ERNW Newsletter 42 / December 2013

Dangers of Disabled Pre-Boot Authentication in Corporate Environments: Attacking Check Point´s Full Disk Encryption with Activated WIL

Version: 1.1

Date: 12/16/2013

Author(s): Felix Wilhelm, Friedwart Kuhn

Table of Content

1 ABSTRACT

In order to protect sensitive data on corporate laptops, most companies are using full disk encryption solutions. While native encryption products like Microsoft Bitlocker, Apple FileVault and open source solutions like TrueCrypt were already heavily scrutinized by security researchers, many popular commercial third party products are to some point still black boxes.

In this paper, we discuss Check Point *Full Disk Encryption* (FDE) with active "Windows Integrated Logon". Checkpoint FDE is a software package that is part of Check Point *Endpoint Security* and offers full disk encryption on Microsoft Windows and Mac OS X systems. The "Windows Integrated Logon" feature reduces total cost of ownership by disabling pre-boot authentication. Check Point warns about security risk associated with using this feature.

We argue that missing TPM integration and integrity checks make Check Point FDE with activated "Windows Integrated Logon" highly insecure against sophisticated attackers. Furthermore, we demonstrate the extraction of AES encryption keys on a running system and subsequent decryption of the encrypted disk.

Our analysis is limited to Check Point FDE v.7.4.9 on Windows operating systems and was performed during a penetration test of an encrypted customer enterprise laptop. Therefore, we concentrate on the client architecture and ignore other aspects like enterprise management interfaces.

2 CHECK POINT FDE ARCHITECTURE

2.1 Boot Process

Figure 1 Default boot process for WinNT systems

Figure 1 shows the components involved in the (today still usual) BIOS-based[1] startup of modern Windows systems. The *Master Boot Record (MBR)* is stored on the first sector of the hard drive and is used to locate the boot partition and the location of the *Volume Boot Record (VBR)*. The VBR is the first piece of code that actually understands the file system on the partition which is NTFS. It locates the Windows Boot Manager (*Bootmgr*) executable, loads it into memory and jumps to it.

The Bootmgr application reads the Boot Configuration Database (BCD) from the disk and is responsible for displaying the boot menu if needed. Furthermore, it can start the Windows recovery process, as well as the safe mode or memory check applications. Until now, the system still runs in 16-bit real mode and the boot manager switches into 32- or 64-bit mode depending on the version of Windows that gets executed.

When a boot entry is selected or the default entry gets chosen, Winload.exe is executed with the passed (default) arguments. Winload.exe is the first application that runs completely in 32-bit or 64-bit protected mode and is responsible for loading everything else needed for kernel initialization. This includes Ntoskrnl.exe, the actual kernel image, as well as all boot device drivers and file system drivers that are needed for a successful boot process. At the end, Winload.exe calls the startup function of Ntoskrnl.exe and the Windows kernel takes over.

Because hardware drivers are only initialized in the last step, all components before the kernel depend on BIOS interrupts to access the system hardware. INT 13h is (also traditionally) the most interesting interrupt for this discussion, because it is used to access the hard drive. It needs to be hooked in order to allow a transparent full disk encryption.

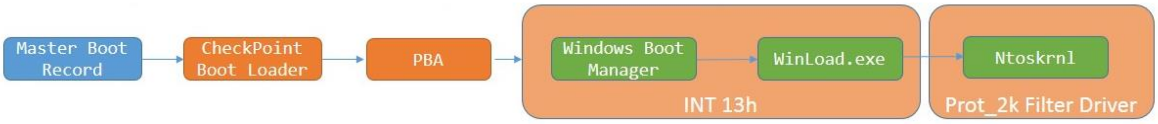

Figure 2 Boot process with activated Check Point FDE

Figure 2 shows the Boot process when Check Point FDE is activated. Instead of Microsoft's normal VBR a proprietary Check Point boot loader is stored at the front of the encrypted partition. It is responsible for initializing everything needed for the Pre-boot Authentication required to boot the system.

[1] *The analyzed Checkpoint FDE v7.4.9 does not support UEFI boot, v7.5 adds support for 64bit platforms. While this paper does not analyze a Check Point FDE installation on a system using a UEFI boot process, all findings are still relevant for such a system. While UEFI and "Secure Boot" can mitigate active attacks against the boot loader, WIL can be bypassed using only passive attacks (see chapter 3).*

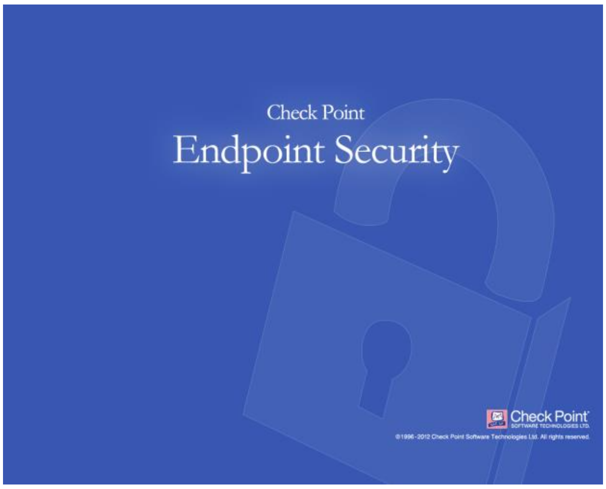

Figure 3 Check Point Bootscreen

In order to allow multiple user accounts, as well as password changes without re-encryption, Check Point FDE encrypted hard drives include a region called *system area*[2] which stores the user database. The user database itself is encrypted using a hardcoded AES-256 key. Decrypting this database makes it possible to enumerate available user IDs, as well as attacking user passwords using brute force attacks. However, the *system area* does not allow direct access to the keys needed for decryption of the rest of the hard drive. These keys are called partition keys. As the name implies, one key exist for each partition on the drive and the keys stay the same even if a user changes his password. When booting with Pre-Boot Authentication, the password entered into the login prompt is padded, combined with a random salt value and used to decrypt the partition keys.

