Advanced Threat Research | Intel Security

Archived: 2026-04-05 19:23:30 UTC

The examination of commercial malware developed by Hacking Team has revealed much to the security community. Of particular interest to platform security researchers at Intel's Advanced Threat Research team (ATR) is the presence of what appears to be a UEFI-based persistent infection mechanism. ATR has been researching vulnerabilities related to system firmware and working with a community of firmware developers and platform manufacturers to mitigate these threats. Others have also posted good information about this issue. Here, we will provide some preliminary analysis of the firmware threat.

Finding a Persistent Rootkit

The trail begins with the mysterious file "Z5WE1X64.fd" which was attached to one of the leaked emails into "Uefi_windows_persistent.zip" compressed file.

uefi

Email-ID	526357
Date	2014-09-25 15:43:28 UTC
From	f.cornelli@hackingteam.com
To	g.cino@hackingteam.com

Attached Files

#	Filename	Size
242336	Uefi_windows_persistent.zip	3.4MiB

Email Body

-- Fabrizio Cornelli
QA Manager

Hacking Team
Milan Singapore Washington DC
www.hackingteam.com <http://www.hackingteam.com>

email: f.cornelli@hackingteam.com
mobile: +39 3666539755
phone: +39 0229060603

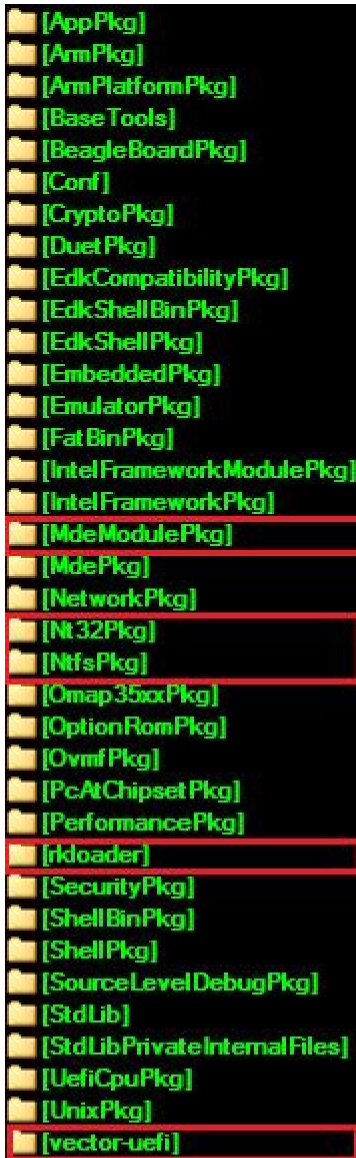
A deeper look into the binary reveals some string constants which point toward firmware image update for UEFI BIOS platforms:

This binary appears to be a firmware update image packed by InsydeH2O tools for UEFI firmware. These tools are applicable to many (though not all) common platforms. Inside the image, we (along with Nikolaj Schlej) found some unexpected sections:

Some suspicious entries are "Ntfs", "rkloader" and a nameless UEFI application (highlighted above). The developers did not hide the real name of the modules, which has a hint about module functionality and enables

detection.

The "rkloader" executable image name suggests a rootkit. In the same leaked archive we find the source code directory "vector-edk" which contains the code of a UEFI-based rootkit.



More than a PoC

The leaked source code goes beyond a research proof-of-concept, revealing a commercial rootkit platform called "]HackingTeam[UEFI Vector" and using real attacks as a part of Hacking Team? RCS malware platform. According to the leaked code and emails, this hacking platform may have already been already sold to some HackingTeam customers. Some of the emails point to specific modes on which the persistent rootkit was tested.

[!JAH-597-79875]: persistent installation vector.

Email-ID	640527
Date	2014-07-10 08:57:15 UTC
From	support@hackingteam.com
To	rcs-support@hackingteam.com

Email Body

Bruno Muschitiello updated #JAH-597-79875

Staff (Owner): Bruno Muschitiello (was: -- Unassigned --) Status: In Progress (was: Open)
persistent installation vector.

