1. Санкт-Петербургский Политехнический Университет Петра Великого
2. **Институт кибербезопасности и защиты информации**

**ЛАБОРАТОРНАЯ РАБОТА № 1**

1. «Изучение безопасности UEFI BIOS на базе платформ Intel»
2. по дисциплине «Принципы построения, проектирования и эксплуатации автоматизированных информационных систем»
3. Выполнил
4. студент гр. 4851003/70801 Гасанов Э.А.

<*подпись*>

1. Преподаватель
2. старший преподаватель Чернов А.Ю.

<*подпись*>

1. Санкт-Петербург
2. 2021

**Цель**

Ознакомление с базовой архитектурой UEFI BIOS и их механизмами защиты.

**Задачи**

1. Изучить процесс загрузки ЭВМ на базе UEFI BIOS в соответствии со спецификацией
2. Получить образ UEFI BIOS вашего ПК/Ноутбука c официального сайта производителя.
3. С использованием утилиты UEFITool открыть образ UEFI BIOS
4. Определить модульный состав и точку входа для всех фаз функционирования UEFI BIOS.
5. С помощью утилиты CHIPSEC определить конфигурацию LPC/SPI-устройства
6. Определить поддержку технологий Intel Boot Guard, Intel BIOS Guard, Secure Boot и их конфигурацию. Сделать предположение о корректности заданной производителем конфигурации данных технологий.
7. В модульном составе UEFI BIOS выявить модуль Setup
8. С помощью утилиты Universal-IFR-Extractor извлечь IFR-информацию из Setup-модуля.
9. Выявить скрытые опции настройки UEFI BIOS через NVRAM-переменные
10. Сделать вывод о безопасности UEFI BIOS на исследуемой платформе.

**Вариант и характеристики**

Материнская плата ASUS UX310UAK-AS.312, AMI

* Intel Core i5 7200U (Kaby Lake – chipset С422)
* Версия BIOS 312; 4/18/2019
* Размер – 6 МБ, так как это урезанный файл обновления (по умолчанию должно быть 8 (8 390 656 байт))
* 8 Гб оперативной памяти; 2 ядра и 4 логических процессора
* Используется плагин для Ida Pro – efiXplorer и efiXloader

**Точка входа в Sec**

Для того чтобы найти точку входа в Sec нужно найти ассемблерный код в реальном режиме, настраивающий таблицу прерываний.

Обычно точка входа экспортируется и этот случай не исключение. Находим ModuleEntrypoint и будем сравнивать исходный код со слитыми реализациями Ivy Bridge и открытой реализацией Tiano Core.

|  |  |
| --- | --- |
| Дизассемблер | Слитый Ivy Bridge/Tiano Core |
| \_ModuleEntryPoint:  \_TEXT\_RE:FFFFF218  \_TEXT\_RE:FFFFF218 fninit  \_TEXT\_RE:FFFFF21A movd mm0, eax | align 4  Flat32Start PROC NEAR C PUBLIC  ;  ; Save BIST state in MM0  ;  fninit; clear any pending Floating point exceptions  movd mm0, eax  файл (Flat32.asm) |

Далее дизассемблер не смог сгенерировать ассемблерный код (рисунок 1).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1 – Не дизассмеблированный код

Поэтому будем пользоваться опкодами. По определению мы знаем, что должна настраиваться таблица прерываний –

( lgdt fword ptr cs:[si] ). Её опкод равен 2e 0f 01 14

Найдём её опкод в дампе (рисунок 2).

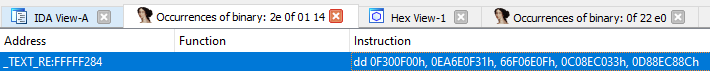


Рисунок 2 – настройка таблицы прерываний.

Так как это единственное вхождение и число 0x0F300F00 совпадают с рисунком выше этот ModuleEntryPoint и есть точка входа SEC.

Теперь воспользуемся созданием кода нажатием на клавишу С по 0x0F300F00. Нашлись те же строчки, что и были в лекции (рисунок 3 и таблицы ниже):



Рисунок 3 – начало sec.

|  |  |
| --- | --- |
| Дизассемблер | Слитый Ivy Bridge |
|  | lgdt fword ptr cs:[si]  mov eax, cr0 ; Get control register 0  or eax, 00000003h ; Set PE bit (bit #0) & MP bit (bit #1)  mov cr0, eax ; Activate protected mode  mov eax, cr4 ; Get control register 4  or eax, 00000600h; Set OSFXSR bit (bit #9) & OSXMMEXCPT bit (bit#10) |

Теперь был совершен переход в защищенный режим

|  |  |
| --- | --- |
| Дизассемблер | Слитый Ivy Bridge |
|  | mov ax, SYS\_DATA\_SEL  mov ds, ax  mov es, ax  mov fs, ax  mov gs, ax  mov ss, ax |

И сам переход в защищенный режим

|  |  |
| --- | --- |
| Дизассемблер | Слитый Ivy Bridge |
|  | jmp fword ptr cs:[si] |

И теперь цикл, до которого не нужно доходить

|  |  |
| --- | --- |
| Дизассемблер | Слитый Ivy Bridge |
|  | TightLoop:  cli  hlt  jmp TightLoop |

Собираем таблицу прерываний руками. Для этого находим в исходнике Flat32.asm упоминание о GDT(PUBLIC BootGDTtable). Смотрим на типы данных :

|  |
| --- |
| BootGDTtable DD 0  DD 0  ;  ; Linear data segment descriptor  ;  LINEAR\_SEL EQU $ - GDT\_BASE ; Selector [0x8]  DW 0FFFFh ; limit 0xFFFFF  DW 0 ; base 0  DB 0  DB 092h ; present, ring 0, data, expand-up, writable  DB 0CFh ; page-granular, 32-bit  DB 0 |

Нажимая на клавишу d превратим тип данных byte в dword.

Изображение выглядит как стол

Автоматически созданное описание



Рисунок 4 – Таблица прерываний.

Чтобы попасть в SecStartup найдём константу, чтобы по ней понять, где происходит передача управления

|  |  |
| --- | --- |
| Дизассемблер | Ivy |
|  | ; Pass BFV into the PEI Core  ;  mov edi, FV\_MAIN\_BASE ; 0FFFFFFFCh  push DWORD PTR ds:[edi]  ;  ; Pass stack size into the PEI Core  ;  push DATA\_STACK\_SIZE  ;  ; Pass Control into the PEI Core  ;  call SecStartup |

Таким образом, sub\_FFFFC6D8 – это SecStartUp

Далее найдём определение заполняемой структуры:

Изображение выглядит как текст

Автоматически созданное описание

|  |  |
| --- | --- |
| Дизассемблер | EDK2 (SecMain.c) ФУНКЦИЯ - SecStartup |
| v12 = 36;  v13 = a3;  v16 = a1;  v14 = -a3;  v18 = a1 - v3;  v15 = a2;  v17 = a2;  result = sub\_FFFFC7EB;  v19 = a2 + a1 - v3;  v20 = a1 >> 1; | SecCoreData.DataSize=(UINT16) sizeof (EFI\_SEC\_PEI\_HAND\_OFF);  SecCoreData.BootFirmwareVolumeBase=BootFirmwareVolume;  SecCoreData.BootFirmwareVolumeSize=(UINTN)((EFI\_FIRMWARE\_VOLUME\_HEADER \*) BootFirmwareVolume)->FvLength;  SecCoreData.TemporaryRamBase=(VOID\*)(UINTN) TempRamBase;  SecCoreData.TemporaryRamSize=SizeOfRam;  SecCoreData.PeiTemporaryRamBase=SecCoreData.TemporaryRamBase;  SecCoreData.PeiTemporaryRamSize=SizeOfRam - PeiStackSize;  SecCoreData.StackBase=(VOID\*)(UINTN)(TempRamBase + SecCoreData.PeiTemporaryRamSize);  SecCoreData.StackSize=PeiStackSize; |

|  |  |
| --- | --- |
| Дизассемблер | EDK2 |
| if ( sub\_FFFFC7EB )  return (\_DWORD (\_\_cdecl \*)(\_DWORD))sub\_FFFFC7EB(&v12);    return result; | InitializeDebugAgent (DEBUG\_AGENT\_INIT\_PREMEM\_SEC, &SecCoreData, SecStartupPhase2);  //  // Should not come here.  //  UNREACHABLE (); |

Таким образом, sub\_FFFFC7EB - это SecStartupPhase2

|  |  |
| --- | --- |
| Дизассемблер | EDK2 |
| int \_\_cdecl SecStartupPhase2(int a1) | SecStartupPhase2( IN VOID \*Context) |

В исходниках EDK происходит прямой каст SecCoreData c переданным Context:

SecCoreData = (EFI\_SEC\_PEI\_HAND\_OFF \*) Context;

По сути, а1 и SecCoreData теперь одно и тоже

Так как эта функция будет использовать указатель на функцию и в последствии ею и вызываться необходимо обратить внимание на

int (\_\_cdecl \*v8)(int, char \*);

Он является указателем на функцию передачи управления на Pei.

|  |  |
| --- | --- |
| Дизассемблер | EDK2 |
| PeiCoreEntryPoint(a1, &byte\_FFFFFFAC[(\_DWORD)&savedregs]); | // Transfer the control to the PEI core  (\*PeiCoreEntryPoint) (SecCoreData, PpiList); |

**PEI**

Точка входа

|  |  |
| --- | --- |
| Дизассемблер | EDK2 (PeiMain.c) |
| int \_\_cdecl sub\_FFF21001(  char \*a1,  int a2,  int a3  ) | PeiCore (  IN CONST EFI\_SEC\_PEI\_HAND\_OFF \*SecCoreDataPtr,  IN CONST EFI\_PEI\_PPI\_DESCRIPTOR \*PpiList,  IN VOID \*Data  ) |

Переданный параметр a1 – это SecCoreDataPtr. В исследуемом исходнике так же происходит явное преобразование типов данных

SecCoreData = (EFI\_SEC\_PEI\_HAND\_OFF \*) SecCoreDataPtr;.

Если в дизассемблере увидим a1 и SecCoreData и Attributes – будем считать это одним и тем же.

a3 - это Data. Но в исходнике преобразуется в OldCoreData, поэтому

a3⬄OldCoreData⬄Data

Мы не будем восстанавливать структуры, но будем ставить соответствие:

