VeraCrypt 1.18 Security Assessment

Technical Report

Ref.	16-08-215-REP
Version	1.0
Date	17 October, 2016
Prepared for	OSTIF
Performed by	Quarkslab

We are building and improving powerful security tools to protect information around the world.

Contents

1. Project information

Document Change Log			
Version	**Date**	**Change**	**Authors**
0.1	18/08/2016	Creation	Marion Videau
0.2	09/09/2016	First draft sent to VeraCrypt	Jean-Baptiste Bédrune Marion Videau
0.3	14/09/2016	Second draft sent to VeraCrypt	Jean-Baptiste Bédrune Marion Videau
0.8	16/09/2016	Draft internally reviewed	Jean-Baptiste Bédrune Marion Videau
0.9	20/09/2016	Reviewed	Fred Raynal
0.9	30/09/2016	Reviewed	Mounir Idrassi
1.0	05/10/2016	Delivered to OSTIF	Jean-Baptiste Bédrune Marion Videau
1.0	13/10/2016	Reviewed	Derek Zimmer
1.0	17/10/2016	**Published**	

Quarkslab		
Quarkslab SAS, 13 rue Saint Ambroise, 75011 Paris, France		
Contact	**Role**	**Contact information**
Frédéric Raynal	CEO and Founder	fraynal@quarkslab.com
Jean-Baptiste Bédrune	R&D Engineer	jbbedrune@quarkslab.com
Marion Videau	R&D Engineer	mvideau@quarkslab.com

Open Source Technology Improvement Fund		
Contact	**Role**	**Contact information**
Derek Zimmer	President and Founder	derek@ostif.org

VeraCrypt Project		
Contact	**Role**	**Contact information**
Mounir Idrassi	Main Developer	mounir.idrassi@idrix.fr

2. Executive Summary

This report describes the results of the security assessment of VeraCrypt 1.18 made by Quarkslab between Aug. 16 and Sep. 14, 2016 and funded by OSTIF. Two Quarkslab engineers worked on this audit, for a total of 32 man-days of study.

The audit followed two lines of work:

- The analysis of the fixes introduced in VeraCrypt after the results of the Open Crypto Audit Project's audit of TrueCrypt 7.1a have been published.
- The assessment of VeraCrypt's features that were not present in TrueCrypt.

The new features of VeraCrypt include:

- Support for non western cryptographic algorithms,
- Compatibility with UEFI for system encryption,
- A better protection of the volume header keys.

VeraCrypt is a hard to maintain project. Deep knowledge of several operating systems, of the Windows kernel, of the system boot chain and good concepts in cryptography are required. The improvements made by IDRIX demonstrate the possession of these skills.

2.1 Fixes

- All the vulnerabilities that have been taken into account have been correctly fixed (except a minor missing fix for one of them). In particular, the problem leading to a privilege escalation discovered by James Forshaw in the TrueCrypt driver just after the OCAP audit has been solved.
- Vulnerabilities which require substantial modifications of the code or the architecture of the project have not been fixed. These include:
  - `TC_IOCTL_OPEN_TEST` multiple issues (need to change the application behavior),
  - `EncryptDataUnits()` lacks error handling (need to design a new logic to retrieve errors),
  - AES implementation susceptible to cache-timing attacks (need to fully rewrite the AES implementations).

- Vulnerabilities leading to incompatibilities with TrueCrypt, as the ones related to cryptographic mechanisms, have not been fixed. Most notable are:
  - Keyfile mixing is not cryptographically sound,
  - Unauthenticated ciphertext in volume headers.

2.2 New Problems

Among the problems found during the audit, some must be corrected quickly:

- The availability of GOST 28147-89, a symmetric block cipher with a 64-bit block size, is an issue. This algorithm must not be used in this context.
- Compression libraries are outdated or poorly written. They must be updated or replaced.
- If the system is encrypted, the boot password (in UEFI mode) or its length (in legacy mode) could be retrieved by an attacker.

Finally, the UEFI loader is not mature yet. However, its use has not been found to cause security problems from a cryptographic point of view.

3. Context and Scope

This report describes the security assessment made by Quarkslab on VeraCrypt. VeraCrypt is a disk encryption software developed by IDRIX. It is derived from the now defunct TrueCrypt project. This audit has been carried out at the request of the Open Source Technology Improvement Fund. Its goal was to evaluate the security of the features brought by VeraCrypt since the publication of the audits on TrueCrypt 7.1a conducted by the Open Crypto Audit Project.

Two people from Quarkslab worked on this audit, for a total of 32 man-days of study:

- Jean-Baptiste Bédrune, Senior Security Researcher,
- Marion Videau, Senior Cryptographer and Chief Scientific Officer.

A first step consisted in verifying that the problems and vulnerabilities identified by *[OCAP1]* and *[OCAP2]* in TrueCrypt 7.1a had been taken into account and fixed.

Then, the remaining study was to identify potential security problems in the code specific to VeraCrypt. Contrary to other TrueCrypt forks, the goal of VeraCrypt is not only to fix the public vulnerabilities of TrueCrypt, but also to bring new features to the software. The innovations introduced by VeraCrypt include:

- The support of UEFI,
- The addition of non-occidental cryptographic algorithms (Camellia, Kuznyechik, GOST 28147-89, Streebog),
- A volume expander,
- A "Personal Iterations Multiplier" impacting the security of the derivation of the volume header encryption keys,
- The support of UNICODE on Windows, and the use of `StrSafe` functions instead of `string.h`,
- The gathering of entropy on mouse movements at each random number generation to improve randomness and a better estimate of the randomness.

Some components of VeraCrypt have not been investigated, as they were already out of the scope of the Open Crypto Audit Project. That includes the OS X and Linux versions of VeraCrypt. Finally, it has been decided conjointly with VeraCrypt's main developer that the features available in the diagnostic tool of the UEFI loader were also out of the scope of this audit.