Ticket ID: JAH-597-79875 URL: https://support.hackingteam.com/staff/index.php?/Tickets/Ticket/View/2994 Name: Peter Balogh Email address: balogh.peter@nbsz.gov.hu Creator: User Department: General Staff (Owner): Bruno Muschitiello Type: Issue Status: In Progress Priority: Normal Template group: Default Created: 10 July 2014 10:51 AM Updated: 10 July 2014 10:57 AM

We are working on this model of notebook:

Acer Aspire E1-570

Kind regards

From: **serge** <s.woon@hackingteam.com>
Date: Mon, Dec 29, 2014 at 5:44 PM
Subject: Re: Signed PO + Proposal
To: nupt@dhag.com.vn
Cc: Marco Bettini <m.bettini@hackingteam.com>, "Hoan Phi Van (Mr.)" <hoanpv@dhag.com.vn>, Daniel Maglietta <d.maglietta@hackingteam.com>, RSALES <rsales@hackingteam.it>, "Phan Tien Hung (Mr.)" <hungpt@dhag.com.vn>

Hi Ms Nu,

Are you saying that you want to test the BIOS infection for all the 4 workstations with AV installed?

I do not have any spare machines to bring for the DAP. I checked that for BIOS infection, we supported the following laptops. If the customer wants to test, please prepare anyone of them:

Dell Latitude 6320Dell Precision T1600Asus X550CAsus F550C

Regards,
Serge

Updating the SPI flash, where system firmware is usually stored, is usually accomplished either through physically attaching a programmer to the chip or through a signed update mechanism built into the firmware. One of the leaked emails contains a presentation (presumably for potential customers) that describes this:

slides

Email-ID	470201
Date	2014-03-14 11:10:20 UTC
From	a.mazzeo@hackingteam.com
To	marco

Attached Files

#	Filename	Size
224546	RCS 9 UEFI.pptx	2.4MiB

Email Body

antonio

--
Antonio Mazzeo
Senior Security Engineer

Here is screenshot of slide which describing infection way with phisical access to the target n machine:

What is?

- Persistent copy of agent/soldier inside system BIOS;
- Require a physical access to target for installation;

]HackingTeam[

Both "agent" and "soldier" are the names of trojan horse applications also found in the leaks. The rootkit reinstalls these applications automatically, from infected firmware. Another slide describes options to modify firmware:

How can I deploy the Agent?

- Via SPI programmer circuit (physical access to motherboard);
- Via Service Mode (recovery device);
- Via firmware upgrade (actually SecureFlash limitation to bypass);

]HackingTeam[

Based on the information we have reviewed so far, we have not identified any new vulnerability used to bypass SPI flash protections. Some of the leaked messages clearly discuss using physical access and a SPI programmer to program the infected image to the SPI flash. There is also mention of a USB image that installs the malware using a UEFI application. This application (which appears to have been ported from an old and modified version of chipsec) erases and re-programs the SPI flash image from software. This method should only work on a system that is not configured to protect writes to the SPI flash. This serious configuration issue has been discussed by multiple researchers over many years, including A Tale of One Software Bypass of Windows 8 Secure Boot, Defeating Signed BIOS Enforcement, and Speed Racer. It is possible that HackingTeam (or others) could have other vulnerabilities that can be used to bypass protections and infect the SPI flash, but so far we have not seen it. This means that physical access and insecure configuration are the most likely attack vectors.

Now that we can see the threat posed by this firmware rootkit, we begin to examine the technical details of its implementation.

What does this malware do?

The infected BIOS image contains a DXE driver named "rkloader". As mentioned earlier, source code (which someone appears to have posted to GitHub) also appears in the leaked archive. During the DXE phase of UEFI firmware execution, modules are enumerated and executed automatically. The main routine of this "rkloader" module just registers a callback to execute the malicious payload.

```
EFI_STATUS
EFIAPI
_ModuleEntryPoint (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
    EFI_EVENT Event;

    DEBUG((EFI_D_INFO, "Running RK loader.\n"));
    InitializeLib(ImageHandle, SystemTable);

    gReceived = FALSE;  // reset event!

    //CpuBreakpoint();

    // wait for EFI EVENT GROUP READY TO BOOT
    gBootServices->CreateEventEx(0x200, 0x10, &CallbackSMI, NULL, &SMBIOS_TABLE_GUID, &Event);

    return EFI_SUCCESS;
}
```

The first parameter of *gBootServices->CreateEventEx()* routine indicates (*EFI_EVENT_GROUP_READY_TO_BOOT*), which is an event that occurs before the OS boot loader has been executed. This allows the malicious payload to gain execution before and OS (or even the boot loader) has had a chance to run.