смещение ⬄ поле структуры

|  |  |
| --- | --- |
| Дизассемблер | EDK2 (PeiMain.c) |
| if ( OldCoreData )  {  Подразумевается, что OldCoreData!=NULL, поэтому в исходнике сразу смотрим в else  if ( !\*(\_DWORD \*)(OldCoreData + 272) )  {  \*(\_DWORD \*)(OldCoreData + 4) = OldCoreData + 88;  \*(\_DWORD \*)(OldCoreData + 80) = OldCoreData + 184;  v3 = \*(\_BYTE \*)(OldCoreData + 260) == 0;  v4 = \*(\_DWORD \*)(OldCoreData + 256); (это HeapOffset)  if ( v3 )  {  \*(\_DWORD \*)(OldCoreData + 72) -= v4;  \*(\_DWORD \*)(OldCoreData + 40) -= v4;  \*(\_DWORD \*)(OldCoreData + 48) -= v4;  \*(\_DWORD \*)(OldCoreData + 28) -= v4;  \*(\_DWORD \*)(OldCoreData + 36) -= v4;  for ( i = 0; i < 0x150; i += 28 )  {  \*(\_DWORD \*)(\*(\_DWORD \*)(OldCoreData + 36) + i + 12) -= \*(\_DWORD \*)(OldCoreData + 256);    \*(\_DWORD \*)(\*(\_DWORD \*)(OldCoreData + 36) + i + 16) -= \*(\_DWORD \*)(OldCoreData + 256);  }  v8 = \*(\_DWORD \*)(OldCoreData + 256);  \*(\_DWORD \*)(OldCoreData + 564) -= v8;  \*(\_DWORD \*)(OldCoreData + 560) -= v8;  }  else  {  \*(\_DWORD \*)(OldCoreData + 72) += v4;  \*(\_DWORD \*)(OldCoreData + 40) += v4;  \*(\_DWORD \*)(OldCoreData + 48) += v4;  \*(\_DWORD \*)(OldCoreData + 28) += v4;  \*(\_DWORD \*)(OldCoreData + 36) += v4;  for ( j = 0; j < 0x150; j += 28 )  {  \*(\_DWORD \*)(\*(\_DWORD \*)(OldCoreData + 36) + j + 12) += \*(\_DWORD \*)(OldCoreData + 256);  \*(\_DWORD \*)(\*(\_DWORD \*)(OldCoreData + 36) + j + 16) += \*(\_DWORD \*)(OldCoreData + 256);  }  v6 = \*(\_DWORD \*)(OldCoreData + 256);  \*(\_DWORD \*)(OldCoreData + 564) += v6;  \*(\_DWORD \*)(OldCoreData + 560) += v6;  v9 = \*(\_DWORD \*)(OldCoreData + 72);  v10 = \*(\_DWORD \*)(OldCoreData + 256);    if ( \*(\_BYTE \*)(OldCoreData + 260) )  \*(\_QWORD \*)(v9 + 48) += v10;  else  \*(\_QWORD \*)(v9 + 48) -= v10;  \*(\_QWORD \*)(v9 + 16) = \*(\_QWORD \*)(OldCoreData + 232) + \*(\_QWORD \*)(OldCoreData + 240);  \*(\_DWORD \*)(v9 + 24) = \*(\_DWORD \*)(OldCoreData + 232);  \*(\_DWORD \*)(v9 + 36) = \*(\_DWORD \*)(OldCoreData + 252);  \*(\_QWORD \*)(v9 + 40) = \*(\_DWORD \*)(v9 + 48)) + 8;  \*(\_BYTE \*)(OldCoreData + 77) = 1;  \*(\_BYTE \*)(OldCoreData + 70) = 1;  v12 = (void (\_\_cdecl \*)(char \*, int, int))ShadowedPeiCore();  \*(\_DWORD \*)(OldCoreData + 272) = v12;  v12(a1, a2, OldCoreData);  sub\_FFF24076();  sub\_FFF24367(v36, a1, 36);  Attributes = v36;  sub\_FFF24367(&DataSize, OldCoreData, 640);  v13 = v51;  v33[0] = v52;  sub\_FFF24367(v50, aPeiServ, 136);  v51 = v13;  v52 = v33[0];  sub\_FFF23928(Attributes, &DataSize);  if ( \*(\_DWORD \*)&VariableGuid->Data4[4] != 17 && !v49 )  sub\_FFF24444(-2147483646, 50466818); | if (OldCoreData == NULL) {  else {  //  // Memory is available to the PEI Core. See if the PEI Core has been shadowed to memory yet.  if (OldCoreData->ShadowedPeiCore == NULL)  {  OldCoreData->Ps = &OldCoreData->ServiceTableShadow;  OldCoreData->CpuIo= &OldCoreData->ServiceTableShadow.CpuIo;  if (OldCoreData->HeapOffsetPositive)  { Отметим ,что тут работаем с HeapOffset !  Строчки 235-251 до цикла for в PeiMain.c  for (Index = 0; Index < OldCoreData->FvCount; Index ++) {  if (OldCoreData->Fv[Index].PeimState != NULL) {  OldCoreData->Fv[Index].PeimState = (UINT8 \*) OldCoreData->Fv[Index].PeimState - OldCoreData->HeapOffset;  }  if (OldCoreData->Fv[Index].FvFileHandles != NULL) {  OldCoreData->Fv[Index].FvFileHandles = (EFI\_PEI\_FILE\_HANDLE \*) ((UINT8 \*) OldCoreData->Fv[Index].FvFileHandles - OldCoreData->HeapOffset);  }  }  OldCoreData->TempFileGuid = (EFI\_GUID \*) ((UINT8 \*) OldCoreData->TempFileGuid - OldCoreData->HeapOffset);  OldCoreData->TempFileHandles = (EFI\_PEI\_FILE\_HANDLE \*) ((UINT8 \*) OldCoreData->TempFileHandles - OldCoreData->HeapOffset);  }  else  Строчки 207-223 до цикла for в PeiMain.c  for (Index = 0; Index < OldCoreData->FvCount; Index ++) {  if (OldCoreData->Fv[Index].PeimState != NULL) {  OldCoreData->Fv[Index].PeimState = (UINT8 \*) OldCoreData->Fv[Index].PeimState + OldCoreData->HeapOffset;  }  if (OldCoreData->Fv[Index].FvFileHandles != NULL) {  OldCoreData->Fv[Index].FvFileHandles = (EFI\_PEI\_FILE\_HANDLE \*) ((UINT8 \*) OldCoreData->Fv[Index].FvFileHandles + OldCoreData->HeapOffset);  }  }  OldCoreData->TempFileGuid=(EFI\_GUID \*) ((UINT8 \*) OldCoreData->TempFileGuid + OldCoreData->HeapOffset);  OldCoreData->TempFileHandles = (EFI\_PEI\_FILE\_HANDLE \*) ((UINT8 \*) OldCoreData->TempFileHandles + OldCoreData->HeapOffset);  HandoffInformationTable =OldCoreData>HobList.HandoffInformationTable;  if (OldCoreData->HeapOffsetPositive)  {  HandoffInformationTable->EfiEndOfHobList = HandoffInformationTable>EfiEndOfHobList + OldCoreData->HeapOffset;    }  else  {  HandoffInformationTable->EfiEndOfHobList =  HandoffInformationTable->EfiEndOfHobList - OldCoreData->HeapOffset;  }  HandoffInformationTable->EfiMemoryTop = OldCoreData->PhysicalMemoryBegin + OldCoreData->PhysicalMemoryLength;  HandoffInformationTable->EfiMemoryBottom = OldCoreData->PhysicalMemoryBegin;  HandoffInformationTable->EfiFreeMemoryTop= OldCoreData>FreePhysicalMemoryTop;  HandoffInformationTable->EfiFreeMemoryBottom = HandoffInformationTable->EfiEndOfHobList + sizeof (EFI\_HOB\_GENERIC\_HEADER);  // After the whole temporary memory is migrated, then we can allocate page in  // permanent memory.  //  OldCoreData->PeiMemoryInstalled = TRUE;  //  // Indicate that PeiCore reenter  //  OldCoreData->PeimDispatcherReenter = TRUE;  OldCoreData->ShadowedPeiCore (SecCoreData, PpiList, OldCoreData);  CpuDeadLoop()  //  // Memory is available to the PEI Core and the PEI Core has been shadowed to memory.  //  CopyMem (&NewSecCoreData, SecCoreDataPtr, sizeof (NewSecCoreData));  SecCoreData = &NewSecCoreData;  CopyMem (&PrivateData, OldCoreData, sizeof (PrivateData));  CpuIo = (VOID\*)PrivateData.ServiceTableShadow.CpuIo;  PciCfg = (VOID\*)PrivateData.ServiceTableShadow.PciCfg;  CopyMem (&PrivateData.ServiceTableShadow, &gPs, sizeof (gPs));  PrivateData.ServiceTableShadow.CpuIo = CpuIo;  PrivateData.ServiceTableShadow.PciCfg = PciCfg;  // Call PEIM dispatcher  //  PeiDispatcher (SecCoreData, &PrivateData);  if (PrivateData.HobList.HandoffInformationTable->BootMode != BOOT\_ON\_S3\_RESUME) {  //  // Check if InstallPeiMemory service was called on non-S3 resume boot path.  //  ASSERT(PrivateData.PeiMemoryInstalled == TRUE);  } |
| Передача в DXE IPL | |
| if ( (int)sub\_FFF2427F(&EFI\_DXE\_IPL\_PPI\_GUID\_FFF25704, 0, 0, &This) < 0 )  {  sub\_FFF24444(-2147483646, 50466817);  CpuDeadLoop();  }  This->GetVariable(This, VariableName, VariableGuid);    return CpuDeadLoop(); | //  // Lookup DXE IPL PPI  //  Status = PeiServicesLocatePpi (  &gEfiDxeIplPpiGuid,  0,  NULL,  (VOID \*\*)&TempPtr.DxeIpl  );  ASSERT\_EFI\_ERROR (Status);  if (EFI\_ERROR (Status))  {  //  // Report status code to indicate DXE IPL PPI could not be found.  //  REPORT\_STATUS\_CODE (  EFI\_ERROR\_CODE | EFI\_ERROR\_MAJOR,  (EFI\_SOFTWARE\_PEI\_CORE | EFI\_SW\_PEI\_CORE\_EC\_DXEIPL\_NOT\_FOUND)  );  CpuDeadLoop ();  }  //  // Enter DxeIpl to load Dxe core.  //  DEBUG ((EFI\_D\_INFO, "DXE IPL Entry\n"));  Status = TempPtr.DxeIpl->Entry (  TempPtr.DxeIpl,  &PrivateData.Ps,  PrivateData.HobList  );  CpuDeadLoop(); |

Прежде чем управление передастся в Dxe, управление передаётся в DxeIpl (согласно исходникам и рисунку 5 )

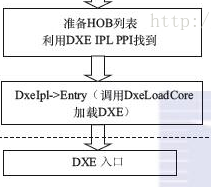


Рисунок 5 – Передача управления.

**DXE IPL PPI**

Last PEIM.(Responsibility of this module is to load the DXE Core from a Firmware Volume.) Последний модуль, загружающий уже сам DXE.

|  |  |
| --- | --- |
| Дизассемблер | EDK2 (DxeLoad.c) |
| int \_\_cdecl sub\_FFEC21C8(int a1)  if ( GetBootModeHob(v1) == 17 )  {  v4 = (const EFI\_PEI\_SERVICES \*\*)sub\_FFEC2AA8();  (\*v4)->NotifyPpi(v4, &NotifyList);  return sub\_FFEC2CE8((EFI\_PEI\_PPI\_DESCRIPTOR \*)&unk\_FFEC53B0);  }  (else)  v2 = sub\_FFEC2AA8();  result = (\*(int (\_\_stdcall \*\*)(int))(\*(\_DWORD \*)v2 + 116))(a1);  v5 = (int \*)&v5;  result = ((int (\_\_cdecl \*)(EFI\_GUID \*, UINTN))PeiServicesLocatePpi)(  &EFI\_PEI\_PERMANENT\_MEMORY\_INSTALLED\_PPI\_GUID\_FFEC52D8,  0);  Важно, что guid на самом деле такой, проверка – в guids.csv Н.Шлея  F894643D-C449-42D1-8EA8-85BDD8C65BDE=EfiPeiMemoryDiscoveredPpiGuid  sub\_FFEC2230(0, 0, 0);  v4->Flags = -2147483632;  v4->Ppi = &off\_FFEC53BC;  v4->Guid = v3;  и внутри функции обёртка sub\_FFEC2DA0 для AllocatePool  return sub\_FFEC2CE8((EFI\_PEI\_PPI\_DESCRIPTOR \*)&unk\_FFEC53B0);  внутри неё InstallPpi | Entry point of DXE IPL PEIM.  This function installs DXE IPL PPI. It also reloads  itself to memory on non-S3 resume boot path.  PeimInitializeDxeIpl (  IN EFI\_PEI\_FILE\_HANDLE FileHandle,  IN CONST EFI\_PEI\_SERVICES \*\*PeiServices  )  BootMode = GetBootModeHob (); //проверка на BOOT\_ON\_S3\_RESUME  else {  //  // Install memory discovered PPI notification to install PPIs for  // decompression and section extraction.  //  Status = PeiServicesNotifyPpi (&mMemoryDiscoveredNotifyList);  ASSERT\_EFI\_ERROR (Status);  }  if (BootMode != BOOT\_ON\_S3\_RESUME) {  Status = PeiServicesRegisterForShadow (FileHandle);  // DXE core load requires permanent memory.  Status = PeiServicesLocatePpi (  &gEfiPeiMemoryDiscoveredPpiGuid,  0,  NULL,  (VOID \*\*) &Dummy  );  // Now the permanent memory exists, install the PPIs for decompression and section extraction.  Status = InstallIplPermanentMemoryPpis (NULL, NULL, NULL);  Внутри используется AllocatePool и заполняется структура  GuidPpi->Flags  GuidPpi->Ppi  GuidPpi->Guid  // Install DxeIpl PPI.  Status = PeiServicesInstallPpi (&mDxeIplPpiList);  return Status; |