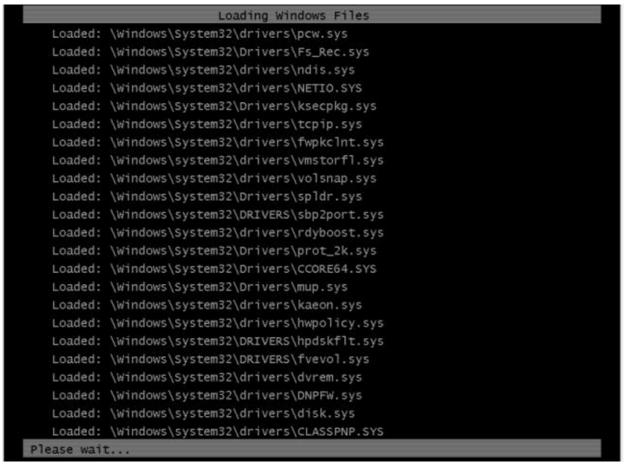

Figure 4 WinLoad.exe loading Check Point prot_2k filter driver

[2] *The* system area *is partially documented in the EAL4 certification report of "Check Point Endpoint Security – Full Disk Encryption", see* https://www.niap-ccevs.org/st/st_vid10194-st.pdf.

The key for the system partition is then used inside an INT13 hook to transparently decrypt all files that are requested from the hard disk. This allows successful execution of Bootmgr and WinLoad.exe, but is not enough to allow the actual Windows kernel to work. Instead a Filter driver called "prot_2k" is loaded which is described in Section 2.4.

2.2 Windows Integrated Logon (WIL)

To reduce the administrative overhead of activated encryption, Check Point offers a feature called „Windows Integrated Logon" (WIL). This removes the additional Pre-Boot Authentication step and directly boots into the Windows logon. In theory this is quite similar to Bitlockers TPM only mode. But while BitLocker with TPM performs so called measurements of the hardware (wherein the hard disk is integrated), thus preventing a boot of the operating system with modified hardware[3], Check Point does not support TPM chips. Because of that, WIL is inherently insecure. The Windows boot process already requires unencrypted partition keys to succeed and without user authentication or cryptographic hardware, this can only be realized by encrypting the partition keys with a (hard coded) key stored inside the Check Point Boot Loader. This flaw enables multiple different attacks described in the next sections.

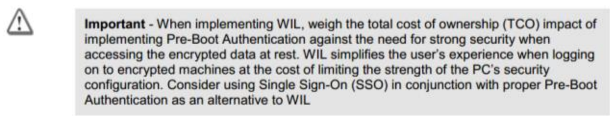
Important - When implementing WIL, weigh the total cost of ownership (TCO) impact of implementing Pre-Boot Authentication against the need for strong security when accessing the encrypted data at rest. WIL simplifies the user's experience when logging on to encrypted machines at the cost of limiting the strength of the PC's security configuration. Consider using Single Sign-On (SSO) in conjunction with proper Pre-Boot Authentication as an alternative to WIL

Figure 5 WIL Security Warning

As seen in Figure 5, the security risks of WIL are known to Check Point and they offer two "features" to increase its security:

- Network Locational Awareness: During the boot process, multiple hardcoded IPs are pinged to make sure that the system is part of the right network. If none responds, WIL is disabled and the User has to authenticate using Pre-Boot Authentication.
- Hardware Hash: Check Point generates a hardware hash out of different BIOS and CPU IDs and checks them during the boot process. If the hash differs WIL is disabled.

But: Network Locational Awareness can be trivially defeated by replying to all ping requests during the boot process. And: Because the hardware hash mechanism is only enforced using software, an attacker can patch the hash comparison or spoof the right ID values when booting the system in another device or a virtual environment.

2.3 Cryptocore

All Check Point applications, that are part of Check Points Endpoint Security solution, share the same FIPS certified cryptographic library called *Cryptocore*[4] . Cryptocore can be compiled for 16-, 32- and 64-bit mode and as a user space DLL or kernel module. It supports a large number of different cryptographic algorithms including symmetric ciphers like AES, DES and Blowfish, asymmetric Ciphers like RSA, as well as cryptographic hash algorithms and PRNGs.

On our test system the Cryptocore library was discovered at multiple locations, named as *cryptocore.dll*, *ccore32.sys* or *ccore64.sys*. All binaries share the same origin and only differ in target architecture and minor details depending on their use as kernel or user mode library.

---

[3] *This prevents for example the boot of the OS within another Computer.*
[4] http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp1959.pdf.

