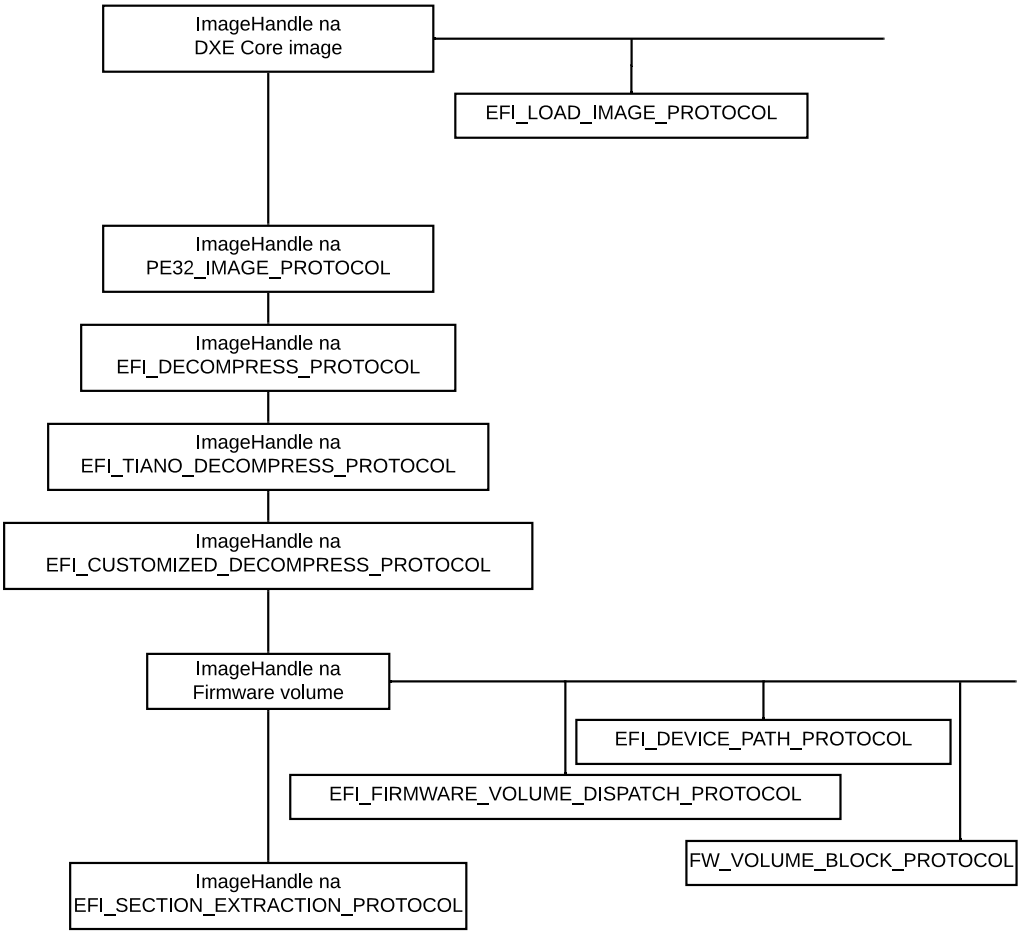
  Argumenty:

  ```
  EFI_HANDLE ImageHandle
  EFI_SYSTEM_TABLE *SystemTable
  ```

  `FwVolDriverInit` vytvorí notifikáciu, ak sa inštaluje do databázy handlou rozhranie protokolov `EFI_FIRMWARE_VOLUME_BLOCK_PROTOCOL_GUID`. Notifikácia spočíva v spúšťaní tzv. „Call-back“ funkcie. Tou je v našom prípade je `NotifyFwVolBlock`. „Call-back“ funkcia nájde v databáze handlou všetky handle, ktoré používajú protokol rozhranie `EFI_FIRMWARE_VOLUME_BLOCK_PROTOCOL`. Pri nájdených handloch nainštaluje protokolové rozhranie `EFI_FIRMWARE_VOLUME_PROTOCOL_GUID`.

