PV204: Disk encryption lab

Apr 10, 2019, Milan Broz <xbroz@fi.muni.cz>

Centre for Research on Cryptography and Security

Introduction

Encryption can provide confidentiality of user data. It can be implemented on different layers, including application, file system or storage device.

Application layer encryption examples are PGP or ZIP compression with password.

Encryption of files (inside filesystem or through independent layer like Linux eCryptfs) provides more generic solution. Yet some parts (like filesystem metadata) are still unencrypted. However this solution provides encrypted data with private key per user. (Every user can have own directory encrypted by own key.)

Encryption of the low-level storage (disk) is called *Full Disk Encryption* (FDE). It is completely transparent to the user (no need to choose what to encrypt – the whole disk is encrypted). The encrypted disk behaves as the same as a disk without encryption.

The major disadvantage is that everyone who knows the password can read the whole disk. Often we combine FDE with another encryption layer. The primary use of FDE is to *provide data confidentiality in power-down mode* (stolen laptop does not leak user data).

Once the disk is unlocked, the main encryption key remains in system, usually directly in system RAM. Exercise II will show how easy is to get this key from memory image of system.

Another disadvantage of FDE is that it usually cannot guarantee integrity of data. Encryption is fully transparent and length-preserving, the ciphertext and plaintext device are of the same size. There is no space to store any integrity information. This allows attacks by direct modification of ciphertext.

The FDE works on the sector level, as the same as the block device. Atomic I/O access units for encrypted devices are sectors. Sectors are encrypted independently by a symmetric cipher. Cipher block size is (in most cases) smaller than the size of a sector. It means that inside sector we need to use an encryption mode (suitable for storage encryption).

It is crucial that the same data in different sectors must produce different ciphertext pattern. This is achieved by a per-sector tweak (IV, Initialization Vector) that is usually derived from logical sector offset (sector number). Combination of properly used IV and encryption mode is critical for the security of the FDE system. *Figure 1* shows an example of wrongly used mode for encryption (here ECB, Electronic Codebook). Another problems could appear with CBC mode [CBC] and predictable IV. Then attacker can use predictability of IV and create watermarks and detect them directly from ciphertext. We will show this flaw on legacy TrueCrypt device in Exercise III.

Today, the most used encryption mode for disks is XTS [XTS]. This mode is constructed such way, that it can safely use predictable sector number directly as an IV. Schema of CBC and XTS encryption mode is for reference in Appendix.

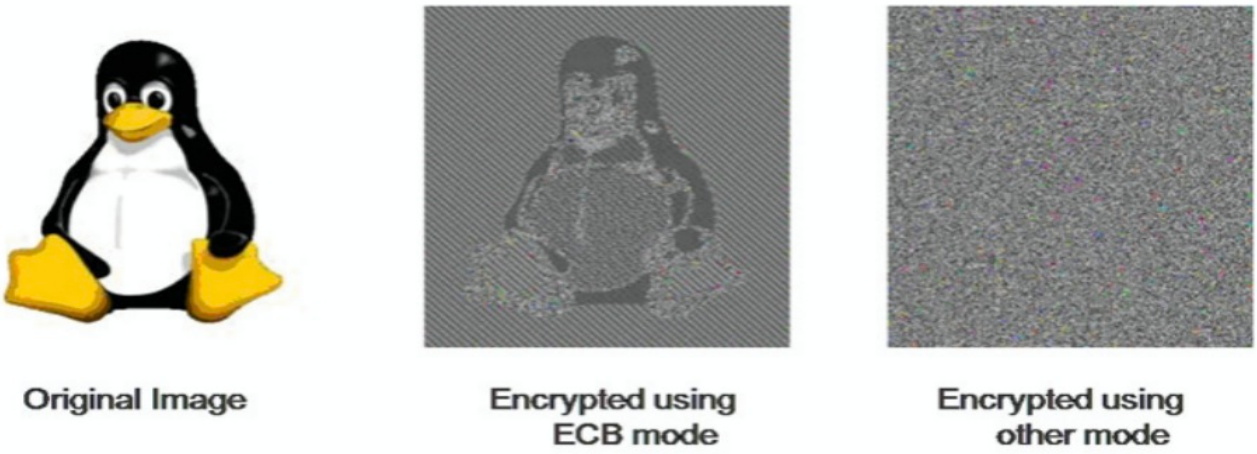


Figure 1. An image encrypted in ECB mode (without tweaked-sector encryption).