Large parts of Cryptocore are actually based on different open source libraries. The AES implementation, which was the main focus point of our analysis, is an extended version of Brian Gladmans open source AES library[5]. The biggest enhancement is support for Intel's AES-NI[6] instructions for faster AES execution as shown in Figure 6:

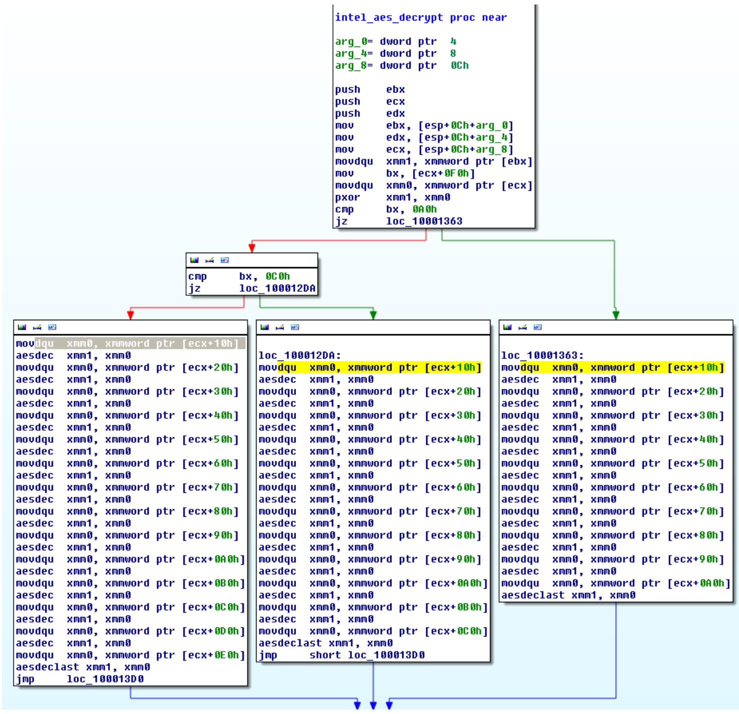

Figure 6 AES-NI 32bit decryption routine

The Cryptocore library exports a large number of functions for the different supported cryptographic operations. Additionally, a function called *initSystem* is exported, that executes a self-test involving all supported algorithms. For AES this includes encryption and decryption with known static keys. While this feature seems to be a requirement for the FIPS 140-2 certification, it does not offer any useful protection against practical attacks, including the ones described in Section 3.

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f

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

2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c 00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f

6f 00 4e 03 72 e4 d5 e3 34 58 00 55 1e 8b ee 15 43 e5 da 9d 49 6a b2 e2-8d 25 c8 f8 ca 7f 0b 06

c7 ac e1 be 82 e3 b2 b3 c2 ea ec b6 9e b3 d0 87 7a 48 06 a0 e4 a6 a7 ee-12 d5 7e 23 cf be c2 74
```

Listing 1 AES 256 keys used during Cryptocore self test

[5] http://gladman.plushost.co.uk/oldsite/AES/.
[6] http://software.intel.com/en-us/articles/intel-advanced-encryption-standard-instructions-aes-ni.

As most modern disk encryptions solutions, Check Point FDE uses block based encryption to encrypt the whole drive. On the analyzed system the used algorithm was AES with a 256 bit key. 512 bytes chunks are encrypted individually using CBC, whereas for each chunk the corresponding disk sector is used as an Initialization Vector. While support for the more secure XTS operation mode exists in Cryptocore, it is not used for disk encryption.

2.4 Filter Driver

After initialization of Ntoskrnl, Check Point FDE uses a filter driver to transparently de- and encrypt read and write requests to files on the hard drive. The driver is named `prot_2k` and creates two devices as shown in Figure 7:

```
ispatch routines:
00] IRP_MJ_CREATE                       fffff88001787fd8    prot_2k+0x1fd8
01] IRP_MJ_CREATE_NAMED_PIPE            fffff88001787fd8    prot_2k+0x1fd8
02] IRP_MJ_CLOSE                        fffff88001787fd8    prot_2k+0x1fd8
03] IRP_MJ_READ                         fffff8800178879c    prot_2k+0x279c
04] IRP_MJ_WRITE                        fffff8800178879c    prot_2k+0x279c
05] IRP_MJ_QUERY_INFORMATION            fffff88001787fd8    prot_2k+0x1fd8
06] IRP_MJ_SET_INFORMATION              fffff88001787fd8    prot_2k+0x1fd8
07] IRP_MJ_QUERY_EA                     fffff88001787fd8    prot_2k+0x1fd8
08] IRP_MJ_SET_EA                       fffff88001787fd8    prot_2k+0x1fd8
09] IRP_MJ_FLUSH_BUFFERS                fffff88001787fd8    prot_2k+0x1fd8
0a] IRP_MJ_QUERY_VOLUME_INFORMATION     fffff88001787fd8    prot_2k+0x1fd8
0b] IRP_MJ_SET_VOLUME_INFORMATION       fffff88001787fd8    prot_2k+0x1fd8
0c] IRP_MJ_DIRECTORY_CONTROL            fffff88001787fd8    prot_2k+0x1fd8
0d] IRP_MJ_FILE_SYSTEM_CONTROL          fffff88001787fd8    prot_2k+0x1fd8
0e] IRP_MJ_DEVICE_CONTROL               fffff88001787fd8    prot_2k+0x1fd8
0f] IRP_MJ_INTERNAL_DEVICE_CONTROL      fffff88001787fd8    prot_2k+0x1fd8
10] IRP_MJ_SHUTDOWN                     fffff88001787fd8    prot_2k+0x1fd8
11] IRP_MJ_LOCK_CONTROL                 fffff88001787fd8    prot_2k+0x1fd8
12] IRP_MJ_CLEANUP                      fffff88001787fd8    prot_2k+0x1fd8
13] IRP_MJ_CREATE_MAILSLOT              fffff88001787fd8    prot_2k+0x1fd8
14] IRP_MJ_QUERY_SECURITY               fffff88001787fd8    prot_2k+0x1fd8
15] IRP_MJ_SET_SECURITY                 fffff88001787fd8    prot_2k+0x1fd8
16] IRP_MJ_POWER                        fffff880017894cc    prot_2k+0x34cc
17] IRP_MJ_SYSTEM_CONTROL               fffff88001787fd8    prot_2k+0x1fd8
18] IRP_MJ_DEVICE_CHANGE                fffff88001787fd8    prot_2k+0x1fd8
19] IRP_MJ_QUERY_QUOTA                  fffff88001787fd8    prot_2k+0x1fd8
1a] IRP_MJ_SET_QUOTA                    fffff88001787fd8    prot_2k+0x1fd8
1b] IRP_MJ_PNP                          fffff880017892e0    prot_2k+0x32e0