The callback then loads another UEFI application, which is hard coded into rkloader source code (GUID named *LAUNCH_APP*).

```
...
EFI_GUID LAUNCH_APP =
{
    0xeaea9aec,
    0xc9c1,
    0x46e2,
    { 0x9d, 0x52, 0x43, 0x2a, 0xd2, 0x5a, 0x9b, 0x0b }
};

...

    NewFilePathProtocol = (EFI_DEVICE_PATH_PROTOCOL *) ((UINT8 *) NewDevicePathProtocol + DevicePathLength);
    NewFilePathProtocol->Type = 0x04;
    NewFilePathProtocol->SubType = 0x06;
    NewFilePathProtocol->Length[0] = 0x14;
    NewFilePathProtocol->Length[1] = 0x00;
    gBootServices->CopyMem(((CHAR8 *)(NewFilePathProtocol) + 4), &LAUNCH_APP, sizeof(EFI_GUID));


    NewDevicePathEnd = (EFI_DEVICE_PATH_PROTOCOL *) ((UINT8 *) NewDevicePathProtocol + DevicePathLength + sizeof(EFI_GUID) + 4);

    NewDevicePathEnd->Type = 0x7f;
    NewDevicePathEnd->SubType = 0xff;
    NewDevicePathEnd->Length[0] = 0x04;
    NewDevicePathEnd->Length[1] = 0x00;

    Status = gBootServices->LoadImage(FALSE, gImageHandle, NewDevicePathProtocol, NULL, 0, &ImageLoadedHandle);

...

EFI_STATUS
EFIAPI
_ModuleEntryPoint (
  IN EFI_HANDLE         ImageHandle,
  IN EFI_SYSTEM_TABLE   *SystemTable
  )
{
    EFI_EVENT Event;

    DEBUG((EFI_D_INFO, "Running RK loader.\n"));
    InitializeLib(ImageHandle, SystemTable);

    gReceived = FALSE;  // reset event!

    // wait for EFI EVENT GROUP READY TO BOOT
    gBootServices->CreateEventEx(0x200, 0x10, &CallbackSMI, NULL, &SMBIOS_TABLE_GUID, &Event);

    return EFI_SUCCESS;
}
```

This GUID identifies a UEFI application module with the name "fsbg". This application contains the main malicious functionality for stealth operations with an NTFS file system.

```
EFI_GUID LAUNCH_APP =
{
    0xeaea9aec,
    0xc9c1,
    0x46e2,
    { 0x9d, 0x52, 0x43, 0x2a, 0xd2, 0x5a, 0x9b, 0x0b }
};
```

```
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = fsbg
  FILE_GUID                      = eaea9aec-c9c1-46e2-9d52-432ad25a9b0b
  MODULE_TYPE                    = UEFI_APPLICATION
  VERSION_STRING                 = 1.0
  ENTRY_POINT                    = UefiMain
```

Here is full list of functions contained in "fsbg.c" module:

```
FIND_XXXXX_FILE_BUFFER_SIZE
CALC_OFFSET
UefiMain
CheckfTA
SetfTA
CheckAL
InstallAgent
InsertFileLock
RemoveFileLock
TestIsUserNotEmpty
FileHandleGetInfo
FileHandleSetPosition
FileHandleIsDirectory
FileHandleFindFirstFile
FileHandleRead
GetHandleListByProtocol
FileHandleFindNextFile
CheckUsers
GetImageFromFv
GetImageEx
UefiMain
```

The main workflow of fsbg dropper module is:

1. Check active infection by predefined UEFI variable "fTA"
2. Initialize Ntfs protocol
3. Find malicious executables in the BIOS image by predefined sections
4. Check for existing users on the machine by existing names in home directory
5. Install multiple malware executable modules: scoute.exe (backdoor) and soldier.exe (RCS agent).

An important note, which we will use for detection, is the use of the UEFI variable "fTA" to mark an infected system.

```
/**
    Leggo in NvRam la variabile fTA
**/
BOOLEAN
EFIAPI
CheckfTA()
{
    EFI_STATUS                    Status = EFI_SUCCESS;

    UINTN  VarDataSize;
    UINT8  VarData;


    VarData=0;
    VarDataSize=sizeof(VarData);
    Status=gRT->GetVariable(L"fTA", &gEfiGlobalFileVariableGuid, NULL, &VarDataSize, (UINTN*)&VarData);
```

The *EFI_GLOBAL_FILE_VARIABLE_GUID* is defined as follows:

```
#define EFI_GLOBAL_FILE_VARIABLE_GUID \
  { \
    0x8BE4DF61, 0x93CA, 0x11d2, {0xAA, 0x0D, 0x00, 0xE0, 0x98, 0x30, 0x22, 0x88} \
  }
```

The "fsbg" module implements file system operations. However the low-level code for interacting specifically with NTFS is factored into a separate module called "Ntfs" which is based on modified open-source implementations of NTFS file system drivers for UEFI. The "fsbg" module loads the "Ntfs" module as a system protocol driver.

```
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = Ntfs
  FILE_GUID                      = f50248a9-2f4d-4de9-86ae-bda84d07a41c
  MODULE_TYPE                    = UEFI_DRIVER
  VERSION_STRING                 = 1.0

  ENTRY_POINT                    = NtfsEntryPoint
  UNLOAD_IMAGE                   = NtfsUnload
```