This study focuses on the source code of VeraCrypt 1.18 and the source code of the VeraCrypt DCS EFI Bootloader 1.18.

The SHA-256 fingerprints of these archives are:

- VeraCrypt_1.18_Source.zip :

  `12c1438a9d2467dcfa9fa1440c3e4f9bd5e886a038231d7931aa2117fef3a5c5`

- VeraCrypt-DCS-EFI-LGPL_1.18_Source.zip :

  `2e8655b3b14ee427320891c08cc7f52239378ee650eb28bad9531371e7c64ae3`

4. Known Vulnerabilities in TrueCrypt 7.1a

This part inventories the vulnerabilities identified in TrueCrypt 7.1a, which is the code base of VeraCrypt. Then, it details if they have been fixed or not in VeraCrypt.

The set of vulnerabilities includes the ones detailed in these three sources:

- The reports of the Open Crypto Audit Project *[OCAP1]*, *[OCAP2]*.
- The audit report of the Fraunhofer Institute for Secure Information Technology for the BSI *[FSIT]*, which is greatly based on the OCAP reports.
- Two problems in the TrueCrypt driver identified by James Forshaw *[P0-537]*, *[P0-538]*.

VeraCrypt's code has been analyzed to check if the vulnerabilities reported in all these sources have been correctly understood and fixed by VeraCrypt's developers.

Note: When pieces of source code are cited, the layout might be changed from the original source code for readability purposes, to make it fit in the page width.

4.1 Vulnerabilities Detailed in OCAP Phase 1

This part lists and comments the vulnerabilities discovered during the first phase of the audit ordered by OCAP. The audit has been performed by iSec Partners *[OCAP1]*.

4.1.1 Weak Volume Header Key Derivation Algorithm

TrueCrypt's volume header keys are derived from the user supplied password with PBKDF2. In its report, iSec advocated to greatly increase the number of iterations of the hash function and, eventually, to migrate this derivation function towards a newer algorithm such as scrypt.

The number of iterations has been increased in VeraCrypt: it was comprised between 1000 and 2000 in TrueCrypt, depending on the hash algorithm and its use case. It is now comprised between 200,000 and 655,331. Furthermore, it can be manually specified.

NIST recommends using a much higher number of iterations for critical keys *[SP800-132]*:

> *For especially critical keys, or for very powerful systems or systems where user-perceived performance is not critical, an iteration count of 10,000,000 may be appropriate.*

This level of security can be reached since the introduction of a "Personal Iterations Multiplier" in VeraCrypt 1.12. The number of iterations when a PIM value is specified is:

- For system drive encryption: Iterations = PIM * 2048.
- For non-system drive and containers encryption: 15000 + (PIM * 1000).

To comply with NIST's recommendations, a PIM value of 4883 for system encryption and of 9985 for containers and non-system partitions can be used.

The default number of iterations in VeraCrypt is a trade-off between security and boot or mount time. Each user can then influence these parameters using the PIM parameter.

This vulnerability is considered fixed. However, a safer derivation algorithm like scrypt (or Argon2) would be a plus.

4.1.2 Sensitive Information Might Be Paged Out from Kernel Stacks

In a situation where the amount of available memory becomes very low, kernel stack pages can be paged out under certain conditions.

The vulnerability and its remediation are correctly documented in iSec's report. This situation has consequences only if the system partition is not encrypted or if the pagination files do not reside on this partition.

VeraCrypt's documentation correctly explains the problem [1]:

> *To prevent the issues described above, encrypt the system partition/drive (for information on how to do so, see the chapter System Encryption) and make sure that all paging files are located on one or more of the partitions within the key scope of system encryption (for example, on the partition where Windows is installed).*

Except for the explanation, nothing else was intended by VeraCrypt to fix the vulnerability.

iSec recommends gathering all sensitive pieces of information at the same place and to lock the corresponding memory area. However, it is very difficult to definitely exclude the possibility of a sensitive piece of information being paged out this way. We rather advise following VeraCrypt's documentation principles which definitely solve the problem.

4.1.3 Multiple Issues in the Bootloader Decompressor

If the system partition is encrypted, at boot time, TrueCrypt's code in the boot sector loads in memory a decompression routine and verifies its checksum. If it succeeds, the decompression routine is called to decompress a bootloader.

The decompression routine suffers from several bugs. Its code is taken from **`puff`**, an implementation of **`inflate`** which was optimized for memory constrained applications and can be found in the **`contrib`** directory of **`zlib`**. The code is a fork and the bug fixes have not been taken into account in TrueCrypt, notably an out of bounds write.

All the problems mentioned in iSec's report have been corrected in VeraCrypt.

It should be noticed that in order to trigger a vulnerability in the decompression routine of the bootloader, an attacker has to first modify the compressed piece of code, which requires an administrator or a physical access to the system. These two attack settings are explicitly excluded from the protection range of TrueCrypt.

Modifying compressed data requires as many rights as modifying the decompression routine. Fixing those bugs make the code more robust but in our opinion, the bugs were not really threatening the application security.

[1] VeraCrypt Documentation - Paging File. https://veracrypt.codeplex.com/wikipage?title=Paging%20File

4.1.4 Windows Kernel Driver Uses `memset()` to Clear Sensitive Data

Some sensitive pieces of information are deleted by calling **memset** in TrueCrypt's driver. Unfortunately, one of the compiler's optimizations consists in removing **memset** calls that it considers useless. Therefore, the function **RtlSecureZeroMemory** needs to be called to securely erase memory, which is what the function **burn** does as a wrapper of **RtlSecureZeroMemory** in the kernel.