- `InitializeSectionExtraction`

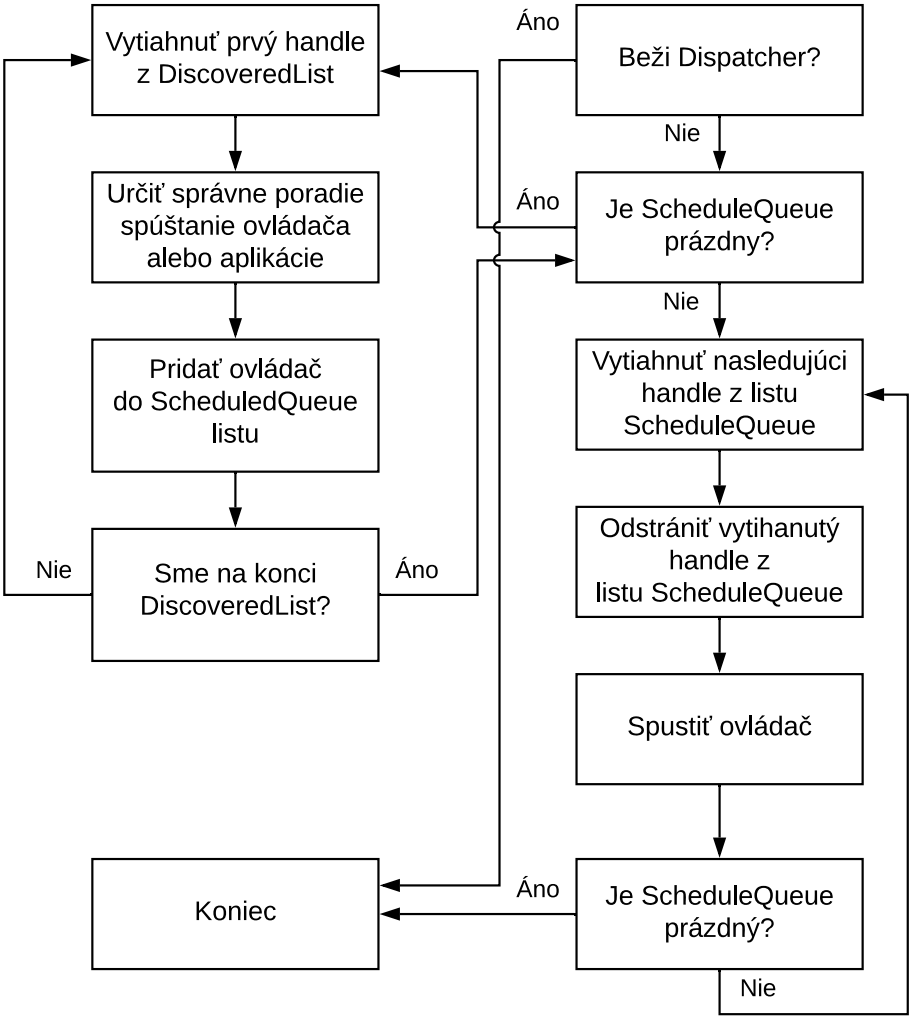
  Argumenty:



Obr. 6.7: Databáza handlou

```
EFI_HANDLE ImageHandle
EFI_SYSTEM_TABLE *SystemTable
```

`InitializeSectionExtraction` inštaluje do databázy handlou protokol rozhranie `EFI_SECTION_EXTRACTION_PROTOCOL` (obr. 6.7). Toto rozhranie protokolu slúži na prečítanie komprimovaných sekcií z DXE ovládačov uložených vo firmware volumes[22].