FDE can be implemented on hardware layer (self-encrypted disk, encryption inside controller chipset) or in software [FDE].
Software implementation is for example Bitlocker for Windows systems, dm-crypt in Linux, FileVault on MacOS or (abandoned) TrueCrypt. There are many other tools implementing this functionality [COMP].

In the following exercises we will use two software FDE tools. The first one is LUKS/dm-crypt [LUKS] which is primary solution for Linux based systems and VeraCrypt/TrueCrypt [TC].

Both tools can create a virtual encrypted block device within a partition or a regular file.
This virtual disk can be formatted as if it was a physical device and mounted.
TrueCrypt also supports trivial steganography and allows to create a hidden virtual volume within another volume.

References

[FDE] Disk encryption
http://en.wikipedia.org/wiki/Disk_encryption

[COMP] Comparison of disk encryption software
http://en.wikipedia.org/wiki/Comparison_of_disk_encryption_software

[CBC] Cipher Block Chaining mode
http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[XTS] IEEE P1619™/D16 Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices (AES-XTS)
http://grouper.ieee.org/groups/1619/email/pdf00086.pdf

[LUKS] Linux Unified Key Setup / cryptsetup
https://gitlab.com/cryptsetup/cryptsetup/

[TC] VeraCrypt (former TrueCrypt), Free open-source disk encryption software
https://veracrypt.codeplex.com/

Exercises

We will use prepared Virtual Machine PV204 (slightly modified Debian Linux).
VM can be run in Oracle VirtualBox.
The virtual machine has preinstalled LUKS disk encryption of the whole system, **cryptsetup** 2.1 (it can open TrueCrypt and VeraCrypt devices as well), **TrueCrypt** 7.1a and **VeraCrypt** 1.23.
.
For experiments you can use image in file **/home/pv204/testdisk.img**.

User name: pv204
Password (also unlocking password for disk)**: pv204**

Note: User pv204 can run superuser commands via sudo without need of entering password, e.g. ***sudo ls /root****.*

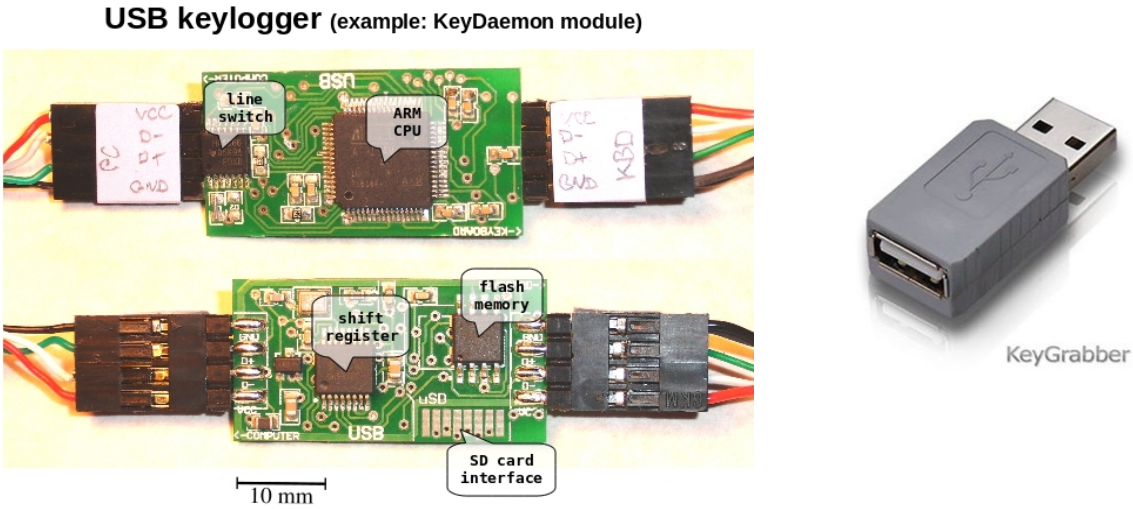
Exercise I: Hardware keylogger use

"If you let your machine out of your sight, it's no longer your machine."