d> !devobj fffffa8001f6fcb0
evice object (fffffa8001f6fcb0) is for:
ProtectNT \Driver\prot_2k DriverObject fffffa8001f6e480
urrent Irp 00000000 RefCount 2 Type 00000022 Flags 00000004
acl fffff9a10009c6c1 DevExt fffffa8001f6fe00 DevObjExt fffffa8001f6ff08
xtensionFlags (0x00000800)  DOE_DEFAULT_SD_PRESENT
haracteristics (0000000000)
evice queue is not busy.
d> !devobj fffffa8001f76040
evice object (fffffa8001f76040) is for:
 \Driver\prot_2k DriverObject fffffa8001f6e480
urrent Irp 00000000 RefCount 0 Type 00000007 Flags 00000010
pb fffffa8001f75570 DevExt fffffa8001f76190 DevObjExt fffffa8001f76298 Dope fffffa8001f76fa0
xtensionFlags (0x00000800)  DOE_DEFAULT_SD_PRESENT
haracteristics (0x00000100)  FILE_DEVICE_SECURE_OPEN
ttachedDevice (Upper) fffffa8001f76c10 \Driver\dvrem
ttachedTo (Lower) fffffa8001f75710 \Driver\Disk
evice queue is not busy.
```

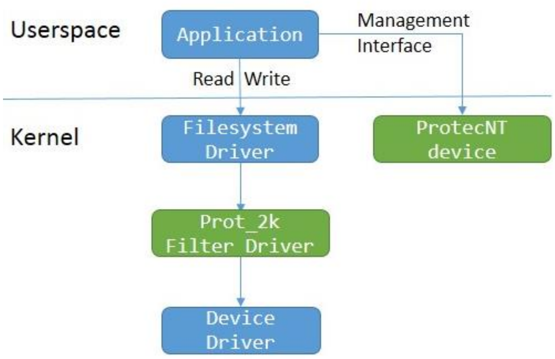

Figure 7 Dispatch routines and devices added by prot_2k

The nameless filter device sits between file system and device drivers and performs the needed encryption routines whenever an `IRP_MJ_READ` or `IRP_MJ_WRITE` request arrives. This happens whenever a user-mode application or another kernel components tries to access a file on disk and it ensures that all upper layer drivers and user space application do not need to be concerned with encryption.

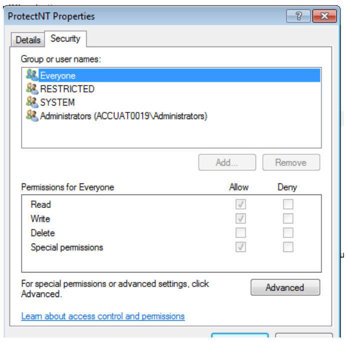

Figure 8 Security properties of the ProtectNT device

In addition to the filter driver which does not get directly used from user space applications, a device named `ProtectNT` is created which can be accessed from unprivileged applications as shown in Figure 8. This opens up an additional attack surface for local privilege escalation attacks, because unprivileged users can try to exploit potential security vulnerabilities in the device code.

When the Windows Kernel initializes the driver, it first performs a cryptographic self-test as described in 2.3. After that, it initializes an *AES key schedule*[7] for en- and decryption using the `aes_decrypt_key` and `aes_encrypt_key` functions included in the aforementioned AES library from Brian Gladman. Of course both key schedules are based on the same partition key. Figure 9 shows the source code of the described key schedule generator. Finally Figure 10 demonstrates the extraction of the used partition key by using a kernel debugger and a breakpoint to the AES encryption function.