Main entry point to last PEIM. This function finds DXE Core in the firmware volume and transfer the control to DXE core.

|  |  |
| --- | --- |
| Дизассемблер | EDK2 (DxeLoad.c) |
| int \_\_cdecl sub\_FFEC26E8(int a1, EFI\_PEI\_SERVICES \*\*PeiServices, int a3)  v5 = sub\_FFEC2B9E();  EFI\_BOOT\_MODE \_\_thiscall sub\_FFEC2B9E(void \*this)  {  const EFI\_PEI\_SERVICES \*\*v1; // eax  EFI\_BOOT\_MODE BootMode; // [esp+0h] [ebp-4h] BYREF  BootMode = (EFI\_BOOT\_MODE)this;  v1 = (const EFI\_PEI\_SERVICES \*\*)sub\_FFEC2AA8();  (\*v1)->GetBootMode(v1, &BootMode);  return BootMode;  }  Уже известно про 17,  if ( v5 == 17 )  {  if ( (int)sub\_FFEC24DD(v6) < 0 ) [Внутри sub\_FFEC24DD лежит sub\_FFEC2CFA]  {  sub\_FFEC3831(0x80000002, 0x3031005u);  v9 = (const EFI\_PEI\_SERVICES \*\*)sub\_FFEC2AA8(v8);  (\*v9)->ResetSystem(v9);  }  }  EFI\_ERROR\_CODE=0x2  EFI\_ERROR\_MAJOR= 0x80000000  do  {  v12 = sub\_FFEC2D17(v4++, &v17);  if ( v12 < 0 ) – если не смогли найти firmware volumes  {  REPORT\_STATUS\_CODE(0x80000002, 0x3021000u);  return -2147483634;  }  FileHandle = 0;  v10 = (const EFI\_PEI\_SERVICES \*\*)sub\_FFEC2AA8(v17); нужна для иниц. PeiService  }  while ( ((\*v10)->FfsFindNextFile(v10, 5u, v11, &FileHandle) &0x80000000) != 0 );  Внутри функции sub\_FFEC2D17 функция FfsFindNextVolume  Назовем её Wrapper\_ FfsFindNextVolume  V3=0  do  {  PeiServicesLocatePpi(&EFI\_PEI\_LOAD\_FILE\_GUID\_FFEC52A8, v3++, (void \*\*)&This);  }  while ( (This->GetVariable(This, (const CHAR16 \*)FileHandle, &VariableGuid, (UINT32 \*)&Attributes, &DataSize, Data) & 0x80000000) != 0 );  (gEfiHobMemoryAllocModuleGuid)  sub\_FFEC2BE1(  VariableGuid.Data1,  \*(\_DWORD \*)&VariableGuid.Data2,  Attributes + (-(int)Attributes & 0xFFF),  (Attributes + (unsigned \_\_int64)(-(int)Attributes & 0xFFF)) >> 32, (align)  DataSize,  v21);  sub\_FFEC285F(DataSize, v21, a3);  sub\_FFEC2AC2(); //внутри while(1)  return -2147483639; | DxeLoadCore (  IN CONST EFI\_DXE\_IPL\_PPI \*This,  IN EFI\_PEI\_SERVICES \*\*PeiServices,  IN EFI\_PEI\_HOB\_POINTERS HobList  )  BootMode = GetBootModeHob ();  //if in S3 Resume, restore configure  if (BootMode == BOOT\_ON\_S3\_RESUME) {  Status = PeiServicesLocatePpi (  &gEfiPeiS3Resume2PpiGuid,  0,  NULL,  (VOID \*\*) &S3Resume  );  if (EFI\_ERROR (Status)) {  //  // Report Status code that S3Resume PPI can not be found  //  REPORT\_STATUS\_CODE (  EFI\_ERROR\_CODE | EFI\_ERROR\_MAJOR,  (EFI\_SOFTWARE\_PEI\_MODULE | EFI\_SW\_PEI\_EC\_S3\_RESUME\_PPI\_NOT\_FOUND)  );  }  Аналог функции DxeIplFindDxeCore - Searches DxeCore in all firmware Volumes and loads the first instance that contains DxeCore.  Traverse all firmware volume instances  //Load the DXE Core from a Firmware Volume.  do {  Status = PeiServicesLocatePpi (&gEfiPeiLoadFilePpiGuid, Instance++, NULL, (VOID \*\*) &LoadFile);  //  // These must exist an instance of EFI\_PEI\_LOAD\_FILE\_PPI to support to load DxeCore file handle successfully.  //  ASSERT\_EFI\_ERROR (Status);  Status = LoadFile->LoadFile (  LoadFile,  FileHandle,  &DxeCoreAddress,  &DxeCoreSize,  &DxeCoreEntryPoint,  &AuthenticationState  );  } while (EFI\_ERROR (Status));  // Add HOB for the DXE Core  //  //Create module allocation HOB for DXE Core  CreateHobMemoryAllocationModule(  PeiServices,  DxeCoreAddress,  DxeCoreSize,  EfiBootServicesCode,  &pFile->Name,  DxeCoreEntryPoint  )  // Transfer control to the DXE Core  // The hand off state is simply a pointer to the HOB list  //  HandOffToDxeCore (DxeCoreEntryPoint, HobList);  // If we get here, then the DXE Core returned. This is an error  // DxeCore should not return.  CpuDeadLoop ();  return EFI\_OUT\_OF\_RESOURCES; |

**DXE**

На этом этапе уже необходимо пользоваться 64-битой Ida, так как файл 64-битный и в его Hex-представлении есть эта запись (PE..d†), свидетельствующая, что это 64-битный файл.

DXE Core Main Entry Point

|  |  |
| --- | --- |
| Дизассемблер | EDK2 (DxeMain.c) |
| void \_\_fastcall sub\_2C8(\_\_int64 a1)  v2 = 0i64;  v3 = sub\_DD40(&EFI\_PEI\_VECTOR\_HANDOFF\_INFO\_PPI\_GUID\_198E0, a1);  if ( v3 )  v2 = v3 + 24;  sub\_3764((\_\_int16 \*\*)&HobStart, &v68, &v67);  qword\_1FB78 = sub\_DAC4(6i64, 120i64, (\_\_int64)aIbiSyst);  off\_1CF98 = (char \*)sub\_DAC4(6i64, 136i64, (\_\_int64)aRuntserv);  \*(\_QWORD \*)(qword\_1FB78 + 88) = off\_1CF98;  sub\_3C7C((\_\_int64 \*)&HobStart, MemoryBaseAddress, MemoryLength);  sub\_A0DC(&DXE\_SERVICES\_TABLE\_GUID\_195A0, aDxeServ);  sub\_A0DC(&EFI\_HOB\_LIST\_GUID\_19590, HobStart);  sub\_A0DC(&EFI\_MEMORY\_TYPE\_INFORMATION\_GUID\_195B0, dword\_1C8F0);  GuidHob = (char \*)GetNextGuidHob(&EFI\_PEI\_VECTOR\_HANDOFF\_INFO\_PPI\_GUID\_198E0, HobStart);  if ( GuidHob )  {  v47 = (\_\_int64)(GuidHob + 24);  v48 = 1i64;  for ( j = GuidHob + 28; \*j != 0x80000000; j += 6 )  ++v48;  v50 = AllocateRuntimeCopyPool(4i64, 24 \* v48, v47);  CoreInstallConfigurationTable(&gEfiVectorHandoffTableGuid\_195F0, v50);  }  sub\_8DD4(&unk\_1FB30, &EFI\_DECOMPRESS\_PROTOCOL\_GUID\_196B0, &off\_1D1C0, 0i64);  (\*(void (\_\_fastcall \*\*)(\_\_int64))qword\_1FB60)(qword\_1FB60); | Main entry point to DXE Core.  VOID EFIAPI DxeMain (IN VOID \*HobStart)  // Setup the default exception handlers  VectorInfoList = NULL;  GuidHob = GetNextGuidHob (&gEfiVectorHandoffInfoPpiGuid, HobStart);  if (GuidHob != NULL) {  VectorInfoList = (EFI\_VECTOR\_HANDOFF\_INFO \*) (GET\_GUID\_HOB\_DATA(GuidHob));  }  // Initialize Memory Services  CoreInitializeMemoryServices (&HobStart, &MemoryBaseAddress, &MemoryLength);  // Allocate the EFI System Table and EFI Runtime Service Table from EfiRuntimeServicesData  // Use the templates to initialize the contents of the EFI System Table and EFI Runtime Services Table  //  gDxeCoreST = AllocateRuntimeCopyPool (sizeof (EFI\_SYSTEM\_TABLE), &mEfiSystemTableTemplate);  ASSERT (gDxeCoreST != NULL);  gDxeCoreRT = AllocateRuntimeCopyPool (sizeof (EFI\_RUNTIME\_SERVICES), &mEfiRuntimeServicesTableTemplate);    ASSERT (gDxeCoreRT != NULL);  gDxeCoreST->RuntimeServices = gDxeCoreRT;  // Initialize the Global Coherency Domain Services  Status = CoreInitializeGcdServices (&HobStart, MemoryBaseAddress, MemoryLength);  // Install the DXE Services Table into the EFI System Tables's Configuration Table  Status = CoreInstallConfigurationTable (&gEfiDxeServicesTableGuid, gDxeCoreDS);    // Install the HOB List into the EFI System Tables's Configuration Table  Status = CoreInstallConfigurationTable (&gEfiHobListGuid, HobStart);    // Install Memory Type Information Table into the EFI System Tables's Configuration Table  Status = CoreInstallConfigurationTable (&gEfiMemoryTypeInformationGuid, &gMemoryTypeInformation);    // Get persisted vector hand-off info from GUIDeed HOB again due to HobStart may be updated, and install configuration table  GuidHob = GetNextGuidHob (&gEfiVectorHandoffInfoPpiGuid, HobStart);  if (GuidHob != NULL)  {  VectorInfoList = (EFI\_VECTOR\_HANDOFF\_INFO \*) (GET\_GUID\_HOB\_DATA(GuidHob));  VectorInfo = VectorInfoList;  Index = 1;  while (VectorInfo->Attribute != EFI\_VECTOR\_HANDOFF\_LAST\_ENTRY) {  VectorInfo ++;  Index ++;  }  VectorInfo = AllocateCopyPool (sizeof (EFI\_VECTOR\_HANDOFF\_INFO) \* Index, (VOID \*) VectorInfoList);  ASSERT (VectorInfo != NULL);  Status = CoreInstallConfigurationTable (&gEfiVectorHandoffTableGuid, (VOID \*) VectorInfo);  ASSERT\_EFI\_ERROR (Status);  }  // Publish the EFI, Tiano, and Custom Decompress protocols for use by other DXE components  Status = CoreInstallMultipleProtocolInterfaces (  &mDecompressHandle,  &gEfiDecompressProtocolGuid, &gEfiDecompress,  NULL  );  // Transfer control to the BDS Architectural Protocol  //  gBds->Entry (gBds); |

**BDS**

Плагин так же нашёл точку входа, но я сделал это сначала вручную, и только потом понял, что можно загрузить весь образ в Loader.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 6 – Точка входа, которую нашёл плагин.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 7 – Точка входа, которую нашёл я.