The "fsbg" module references locations for malicious OS applications in the file system:

```
#define FILE_NAME_SCOUT L"\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\"
#define FILE_NAME_SOLDIER L"\\AppData\\Roaming\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\"
#define FILE_NAME_ELITE  L"\\AppData\\Local\\"
#define DIR_NAME_ELITE L"\\AppData\\Local\\Microsoft\\"


// (20 * (6+5+2))+1) unicode characters from EFI FAT spec (doubled for bytes)
#define MAX_FILE_NAME_LEN 512
#define FIND_XXXXX_FILE_BUFFER_SIZE (SIZE_OF_EFI_FILE_INFO + MAX_FILE_NAME_LEN)
#define CALC_OFFSET(type, base, offset) (type)((UINTN)base + (UINT32) offset)

#ifdef FORCE_DEBUG
UINT16 g_NAME_SCOUT[] =   L"scoute.exe";
UINT16 g_NAME_SOLDIER[] = L"soldier.exe";
UINT16 g_NAME_ELITE[]   = L"elite";
#else
//32 byte per inserire 16 caratteri unicode
UINT16 g_NAME_SCOUT[] =   L"6To_60S7K_FU06yjEhjh5dpFw96549UU";
UINT16 g_NAME_SOLDIER[] = L"kdfas7835jfwe09j29FKFLDOR3r35fJR";
UINT16 g_NAME_ELITE[]   = L"eorpekf3904kLDKQO023iosdn93smMXK";
#endif
```

From the above, it is easy to see how the this rootkit fulfills the attacker? goals of persistent infection. Even if the OS is reinstalled or the hard drive is replaced, the malicious DXE module continues to execute automatically and drops additional malicious files into the file system. While this appears to be Windows-specific, the modular architecture would allow for other file systems or applications to be used with a different OS. (Though, so far, we have not seen an implementation for any other OS.)

Firmware Rootkit Detection

We released CHIPSEC framework, which contains various tests and tools for platform security assessment, including some forensic capabilities. We leverage CHIPSEC in our examples below, but other tools can also be

used to accomplish the same thing.

Testing UEFI Firmware Protections in Flash Memory

Installing this firmware rootkit involves rewriting SPI flash. The system firmware is responsible for securely configuring the protections on SPI flash in order to prevent this. CHIPSEC contains configuration checks that users can easily run:

python chipsec_main.py -m common.bios_wp

If this test fails, it may be possible for software running on the system to modify the BIOS in the SPI flash due to insecure configuration of the hardware protections. **This does not mean that the system is infected, but it would be harder to infect a system that passes this test than one that fails.**

Signs of Infection

As we will detail in our next posthave explained above, the rootkit creates a UEFI variable called "fTA" and even uses this to check if it has already been installed. A user can list the UEFI variables on a system and search for this variable.

python chipsec_util.py uefi var-list

python chipsec_util.py uefi var-find fTA

After running the var-list command, users should look in the file efi_variables.lst for a variable with the name "fTA" and GUID *8BE4DF61-93CA-11d2-aa0d-00e098302288*. The var-find command in chipsec will specifically look for this variable. **Presence of the "fTA" variable indicates a system that might have been infected.**

Also, a user can read the entire image from the SPI flash and look inside it for unexpected changes. Ideally, a hardware programmer should be attached to independently capture an image of SPI flash from a suspect system. This should avoid advanced hiding techniques. (While not recommended, the image can be read from software using the chipsec command python chipsec_util.py spi dump spi.bin)

After an image has been captured from a suspect system and a known-good system of the same model, offline analysis that compares both images can begin. It helps to parse the firmware volumes and NVRAM variables that make up the image. With CHIPSEC, this can be done using the following command:

python chipsec_util.py decode spi.bin [fw type]

Note that NVRAM variables often change between platforms and even between reboots of the same platform. UEFI-based firmware Experience is recommended for this analysis. However, by comparing a suspect image with a known-good one, it is possible to find the extra modules added by the rootkit.

Finally, on an infected Windows system, malicious files (particularly scoute.exe or soldier.exe) may be stored in the following locations:

- AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup
- AppData\Local

- AppData\Local\Microsoft

This should be a good start for finding persistently infected systems.

Source: https://web.archive.org/web/20170313124421/http://www.intelsecurity.com/advanced-threat-research/content/data/HT-UEFI-rootkit.html