```
AES_RETURN aes_encrypt_key256(const unsigned char *key, aes_encrypt_ctx cx[1])
{   uint_32t    ss[8];

    cx->ks[0] = ss[0] = word_in(key, 0);
    cx->ks[1] = ss[1] = word_in(key, 1);
    cx->ks[2] = ss[2] = word_in(key, 2);
    cx->ks[3] = ss[3] = word_in(key, 3);
    cx->ks[4] = ss[4] = word_in(key, 4);
    cx->ks[5] = ss[5] = word_in(key, 5);
    cx->ks[6] = ss[6] = word_in(key, 6);
    cx->ks[7] = ss[7] = word_in(key, 7);

    uint_32t i;
    for(i = 0; i < 6; ++i)
    ke8(cx->ks,  i);
    kef8(cx->ks, 6);
    cx->inf.l = 0;
    cx->inf.b[0] = 14 * 16;
    return EXIT_SUCCESS;
}
```

Figure 9 Brian Gladman's AES key schedule generator

[7] An AES key schedule *is a data structure that extends the 128, 194 or 256 bit long key into multiple separate round keys needed by the actual AES en- and decryption process. See* http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

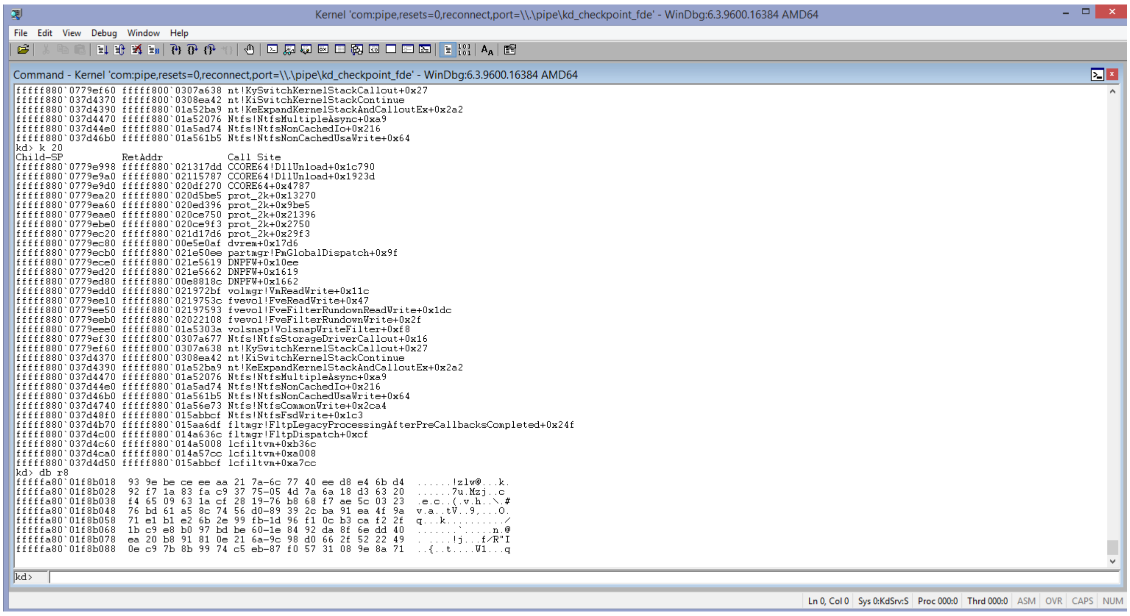

Figure 10 AES partition key schedule used for encryption during write request

3 ATTACKS AGAINST CHECK POINT FDE

In this chapter, possible attacks against Check Point FDE are highlighted. The first section describes general attacks that can be executed against any disk encryption solution that does not use Pre-Boot Authentication. Section 3.2 recaps hardware based DMA attacks and how they can be used to attack encrypted system. This is followed by a demonstration of how simple virtualization software can be used to completely bypass Check Point FDE with activated WIL. The last section discusses techniques that can be used to extract the encryption key out of a running system.

3.1 General Attacks Against FDE Solutions

All Full Disk Encryption solutions that works without Pre-Boot Authentication suffers from multiple weaknesses, which make them unsuitable for (highly) security sensitive environments:

- Weak Domain/Logon Credentials Security: Many corporate environments allow interactive logons to a single laptop with arbitrary valid domain credentials. While a login with a previously unused domain account requires a working networking connection to the domain, this might be possible for a sophisticated attacker. If the attacker has access to a highly privileged account or can use a privilege escalation exploit to gain admin rights, the encryption is practically bypassed. If the logon credentials are password based, password security and a robust password policy comes into play.
- Driver Vulnerabilities: Because the complete Kernel is already running during Windows Login process, vulnerabilities in hardware drivers can be exploited to bypass the drive encryption. An example for such vulnerabilities is the USB bugs patched with MS13-081[8].

3.2 DMA Attacks

DMA (Direct Memory Access) attacks using Firewire Ports were first publicly discussed in 2004[9]. They exploit a feature of the Firewire specification that enables read and write access to physical memory:

```
"physical requests, including physical read, physical write [..] are handled directly by
the Host Controller without assistance by system software." (1394 Open Host Controller
Interface Specification)
```

Because these requests are executed independently from software running on the target system, the operating system cannot protect itself against the extraction of sensitive data or even the malicious manipulation through memory modifications.

This opens up many critical attack vectors, if an attacker is able to connect his computer to a running target computer via Firewire. This is even possible when the laptop does not have an internal Firewire port: Windows automatically loads the needed driver when a Firewire adapter is inserted into PCMICA or ExpressCard slots.

After a successful connection is established, an attacker has several possibilities:

- Read the complete[10] RAM to extract sensitive information. The memory includes data of running applications, as well as data managed by the OS. This means: it might be possible to gain access to the content of an open Email or

[8] http://technet.microsoft.com/de-de/security/bulletin/ms13-081.
[9] http://md.hudora.de/presentations/firewire/PacSec2004.pdf.
[10] *Because the underlying protocol only supports 32-bit addressing only the first 4GB can be extracted.*

even to user passwords. As we will discuss in Section 3.4, this also enables the extraction of the constant partition keys used for disk encryption.

- Manipulate internal data structures of the OS to bypass login restriction or password checks.

While these attacks seem complex, there are several publicly available tools that perform them fully automatically. Figure 11 shows *Inception*[11] a tool that can dump the memory content of the target machine using the described Firewire attack. More importantly, it is able to patch the login functionality of different operating systems. In this screenshot, the Microsoft Windows login is modified to allow a login with a valid user name and any password.