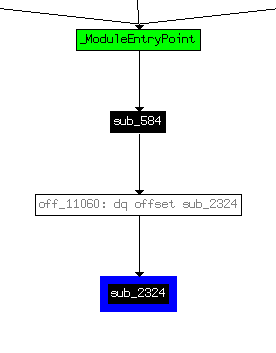


Рисунок 8 – Граф, где косвенно вызывается функция 2324, которая проделывает все инициализации.

|  |  |
| --- | --- |
| Дизассемблер | EDK2(BdsEntry.c) +BDS.c Ivy Bridge. Функция- BdsEntry |
| if ( ((\_\_int64 (\_\_fastcall \*)(\_\_int64, \_\_int64, CHAR16 \*\*))BootServices->AllocatePool)(  6i64,  2 \* v6 + 2,  &v2->FirmwareVendor) >= 0 )  {  FirmwareVendor = gST\_12838->FirmwareVendor;  do  {  v9 = \*v5++;  \*FirmwareVendor++ = v9;  }  while ( v9 );  }  gST\_12838->FirmwareRevision = 0x5000C;  return gBS\_12840->InstallMultipleProtocolInterfaces(  &Handle,  &EFI\_BDS\_ARCH\_PROTOCOL\_GUID\_FA30,  &off\_11060,  &gAmiBdsConnectPolicyProtocolGuid\_FB10,  &off\_110B0,  0i64);  Теперь проверим функцию 2324. Она хитрая и вызывает кучу других функций-инициализаций, подобно тем, что есть в BDSEntry.  DataSize = 0i64;  if ( gRT\_12848->GetVariable(aTimeout, &EFI\_GLOBAL\_VARIABLE\_GUID\_11000, 0i64, &DataSize, 0i64) == 0x800000000000000Eui64 )  gRT\_12848->SetVariable(aTimeout, &EFI\_GLOBAL\_VARIABLE\_GUID\_11000, 7u, 2ui64, &unk\_111A0);  EfiCreateProtocolNotifyEvent(  &EFI\_AMI\_LEGACYBOOT\_PROTOCOL\_GUID\_10A78,  (EFI\_TPL)sub\_14F0,  0i64,  &NotifyContext,  (void \*\*)&DataSize);  gBS\_12840->LocateHandleBuffer(AllHandles, 0i64, 0i64, &qword\_12640, &qword\_12648);  result = funcs\_23EA[0];  v1 = 0i64;  while ( result )  {  result();  result = funcs\_23EA[++v1]; --здесь происходит последовательный вызов функций инициализаций как в edk2  }  return result; | Service routine for BdsInstance->Entry(). Devices are connected, the  consoles are initialized, and the boot options are tried.  @param This Protocol Instance structure.  FirmwareVendor = (CHAR16 \*) PcdGetPtr (PcdFirmwareVendor);  gST->FirmwareVendor = AllocateRuntimeCopyPool (StrSize (FirmwareVendor), FirmwareVendor);  ASSERT (gST->FirmwareVendor != NULL);  gST->FirmwareRevision = PcdGet32 (PcdFirmwareRevision);  (это уже из функции initialize )  Status = gBS->InstallMultipleProtocolInterfaces (  &Handle,  &gEfiBdsArchProtocolGuid, &gBds,  NULL  );  //  // Initialize L"Timeout" EFI global variable.  //  BootTimeOut = PcdGet16 (PcdPlatformBootTimeOut);  if (BootTimeOut != 0xFFFF) {  //  // If time out value equal 0xFFFF, no need set to 0xFFFF to variable area because UEFI specification  // define same behavior between no value or 0xFFFF value for L"Timeout".  //  BdsDxeSetVariableAndReportStatusCodeOnError (  EFI\_TIME\_OUT\_VARIABLE\_NAME,  &gEfiGlobalVariableGuid,  EFI\_VARIABLE\_BOOTSERVICE\_ACCESS | EFI\_VARIABLE\_RUNTIME\_ACCESS | EFI\_VARIABLE\_NON\_VOLATILE,  sizeof (UINT16),  &BootTimeOut  );  } |

Изображение выглядит как стол

Автоматически созданное описание

Рисунок 9 – Последовательный вызов функций инициализаторов из функции funcs\_23EA, которая раз за разом вызывается из 2324.

|  |  |
| --- | --- |
| gRT\_12848->SetVariable(aOsindicationss, &EFI\_GLOBAL\_VARIABLE\_GUID\_11000, 6u, 8ui64, &unk\_10FF0);  gBS\_12840->InstallProtocolInterface(  (EFI\_HANDLE \*)(UINT32)&Handle,  (EFI\_GUID \*)&gEfiBootManagerPolicyProtocolGuid\_FB00.Data1,  EFI\_NATIVE\_INTERFACE,  &MaximumVariableSize);  result = gBS\_12840->LocateProtocol(&AMI\_POST\_MANAGER\_PROTOCOL\_GUID\_11010, 0i64, &Interface);  sub\_2268 | OsIndication &= ~((UINT64) (EFI\_OS\_INDICATIONS\_BOOT\_TO\_FW\_UI | EFI\_OS\_INDICATIONS\_START\_PLATFORM\_RECOVERY));  Status = gRT->SetVariable (  EFI\_OS\_INDICATIONS\_VARIABLE\_NAME,  &gEfiGlobalVariableGuid,  EFI\_VARIABLE\_BOOTSERVICE\_ACCESS | EFI\_VARIABLE\_RUNTIME\_ACCESS | EFI\_VARIABLE\_NON\_VOLATILE,  sizeof(UINT64),  &OsIndication  ); |

Также важны sub\_10AC – Sysprep####

Sub\_2408 – Boot#### и bootorder – Эта функция подчеркнута красным на рисунке 9. Она говорит о том, в каком порядке запускать ОС.

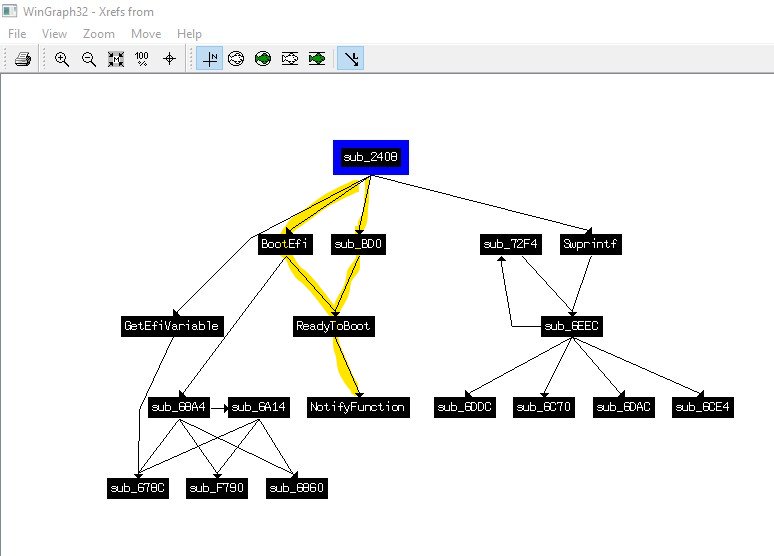


Рисунок 10 – Граф вызовов для передачи управления в ОС.

После этого запускается ОС

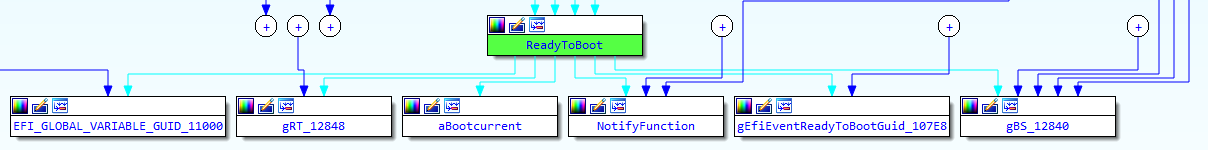


Рисунок 11 – Proximity view для доказательства, что функция конечная.

Далее ниже представлено ещё найденные соответствия функций из дизассемблера и исходников. Слева дизассемблер, справа и EDK2, и BDS.c Ivy Brigde.

|  |  |
| --- | --- |
| \_\_int64 \_\_fastcall sub\_7E0(char a1, char a2, \_\_int64 \*a3, EFI\_HANDLE \*\*a4)  {  result = gBS\_12840->LocateHandleBuffer(ByProtocol, &EFI\_PCI\_IO\_PROTOCOL\_GUID\_FA60, 0i64, &NoHandles, &Buffer);  if ( result >= 0 )  {  \*a3 = 0i64;  v9 = 0i64;  if ( !NoHandles )  goto LABEL\_12;  do  {  if ( (gBS\_12840->HandleProtocol(Buffer[v9], &EFI\_PCI\_IO\_PROTOCOL\_GUID\_FA60, &Interface) & 0x8000000000000000ui64) == 0i64 )  {  (\*((void (\_\_cdecl \*\*)(EFI\_PHYSICAL\_ADDRESS, UINTN))Interface + 6))((EFI\_PHYSICAL\_ADDRESS)Interface, 2ui64);  if ( v15 == a1 && v14 == a2 )  {  v10 = \*a3;  Buffer[v10] = Buffer[v9];  \*a3 = v10 + 1;  }  }  ++v9;  }  while ( v9 < NoHandles );  if ( \*a3 )  {  \*a4 = Buffer;  return 0i64;  }  else  {  LABEL\_12:  gBS\_12840->FreePool(Buffer);  return 0x800000000000000Eui64;  }  }  return result;  \_\_int64 \_\_fastcall sub\_D0C(EFI\_DEVICE\_PATH\_PROTOCOL \*DevicePath, \_\_int16 a2, \_\_int64 a3, int a4)  {    if ( (gBS\_12840->LoadImage(1u, ::Handle, DevicePath, 0i64, 0i64, &Handle) & 0x8000000000000000ui64) != 0i64 )  {  }  result = gBS\_12840->HandleProtocol(Handle, &EFI\_LOADED\_IMAGE\_PROTOCOL\_GUID\_FBA0, &Interface);  if ( result >= 0 )  {  if ( \*((\_DWORD \*)Interface + 20) == 1 )  {  if ( a4 )  {  \*((\_DWORD \*)Interface + 12) = a4;  \*((\_QWORD \*)Interface + 7) = a3;  }  sub\_B30(a2);  return gBS\_12840->StartImage(Handle, 0i64, 0i64);  }  else  {  return 0x8000000000000003ui64;  }  }  return result;  }  \_\_int64 \_\_fastcall sub\_B30(\_\_int16 a1)  {  \_\_int64 result; // rax  \_\_int16 Data; // [rsp+40h] [rbp+8h] BYREF  EFI\_EVENT Event; // [rsp+48h] [rbp+10h] BYREF  Data = a1;  if ( a1 != -1 )  gRT\_12848->SetVariable(  aBootcurrent,  &EFI\_GLOBAL\_VARIABLE\_GUID\_11000,  6u,  2ui64,  &Data);  result = gBS\_12840->CreateEventEx(0x200u, 8ui64, NotifyFunction, 0i64, &gEfiEventReadyToBootGuid\_107E8, &Event);  if ( result >= 0 )  {  gBS\_12840->SignalEvent(Event);  return gBS\_12840->CloseEvent(Event);  }  return result;  } | VOID  EFIAPI  CheckDeferredLoadImageOnReadyToBoot (  IN EFI\_EVENT Event,  IN VOID \*Context  )  Status = gBS->LocateHandleBuffer (  ByProtocol,  &gEfiDeferredImageLoadProtocolGuid,  NULL,  &HandleCount,  &Handles  );  if (EFI\_ERROR (Status)) {  return;  }  for (Index = 0; Index < HandleCount; Index++) {  Status = gBS->HandleProtocol (Handles[Index], &gEfiDeferredImageLoadProtocolGuid, (VOID \*\*) &DeferredImage);  if (EFI\_ERROR (Status)) {  continue;  }  if (DevicePathStr != NULL) {  FreePool (DevicePathStr);  }  }  }  EFI\_STATUS BootEfi(EFI\_DEVICE\_PATH\_PROTOCOL \*Dp, UINT16 Number, VOID \*pOptions, UINT32 Size)  {  EFI\_STATUS Status;  EFI\_HANDLE Handle;  EFI\_LOADED\_IMAGE\_PROTOCOL \*Image;  Status=pBS->LoadImage(TRUE, TheImageHandle, Dp, NULL, 0, &Handle);  if (EFI\_ERROR(Status)) return Status;  Status=pBS->HandleProtocol(Handle,&gEfiLoadedImageProtocolGuid,&Image);  if (!EFI\_ERROR(Status) && Size)  {  Image->LoadOptionsSize = Size;  Image->LoadOptions = pOptions;  }  ReadyToBoot(Number);  return pBS->StartImage(Handle, NULL, NULL);  }  VOID ReadyToBoot(UINT16 OptionNumber)  {  //signal EFI\_EVENT\_SIGNAL\_READY\_TO\_BOOT  EFI\_EVENT ReadToBootEvent;  EFI\_STATUS Status;  if (OptionNumber!= (UINT16)-1)  pRS->SetVariable(  L"BootCurrent",  &EfiVariableGuid,  EFI\_VARIABLE\_BOOTSERVICE\_ACCESS | EFI\_VARIABLE\_RUNTIME\_ACCESS,  sizeof(OptionNumber),  &OptionNumber  );  Status = CreateReadyToBootEvent(  TPL\_CALLBACK,  NULL,  NULL,  &ReadToBootEvent  );  if (!EFI\_ERROR(Status)) {  pBS->SignalEvent(ReadToBootEvent);  pBS->CloseEvent(ReadToBootEvent);  }  }  AMI bios |