Obr. 6.8: Diagram DXE dispatchra

- `InitializeDispatcher`

  `InitializeDispatcher` vytvorí notifikáciu, ak sa inštaluje do databázy handlou protokol rozhranie `EFI_FIRMWARE_VOLUME_PROTOCOL_GUID`. Notifikácia spočíva v spúšťaní tzv. „Call-back“ funkcie. Tou je v našom prípade je `FwVolEventProtocolNotify`. Pre konkrétny firmware volume prehľadá všetky UEFI ovladače a aplikácie a priradí ich do globálneho listu `DiscoveredList`. Ak však nájde tzv. „apriori“ súbor, zaradí ich (podľa apriori súboru) do globálneho list `ScheduledQueue` v správnom poradí. Apriori súbor uchováva zoznam UEFI ovládačov a aplikácií, ako majú byť načítané a spúšťané. v spraváno poradí.

- `Dispatcher`

  `Dispatcher` je zodpovedný za spúšťanie UEFI ovladačov alebo aplikacií v správnom poradí. Handle na UEFI ovladač alebo aplikáciu nájde buď v globálnom liste `ScheduledQueue` alebo `DiscoveredList`. Na obr. 6.8 vidíme zjednodušený diagram opisujúci chod funkcie `Dispatcher`.

- `AllEfiServicesAvailable`

  `AllEfiServicesAvailable` ošetruje, či sa v globálnom poli `ArchProtocols` nachádzajú všetky ukazovatele na jednotlivé architektonické protokoly.

- `ReportStatusCode`

  Argumenty:

  ```
  EFI_STATUS_CODE_VALUE Value
  ```

  Funkciu `ReportStatusCode` sme už opísali vyššie v sekci 6.2. V prípade zobrazenom na obr. 6.9 je hodnota `3041001h`, ktorá vyjadruje nasledovné:

  - `03000000h EFI_SOFTWARE`
  - `40000h EFI_SOFTWARE_DXE_CORE`
  - `1000h EFI_SUBCLASS_SPECIFIC`
  - `1h EFI_SW_DXE_CORE_PC_HANDOFF_TO_NEXT`

  Vypísane hodnoty vyjadrujú, že nasleduje odovzdanie kontroly z DXE fázy na „Boot Device Selection“ (BDS) fázu.

- `EFI_BDS_ARCH_PROTOCOL.Entry`

  `EFI_BDS_ARCH_PROTOCOL.Entry` je posledná volána funkcia v DXE fáze. Nastáva vnej predanie kontroly na BDS fázu, ako ukazuje obr. 6.9.

```
loc_18000063A:          ; interface
lea     rdx, EfiPeiFlushInstructionCacheGuid
lea     rcx, EFI_PEI_FLUSH_INSTRUCTION_CACHE_GUID_0x18000bd90 ; ProtocolGuid
call    GetProtocolInterface
lea     rdx, EfiPeiPeCoffLoaderGuid ; interface
lea     rcx, EFI_PEI_PE_COFF_LOADER_GUID_0x18000bd80 ; ProtocolGuid
call    GetProtocolInterface
lea     rdx, EfiPeiTransferControlGuid ; interface
lea     rcx, EFI_PEI_TRANSFER_CONTROL_GUID_0x18000bd70 ; ProtocolGuid
call    GetProtocolInterface
call    NotifyOnArchProtocolInstallation
mov     rdx, cs:EfiSystemTable ; SystemTable
mov     rcx, cs:DxeCoreImageHandle ; ImageHandle
call    FwVolBlockDriverInit
mov     rdx, cs:EfiSystemTable ; SystemTable
mov     rcx, cs:DxeCoreImageHandle ; ImageHandle
call    FwVolDriverInit
mov     rdx, cs:EfiSystemTable ; SystemTable
mov     rcx, cs:DxeCoreImageHandle ; ImageHandle
call    InitializeSectionExtraction
call    InitializeDispatcher
call    Dispatcher
call    AllEfiServicesAvailable
mov     ecx, 3041001h   ; Type
call    ReportStatusCode
mov     r11, cs:This
mov     rcx, r11        ; _QWORD
call    [r11+EFI_BDS_ARCH_PROTOCOL.Entry]
```