In a context of disk encryption specialized hw device can be used to attack entered passphrase (BIOS password, disk encryption unlocking passphrase).

Hardware keylogger is a device which intercepts data entered through keyboard and store it internally for later analysis. The device can also transmit data using wireless network or another hidden channel. If properly masked (e.g. in keyboard controller) it is very hard to reveal it by user because device is completely transparent to BIOS and operating system (and power consumption is minimal).

Figure 2. Example keylogger module



We will use simple example of commercially available USB keylogger module (2010/2014) intended for built-in applications. Used version has only 4MB internal memory for log but extended version provides SD card for log store (enough storage for almost the whole physical life of the keyboard).

Log retrieval is activated by an activation shortcut, which switches keylogger into USB drive emulation mode and provides direct access to log storage (file).

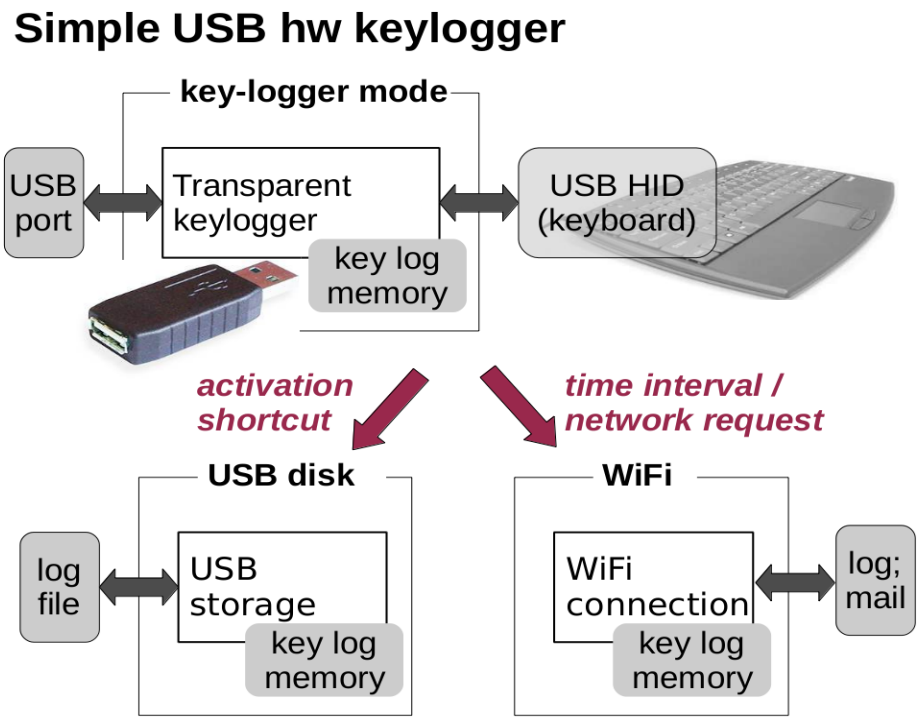


Figure 3. Log retrieval from the module

Your task

1. Connect provided USB keyboard through cable with USB keylogger module.
2. Check that system recognizes keyboard (as the same as it is connected directly).
3. Simulate some work where you need enter sensitive data, e.g. create new TrueCrypt encrypted device, boot testing virtual machine and log in or try to enter some password in browser.
4. Switch to host system file explorer and press activation shortcut :
   'K' + 'B' + 'S' key together.
   The keyboard should detach and you should see new virtual USB flash disk.
5. Investigate "LOG.txt" file on the disk and compare it with your work in step 3.
6. Please delete the "LOG.txt" so your colleagues will not see your "sensitive" data :-)

Exercise II: Obtaining encryption key from VM memory image

For most of the software-based disk encryption applications the encryption run on the CPU, usually inside OS context (optionally with HW acceleration like AES-NI instructions).