Two examples of sensitive data deletion with **memset** are presented in iSec's report. The first one is fixed in VeraCrypt:

Listing 4.1: **src/Driver/DriveFilter.c:113**

```
BootArgs = *bootArguments;
BootArgsValid = TRUE;
burn (bootArguments, sizeof (*bootArguments));
```

As is the case with the second one:

Listing 4.2: **src/Driver/DriveFilter.c:453**

```
// Erase boot loader scheduled keys
if (mappedCryptoInfo)
{
   burn (mappedCryptoInfo, BootArgs.CryptoInfoLength);
   MmUnmapIoSpace (mappedCryptoInfo, BootArgs.CryptoInfoLength);
   BootArgs.CryptoInfoLength = 0;
}
```

However, code has been added in this function by VeraCrypt. There is now an execution path where a check on a hidden volume can call **TC_THROW_FATAL_EXCEPTION**, a wrapper of **KeBugCheckEx** :

Listing 4.3: **src/Driver/DriveFilter.c:391**

```
   mappedCryptoInfo = MmMapIoSpace (cryptoInfoAddress, BootArgs.CryptoInfoLength,
                                    MmCached);
   if (mappedCryptoInfo)
   {
      ...
   }
}

pim = (int) (BootArgs.Flags >> 16);

if (ReadVolumeHeader (!hiddenVolume, header, password, pkcs5_prf, pim, FALSE,
                      &Extension->Queue.CryptoInfo, Extension->HeaderCryptoInfo) == 0)
{
   // Header decrypted
   status = STATUS_SUCCESS;
   Dump ("Header decrypted\n");

   // calculate Fingerprint
   ComputeBootLoaderFingerprint (Extension->LowerDeviceObject, header);

   if (Extension->Queue.CryptoInfo->hiddenVolume)
```

```
   {
      Dump ("Hidden volume start offset = %I64d\n",
            Extension->Queue.CryptoInfo->EncryptedAreaStart.Value
               + hiddenPartitionOffset);
      ...

      if (Extension->Queue.CryptoInfo->VolumeSize.Value >
          hiddenPartitionOffset - BootArgs.DecoySystemPartitionStart)
         TC_THROW_FATAL_EXCEPTION;
      ...
      // Erase boot loader scheduled keys
   if (mappedCryptoInfo)
   {
      burn (mappedCryptoInfo, BootArgs.CryptoInfoLength);
```

If an exception is raised, the call to **burn** will never be reached and **mappedCryptoInfo** will not be deleted. The corresponding data could possibly be written in a crash dump. Therefore, **mappedCryptoInfo** must be erased before raising the exception.

However, it must be mentioned that TrueCrypt's documentation prescribes the deactivation of crash dumps creation if the system partition is not encrypted. Therefore, if the system is correctly configured, the unreachable **burn** call will not cause a security problem.

4.1.5 TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG Kernel Pointer Disclosure

The **TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG** ioctl returns the address of **BootDriveFilterExtension**, which is a pointer to the boot drive's extension object. The function does not check if the call comes from the userspace. An attacker can recover the pointer's address from the userspace.

VeraCrypt has fixed this problem:

Listing 4.4: **src/Driver/Ntdriver.c:1690**

```
case TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG:
   if (    (ValidateIOBufferSize (Irp, sizeof(GetSystemDriveDumpConfigRequest),
                                  ValidateOutput))
        && (Irp->RequestorMode == KernelMode)
        )
   {
```

If the call does not come from the kernel space, the ioctl returns **STATUS_INVALID_PARAMETER** and does not satisfy the request.

4.1.6 IOCTL_DISK_VERIFY Integer Overflow

An addition of parameters controlled by the user and processed by the **IOCTL_DISK_VERIFY** ioctl leads to an integer overflow. Verification can be avoided thanks to this vulnerability and large amounts of memory can be allocated in the non-paged pool.

By calling **IOCTL_DISK_VERIFY** several times, an attacker asks for large amounts of memory in order to fill the kernel memory space. The consequence is a denial of service and, most probably, several malfunctions requiring a system reboot.

To detect the overflow, VeraCrypt has replaced the addition with a function from **IntSafe**, **ULongLongAdd**:

Listing 4.5: **src/Driver/Ntdriver.c:809**

```
ullStartingOffset = (ULONGLONG) pVerifyInformation->StartingOffset.QuadPart;
hResult = ULongLongAdd(ullStartingOffset,
          (ULONGLONG) Extension->cryptoInfo->hiddenVolume ?
          Extension->cryptoInfo->hiddenVolumeOffset :
          Extension->cryptoInfo->volDataAreaOffset,
          &ullNewOffset);
if (hResult != S_OK)
   Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
else if (S_OK != ULongLongAdd(ullStartingOffset,
                              (ULONGLONG) pVerifyInformation->Length,
                              &ullEndOffset))
   Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
```

Note: Fraunhofer's report analysis of the bug's origin is not accurate:

The content of the variable comes from the method ''ExInterlockedRemoveHeadList'', which is part of Microsoft ''Ntoskrnl.lib'' library. Thus, this vulnerability dependents on whether the method ''ExInterlockedRemoveHeadList()'' intercepts an integer overflow.

The claim feels like a complete nonsense.

4.1.7 `TC_IOCTL_OPEN_TEST` Multiple Issues

The **TC_IOCTL_OPEN_TEST** ioctl opens a user-specified file with the function **ZwCreateFile** without checking if the user has the correct access rights and then reads its content. TrueCrypt made this choice in order to detect whether its bootloader is present on the disk without the need for administrator's privileges.

This behavior allows several kinds of information leakage, such as the possibility to check for the presence of a file normally not accessible for the user.

VeraCrypt does not fix this issue. It even adds a new information leakage source: it is now possible to check the SHA-256 hash of the first 512 bytes of a file.

A similar issue can be found in the **TC_IOCTL_GET_SYSTEM_DRIVE_CONFIG** ioctl. It has not been fixed either.

VeraCrypt should consider these issues and fix them.

4.1.8 `MainThreadProc()` Integer Overflow

The **MainThreadProc** function handles userspace-controlled data. Its code contains an integer overflow vulnerability which can be triggered when handling requests from **IRP_MJ_READ** and **IRP_MJ_WRITE**.