```
 _|  _|      _|    _|_|_|  _|_|_|_|  _|_|_|    _|_|_|  _|    _|_|    _|      _|
 _|  _|_|    _|  _|        _|        _|    _|    _|    _|  _|    _|  _|_|    _|
 _|  _|  _|  _|  _|        _|_|_|    _|_|_|      _|    _|  _|    _|  _|  _|  _|
 _|  _|    _|_|  _|        _|        _|          _|    _|  _|    _|  _|    _|_|
 _|  _|      _|    _|_|_|  _|_|_|_|  _|          _|    _|    _|_|    _|      _|

v.0.2.4 (C) Carsten Maartmann-Moe 2013
Download: http://breaknenter.org/projects/inception | Twitter: @breaknenter

[*] Available targets:
-----------------------------------------------------------------------------------
[1] Windows 8: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[2] Windows 7: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[3] Windows Vista: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[4] Windows XP: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[5] Mac OS X: DirectoryService/OpenDirectory unlock/privilege escalation
[6] Ubuntu: libpam unlock/privilege escalation
[7] Linux Mint: libpam unlock/privilege escalation
-----------------------------------------------------------------------------------
[?] Please select target (or enter 'q' to quit): 2
[*] Selected target: Windows 7: msv1_0.dll MsvpPasswordValidate unlock/privilege
    escalation
[*] DMA shields should be down by now. Attacking...
^C================================================>                ]  160 MiB ( 75%)
```

Figure 11 Inception DMA attack

While DMA attacks are a problem for almost all currently used operating systems and disk encryption solutions, Check Point FDE offers additional attack vectors, because it does not recognize hardware manipulation. This means that an attacker can reduce the amount of system memory to enable an attack, add additional firewire devices or even boot the hard drive in a completely different system, which is specially prepared to enable a successful DMA attack. Of course, DMA attacks only work against running systems, therefore a turned off system with deactivated WIL is safe against this type of attack.

[11] http://www.breaknenter.org/projects/inception/.

3.3 Attacks Through Virtualization

Because Check Point's default configuration does not include any hardware checks, it is possible to boot an encrypted hard drive using standard virtualization software like VMware or VirtualBox. While this wouldn´t be an exposure with activated Pre-Boot Authentication, it completely bypasses any security measures for systems with activated WIL.

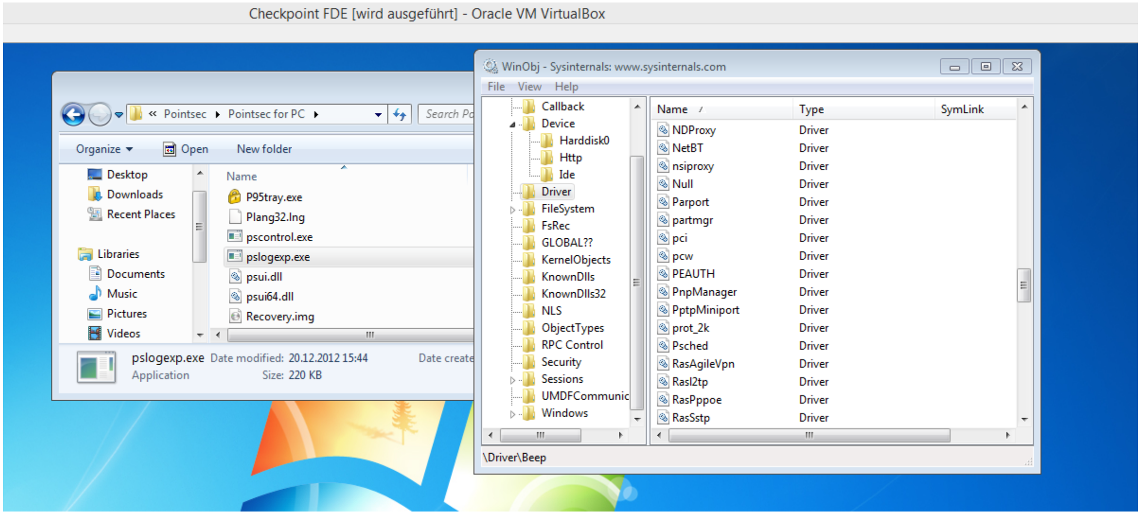

Figure 12 Sucessful Boot of encrypted system inside Virtualbox

Due to the nature of virtualization technology, the VM host has complete control over all aspects of the virtualized system. This includes the virtualized system memory and enables an attack that can be executed even by unsophisticated attackers:

- Boot the encrypted hard drive inside VMware Workstation (or similar virtualization software that supports raw memory snapshots)
- Pause the system when it finishes booting and displays the Windows login form. This creates multiple files describing the current state of the system. One of them is stored inside the VM directory and ends with *.vmem*. It is a 1:1 representation of the virtual system memory.
- Use inception to perform a DMA attack against the *vmem* file as shown in Figure 13.
- Continue execution of the VM and login with a valid username and any password.