Ещё немного найденных соответствий:

\_\_int64 sub\_E74() – RunDrivers

sub\_6A90 – GetEfiVariable

void \*\_\_fastcall sub\_1264 – DiscoverPartition

EFI\_STATUS \_\_fastcall sub\_14A8 – SignalProtocolEvent

\_\_int64 \_\_fastcall sub\_18F0 – ConnecConsoleVariable

void \_\_fastcall sub\_680 – ConnectDevicePath

\_\_int64 sub\_19E0 – InstallConOutStartedProtocol

\_\_int64 sub\_1BF0( - ConnectConInVariable

\_\_int64 sub\_1BFC() – InstallConInStartedProtocol

**Lpc/spi конфигурации.Сhipsec**

Сначала был скомпилирован драйвер и загружен в виртуальную ОС с отключенными подписями (как того требует документация Chipsec). Ниже представлены результаты успешных тестов и их пояснения.

Таблица 1 – Пройденные тесты.

|  |  |
| --- | --- |
| 1.Bypassing pre-boot authenti ation passwords by instrumenting the BIOS keyboard bufer | Программисты BIOS могут неправильно понимать его работу и не очищают буфер, из-за чего пользовательский ввод пароля(в незашифрованном виде) остаётся в памяти по определённому адресу |
| #### common.bios\_kbrd\_buffer    [\*] running module: chipsec.modules.common.bios\_kbrd\_buffer  [x][ =======================================================================  [x][ Module: Pre-boot Passwords in the BIOS Keyboard Buffer  [x][ =======================================================================  [\*] Keyboard buffer head pointer = 0x0 (at 0x41A), tail pointer = 0x0 (at 0x41C)  [\*] Keyboard buffer contents (at 0x41E):  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |  [\*] Checking contents of the keyboard buffer..  [+] PASSED: Keyboard buffer looks empty. Pre-boot passwords don't seem to be exposed | |
|  | |
| 2.SMI Events Configuration | Проверка на (запрет)lock изменения бита SMI, чтобы не позволять изменять состояние этого регистра, защищает от произвольного исполнения кода |
| #### common.bios\_smi    [\*] running module: chipsec.modules.common.bios\_smi  [x][ =======================================================================  [x][ Module: SMI Events Configuration  [x][ =======================================================================  [+] SMM BIOS region write protection is enabled (SMM\_BWP is used)    [\*] Checking SMI enables..  Global SMI enable: 1  TCO SMI enable : 1  [+] All required SMI events are enabled    [\*] Checking SMI configuration locks..  [+] TCO SMI configuration is locked (TCO SMI Lock)  [+] SMI events global configuration is locked (SMI Lock)    [+] PASSED: All required SMI sources seem to be enabled and locked | |
|  | |
| 3.BIOS Interface Lock (bit) (Top Swap Mode) | Механизм защиты (Top Block Swap[boot block] перемещается в другое место в памяти ,если вдруг при обновлении boot block’a пропадёт питание) становится дырой безопасности, так как разработчики не отключают возможность вызова этой swap функции перед передачей управления в ОС, что может привести к использованию swap’а вредоносным кодом. Поэтому нужно делать lock down bit на swap функцию |
| #### common.bios\_ts    [\*] running module: chipsec.modules.common.bios\_ts  [x][ =======================================================================  [x][ Module: BIOS Interface Lock (including Top Swap Mode)  [x][ =======================================================================  [\*] BiosInterfaceLockDown (BILD) control = 1  [\*] BIOS Top Swap mode is enabled (TSS = 1)  [\*] RTC TopSwap control (TS) = 1  [+] PASSED: BIOS Interface is locked (including Top Swap Mode) | |
|  | |
| 4.BIOS Region Write Protection | Защита от записи(перезаписи) BIOS |
| #### common.bios\_wp    [\*] running module: chipsec.modules.common.bios\_wp  [x][ =======================================================================  [x][ Module: BIOS Region Write Protection  [x][ =======================================================================  [\*] BC = 0xFFFFFFFF << BIOS Control (b:d.f 00:31.5 + 0xDC)  [00] BIOSWE = 1 << BIOS Write Enable  [01] BLE = 1 << BIOS Lock Enable  [02] SRC = 3 << SPI Read Configuration  [04] TSS = 1 << Top Swap Status  [05] SMM\_BWP = 1 << SMM BIOS Write Protection  [06] BBS = 1 << Boot BIOS Strap  [07] BILD = 1 << BIOS Interface Lock Down  [-] BIOS region write protection is disabled!    [\*] BIOS Region: Base = 0x07FFF000, Limit = 0x07FFFFFF  SPI Protected Ranges  ------------------------------------------------------------  PRx (offset) | Value | Base | Limit | WP? | RP?  ------------------------------------------------------------  PR0 (84) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR1 (88) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR2 (8C) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR3 (90) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR4 (94) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1    [+] PASSED: SPI Protected Ranges are configured to write protect BIOS | |
|  | |
| 5.IA32 Feature Control Lock | Lock bit в структуре выставляется в 1, чтобы отключить виртуализацию в ядре  typedef struct  \_IA32\_FEATURE\_CONTROL\_MSR  {  unsigned Lock :1;  unsigned VmxonInSmx :1;  unsigned VmxonOutSmx :1;  unsigned Reserved2 :29;  unsigned Reserved3 :32;  } IA32\_FEATURE\_CONTROL\_MSR; |
| #### common.ia32cfg    [\*] running module: chipsec.modules.common.ia32cfg  [x][ =======================================================================  [x][ Module: IA32 Feature Control Lock  [x][ =======================================================================  [\*] Verifying IA32\_Feature\_Control MSR is locked on all logical CPUs..  [\*] cpu0: IA32\_Feature\_Control Lock = 1  [\*] cpu1: IA32\_Feature\_Control Lock = 1  [\*] cpu2: IA32\_Feature\_Control Lock = 1  [\*] cpu3: IA32\_Feature\_Control Lock = 1  [+] PASSED: IA32\_FEATURE\_CONTROL MSR is locked on all logical CPUs | |
|  | |
| 6.ME Manufacturing Mode | Intel ME Manufacturing Mode — сервисный режим работы, предназначенный для конфигурирования, настройки и тестирования конечной платформы на стадии производства; он обязательно должен быть отключен перед поступлением оборудования в продажу и отгрузкой пользователю. |
| #### common.me\_mfg\_mode    [\*] running module: chipsec.modules.common.me\_mfg\_mode  [x][ =======================================================================  [x][ Module: ME Manufacturing Mode  [x][ =======================================================================  [+] PASSED: ME is not in Manufacturing Mode | |
|  | |
| 7. Attributes of Secure Boot EFI Variables | Проверка, что все Secure Boot uefi переменные аутентифицированы и не модифицировались |
| #### common.secureboot.variables    [\*] running module: chipsec.modules.common.secureboot.variables  [x][ =======================================================================  [x][ Module: Attributes of Secure Boot EFI Variables  [x][ =======================================================================  [\*] Checking protections of UEFI variable 8BE4DF61-93CA-11D2-AA0D-00E098032B8C:SecureBoot  [\*] Checking protections of UEFI variable 8BE4DF61-93CA-11D2-AA0D-00E098032B8C:SetupMode  [\*] Checking protections of UEFI variable 8BE4DF61-93CA-11D2-AA0D-00E098032B8C:PK  [+] Variable 8BE4DF61-93CA-11D2-AA0D-00E098032B8C:PK is authenticated (TIME\_BASED\_AUTHENTICATED\_WRITE\_ACCESS)  [\*] Checking protections of UEFI variable 8BE4DF61-93CA-11D2-AA0D-00E098032B8C:KEK  [+] Variable 8BE4DF61-93CA-11D2-AA0D-00E098032B8C:KEK is authenticated (TIME\_BASED\_AUTHENTICATED\_WRITE\_ACCESS)  [\*] Checking protections of UEFI variable D719B2CB-3D3A-4596-A3BC-DAD00E67656F:db  [+] Variable D719B2CB-3D3A-4596-A3BC-DAD00E67656F:db is authenticated (TIME\_BASED\_AUTHENTICATED\_WRITE\_ACCESS)  [\*] Checking protections of UEFI variable D719B2CB-3D3A-4596-A3BC-DAD00E67656F:dbx  [+] Variable D719B2CB-3D3A-4596-A3BC-DAD00E67656F:dbx is authenticated (TIME\_BASED\_AUTHENTICATED\_WRITE\_ACCESS)    [\*] Secure Boot appears to be disabled  [+] PASSED: All Secure Boot UEFI variables are protected | |
|  | |
| 8.SPI Flash Descriptor Security Override Pin-Strap | Проверяет значение бита Flash Descriptor Security Override Pin Strap(FDOPSS) регистра HSFS контролера SPI, сигнализирующего о перезаписи SPI Flash Descriptor |
| #### common.spi\_fdopss    [\*] running module: chipsec.modules.common.spi\_fdopss  [x][ =======================================================================  [x][ Module: SPI Flash Descriptor Security Override Pin-Strap  [x][ =======================================================================  [\*] HSFS = 0xFFFFFFFF << Hardware Sequencing Flash Status Register (SPIBAR + 0x4)  [00] FDONE = 1 << Flash Cycle Done  [01] FCERR = 1 << Flash Cycle Error  [02] AEL = 1 << Access Error Log  [05] SCIP = 1 << SPI cycle in progress  [11] WRSDIS = 1 << Write status disable  [12] PR34LKD = 1 << PRR3 PRR4 Lock-Down  [13] FDOPSS = 1 << Flash Descriptor Override Pin-Strap Status  [14] FDV = 1 << Flash Descriptor Valid  [15] FLOCKDN = 1 << Flash Configuration Lock-Down  [16] FGO = 1 << Flash cycle go  [17] FCYCLE = F << Flash Cycle Type  [21] WET = 1 << Write Enable Type  [24] FDBC = 3F << Flash Data Byte Count  [31] FSMIE = 1 << Flash SPI SMI# Enable  [+] PASSED: SPI Flash Descriptor Security Override is disabled | |
|  | |
| 9.SPI Flash Controller Configuration Locks | Конфигурация SPI контроллера(включая protected ranges (PR0-PR4)) блокируется HSFS[FLOCKDN] до перезагрузки, а если этого не сделать, то конфигурацию контролера можно перепрограммировать |
| #### common.spi\_lock    [\*] running module: chipsec.modules.common.spi\_lock  [x][ =======================================================================  [x][ Module: SPI Flash Controller Configuration Locks  [x][ =======================================================================  [\*] HSFS = 0xFFFFFFFF << Hardware Sequencing Flash Status Register (SPIBAR + 0x4)  [00] FDONE = 1 << Flash Cycle Done  [01] FCERR = 1 << Flash Cycle Error  [02] AEL = 1 << Access Error Log  [05] SCIP = 1 << SPI cycle in progress  [11] WRSDIS = 1 << Write status disable  [12] PR34LKD = 1 << PRR3 PRR4 Lock-Down  [13] FDOPSS = 1 << Flash Descriptor Override Pin-Strap Status  [14] FDV = 1 << Flash Descriptor Valid  [15] FLOCKDN = 1 << Flash Configuration Lock-Down  [16] FGO = 1 << Flash cycle go  [17] FCYCLE = F << Flash Cycle Type  [21] WET = 1 << Write Enable Type  [24] FDBC = 3F << Flash Data Byte Count  [31] FSMIE = 1 << Flash SPI SMI# Enable  [+] SPI write status disable set.  [+] SPI Flash Controller configuration is locked  [+] PASSED: SPI Flash Controller locked correctly. | |
|  | |
| 10.Access Control of EFI Variables | Проверяет атрибуты доступа глобальных переменных UEFI на соответствие спецификации |
| #### common.uefi.access\_uefispec    [\*] running module: chipsec.modules.common.uefi.access\_uefispec  [x][ =======================================================================  [x][ Module: Access Control of EFI Variables  [x][ =======================================================================  [\*] Testing UEFI variables ..  [\*] Variable PK (NV+BS+RT+TBAWS)  [\*] Variable KEK (NV+BS+RT+TBAWS)  [\*] Variable db (NV+BS+RT+TBAWS)  [\*] Variable SbConfigState (NV+BS+RT+TBAWS)  [\*] Variable Lang (NV+BS+RT)  [\*] Variable PlatformLang (NV+BS+RT)  [\*] Variable ConsoleOutMode (NV+BS+RT)  [\*] Variable Boot0000 (NV+BS+RT)  [\*] Variable Boot0001 (NV+BS+RT)  [\*] Variable Boot0002 (NV+BS+RT)  [\*] Variable Boot0003 (NV+BS+RT)  [\*] Variable CurrentPolicy (NV+BS+RT+TBAWS)  [\*] Variable UnlockIDCopy (NV+BS+RT)  [\*] Variable Boot0004 (NV+BS+RT)  [\*] Variable Timeout (NV+BS+RT)  [\*] Variable HDDP (NV+BS+RT)  [\*] Variable OfflineUniqueIDRandomSeed (NV+BS+RT)  [\*] Variable OfflineUniqueIDRandomSeedCRC (NV+BS+RT)  [\*] Variable dbx (NV+BS+RT+TBAWS)  [\*] Variable SignatureSupport (BS+RT)  [\*] Variable PKDefault (BS+RT)  [\*] Variable KEKDefault (BS+RT)  [\*] Variable dbDefault (BS+RT)  [\*] Variable dbxDefault (BS+RT)  [\*] Variable SetupMode (BS+RT)  [\*] Variable SecureBoot (BS+RT)  [\*] Variable MTC (NV+BS+RT)  [\*] Variable BootOptionSupport (BS+RT)  [\*] Variable OsIndicationsSupported (BS+RT)  [\*] Variable LangCodes (BS+RT)  [\*] Variable PlatformLangCodes (BS+RT)  [\*] Variable ConOutDev (BS+RT)  [\*] Variable ConIn (NV+BS+RT)  [\*] Variable ConOut (NV+BS+RT)  [\*] Variable ConInDev (BS+RT)  [\*] Variable ErrOut (NV+BS+RT)  [\*] Variable MemoryOverwriteRequestControl (NV+BS+RT)  [\*] Variable MemoryOverwriteRequestControlLock (NV+BS+RT)  [\*] Variable BootOrder (NV+BS+RT)  [\*] Variable MemoryTypeInformation (NV+BS+RT)  [\*] Variable BootCurrent (BS+RT)    [+] PASSED: All checked EFI variables are protected according to spec. | |