By choosing carefully crafted values, data whose sizes are controlled by the user are copied in a buffer sent back to the user. It is a typical information leakage.

VeraCrypt has fixed this vulnerability by means of an overflow detection function from **IntSafe**.

Listing 4.6: **src/Driver/EncryptedIoQueue.c:571**

```
ULONG alignedLength;
LARGE_INTEGER alignedOffset;
hResult = ULongAdd(item->OriginalLength, ENCRYPTION_DATA_UNIT_SIZE, &alignedLength);
if (hResult != S_OK)
{
   CompleteOriginalIrp (item, STATUS_INVALID_PARAMETER, 0);
   continue;
}
```

4.1.9 `MountVolume()` Device Check Bypass

The **VolumeThreadProc** function in **Ntdriver.c** checks the validity of a volume's name, which is a userspace-controlled value, by comparing it to the string **\Device** without prior case checks.

Thus, according to the iSec report, a filename beginning with **\device** is not considered as a device. This will result in an unexpected code path being followed in the function **TCOpenVolume()**.

VeraCrypt has fixed the issue by adding a function **IsDeviceName** which provides a case-insensitive comparison to the string **\device**:

Listing 4.7: **src/Driver/Ntdriver.c:1921**

```
BOOL IsDeviceName(wchar_t wszVolume[TC_MAX_PATH])
{
   if (   (wszVolume[0] == '\\')
      &&  (wszVolume[1] == 'D' || wszVolume[1] == 'd')
      &&  (wszVolume[2] == 'E' || wszVolume[2] == 'e')
      &&  (wszVolume[3] == 'V' || wszVolume[3] == 'v')
      &&  (wszVolume[4] == 'I' || wszVolume[4] == 'i')
      &&  (wszVolume[5] == 'C' || wszVolume[5] == 'c')
      &&  (wszVolume[6] == 'E' || wszVolume[6] == 'e')
      )
   {
      return TRUE;
   }
   else
      return FALSE;
}
```

4.1.10 `GetWipePassCount()` / `WipeBuffer()` Can Cause BSOD

The **GetWipePassCount** function returns the number of wipe passes to execute on a given volume from a wipe algorithm identifier. If the identifier is unknown, **TC_THROW_FATAL_EXCEPTION** is called. In the driver's code, this function is a wrapper of **KeBugCheckEx**.

The function can be reached from two ioctls using a controlled identifier value. If the identifier value is not in the expected case list, a BSOD is triggered. Handling the case differently, for example by issuing an error message, seems more reasonable.

VeraCrypt has fixed this vulnerability by returning an invalid value (-1). A calling function checks against this value instead of causing a BSOD.

4.1.11 `EncryptDataUnits()` Lacks Error Handling

The **`EncryptDataUnits`** is a key function of the TrueCrypt security. Indeed, it orchestrates all the encryption and decryption operations of the volumes in the BIOS and the driver.

Here is its prototype:

```
void EncryptDataUnits (unsigned __int8 *buf, const UINT64_STRUCT *structUnitNo,
                       uint32 nbrUnits, PCRYPTO_INFO ci);
```

The function encrypts data in-place and does not return any value which would allow to detect an error occurring during encryption.

If the encryption operation fails, plaintext data are written to disk. The same situation occurs for the **`DecryptDataUnits`** function: if it fails, corrupted data are read.

iSec recommends redesigning this functionality to make it more robust. It is a very relevant recommendation which would imply heavy modifications in the current source code.

VeraCrypt has not taken this recommendation into account.

4.1.12 Conclusion on Vulnerabilities Detailed in OCAP Phase 1

Most vulnerabilities presented in iSec's report have been fixed in VeraCrypt. The first medium severity problem that has not been fixed is related to the kernel stack page mechanism; it was already a known problem and it was documented by TrueCrypt to allow users to get a secure configuration. The problem related to the erasure of sensitive data should be fixed, even if it does not lower the security of the product when used according to the documentation's recommendations.

Vulnerability	Class	Severity	Status
Weak Volume Header key derivation algorithm	Cryptography	Medium	Fixed
Sensitive information might be paged out from kernel stacks	Data Exposure	Medium	Not fixed
Multiple issues in the bootloader decompressor	Data Validation	Medium	Fixed
Windows kernel driver uses **`memset()`** to clear sensitive data	Data Exposure	Medium	Partially fixed
`TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG` kernel pointer disclosure	Data Exposure	Low	Fixed
`IOCTL_DISK_VERIFY` integer overflow	Data Validation	Low	Fixed
`TC_IOCTL_OPEN_TEST` multiple issues	Data Exposure	Low	Not fixed
`MainThreadProc()` integer overflow	Denial of Service	Informational	Fixed
`MountVolume()` device check bypass	Data Validation	Informational	Fixed
`GetWipePassCount()` / **`WipeBuffer()`** can cause BSOD	Denial of Service	Informational	Fixed
`EncryptDataUnits()` lacks error handling	Error Reporting	Informational	Not fixed

4.2 Vulnerabilities Detailed in OCAP Phase 2

This section lists and comments on the vulnerabilities discovered during the second phase of the audit ordered by OCAP. The audit has been performed by Cryptography Services of NCC Group *[OCAP2]*.

4.2.1 `CryptAcquireContext` May Silently Fail in Unusual Scenarios

The **`CryptAcquireContext`** function belongs to Windows' CryptoAPI. It is used in conjunction with **`CryptoGenRandom`** to generate random numbers.

The **`CryptAcquireContext`** function is used to get a context to a user's key container. The function is called with incorrect parameters. In some situations, initializing a key container can fail and the function call will fail. TrueCrypt does not use a key container and uses the **`CryptAcquireContext`** function exclusively to get a handle to a Cryptographic Service Provider to generate random numbers.

NCC Group's report gives a set of correct call parameters. They have been taken into account by VeraCrypt. The vulnerability has been fixed for all sensitive operations. There remains a single case where the code is unchanged: the random generation of colors in the donation page, near the end of the installation program, on Windows. We consider the vulnerability fixed.