```
felix@spoon ~/inception (git)-[master] % sudo python3.2 incept -f memdump.bin

INCEPTION

v.0.2.4 (C) Carsten Maartmann-Moe 2013
Download: http://breaknenter.org/projects/inception | Twitter: @breaknenter

[*] Available targets:
-------------------------------------------------------------------------------
[1] Windows 8: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[2] Windows 7: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[3] Windows Vista: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[4] Windows XP: msv1_0.dll MsvpPasswordValidate unlock/privilege escalation
[5] Mac OS X: DirectoryService/OpenDirectory unlock/privilege escalation
[6] Ubuntu: libpam unlock/privilege escalation
[7] Linux Mint: libpam unlock/privilege escalation
-------------------------------------------------------------------------------
```

Figure 13 Using Inception on memory dump file

This simple attack is sufficient against most systems with active WIL, but it has several downsides in the attacker´s perspective. An attacker requires a valid user name and has to login interactively into the system. Furthermore, many physical systems do not load the device driver needed by the VMware disk controller by default. This results in a blue screen with error code 0x7B, when the windows kernel tries to access the boot device. However, it is possible to extract the encryption keys out of the running system, even if it is not completely booted. These keys can then be used for offline decryption of the hard drive, which bypasses all the listed problems.

3.4 Attack Through Key Extraction

A major weakness of current encryption software is the need for keeping the encryption key in system memory during normal operation.[12] This means that an attacker who can read memory of a running system may be able to extract these keys and decrypt the disk. Because encryption keys are normally not coupled to user passwords an extracted encryption key stays valid over the whole life time of the encrypted disk.

While this problem is not limited to Check Point FDE and affects all major encryption solutions, systems using Check Point FDE with activated WIL are rendered completely insecure. As shown before in Figure 10, the partition key schedule of our test system can be extracted using WinDBG[13]

```
D:\>findaes.exe "C:\Users\Felix\Documents\Virtual Machines\Windows 7 x64\Windows
 7 x64-ee17a77d.vmem"
Searching C:\Users\Felix\Documents\Virtual Machines\Windows 7 x64\Windows 7 x64-
ee17a77d.vmem
Found AES-256 key schedule at offset 0x4b7aeb8:
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a
 1b 1c 1d 1e 1f
Found AES-256 key schedule at offset 0x4b7cf70:
56 38 ef 45 d9 28 e6 11 80 05 20 ba 19 8d 82 d4 47 dc 49 ed 42 e2 4a 40 4c 6a dc
 60 48 31 63 35
Found AES-256 key schedule at offset 0x4b7f088:
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
Found AES-256 key schedule at offset 0x7c306308:
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
Found AES-256 key schedule at offset 0x7c3081f0:
56 38 ef 45 d9 28 e6 11 80 05 20 ba 19 8d 82 d4 47 dc 49 ed 42 e2 4a 40 4c 6a dc
 60 48 31 63 35
Found AES-256 key schedule at offset 0x7c30a138:
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a
 1b 1c 1d 1e 1f
Found AES-256 key schedule at offset 0x7f4ac018:
93 9e be ce ee aa 21 7a 6c 77 40 ee d8 e4 6b d4 92 f7 1a 83 fa c9 37 75 05 4d 7a
 6a 18 d3 63 20
Found AES-256 key schedule at offset 0x7f4b2018:
93 9e be ce ee aa 21 7a 6c 77 40 ee d8 e4 6b d4 92 f7 1a 83 fa c9 37 75 05 4d 7a
 6a 18 d3 63 20
Found AES-256 key schedule at offset 0x7f4b8018:
93 9e be ce ee aa 21 7a 6c 77 40 ee d8 e4 6b d4 92 f7 1a 83 fa c9 37 75 05 4d 7a
 6a 18 d3 63 20
Found AES-256 key schedule at offset 0x7fbb0018:
93 9e be ce ee aa 21 7a 6c 77 40 ee d8 e4 6b d4 92 f7 1a 83 fa c9 37 75 05 4d 7a
 6a 18 d3 63 20
```

Figure 14 Extracting different example keys and the partition key using FindAES

Figure 14 demonstrates the extraction of the same key using the "findaes.exe"[14] tool. This key can than be used to decrypt the mounted hard drive from a different system using standard cryptographic libraries, as shown below.

[12] *Neither popular encryption software nor popular operating systems support holomorphic encryption.*
[13] *http://msdn.microsoft.com/en-us/library/windows/hardware/ff551063(v=vs.85).aspx*
[14] http://sourceforge.net/projects/findaes/