Далее идут проваленные тесты

Таблица 2 – Проваленные тесты.

|  |  |
| --- | --- |
| 1.Host Bridge Memory Map Locks | Проверка памяти на правильность конфигурации, например, что регистры правильно сконфигурированы и заблокированы от изменений. |
| #### common.memconfig    [\*] running module: chipsec.modules.common.memconfig  [x][ =======================================================================  [x][ Module: Host Bridge Memory Map Locks  [x][ =======================================================================  [\*]  [\*] Checking register lock state:  [-] PCI0.0.0\_BDSM = 0x 0 - UNLOCKED - Base of Graphics Stolen Memory  [-] PCI0.0.0\_BGSM = 0x 0 - UNLOCKED - Base of GTT Stolen Memory  [+] PCI0.0.0\_DPR = 0x 13001 - LOCKED - DMA Protected Range  [-] PCI0.0.0\_GGC = 0x 0 - UNLOCKED - Graphics Control  [-] PCI0.0.0\_MESEG\_MASK = 0x 0 - UNLOCKED - Manageability Engine Limit Address Register  [-] PCI0.0.0\_PAVPC = 0x 30011000 - UNLOCKED - PAVP Configuration  [-] PCI0.0.0\_REMAPBASE = 0x 0 - UNLOCKED - Memory Remap Base Address  [-] PCI0.0.0\_REMAPLIMIT = 0x 0 - UNLOCKED - Memory Remap Limit Address  [-] PCI0.0.0\_TOLUD = 0x 0 - UNLOCKED - Top of Low Usable DRAM  [-] PCI0.0.0\_TOM = 0x 0 - UNLOCKED - Top of Memory  [-] PCI0.0.0\_TOUUD = 0x 0 - UNLOCKED - Top of Upper Usable DRAM  [-] PCI0.0.0\_TSEGMB = 0x 0 - UNLOCKED - TSEG Memory Base  [\*]  [-] FAILED: Not all memory map registers are locked down | |
|  | |
| 2.Check MSR\_LT\_LOCK\_MEMORY | Этот модуль проверяет, заблокирована ли конфигурация памяти для защиты SMM. |
| #### common.memlock    [\*] running module: chipsec.modules.common.memlock  [x][ =======================================================================  [x][ Module: Check MSR\_LT\_LOCK\_MEMORY  [x][ =======================================================================  [X] Checking MSR\_LT\_LOCK\_MEMORY status  [\*] cpu0: MSR\_LT\_LOCK\_MEMORY[LT\_LOCK] = 0  [\*] cpu1: MSR\_LT\_LOCK\_MEMORY[LT\_LOCK] = 0  [\*] cpu2: MSR\_LT\_LOCK\_MEMORY[LT\_LOCK] = 0  [\*] cpu3: MSR\_LT\_LOCK\_MEMORY[LT\_LOCK] = 0  [-] FAILED: Check failed. MSR\_LT\_LOCK\_MEMORY isn't configured correctly | |
|  | |
| 3.Memory Remapping Configuration | Проверяет возможность перекрытия области SMRAM |
| #### common.remap    [\*] running module: chipsec.modules.common.remap  [x][ =======================================================================  [x][ Module: Memory Remapping Configuration  [x][ =======================================================================  [\*] Registers:  [\*] TOUUD : 0x0000000000000000  [\*] REMAPLIMIT: 0x0000000000000000  [\*] REMAPBASE : 0x0000000000000000  [\*] TOLUD : 0x00000000  [\*] TSEGMB : 0x00000000    [\*] Memory Map:  [\*] Top Of Upper Memory: 0x0000000000000000  [\*] Remap Limit Address: 0x00000000000FFFFF  [\*] Remap Base Address : 0x0000000000000000  [\*] 4GB : 0x0000000100000000  [\*] Top Of Low Memory : 0x0000000000000000  [\*] TSEG (SMRAM) Base : 0x0000000000000000    [\*] checking memory remap configuration..  [\*] Memory Remap is enabled  [-] Remap window configuration is not correct  [+] All addresses are 1MB aligned  [\*] checking if memory remap configuration is locked..  [-] TOUUD is not locked  [-] TOLUD is not locked  [-] REMAPBASE and REMAPLIMIT are not locked  [-] FAILED: Memory Remap is not properly configured/locked. Remaping attack may be possible | |
|  | |
| 4.SPI Flash Region Access Control | Проверка на права доступа SPI Flash Region Access к Flash Descriptor |
| #### common.spi\_access    [\*] running module: chipsec.modules.common.spi\_access  [x][ =======================================================================  [x][ Module: SPI Flash Region Access Control  [x][ =======================================================================  SPI Flash Region Access Permissions  ------------------------------------------------------------    BIOS Region Write Access Grant (FF):  FREG0\_FLASHD: 1  FREG1\_BIOS : 1  FREG2\_ME : 1  FREG3\_GBE : 1  FREG4\_PD : 1  FREG5 : 1  BIOS Region Read Access Grant (FF):  FREG0\_FLASHD: 1  FREG1\_BIOS : 1  FREG2\_ME : 1  FREG3\_GBE : 1  FREG4\_PD : 1  FREG5 : 1  BIOS Region Write Access (FF):  FREG0\_FLASHD: 1  FREG1\_BIOS : 1  FREG2\_ME : 1  FREG3\_GBE : 1  FREG4\_PD : 1  FREG5 : 1  BIOS Region Read Access (FF):  FREG0\_FLASHD: 1  FREG1\_BIOS : 1  FREG2\_ME : 1  FREG3\_GBE : 1  FREG4\_PD : 1  FREG5 : 1  [\*] Software has write access to Platform Data region in SPI flash (it's platform specific)  [!] WARNING: Software has write access to GBe region in SPI flash  [-] Software has write access to SPI flash descriptor  [-] Software has write access to Management Engine (ME) region in SPI flash  [-] FAILED: SPI Flash Region Access Permissions are not programmed securely in flash descriptor  [!] System may be using alternative protection by including descriptor region in SPI Protected Range Registers | |
|  | |
| 5.SPI Flash Region Access Control | SPI Flash Descriptor обозначает разрешения на чтение/запись для устройств, которые хотят получить доступ в flash память. Этот модуль читает Flash Descriptor и проверяет, что программное обеспечение не может изменить дескриптор Flash. Если ПО может писать во Flash Descriptor, то оно может обойти любую защиту, установленную Flash Descriptor’ом. Используется в дебаге и должно быть отключено в продакшене. Другими словами, этот модуль проверяет, что ПО не может писать во Flash Descriptor. |
| #### common.spi\_desc    [\*] running module: chipsec.modules.common.spi\_desc  [x][ =======================================================================  [x][ Module: SPI Flash Region Access Control  [x][ =======================================================================  [\*] FRAP = 0xFFFFFFFF << SPI Flash Regions Access Permissions Register (SPIBAR + 0x50)  [00] BRRA = FF << BIOS Region Read Access  [08] BRWA = FF << BIOS Region Write Access  [16] BMRAG = FF << BIOS Master Read Access Grant  [24] BMWAG = FF << BIOS Master Write Access Grant  [\*] Software access to SPI flash regions: read = 0xFF, write = 0xFF  [-] Software has write access to SPI flash descriptor    [-] FAILED: SPI flash permissions allow SW to write flash descriptor  [!] System may be using alternative protection by including descriptor region in SPI Protected Range Registers | |

Таблица 3 – Предупреждения.