4.2.2 AES Implementation Susceptible to Cache-Timing Attacks

NCC Group's report claims that some implementations of AES located in the files **`AesSmall.c`**, **`AesSmall_x86.asm`**, **`Aes_x86.asm`** and **`Aes_x64.asm`** are susceptible to cache-timing attacks.

VeraCrypt did not implement any of the proposed countermeasures.

The implementations located in **`AesSmall.c`** and **`AesSmall_x86.asm`** are only used during boot time, as detailed in Fraunhofer's report. The only attack scenario where it would be possible to take advantage of a cache-timing is the case of physical machine hosting two virtual machines, one with a system entirely encrypted and the other controlled by an attacker.

If the AES-NI instruction set is available, an AES-NI implementation will be used. This implementation is not susceptible to cache-timing attacks. Otherwise, the functions from **`Aes_x86.asm`** and **`Aes_x64.asm`** are used. We consider that both these implementations would need being fixed first.

NCC Group's report only focuses on AES. We did not check if other implementations are susceptible to such attacks.

The severity of this vulnerability was judged "High" in NCC Group's report. We would like to stress the fact that VeraCrypt's security model [2] makes it clear that :

> *VeraCrypt does not: Secure any data on a computer if the computer contains any malware (e.g. a virus, Trojan horse, spyware) or any other piece of software (including VeraCrypt or an operating system component) that has been altered, created, or can be controlled, by an attacker.*

The documentation does not specify whether the term "computer" is limited to a physical machine or if it can be a virtual machine sharing a physical machine with other VM.

[2] VeraCrypt Documentation - Security Model. https://veracrypt.codeplex.com/wikipage?title=Security%20Model

4.2.3 Keyfile Mixing Is Not Cryptographically Sound

Keyfiles can be used along the user passphrase to generate the key used to mount a volume. The key derivation algorithm used to process the keyfiles and generate the volume key is not cryptographically sound.

As a preliminary remark, notice that for each keyfile, only the first megabyte is used.

Keyfile data is used to fill a 64-byte circular buffer named the keyfile pool. When starting the key derivation process, a cursor is set at the beginning of the keyfile pool and will be moved after each update.

The derivation process from the keyfiles relies on a CRC-32 function. For each keyfile, each byte is read and submitted to the CRC-32 function to update its current value. The 4-byte value is extracted and each byte is added to the corresponding byte located at the cursor's position in the keyfile pool. The cursor position is updated accordingly.

Once this process is finished, the resulting 512 bits are XORed with the passphrase padded with zeroes on 512 bits.

CRC-32 is not a cryptographic hash function. Using it in this key derivation mechanism creates undesirable properties:

- From a set of valid keyfiles, it is possible to create another distinct one (*i.e.* allowing to mount a volume),
- It is possible to create keyfiles that do not modify the keyfile pool,
- From a set of keyfiles, it is possible to create a new keyfile which removes the security brought by the former set of keyfiles (*i.e.* which zeroes the keyfile pool),
- From a known passphrase, it is possible to create a keyfile which removes the security brought by this passphrase.

This problem has been previously reported to TrueCrypt's authors by Sogeti *[SOGETI]* in 2008 and by the Ubuntu Privacy Remix Team *[UPR]* in 2011. The *[UPR]* and the *[FSIT]* reports describe interesting attack scenarios. TrueCrypt's developers denied the problem since the attacks require the prior knowledge of a secret or the prior manipulation of a machine. Here is an excerpt of their answer to the Ubuntu Privacy Remix Team :

> *It is a basic security requirement that cryptographic keys (whether passwords, keyfiles, or master keys) must be secret and unknown to attackers. Your attack violates this requirement and is therefore invalid/bogus.*

NCC Group's report recommends the use of HMAC with a cryptographic hash function. We concur. This recommendation should really be implemented. For the moment the problem has not been corrected, probably because it would break backward compatibility.

4.2.4 Unauthenticated Ciphertext in Volume Headers

To provide integrity for a volume header in plaintext, two CRC-32 on decrypted header data and a 4-byte ASCII string ("TRUE" in TrueCrypt and "VERA" in VeraCrypt) are used. As mentioned earlier, CRC-32 is not a cryptographic integrity mechanism but rather an error detection mechanism, which is meant to prevent accidents but not attacks. The same goes for the comparison of the 4-byte decrypted ASCII string to a fixed string. This problem was already mentioned by Sogeti in 2008 *[SOGETI]*.

NCC Group's report mentions that an existential forgery is possible with approximately 2^{32} queries. For the sake of clarity we provide an idea of such an attack.

An existential forgery does not imply that the header produced and accepted by the integrity verification will have all fields coherent, simply that a header will be declared legitimate while produced by the attacker. A VeraCrypt header contains 2 CRCs and a few fields that are subjected to strict checks. Most of them are located in the header's first encrypted block. The first CRC, located in the first encrypted block, is dependent on the encryption keys and if they are left untouched, this CRC is also untouched. The fourth encrypted block contains the `SectorSize` field which is subjected to a check. If we leave this block untouched, the field will be valid. Then there remains 10 encrypted blocks that can be manipulated as the XTS mode leaves all the blocks independent from each other. As the goal for the attacker is to get a 32-bit value CRC right from manipulating ten 128-bit blocks, it can succeed after 2^{32} queries.

NCC Group's report recommends replacing these mechanisms with a Message Authentication Code (MAC). The user passphrase could be used to derive a MAC key in addition to the encryption keys. The MAC of the header would be checked before mounting the volume. Implementing such a mechanism should be done in VeraCrypt. Mechanisms using CRC-32 are still present in TrueCrypt after many years of warnings for obvious compatibility reasons. They should nonetheless be replaced by real up-to-date authentication mechanisms.