In this operating mode the encryption key is present in RAM during the operation. An attacker can obtain image of physical memory and try to search for the encryption key. This image can be either obtained though software (we will use VirtualBox debugger function) or through hardware (Cold boot attack, i.e. recover memory content after force reboot/short powerdown or use FireWire hw debug functions). The obtained key can be directly used for storage decryption (without password).

Next problem is to identify candidate keys in memory image. This can be done by recognizing internal OS structures in memory or by a more generic approach, like reconstructing AES keys by identifying specific round keys [COLD].

Your Task

1. Copy provided image and prepare Virtual Machine with some active and mounted encrypted devices and understand the storage stack inside.

   You should have two encrypted and active mappings for this exercise in VM. One is system encryption itself (LUKS/dm-crypt) and second will be VeraCrypt device created from disk image **testdisk.img**.

   a) Start and login to VM.
   b) Run VeraCrypt GUI (on desktop) and create new image **testdisk.img** (please use AES only for encryption algorithm).
   c) Mount image through VeraCrypt GUI and investigate used parameters (Volume Properties). Keep it mounted to investigate key in memory later.
   d) Investigate and understand storage stack, dump parameters about encrypted virtual devices. Use **lsblk** command to get the whole picture:

```
root@pv204:/home/pv204# lsblk
NAME                  MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINT
loop0                   7:0    0   32M  0 loop
└─veracrypt1          254:3    0 31.8M  0 dm    /media/veracrypt1
sda                     8:0    0   16G  0 disk
├─sda1                  8:1    0  243M  0 part  /boot
├─sda2                  8:2    0    1K  0 part
└─sda5                  8:5    0 15.8G  0 part
  └─sda5_crypt        254:0    0 15.8G  0 crypt
    ├─pv204--vg-root  254:1    0 15.1G  0 lvm   /
    └─pv204--vg-swap_1 254:2   0  708M  0 lvm   [SWAP]
```

   In this example, **sda5** is your system encrypted partition, **loop0** is the VeraCrypt (or TrueCrypt) device mapped to **testimage.img**.

Check VeraCrypt device - note the header is encrypted so you cannot get information using **blkid**. Use cryptsetup TrueCrypt/VeraCrypt extension to get more info about device on-disk header:

```
$ sudo blkid testdisk.img
```

```
$ sudo cryptsetup tcryptDump testdisk.img –veracrypt –dump-master-key
```

```
Enter passphrase for testdisk.img:
TCRYPT header information for testdisk.img
Cipher chain:   aes
Cipher mode:    xts-plain64
Payload offset: 256
MK bits:        512
MK dump:        a8 8e 10 08 2d a1 a1 c3 b3 bf f0 4d 61 ab 16 1b
                80 3d 4a 17 ce 56 eb 02 a5 df 72 41 55 b5 4b 9c
                bb d0 19 94 01 be 3b 97 11 6f e7 04 29 ba bb c4
                19 17 8f 2e 7b 30 f9 f3 8c 10 41 35 6a 89 0d 91
```

Note: always add `--veracrypt` option to cryptsetup optiond (otherwise cryptsetup will not recognize VeraCrypt format).

Now, try to display encrypted keys directly from kernel (superuser can display full device-mapper dm-crypt mapping table for active devices).

```
pv204@pv204:~$ sudo dmsetup table --showkeys veracrypt1
0 65024 crypt aes-xts-plain64 a88e10082da1a1c3b3bff04d61ab161b803d4a17ce56eb02a5df724155b54b9cbbd0199
401be3b97116fe70429babbc419178f2e7b30f9f38c1041356a890d91 256 7:0 256
```

e) Repeat the same exercise for LUKS device (**/dev/sda5** mapped to **sda5_crypt**), just use **`luksDump`** command. Note LUKS has visible header on-disk.

2. While the VM is still running, obtain snapshot of memory core image.
   You can use **Vbox_save_memcore.bat** script which uses internal VirtualBox debugger to dump VM memory core (it is in fact ELF format dump, but it is enough for our use).
3. Analyze the memory core image with the provided aeskeyfind program. Note this program search for specific AES structures. (In reality you have key in memory more times but in different format, just think about screen buffer with text dumps executed in step 1.)
4. Compare possible encryption keys with information you obtained in step 1.
   See XTS encryption mode definition [XTS] and think why you see two separate AES keys for one device.