Obr. 6.9: Koniec hlavnej DXE Core funkcie

Kapitola 7

Diskusia

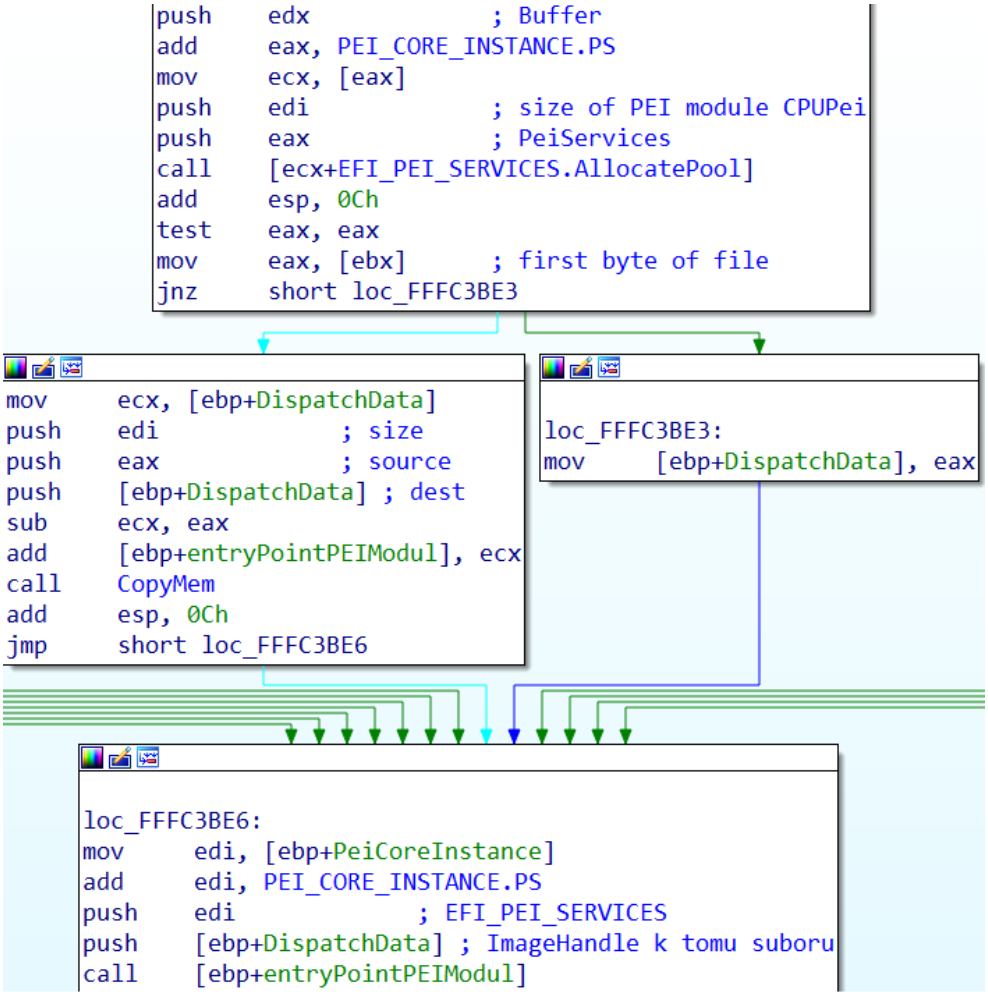
Pri analýze modulov PEI Core a DXE Core sme použil disassembler IDA Pro 7.4. Spustiteľné súbory PEI Core a DXE Core sme analyzovali staticky bez využitia debuggera. Debugovanie spustiteľného súboru pre PEI Core a DXE Core môžeme vykonať napríklad cez hardware rozhranie „Joint Test Action Group “ (JTAG) podľa návodu [23]. JTAG je hardware rozhranie, ktoré poskytuje komunikáciu medzi počítačom a čipom v základnej doske [24]. Debugovanie DXE Core a DXE modulov je možné aj pomocou špeciálneho emulátora napr. [25]

Počas analýzy PEI Core modulu sme si všimli, ako programátori pristupovali k bezpečnostným zvyklostiam, či vytvorený kód odpovedá špecifikáciám a existujúcim open source UEFI verziám.