A major difficulty to achieve this improvement is due to the lack of space after the header to store a MAC value in the case of system encryption with MBR. A possible idea would be to use the 960 zero bits located in the header to store such a MAC but whether to use a MAC-then-encrypt or an encrypt-then-MAC construction taking into account all the constraints of the project must be studied in detail.

4.2.5 Conclusion on Vulnerabilities Detailed in OCAP Phase 2

A single vulnerability from the list reported by OCAP Phase 2 analysis has been corrected. The fix was the simplest one to implement.

Re-designing and re-writing AES functions immune against cache-timing attacks would require a more significant effort. Using VeraCrypt on a machine with a CPU providing AES-NI instructions is an available workaround.

Note: We did not check the implementations of other cryptographic algorithms in VeraCrypt against cache-timing attacks.

The last two vulnerabilities have been known for a long time. Fixing them would break backward compatibility with existing volumes but would bring a better security to the product. We recommend to fix them.

Vulnerability	Class	Severity	Status
`CryptAcquireContext` may silently fail in unusual scenarios	Cryptography	High	Fixed
AES implementation susceptible to cache-timing attacks	Cryptography	High	Not fixed
Keyfile mixing is not cryptographically sound	Cryptography	Low	Not fixed
Unauthenticated ciphertext in volume headers	Cryptography	Undetermined	Not fixed

4.3 Vulnerabilities Reported by James Forshaw

Two vulnerabilities have been discovered by James Forshaw of Google Project Zero. Both are located in TrueCrypt's driver. The impact of the first one (CVE-2015-7359) is rather anecdotal. The second one (CVE-2015-7358) leads to an interesting privilege escalation.

4.3.1 Incorrect Impersonation Token Handling EoP (CVE-2015-7359)

The vulnerability and its impact are documented in detail in Project Zero's issue tracker *[P0-537]*. It is located in the user token verification routine in TrueCrypt's driver.

In two different locations, TrueCrypt's driver acquires the security context of the current user with a call to **`SeCaptureSubjectContext`** and extracts the active token with a call to **`SeQuerySubjectContextToken`**. The driver does not apply any impersonation level verification, which causes the problem.

By using the impersonation level **`SecurityIdentification`**, a user can impersonate another one. This should only be possible from the level **`SecurityImpersonation`**.

The problem is present in two functions of the driver: **`IsVolumeAccessibleByCurrentUser`** and **`MountDevice`**. In **`IsVolumeAccessibleByCurrentUser`**, the bug allows an attacker to unmount another user's volumes and to get information on the mounted volumes. In **`MountDevice`**, the bug does not lead to any attack.

As stated by James Forshaw, the problem is anecdotal compared to the problems already coming from letting volumes mounted on a shared machine. It has been fixed by VeraCrypt, following Forshaw's recommendations: the impersonation level is verified.

An example of the fix is shown below. The other one is identical.

Listing 4.8: **`src/Driver/Ntdriver.c:2756`**

```
SeCaptureSubjectContext (&subContext);
SeLockSubjectContext(&subContext);
if (subContext.ClientToken && subContext.ImpersonationLevel
    >= SecurityImpersonation)
   accessToken = subContext.ClientToken;
else
   accessToken = subContext.PrimaryToken;
```

4.3.2 Drive Letter Symbolic Link Creation EoP (CVE-2015-7358)

The second vulnerability reported by James Forshaw allows a user to get system privileges from an application running with user privileges or within a low-integrity sandbox *[P0-538]*.

The bug origin is very simple. On the contrary, its consequences in terms of security and its exploitation are not. James Forshaw wrote a detailed blog article on how to exploit this vulnerability and how **`DosDevices`** are managed since NT 3.1 *[P0-BLOG]*.

The vulnerability is located in the **`IsDriveLetterAvailable`** function of the **`Ntdriver.c`** file.

Listing 4.9: **Driver/Ntdriver.c:2881 in TrueCrypt7.1a**

```
BOOL IsDriveLetterAvailable (int nDosDriveNo)
{
   OBJECT_ATTRIBUTES objectAttributes;
   UNICODE_STRING objectName;
   WCHAR link[128];
   HANDLE handle;

   TCGetDosNameFromNumber (link, nDosDriveNo);
   RtlInitUnicodeString (&objectName, link);
   InitializeObjectAttributes (&objectAttributes, &objectName,
                               OBJ_KERNEL_HANDLE | OBJ_CASE_INSENSITIVE,
                               NULL, NULL);

   if (NT_SUCCESS (ZwOpenSymbolicLinkObject (&handle, GENERIC_READ, &objectAttributes)))
   {
      ZwClose (handle);
      return FALSE;
   }

   return TRUE;
}
```

The variable **nDosDriveNo** is an integer ranging from 0 to 25 and represents a drive letter from *A* to *Z*. The function **TCGetDosNameFromNumber** builds a path to the symbolic link **\DosDevices\X:**, where **X** stands for the drive letter associated to **nDosDriveNo**.

If **ZwOpenSymbolicLinkObject** fails for any reasons, then the drive letter passed as a parameter is considered available, even if it already exists.

VeraCrypt has fixed this issue, following James Forshaw's recommendations: the function returns **TRUE** only if the object **\DosDevices\X:** does not exist.

Moreover, in order to avoid any further problems with **\DosDevices**, the Global MS-DOS device names are used.

```
-#define DOS_MOUNT_PREFIX DRIVER_STR("\\DosDevices\\")
+#define DOS_MOUNT_PREFIX DRIVER_STR("\\GLOBAL??\\")
  // Explicitely use Global MS-DOS device names to avoid security issues
```

This modification has triggered side effects with Windows's mount manager. Thus new modifications have been implemented. In our opinion, they do not bring vulnerabilities. We consider this problem fixed.

4.3.3 Conclusion on Vulnerabilities Reported by James Forshaw

Both vulnerabilities are corrected in VeraCrypt.