|  |  |  |
| --- | --- | --- |
| 1.Checks for Branch Target Injection / Spectre v2 (CVE-2017-5715) | | Проверка на спекулятивное выполнение команд для чтения данных через сторонний канал |
| #### common.cpu.spectre\_v2    [\*] running module: chipsec.modules.common.cpu.spectre\_v2  [x][ =======================================================================  [x][ Module: Checks for Branch Target Injection / Spectre v2 (CVE-2017-5715)  [x][ =======================================================================  [\*] CPUID.7H:EDX[26] = 1 Indirect Branch Restricted Speculation (IBRS) & Predictor Barrier (IBPB)  [\*] CPUID.7H:EDX[27] = 1 Single Thread Indirect Branch Predictors (STIBP)  [\*] CPUID.7H:EDX[29] = 1 IA32\_ARCH\_CAPABILITIES  [+] CPU supports IBRS and IBPB  [+] CPU supports STIBP  [\*] checking enhanced IBRS support in IA32\_ARCH\_CAPABILITIES...  [\*] cpu0: IBRS\_ALL = 0  [-] CPU doesn't support enhanced IBRS  [!] WARNING: CPU supports mitigation (IBRS) but doesn't support enhanced IBRS  [!] OS may be using software based mitigation (eg. retpoline)  [-] Retpoline is NOT enabled by the OS | | |
|  | | |
| 2.SMM\_Code\_Chk\_En (SMM Call-Out) Protection | | Проверка регистра SMM\_Code\_Chk\_En на то, что он выставлен в 1 и locked. В этому случае эта конфигурация устраняет уязвимости, связанных с вызовами SMM. Если обработчик SMI вызывает какой-либо код, находящийся вне SMRAM, то атакующий с правами на запись в физическую память может подменить вызываемый код на свой, и выполнить его таким образом в режиме SMM. |
| #### common.smm\_code\_chk    [\*] running module: chipsec.modules.common.smm\_code\_chk  [x][ =======================================================================  [x][ Module: SMM\_Code\_Chk\_En (SMM Call-Out) Protection  [x][ =======================================================================  [\*] MSR\_SMM\_FEATURE\_CONTROL = 0x00000000 << Enhanced SMM Feature Control (MSR 0x4E0 Thread 0x0)  [00] LOCK = 0 << Lock bit  [02] SMM\_Code\_Chk\_En = 0 << Prevents SMM from executing code outside the ranges defined by the SMRR  [\*] MSR\_SMM\_FEATURE\_CONTROL = 0x00000000 << Enhanced SMM Feature Control (MSR 0x4E0 Thread 0x0)  [00] LOCK = 0 << Lock bit  [02] SMM\_Code\_Chk\_En = 0 << Prevents SMM from executing code outside the ranges defined by the SMRR  [\*] MSR\_SMM\_FEATURE\_CONTROL = 0x00000000 << Enhanced SMM Feature Control (MSR 0x4E0 Thread 0x0)  [00] LOCK = 0 << Lock bit  [02] SMM\_Code\_Chk\_En = 0 << Prevents SMM from executing code outside the ranges defined by the SMRR  [\*] MSR\_SMM\_FEATURE\_CONTROL = 0x00000000 << Enhanced SMM Feature Control (MSR 0x4E0 Thread 0x0)  [00] LOCK = 0 << Lock bit  [02] SMM\_Code\_Chk\_En = 0 << Prevents SMM from executing code outside the ranges defined by the SMRR  WARNING: [\*] SMM\_Code\_Chk\_En is not enabled.  This can happen either because this feature is not supported by the CPU or because the BIOS forgot to enable it.  Please consult the Intel SDM to determine whether or not your CPU supports SMM\_Code\_Chk\_En. | | |
|  | | |
| 3.SMM TSEG Range Configuration Check | Проверяется конфигурация и защита SMRAM от DMA атак. Атакующий с правами администратора инициирует DMA с контролируемой им областью памяти с одной стороны и SMRAM с другой, и получает полный доступ к ней. TSEG (т.е. «топовый сегмент») – в нём находятся код, исполняемого в SMM | |
| #### common.smm\_dma    [\*] running module: chipsec.modules.common.smm\_dma  [x][ =======================================================================  [x][ Module: SMM TSEG Range Configuration Check  [x][ =======================================================================  [\*] TSEG : 0x0000000000000000 - 0x-000000000000001 (size = 0x00000000)  [\*] SMRR is not supported    [\*] checking TSEG range configuration..  [!] WARNING: TSEG is properly configured but can't determine if it covers entire SMRAM | | |
|  | | |
| 4.S3 Resume Boot-Script Protections | Проверка защиты S3 Resume Boot-Script | |
| #### common.uefi.s3bootscript    [\*] running module: chipsec.modules.common.uefi.s3bootscript  [x][ =======================================================================  [x][ Module: S3 Resume Boot-Script Protections  [x][ =======================================================================  [\*] SMRAM: Base = 0x0000000000000000, Limit = 0x-000000000000001, Size = 0x00000000  [+] Didn't find any S3 boot-scripts in EFI variables  [!] WARNING: S3 Boot-Script was not found. Firmware may be using other ways to store/locate it, or OS might be blocking access. | | |

**Secure boot**

Корректность конфигурации Secure boot подтверждается путём использования AMI HSTI(Hardware Security Test Interface). HSTI - это не список требований для OEM-производителей; это интерфейс, гарантирующий эффективное программное тестирование безопасности микрокода, оборудования и параметров конфигурации.

Результаты HSTI отправляются в Windows для сертификации.

Была найдена точка входа для Ami HSTI и использование Secure boot , и криптографии(GUID) соответственно.



Рисунок 12 – Использование Secure boot переменной.



Рисунок 13 – Использование Secure boot переменной.

Secure boot работает в режиме User(рисунок 14). Если бы он был в режиме Setup (например, *"Platform in Setup Mode!  
Secure Boot can be enabled when Platform is in User Mode. Repeat operation after enrolling Platform Key (PK)"*). То он был бы настроен неверно. Но у меня настроен верно.

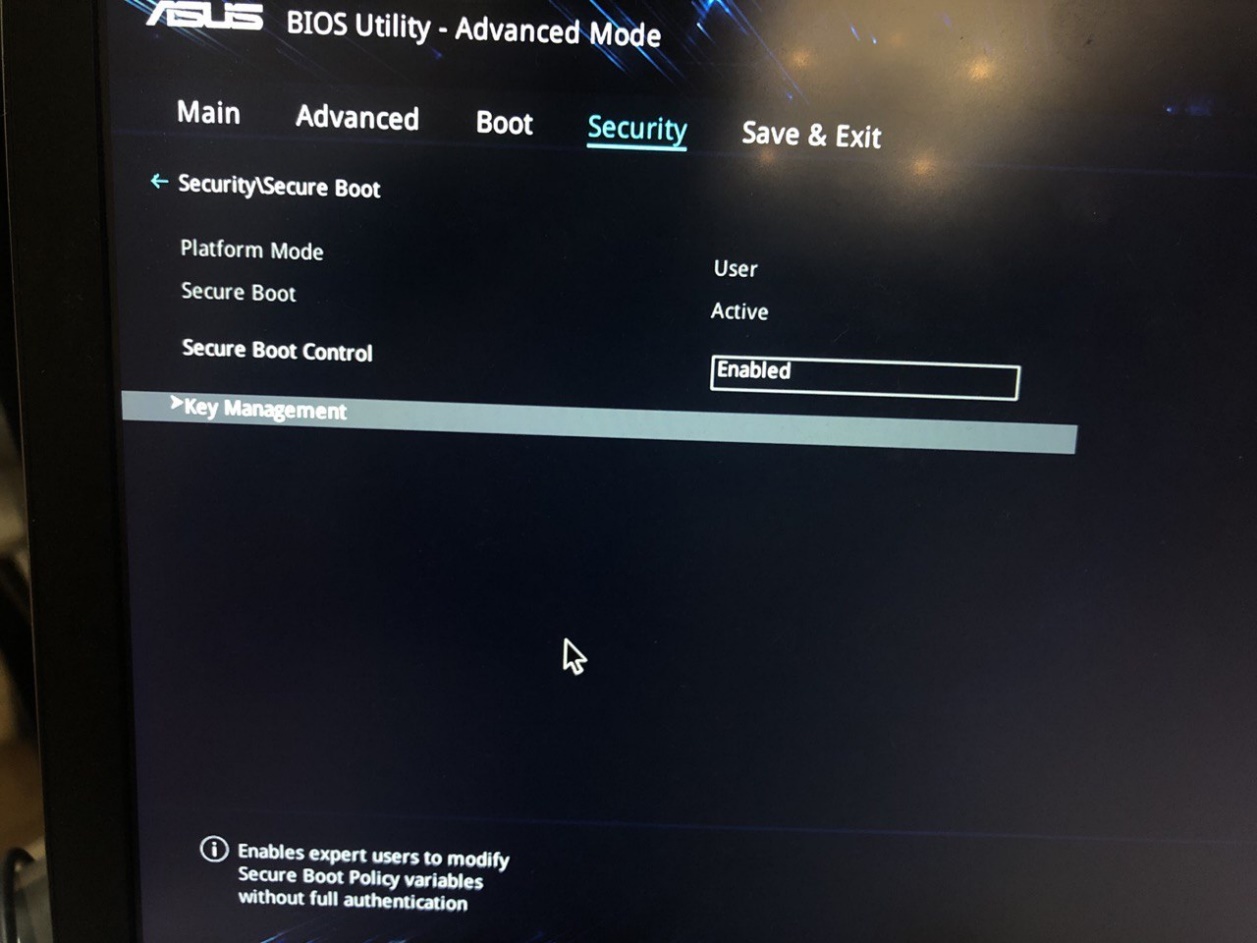


Рисунок 14 – Secure boot включен и настроен.



Рисунок 15 – Настройки ключей Secure boot.

Для Intel boot guard и Intel bios guard присутствует такой же HSTI-тест.



Рисунок 16 – Наличие технологии.

Для оставшихся технологий (Intel boot guard, bios guard) мы воспользуемся инструментами (Converged Security) Management Engine - (CS)ME, ( Intel System Tool Kit). Среди них нам понадобятся Flash Programming Tool и Flash Image Tool. Первым инструментом мы снимем дамп, так как Flash Image Tool не распознает файл обновления с сайта ASUS (с которым мы работали) (рисунок 17).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 – Неудачный сбор дампа – bios lock.

Тогда поступим иным способом. Исходим из определения чипсета в списке чиспсетов Intel C422 ⬄ Kaby Lake. Тогда во Flash Image Tool и выберем эту конфигурацию (рисунок 18).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 18 – Выбрана конфигурация исследуемого компьютера.

В силу того, что снять дамп образа bios не получилось (а программатора под рукой нет). Мы можем только предположить (опираясь на тесты chipsec) конфигурацию средств защиты. А Flash image tool дал нам критерии для построения предположений.

Таблица 4 – Boot Guard конфигурация.

|  |  |  |
| --- | --- | --- |
| Название конфигурации | Значение | Комментарий |
| Key Manifest ID | 256-хэш от открытого ключа OEM | Этот хэш от открытого ключа используется ACM (Authenticated Code Module), чтобы верефицировать Boot Policy Manifest |
| Boot Guard Profile Configuration | No\_FVME технология Intel BG выключена  VM включены оба режима, без выключения системы  FVE включён режим VerifyBoot, немедленное выключение  FVME включены оба режима, немедленное выключение | Здесь выбирается профиль  typedef struct BG\_PROFILE  {  unsigned long Force\_Boot\_Guard\_ACM : 1;  unsigned long Verified\_Boot : 1;  unsigned long Measured\_Boot : 1;  unsigned long Protect\_BIOS\_Environment : 1;  unsigned long Enforcement\_Policy : 2; // 00b – do nothing  // 01b – shutdown with timeout  // 11b – immediate shutdown  unsigned long : 26;  }; |
| CPU Debugging | Disabled | Возможность отладки не поддерживается согласно CHIPSEC тесту  #### common.debugenabled  [\*] running module: chipsec.modules.common.debugenabled  [\*] NOT IMPLEMENTED: CPU Debug features are not supported on this platform  Skipping module chipsec.modules.common.debugenabled since it is not supported in this platform |
| BSP Initialization | Наверное, disabled | Определяет поведение BSP при принятии INIT сигнала – вход в невосстановимое выключение (unrecoverable shutdown) |

Таблица 5 - Bios guard

|  |  |  |
| --- | --- | --- |
| BIOS Guard Protection Override Enabled | Enable (ниже доказательство в IFR формах) | Intel BIOS Guard,используя,Embended controller, контролирует доступ на чтение/запись в SPI Flash  #### common.bios\_wp  PR0 (84) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR1 (88) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR2 (8C) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR3 (90) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1  PR4 (94) | FFFFFFFF | 07FFF000 | 07FFFFFF | 1 | 1    [+] PASSED: SPI Protected Ranges are configured to write protect BIOS |

**Определение модуля Setup**

Находим модуль Setup в UEFItool (рисунок 19).