Určitú mieru neistoty vzbudzuje úsek na obr. 7.1, ktorý sa nenachádza v open source UEFI verzii „EFI Development Kit“ (EDK). V tomto úseku PEI Core hľadá špecifický PEI modul „CPUPei“ s identifikátorom GUID 2BB5AFA9-FF33-417B-8497-CB773C2B93BF. Ak modul nájde, tak pomocou funkcie `AllocatePool` alokuje pool veľkosti 12928 bajtov. Po preskúmaní funkcie `AllocatePool` sme zistili, že sa vytvára HOB štruktúra s typom `EFI_HOB_TYPE_MEMORY_POOL`. DXE Core a ostatné DXE moduly túto HOB štruktúru ignorujú podľa dokumentácie [15]. Po úspešnom alokovaní pamäte prekopíruje modul CPUPei do alokovanej pamäte a zavolá začiatok funkcie modulu CPUPei. Po skončení modulu CPUPei by sa mala alokovaná pamäť uvoľniť, čo však v tomto prípade neplatí. Aj keď si všimneme používané PEI funkcie v `EFI_PEI_SERVICES`, nikde nenachádzame funkciu, ktorá by uvoľňovala vytvorenú pamäť. To nás privádza k, myšlienke čo sa stane, ak vytvoríme veľké množstvo pool pamäti pomocou funkcie `AllocatePool` a zaplníme tým pamäťové miesto. A ako zaplnené pamäťové miesto bezpečne uvoľníme?

Inak sa spustiteľné súbory PEI Core a DXE Core modulu zdali byť prehľadné, dobre čitateľné a programátori ošetrovali návratové hodnoty. V spustiteľnom súbore DXE Core modulu sme nenašli žiadne úseky kódu, ktoré sa ne-

nachádzali v open source UEFI verzií „EFI Development Kit“ (EDK). Môžeme s istotou povedať, že spustiteľný súbor DXE Core bol skompilovaný podľa open source UEFI verzie „EFI Development Kit“ (EDK).



Obr. 7.1: Alokovanie Pool pamäti pre CPUPei modul

Záver

V tejto diplomovej práci sme prostredníctvom nástrojov reverzného inžinierstva analyzovali hlavné UEFI moduly v PEI a DXE fáze. Najskôr sme získali Firmware zo SPI flash pamäte. Pomocou open-source softwaru Chipsec sa nám podarilo získať image UEFI firmwara. Získaný image UEFI firmwara sme otvorili pomocou UEFITool, ktorý vyextrahoval image UEFI firmwara do jednotlivých súborov. Z vyextrahovaného imagu UEFI firmwara sme získali PEI Core a moduly DXE Core. Jednotlivé súbory sú spustiteľné Windows súbory, kde pomocou disassemblera a špecialnych emulátorov existuje možnosť vykonať ich analýzu. S nástrojom IDA Pro a za pomoci metódy reverzného inžinierstva sme dokázali vykonať statickú analýzu získaných modulov. Výsledky analýzy sme popísali v kapitole päť a šesť.

DXE Core modul sa skladá z piatich hlavných komponentov. Po jeho preskúmaní pomocou techniky reverzného inžinierstva sme došli k záveru, že presne zodpovedá dokumentácií. Kód je písaný zrozumitel'ne a dodržuje programátorské zvyky, nenašli sme dôvody na znepokojenie. Po aktivácií týchto komponentov UEFI priamo spustí štartovacie súbory operačného systému a predá mu tým kontrolu nad systémom. Ďalší priebeh štartu počítača je už v réžii OS. Priebeh analýzy v nás vzbudzuje dôveru, pretože až do tejto fázy je štart systému riešený bezpečne a nie sú viditeľné žiadne známky škodlivého alebo rizikového chovania.