References

[COLD] *Lest We Remember*: Cold Boot Attacks on Encryption Keys
https://citp.princeton.edu/research/memory/

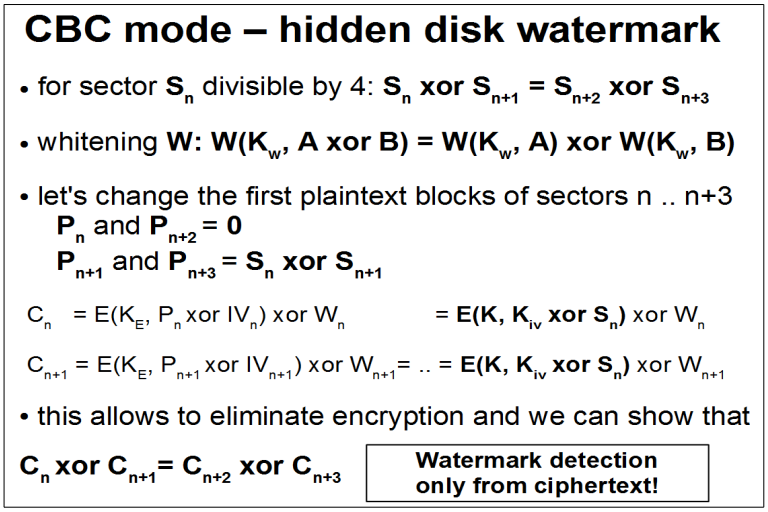
[XTS] IEEE P1619™/D16 Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices (AES-XTS)
http://grouper.ieee.org/groups/1619/email/pdf00086.pdf

Optional/Advanced Exercise III: Revealing TrueCrypt hidden disk existence (CBC mode)

The goal of this exercise is to show that old CBC mode in TrueCrypt is vulnerable to watermarking attack and that such attack could be used to reveal hidden disk existence just from ciphertext analysis.

Theory

The TrueCrypt (in version before 4.1) used CBC mode with Initialization Vector calculated from the sector number and xored with secret key K_{IV}. Also there is additional whitening which is calculated using several CRC32 operations over secret key K_W xored with sector number. Final whitening value is then applied to the whole plaintext sector with plain xor function. Unfortunately, both operations are not secure (IV is still partially predictable because for consecutive sectors we know which bits will change, and the whitening is just linear transformation, CRC32 is not cryptographically secure). The figure shows simplified attack with P (plaintext block) and C (ciphertext block). For more info see [TC4].



This attack allows construct a special plaintext, which if aligned to proper sectors can propagate special pattern (watermark) into ciphertext. Because filesystem aligns start of files to sector offset, we can just use specially formatted file in hidden disk and then search for the watermark on ciphertext device. As the exercise will show, this special file can be text file, so forcing user to store such file in the hidden disk is not too complicated (imagine mail or picture in browser cache).

Your Task

For the task use TrueCrypt installed in Virtual Machine (VeraCrypt no longer supports CBC mode). The exercise files are already copied to /home/task3 inside VM.

1. Use the provided TrueCrypt container in **/home/task3/tc_hidden_cbc.tc** and try to open it in TrueCrypt.
   The password to outer (decoy) volume is "password", password to hidden volume is "hidden".
2. Using the provided program **/home/task3/create_file <name>** (which implements attack above) generate special file and copy it to hidden TrueCrypt disk.
3. Dismount the TrueCrypt device and run **/home/task3/detect_file** over the TrueCrypt container image. It should find the pattern in special file and print the offset of this pattern.
4. See the create_file and detect_file source code (/home/task3/src) to better understand internal operation.

References

[TC4] sci-crypt mailinglist, *TrueCrypt 4.0 Out*
https://groups.google.com/forum/#!topic/sci.crypt/3DxOChZ0lrQ[1-25-false]

Assignment

There is **assignment.zip** in study materials which contains **pv204_assignment.tc** (TrueCrypt virtual volume compatible with TrueCrypt 7 and VeraCrypt 1.x).