Vulnerability	Severity	Status
Incorrect Impersonation Token Handling EoP	Low	Fixed
Drive Letter Symbolic Link Creation EoP	High	Fixed

5. VeraCrypt's Modifications Assessment

5.1 The Length of the Password Can Be Computed When Encryption Is Activated

Class	Severity	Difficulty
Data Exposure	Low	Medium

VeraCrypt can encrypt the hard drive partition where the Operating System is installed. We deal here with the start up from the BIOS only, not the UEFI.

The original bootloader is replaced with a specific one asking for the password of the partition to start on. It then decrypts the system partition and the usual boot goes on.

Keystrokes are saved in a 32-byte circular buffer in the BIOS Data Area, located at address **0040:001Eh**. Each keystroke being stored on 2 bytes, the first one being the ASCII code and the second one the BIOS scan code, 16 inputs can be saved.

Once the system is started, if this buffer has not been cleared, it is possible to retrieve the user password. VeraCrypt prevents this potential leak by zeroing the buffer with the **ClearBiosKeystrokeBuffer** function.

Listing 5.1: **src/Boot/Windows/BootConsoleIo.cpp:291**

```
void ClearBiosKeystrokeBuffer ()
{
   __asm
   {
      push es
      xor ax, ax
      mov es, ax
      mov di, 0x41e
      mov cx, 32
      cld
      rep stosb
      pop es
   }
}
```

However, 2 pointers related to the keystroke buffer are located just before it and are not erased. The first one points to the last character of the buffer, the other one to the address where the next character is going to be written. Using the value of these 2 pointers, one can gain information on the length of the password.

If the user properly entered his password with no mistake at boot time, followed by "Enter", the first pointer value is then **001Eh** + (2 * (len(password) + 1) mod 32). Since each character is stored on 2 bytes, one can compute the length of the password modulo 16.

This information leak might not look critical as the system needs to be booted and a privileged access is required to read BIOS memory. Nonetheless, this should be fixed for 2 reasons:

- The risk has been considered since the password is zeroed.
- If the running system is compromised, recovering the keys encrypting the system is a known damage, but it should not leak information about the user password since it can

be used to quicken the password bruteforce. And the same password could be used on other systems too.

5.2 Data Compression: Too Many Different Critical Issues

Some compression functions are used at several places in the project's source code:

- To decompress the bootloader when the hard drive is encrypted.
- To create and check the recovery disks if the system is encrypted and uses UEFI.
- During the installation to extract programs.

It appears that all compression functions have issues.

5.2.1 Out-of-Date `inflate` and `deflate`

Class	Severity	Difficulty
Patching	High	High

TrueCrypt forked in 2007 a version of the **inflate** library to decompress data with the format specified in RFC 1951 in order to create self-extracting installation packages. Compression being made with **gzip**, **inflate** only is required for decompression. The installation package uses **inflate** to extract files at install time.

The version of **inflate** used by TrueCrypt was already obsolete and vulnerable but that was not a security issue as the user had to execute the installer to be compromised.

VeraCrypt next added another copy of **inflate** coming from **XUnzip**. **XZip** and **XUnzip** are 2 modules allowing to create and extract Zip archives. They embed **inflate** and **deflate**, in more than obsolete versions as we can read from **XUnzip** copyright:

```
extern const char inflate_copyright[] =
   " ";//inflate 1.1.3 Copyright 1995-1998 Mark Adler ";
// If you use the zlib library in a product, an acknowledgment is welcome
// in the documentation of your product. If for some reason you cannot
// include such an acknowledgment, I would appreciate that you keep this
// copyright string in the executable of your product.
```

The used version is for instance vulnerable to CVE-2002-0059.

The functions **inflate** and **deflate** should be merged and replaced by up-to-date versions, like the ones included in the up-to-date **zlib** library. TrueCrypt did fork the code, but chose not to fix security issues affecting it. VeraCrypt includes a **zlib** version already vulnerable.

It seems better to add a dependency to **zlib**, which would at least ensure a proper up-to-date code base.

The same type of issue affects the decompressor for the bootloader. Its code comes from **puff**, an optimized implementation of **inflate** for applications with little memory capacity. It is available in the **contrib** directory of **zlib**. A minor bug has been fixed in the latest version of **puff**, distributed with zlib 1.2.5.1, but not in VeraCrypt's. Also, calls to **longjmp** and **setjmp** have been removed from the original code, leading to an out-of-bounds read during bootloader decompression. This is because these functions are not supported by Visual C++ 1.52, and

no workaround has been implemented by IDRIX. The size constraint on the bootloader, which forces the decompressor to fit on 4 disk sectors of 512 bytes each, makes the problem difficult to fix. Note that it does not lead to a vulnerability.

5.2.2 `XZip` and `XUnzip` Need to Be Completely Re-Written

Class	Severity	Difficulty
Data Validation	High	High

Since version 1.18 and UEFI support, VeraCrypt can create rescue disks. They are different from the ones supported by BIOS. They contain UEFI loaders allowing to reinstall the bootloader for instance.

The format for the image is Zip. The library used to create them is **`XZip`**, as mentioned earlier. It seems to come from a 2007 article published on CodeProject [1].

Obvious bugs are present in the code, as demonstrated in the following example:

Listing 5.2: **`src/Common/XZip.cpp:3130`**

```
BOOL AddFolderContent(HZIP hZip, TCHAR* AbsolutePath, TCHAR* DirToAdd)
{
   HANDLE hFind; // file handle
   WIN32_FIND_DATA FindFileData;
   TCHAR PathToSearchInto [MAX_PATH] = {0};

   if (NULL != DirToAdd)
   {
      ZipAdd(hZip, DirToAdd, 0, 0, ZIP_FOLDER);
   }

   // Construct the path to search into "C:\\Windows\\System32\\*"
   _tcscpy(PathToSearchInto, AbsolutePath);
   _tcscat(PathToSearchInto, _T("\\"));
   _tcscat(PathToSearchInto, DirToAdd);
   _tcscat(PathToSearchInto, _T("\\*"));
```

- AddFolderContent is exported by the library. The length of **`AbsolutePath`** is not checked before the call to **`_tcscpy`**.
- It checks that **`DirToAdd`** is not null but will call **`_tcscat`** with **`DirToAdd`** as argument all the time.