Určitú mieru neistoty však vzbudzuje práca s pamäťou v PEI fázy, ktorá obsahuje nedokumentované časti spustiteľného súboru. Tu sa nedodržiavajú správne programátorské zvyky. Domnievame sa, že by bolo zaujímavé zistiť, prečo bol zvolený takýto postup pri programovaní tejto fázy. Taktiež by bolo zaujímavé získať UEFI image z rôznych hardwarových značiek a preskúmať, s akými riešeniami inicializujú a pripravujú pamäť pre ďalšie UEFI fázy. Následne v nich skúmať taktiež správnosť a efektívnosť riešenia z pohľadu bezpečnosti.

Literatúra

[1] Joshi, A.; Gillespie, K.: UEFI on Dell BizClient Platforms. Januar 2013: s. 4 – 5.

[2] Rothman, M.; Zimmer, V.; Lewis, T.: *Harnessing the UEFI Shell: Moving the Platform Beyond DOS, Second Edition*. De Gruyter, Incorporated, 2017, ISBN 9781501505751. Dostupné z: `https://books.google.cz/books?id=s11SDgAAQBAJ`

[3] UEFI Forum: *Working Groups*. [Cited 2020-18-1]. Dostupné z: `https://uefi.org/workinggroups`

[4] Corporation, Compaq Computer and B, Revision: *Advanced Configuration and Power Interface Specification*. 2000, [Cited 2020-01-02]. Dostupné z: `http://www.acpi.info/`

[5] UEFI Forum: *UEFI Security working groups*. [Cited 2020-18-1]. Dostupné z: `https://uefi.org/security`

[6] Ecker, W.; Müller, W.; Dömer, R.: *Hardware-dependent Software: Principles and Practice*. Springer Netherlands, 2009, ISBN 9781402094361. Dostupné z: `https://books.google.cz/books?id=59WsE36uAzgC`

[7] Fan, X.: *Real-time embedded systems : design principles and engineering practices*. Kidlington, Oxford, UK: Newnes, 2015, ISBN 978-0-12-801507-0.

[8] Zimmer, V.; Rothman, M.; Marisetty, S.: *Beyond BIOS: Developing with the Unified Extensible Firmware Interface*. Intel Press, 2010, ISBN 9781934053294. Dostupné z: `https://books.google.cz/books?id=t_iwuAAACAAJ`

[9] Tianocore: *EDK II Driver Writer's Guide for UEFI 2.3.1*. April 2018.

[10] Matrosov, A.; Rodionov, E.; Bratus, S.: *Rootkits and bootkits: reversing modern malware and next generation threats*. No Starch Press, 2019.

[11] Intel Corporation: *Intel® 7 Series/C216 Chipset Family SPI Programming Guide*. June 2012.

[12] CHIPSEC: *CHIPSEC Platform Security Assesment Framework*. Dostupné z: `https://uefi.org/security`

[13] LongSoft: *UEFITool*. [Cited 2020-1-3]. Dostupné z: `https://github.com/LongSoft/UEFITool`

[14] Tianocore: *EDK II Build Specification*. August 2019, [Cited 2020-18-1]. Dostupné z: `https://edk2-docs.gitbooks.io/edk-ii-build-specification/content/2_design_discussion/23_boot_sequence.html`

[15] Intel Corporation: *Intel Platform Innovation Framework for EFI Hand-Off Block (HOB) Specification*. September 2003, [Cited 2020-01-02]. Dostupné z: `https://www.intel.com/content/www/us/en/architecture-and-technology/unified-extensible-firmware-interface/efi-specifications-general-technology.html`

[16] Wu, W.; Wu, H.; Lan, J.; aj.: UEFI Firmware Storage System Analysis. In *2016 5th International Conference on Environment, Materials, Chemistry and Power Electronics*, Atlantis Press, 2016/08, ISBN 978-94-6252-197-1, ISSN 2352-5401. Dostupné z: `https://doi.org/10.2991/emcpe-16.2016.40`

[17] UEFI Forum: *Platform Initialization Specification Volume 1:Pre-EFI Initialization Core Interface*. January 2019.