Note: the same files are in `/home/assignment` directory in provided VM used in Exercise II.

The volume is protected by a 9-character long password, which begins with "**pv204_**XXX" where X means digit [0-9].

Your task

1. Find the password and unlock the volume.

2. Investigate the master encryption keys and primary header salt.
   What's wrong with them?

3. Describe your findings in text file:

   - The password to the volume.

   - How did you find the password.
   Attach the script or source code of your password search utility.

   - Describe which cipher chain (names of ciphers) is used for the volume.

   - Write in hexa format volume **salt** (for the primary TrueCrypt header, from the first sector).

   - Write in hexa **master keys** for all used ciphers.

   - Shortly describe (max. 10 sentences) what is wrong with the keys (or salt).
   Are the keys independent and randomly generated?

   - Bonus: try to check how the key(s) were generated (hint: just use Google).

   Pack description text file and scripts or source files to **<UCO>.zip** archive and upload it to IS.

Notes

The volume uses cipher chain (more cipher in sequence), note the most password crackers do NOT support this mode.
You can simply use wrapper script for TrueCrypt or cryptsetup commandline or generate candidate password list and use dictionary search example in cryptsetup sources.

To display hexa keys you can use commands from Exercise II.
To display salt just check TrueCrypt header format (to see the salt position) and use hexa editor, hexdump or similar tool. See Appendix for the on-disk format description.

Appendix – TrueCrypt on-disk header format

TrueCrypt Volume Format Specification

Offset (bytes)	Size (bytes)	Encryption Status[*]	Description
0	64	Unencrypted[‡]	Salt
64	4	Encrypted	ASCII string “TRUE”
68	2	Encrypted	Volume header format version (5)
70	2	Encrypted	Minimum program version required to open the volume
72	4	Encrypted	CRC-32 checksum of the (decrypted) bytes 256–511
76	16	Encrypted	Reserved (must contain zeroes)
92	8	Encrypted	Size of hidden volume (set to zero in non-hidden volumes)
100	8	Encrypted	Size of volume
108	8	Encrypted	Byte offset of the start of the master key scope
116	8	Encrypted	Size of the encrypted area within the master key scope
124	4	Encrypted	Flag bits (bit 0 set: system encryption; bit 1 set: non-system in-place-encrypted volume; bits 2-31 are reserved)
128	4	Encrypted	Sector size (in bytes)
132	120	Encrypted	Reserved (must contain zeroes)
252	4	Encrypted	CRC-32 checksum of the (decrypted) bytes 64–251
256	*Var.*	Encrypted	Concatenated primary and secondary master keys[§]
512	65024	Encrypted	Reserved (for system encryption, this item is omitted[††])
65536	65536	Encrypted / Unencrypted[‡]	Area for hidden volume header (if there is no hidden volume within the volume, this area contains random data[**]). For system encryption, this item is omitted.[††] See bytes 0–65535.
131072	*Var.*	Encrypted	Data area (master key scope). For system encryption, offset may be different (depending on offset of system partition).
S–131072[†]	65536	Encrypted / Unencrypted[‡]	Backup header (encrypted with a different header key derived using a different salt). For system encryption, this item is omitted.[††] See bytes 0–65535.
S–65536	65536	Encrypted / Unencrypted[‡]	Backup header for hidden volume (encrypted with a different header key derived using a different salt). If there is no hidden volume within the volume, this area contains random data.[**] For system encryption, this item is omitted.[††] See bytes 0–65535.

[*] The encrypted areas of the volume header are encrypted in XTS mode using the primary and secondary header keys. For more information, see the section *Encryption Scheme* and the section *Header Key Derivation, Salt, and Iteration Count*.
[†] *S* denotes the size of the volume host (in bytes).
[‡] Note that the salt does not need to be encrypted, as it does not have to be kept secret [7] (salt is a sequence of random values).
[§] Multiple concatenated master keys are stored here when the volume is encrypted using a cascade of ciphers (secondary master keys are used for XTS mode).
[**] See below in this section for information on the method used to fill free volume space with random data when the volume is created.
[††] Here, the meaning of "system encryption" does not include a hidden volume containing a hidden operating system.