```
felix@spoon ~/aespython/pythonaes (git)-[master] % python checkpoint_working.py
 /media/felix/Elements/disk_100000h_9A88294Ch.dump | strings -n 8 | grep -A5 -B
5 KERNEL32.dll
GetSystemTimeAsFileTime
TerminateProcess
GetCurrentProcess
UnhandledExceptionFilter
SetUnhandledExceptionFilter
KERNEL32.dll
appidapi.dll
AppIDDecodeAttributeString
AppIDEncodeAttributeString
AppIDFreeAttributeString
AppIDGetFileAttributes
```

Figure 15 Decrypting Check Point FDE disk using extracted partition key

4 CONCLUSION & MITIGATING CONTROLS

Due to the vulnerabilities described in Section 3, Check Point FDE with WIL should not be used for security sensitive environments. While features like hardware verification (Section 2.2) can hinder unsophisticated attackers, they are not sufficient without hardware integration like TPM.

If you use Check Point FDE or other/similar encryption solutions in corporate environments consider and keep in mind the following aspects:

- **Carefully define focus (FDE, E-Mail, removable media etc.) and security requirements for the encryption solution** (integration in your hard- and software environment, password/credential management, operational feasibility[15], certification (CC et. al.) etc.)
- **Evaluate various solutions against your (security) requirements.** Common FDE solution providers for Windows based operating systems (apart from Check Point) are McAfee (Endpoint Encryption), Microsoft (BitLocker), Secude (Full Disk Encryption), Symantec (PGP) etc. Evaluate only vendors which implement reliable encryption technology.[16]
- **Follow vendor recommendations**. Check Point does not recommend the activation of WIL and recommends its "Unlock on LAN" feature as a safer alternative for environments where PBA is not feasible. While we did not analyze this feature and therefore cannot guarantee a safe implementation, the overall architecture seems much more promising than WIL. However, for security sensitive environments all vendors recommend:
- **Implement Pre-Boot Authentication.**
  Implement Pre-Boot Authentication.
  Implement Pre-Boot Authentication.[17]
  Lack of implemented Pre-Boot Authentication and resulting vulnerabilities (= encryption depends only on user credentials) have been the major reasons for enterprise client security issues of security assessments we performed in corporate environments. Enterprise solutions (such as the above mentioned solutions) enable PBA implementation (and associated password management) together with Active Directory integration. So the user has to remember only one password (or PIN) in order to successfully logon.
- **Implement a robust password policy**. Encryption software solutions rely in many cases on passwords in the way that encryption keys are often derived from passwords. So make sure that users use a secure password.
- **Control interfaces**. In particular interfaces that permit Direct Memory Access (DMA) like Firewire or Thunderbolt should be controlled by a technically implemented policy. In Windows environments this may be easily implemented via SPB2 driver blocking through Group Policy.[18]
- Last but not least: **Define and implement an encryption policy**. In order to ensure compliance to organizational (encryption) requirements, define an organization-wide encryption policy. One of the chapters of this policy should contain information of how to implement, configure and maintain FDE on corporate hardware (at least on notebooks). This policy should rely on current encryption technology (see above). This policy should consider amongst others, the following aspects:
  - *Do not leave powered-on systems unattended*

[15] *See* http://www.insinuator.net/2011/05/evaluating-operational-feasibility/.
[16] *See the ENISA recommendations:* http://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report/at_download/fullReport.
[17] *Yes, you saw that correctly ;-) We repeated this recommendation three times!*
[18] *See* http://support.microsoft.com/kb/2516445. *In our observation, these GPO settings are sometimes more effective then 3rd. party vendor software solutions. For Hardening of Windows environments see our Newsletter Nr. 40 at* https://www.ernw.de/newsletter/newsletter-40-july-2012-windows-server-2008-r2-und-active-directory-bsi-compliant-gehartet/index.html.

- *Compromised systems require a re-encryption*
- etc.

Don´t forget, that an encryption policy requires a **data classification** or more generally spoken a classification of your assets.[19]

[19] *ERNW has written encryption policies for our customers and ERNW has done data classification for customers as well. If you need help, please contact us.*

5 Further Research & Related Work

5.1 Further Research

While the attacks presented in Section 3 are sufficient to break encrypted systems when WIL is activated, further research could improve our results in several ways:

- Offline Decryption: Currently, the extraction of the partition key still requires running the Check Point boot loader inside a virtual environment. Due to the aforementioned design flaws of WIL the partition key has to be stored inside the system area, possibly encrypted with a static key. A complete reverse engineering of the boot loader and system area would make it possible to extract the needed partition key out of a disk image without the need for any virtualization technology.
- Brute Force Attacks: Currently no tools exist to perform a brute force attack against encrypted systems that do not use WIL. Because a direct attack on the random partition key is infeasible, this would require a deeper understanding of the system area and the mechanism used to decrypt partition keys using the entered user password. Weak user passwords could then be cracked using high speed offline brute force attacks.
- Driver Vulnerabilities: The prot_2k device driver opens a large attack surface to unprivileged users. This could potentially be used for privilege escalation attacks by a sophisticated attacker. A complete analysis of the device driver interfaces should be conducted.

5.2 Related Work

Self-Encrypting Disks pose Self-Decrypting Risks - *https://www1.informatik.uni-erlangen.de/filepool/projects/sed/seds-at-risks.pdf.*

Full Disk Encryption Crash Course *http://events.ccc.de/congress/2008/Fahrplan/attachments/1190_Full-Disk-Encryption_Crash-Course_Paper.pdf.*

Infiltrate the Vault: Security Analysis and Decryption of Lion Full Disk Encryption *http://eprint.iacr.org/2012/374.pdf*

Last but not least, we Remember: Cold Boot Attacks on Encryption Keys *https://citp.princeton.edu/research/memory/.*