[18] Tianocore: *PI Boot Flow*. [Cited 2020-02-02]. Dostupné z: `https://github.com/tianocore/tianocore.github.io/wiki/PI-Boot-Flow`

[19] *Memory pools*. [Cited 2020-18-3]. Dostupné z: `https://talloc.samba.org/talloc/doc/html/libtalloc__pools.html`

[20] What is Memory Descriptor Lists? [Cited 2020-06-04]. Dostupné z: `http://dominique122.blogspot.com/2015/04/what-is-5-memory-descriptor-lists.html`

[21] UEFI Forum: *Unified Extensible Firmware Interface Specification*. January 2016.

[22] UEFI Forum: *Intel® Platform Innovation Framework for EFI Driver Execution Environment Core Interface Specification (DXE CIS)*. December 2004.

[23] Reed, T.: Debug UEFI code by single-stepping your Coffee Lake-S hardware CPU. [Cited 2020-07-04]. Dostupné z: https://casualhacking.io/blog/2019/6/2/debug-uefi-code-by-single-stepping-your-coffee-lake-s-hardware-cpu

[24] Senrio: *JTAG Explained: Why IoT, Software Security Engineers, and Manufacturers Should Care*. [Cited 2020-10-04]. Dostupné z: https://blog.senr.io/blog/jtag-explained

[25] CHIPSEC: *EFI DXE Emulator*. [Cited 2020-07-04]. Dostupné z: https://github.com/assafcarlsbad/efi_dxe_emulator

Dodatok **A**

Zoznam použitých skratiek

ACPI Advanced Configuration and Power Interface

AL After Life

BDS Boot Device Selection

BFV Boot Firmware Volume

BIOS Basic Input Output System

CAR Cache-as-RAM

COFF The Common Object File Format

CSM Compatibility Support Module

DEPEX Dependency Expresion

DXE Driver Execution Enviroment

EBC EFI Byte Code

EDK EFI Development Kit

EFI Extensible Firmware Interface

FV Firmware Volume

GBE Gigabit Ethernet

GUID Globally Unique Identifier

HAL Hardware Abstraction Layer

HII Human Interface Infrastructure

HOB Hand-Off Blocks

IBV Independent BIOS Vendor

ICWG Industry Communications Work Group

IPF Intel Processor family

IPL Initial Program Loader

ISV Independent Software Vendor

MCA Machine Check Architecture

ME Intel Management Engine

ODM Original Design Manufacturer

OEM Original Equipment manufacturer

PCI Peripheral Component interconnect

PDR Platform Data Region

PE Portable Executable

PEI Pre-EFI Initialization

PEIM PEI Module

PHIT Phase Hand-off Information Table

PI Platform Initialization

PIWG Platform Initialiazation Work Group

PMI Platform Management Interupt

PPI PEIM-to-PEIM

RAS Reliability, Availability, Supportability

RT Run Time

SAL System Abstraction Layer

SEC Security

SMBIOS System Management BIOS

SMM System Management Mode

SPI Serial Peripheral Interface

SST System Table

TLS Transient System Load

UART Universal Asynchronous Receiver-Transmitter

UEFI Unified Extensible Firmware Interface

USRT UEFI Security Response Team

USWG UEFI Specification Work Group

UTWG UEFI Testing Work Group

XIP Execute-in-Place

Dodatok **B**

Obsah priloženého CD

- `ASM` ..... adresár so exportovanými IDA súbormi
  - `PEICore.asm` ..... disassemblovaný spustiteľný súbor PEICore.efi
  - `DXECore.asm` ..... disassemblovaný spustiteľný súbor DXECore.efi
- `EFI` ..... EFI spustiteľné súbory
  - `PEICore.efi`
  - `DXECore.efi`
- `text` ..... text práce
  - `thesis.pdf` ..... text tejto práce vo formáte PDF