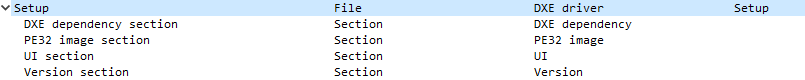


Рисунок 19 – модуль Setup.

Запускаем утилиту IFR-Extractor.

|  |
| --- |
| C:\Users\Seven\Downloads\ifr>ifrextract.exe Section\_PE32\_image\_Setup\_Setup\_body.efi out2  Input: Section\_PE32\_image\_Setup\_Setup\_body.efi  Output: out2  Protocol: UEFI |

И получаем следующие строки

|  |
| --- |
| 0x25930 VarStore: VarStoreId: 0x32 [EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9], Size: 0x4, Name: UsbControllerNum  {24 27 43 D6 87 EC A4 EB B5 4B A1 E5 3F 3E 36 B2 0D A9 32 00 04 00 55 73 62 43 6F 6E 74 72 6F 6C 6C 65 72 4E 75 6D 00}  0x25957 VarStore: VarStoreId: 0x33 [EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9], Size: 0x21, Name: UsbSupport  {24 21 43 D6 87 EC A4 EB B5 4B A1 E5 3F 3E 36 B2 0D A9 33 00 21 00 55 73 62 53 75 70 70 6F 72 74 00} |

Необходимо упомянуть, что меню setup – это интерфейс к переменным в NVRAM. Переменные могут находиться в разных блоках (т.н. varstore), но настройки, к котором есть доступ из Setup, чаще всего хранятся в блоке SETUP\_DATA, который в свою очередь целиком хранится в переменной по имени Setup.

Тогда для того, чтобы найти скрытые опции настройки UEFI, достаточно заменить один байт в NVRAM, но при условии чтобы guid совпадали.

|  |
| --- |
| 24 27 43 D6 87 EC A4 EB B5 4B A1 E5 3F 3E 36 B2 0D A9  24 21 43 D6 87 EC A4 EB B5 4B A1 E5 3F 3E 36 B2 0D A9 |

NVRAM-переменные и IFR-текст представлены для сравнения в таблице 6.

Таблица 6 - NVRAM-переменные и Varstore.

|  |  |  |  |
| --- | --- | --- | --- |
| NVRAM-GUID | NVRAM-name | Varstore-GUID | Varstore-name |
| EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9  EfiSetupVariableGuid | Setup | EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9 | Setup  SetupCpuFeatures  SetupAmtFeatures  AsusSetupIoInterface  SetupPlatformData  NBPlatformData  NBGopPlatformData  SdioDevConfiguration  UsbMassDevNum  UsbMassDevValid  UsbControllerNum  UsbSupport  SataPresent  SetupVolatileData |
| 8BE4DF61-93CA-11D2-AA0D-00E098032B8C  EfiGlobalVariableGuid | PlatformLang | 8BE4DF61-93CA-11D2-AA0D-00E098032B8C | PlatformLang  PlatformLangCodes |
| 8BE4DF61-93CA-11D2-AA0D-00E098032B8C  EfiGlobalVariableGuid | Timeout | 8BE4DF61-93CA-11D2-AA0D-00E098032B8C | Timeout  BootOrder  DriverOrder  VendorKeys  SetupMode  SecureBoot  AuditMode  DeployedMode |
| C811FA38-42C8-4579-A9BB-60E94EDDFB34  AmiTseSetupGuid | AMITSESetup | C811FA38-42C8-4579-A9BB-60E94EDDFB34 | AMITSESetup |
| 90D93E09-4E91-4B3D-8C77-C82FF10E3C81 | CpuSmm | 90D93E09-4E91-4B3D-8C77-C82FF10E3C81 | CpuSmm |
| 5432122D-D034-49D2-A6DE-65A829EB4C74 | MeSetupStorage | 5432122D-D034-49D2-A6DE-65A829EB4C74 | MeSetupStorage |
| 64192DCA-D034-49D2-A6DE-65A829EB4C74 | IccAdvancedSetupDataVar | 64192DCA-D034-49D2-A6DE-65A829EB4C74 | IccAdvancedSetupDataVar |
| 5E9A565F-CDC0-413B-AD13-1FE8713FFDCD | PcieSataModVar | 5E9A565F-CDC0-413B-AD13-1FE8713FFDCD | PcieSataModVar |
| EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9  EfiSetupVariableGuid | AsusSetupIoInterface |  | AsusSetupIoInterface  SetupPlatformData  NBPlatformData  NBGopPlatformData  SdioDevConfiguration  UsbMassDevNum  UsbMassDevValid  UsbControllerNum  UsbSupport  SataPresent  SetupVolatileData |
| 69ECC1BE-A981-446D-8EB6-AF0E53D06CE8 | NewOprionPolicy | 69ECC1BE-A981-446D-8EB6-AF0E53D06CE8 | NewOptionPolicy |
| D1405D16-7AFC-4695-BB12-41459D3695A2  EfiNetworkStackSetupGuid | NetworkStackVar | D1405D16-7AFC-4695-BB12-41459D3695A2 | NetworkStackVar |
| EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9  EfiSetupVariableGuid | SdioDevConfiguration | EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9 | SdioDevConfiguration  UsbMassDevNum  UsbMassDevValid  UsbControllerNum  UsbSupport  SataPresent  SetupVolatileData |
| 7B59104A-C00D-4158-87FF-F04D6396A915 | SecureBootSetup | 7B59104A-C00D-4158-87FF-F04D6396A915 | SecureBootSetup  SecureVarPresent |
| EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9  EfiSetupVariableGuid | UsbSupport | EC87D643-EBA4-4BB5-A1E5-3F3E36B20DA9 | Setup  SetupCpuFeatures  SetupAmtFeatures  AsusSetupIoInterface  SetupPlatformData  NBPlatformData  NBGopPlatformData  SdioDevConfiguration  UsbMassDevNum  UsbMassDevValid  UsbControllerNum  UsbSupport  SataPresent  SetupVolatileData |
| ACA9F304-21E2-4852-9875-7FF4881D67A5 | PCI\_COMMON |  |  |
| 80E1202E-2697-4264-9CC9-80762C3E5863 | Setup |  |  |

Воспользуемся Universal IFR Extractor для поиска скрытых из меню Setup переменных.

Но прежде покажем, как выглядят в Setup присутствующая и включенная настройка, например, AES New Instructions (рисунок 20).

Изображение выглядит как текст, монитор, черный, экран

Автоматически созданное описание

Рисунок 20 – Включенный AES New Instructions.

|  |
| --- |
| Variable 0xC05 equals 0x1 {12 06 05 0C 01 00}  Setting: Intel AES-NI, Variable: 0x5B9 {05 91 5A 17 5B 17 C0 0B 01 00 B9 05 10 10 00 01 00}  Option: Disabled, Value: 0x0 {09 07 04 00 00 00 00}  Option: Enabled, Value: 0x1 {09 07 03 00 30 00 01}  End of Options {29 02} |

Это был пример включенной и видимой в Setup настройки.

Рассмотрим cкрытые настройки в меню Setup(не отображается в меню Setup). Для этого обратимся к спецификации процессора Kaby Lake и увидим SGX (рисунок 21).

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 21 – Спецификация процессора.

|  |
| --- |
| Variable 0xCE8 equals 0x0 {12 06 E8 0C 00 00}  Setting: SW Guard Extensions (SGX), Variable: 0x5DC {05 91 70 03 71 03 1E 27 01 00 DC 05 14 10 00 02 00}  Option: Software Controlled, Value: 0x2 {09 07 63 03 30 00 02}  Option: Enabled, Value: 0x1 {09 07 03 00 00 00 01}  Option: Disabled, Value: 0x0 {09 07 04 00 00 00 00}  End of Options {29 02} |

Это пример выключенной и невидимой в меню Setup настройки.

Рассмотрим ещё один пример. Этой настройки нет в Setup и невидима, но она включена

|  |
| --- |
| Variable 0xC05 equals 0x1 {12 06 05 0C 01 00}  Setting: Intel Bios Guard Support, Variable: 0x1003 {05 91 87 15 86 15 0E 03 01 00 03 10 10 10 00 01 00}  Default: 8 Bit, Value: 0x0 {5B 06 00 00 00 00}  Option: Disabled, Value: 0x0 (default MFG) {09 07 04 00 20 00 00}  Option: Enabled, Value: 0x1 {09 07 03 00 00 00 01}  End of Options {29 02} |

Аналогично с Intel® Hyper-Threading Technology

|  |
| --- |
| Variable 0x72 equals 0x1 {12 06 72 00 01 00}  Setting: Hyper-Threading, Variable: 0x4F0 {05 91 9D 01 9E 01 6A 00 01 00 F0 04 10 10 00 01 00}  Option: Disabled, Value: 0x0 {09 07 04 00 00 00 00}  Option: Enabled, Value: 0x1 {09 07 03 00 30 00 01}  End of Options {29 02} |

**Вывод**

В данной лабораторной работе был исследован UEFI ноутбука ASUS UX310UAK-AS.312 на базе процессора Intel Core i5-7200U Kaby Lake. После того как был изучен пошаговый процесс загрузки UEFI вплоть до передачи управления в ОС, были рассмотрены средства защиты, которые присутствуют в рассматриваемом ноутбуке.

Первый из рассмотренных – Secure Boot. Этот защитный механизм, разрешающий запуск только аутентифицированных компонентов (находящихся в db) и не разрешающий в ином случае(dbx), по результатам проведенной выше проверке настроен верно. Следующие два механизма Boot Guard и Bios Guard. Boot Guard нужен, чтобы защищать root-of-trust Secure Boot’а (от атак нацеленных на прошивку) путём запуска только проверенной прошивки. Intel BIOS Guard, используя Embedded controller, контролирует доступ на чтение/запись в SPI Flash. К сожалению, не удалось снять дамп с flash из-за bios lock, поэтому пришлось лишь примерно, основываясь на тесты Chipsec, оценить реализацию обеих вышеупомянутых технологий.

Таким образом, с учетом включенного Bios Guard, PR-регистров (тест common.bios\_wp) для защиты от записи в BIOS, перезаписать UEFI будет невозможно.

Хоть мы и не смогли получить дамп, чтобы узнать конфигурацию Bios Guard, мы смогли найти доказательство, что эта технология присутствует и включена, но не видна (скрытая настройка Setup). А также и скрытый Intel SGX.

В качестве рекомендаций по настройке механизмов защиты можно установить защиту от CPU SMM Cache Poisoning(Эта атака может изменять значения в SMRAM. Если ПО в ring0 может сделать SMRAM кэшируемым, а затем заполнить строки кэша на SMBASE кодом эксплойта, то при срабатывании прерывания SMI ЦП может выполнить код эксплойта из кеша. System Management Mode Range Registers (SMRR) не дают кэшироваться и блокируют доступ к SMRAM, когда ЦП не находится в SMM. Поэтому эти регистры должны быть включены и настроены в BIOS.). В рассматриваемом устройстве этой реализации нет, согласно тесту Chipsec(CPU does not support SMRR range protection of SMRAM. NOT IMPLEMENTED: CPU does not support SMRR range protection of SMRAM).

Так же оказалось, что Memory Remap не заблокирован, что может привести к Remaping attack. Если переместить границу TOLUD (Top Of Low Usable DRAM) за SMRAM и затем воспользуемся механизмом Remapping, чтобы оказаться в SMRAM, так как происходит перекрытие (covered by MMIO). Туда можно загрузить вредоносный код. Поэтому Memory Remap должен быть закрыт.

Подводя итог, можно сделать вывод, что ASUS идёт по верному пути в плане безопасности материнских плат. Однако ничто не идеально и случается проявление человеческого фактора, из-за которого разработчики могу нечаянно забыть, например, поставить запрет на изменение того или иного регистра или вовсе не настроить его.