As explained earlier, known vulnerabilities are present in the copied **`inflate`** and **`deflate`** .

We strongly recommend to either rewrite this library and use an up-to-date version of **`zlib`**, or preferably, use another component to handle Zip files.

Security consequences are explained in *Memory Corruption Can Occur When the Recovery Disk Is Read*.

[1] XFile - Extending the Win32 File API for Server Applications. http://www.codeproject.com/Articles/4093/XFile-Extending-the-Win-File-API-for-Server-Appl.

5.3 Integer Overflow When Computing the Number of Iterations for PBKDF2 When PIM Is Used

Class	Severity	Difficulty
Data Validation	High	High

VeraCrypt has added a new security parameter called PIM (Personal Iterations Multiplier) in order to change the number of rounds in PBKDF2 during key derivation used to encrypt the header of a volume.

The function computing the iteration count for PBKDF2 is **get_pkcs5_iteration_count**. In the function code, the case of each supported hash function is treated separately. However, when the PIM is used, the number of rounds for PBKDF2 uses the same formula for all of them, namely: 15000 + PIM * 1000 (computation is different for the system partition).

The computation made in function **get_pkcs5_iteration_count** can overflow for a large number of PIM, which can lead to a wrong feeling about the security strength.

The overflow is quite straightforward:

Listing 5.3: **src/Common/Pkcs5.c:1158**

```
int get_pkcs5_iteration_count (int pkcs5_prf_id, int pim, BOOL truecryptMode,
                               BOOL bBoot)
{
   if ( (pim < 0)
        || (truecryptMode && pim > 0) /* No PIM for TrueCrypt mode */
        )
   {
      return 0;
   }

   switch (pkcs5_prf_id)
   {
      ...
      case SHA512:
         return truecryptMode? 1000 : ((pim == 0)? 500000 : 15000 + pim * 1000);
```

With a PIM value of 8589920, the return value will be 408, which is obviously weaker than the expected value of 8589935000.

On Windows, a check is performed on the GUI when the volume is created: a message box appears if the PIM number is greater than 2147468, the limit value for the overflow.

This is not true for Linux and Mac OS X. One can then specify a PIM in order to trigger the overflow. Fortunately, these versions do not allow to mount these volumes as an error message is displayed when trying. However, they can be mounted on Windows, since there is no verification at mount time in the Windows version.

A specific use could weaken the security of an encrypted container, but is unlikely to occur. A user must create a volume under Linux or OS X, specify a PIM triggering the overflow, and then use this volume under Windows. The resulting number of iterations can then be very small, while the user keeps feeling secure.

We advise unifying the application behavior. Checks on the PIM must not be performed in the code related to the UI, but in the core functions of the program.

5.4 PIN Code on Command Line

Class	Severity	Difficulty
Data Exposure	Informational	Low

A smart card or a security token can be used to unlock and mount a volume. The user has to provide a PIN code. VeraCrypt added a feature which was not available in TrueCrypt: passing the PIN code on the command line, using parameter **`--token-pin`** (or **`/tokenpin`** for Windows).

VeraCrypt already allowed to provide the password on the command line using the **`--password`** parameter. This feature was documented as potentially insecure:

> *Warning: This method of entering a volume password may be insecure, for example, when an unencrypted command prompt history log is being saved to unencrypted disk.*

The current documentation does not mention the risks for argument **`--token-pin`**.

More generally, we believe such parameters should not be available and are a bad practice. If this feature should be kept anyway, the same security warning should be provided.

6. New Cryptographic Mechanisms Assessment

New cryptographic primitives for hashing and encryption have been added in VeraCrypt 1.18. The purpose of these additions is to include non-western algorithms in the project. The newly added algorithms are:

- Camellia, a symmetric block cipher with a block size of 128 bits. Camellia has been developed by two Japanese companies, Mitsubishi Electric and NTT. It is derived from the AES candidate E2. VeraCrypt uses Camellia with 256-bit keys only.
- GOST89 28147-89, also known as Magma. It is a Russian symmetric block cipher algorithm designed in the 70s. It uses a 256-bit key and has a block size of 64 bits. It used to be the Soviet alternative to DES.
- Kuznyechik, a symmetric block cipher algorithm with a block size of 128 bits and a key size of 256 bits. Kuznyechik is a Russian algorithm specified in GOST R 34.12-2015. It is the successor of GOST 28147-89.
- Streebog-512, a hash function defined in GOST R 34.12 2012. It is the Russian alternative to SHA-3.

The implementation of these new algorithms has been analyzed. Camellia, Kuznyechik and Streebog-512 are correctly implemented. Nevertheless, several problems have been identified.

6.1 GOST 28147-89 Must Be Removed from VeraCrypt

Class	Severity	Difficulty
Cryptography	High	High

The XTS mode is specified in *[IEEE07]* [1]. It is only specified for 128-bit blocks (and more specifically intended to be used with AES) and is implemented for 128-bit blocks in VeraCrypt. It is derived from the XEX mode *[Ro04]*.

GOST 28147-89 is a 64-bit block cipher specified in *[GOST89]*. To fit inside the XTS mode, GOST is "expanded" into a 128-bit block cipher by putting it into a CBC mode for two blocks with a null IV (see Fig. 6.1).

Listing 6.1: **VeraCrypt/src/Crypto/GostCipher.c:234**

```
void gost_encrypt(const byte *in, byte *out, gost_kds *ks, int count){
#if defined(_M_AMD64)
  gost_encrypt_128_CBC_asm(in, out, ks, (uint64)count);
#else
  while (count > 0) {
    // encrypt two blocks in CBC mode
    gost_encrypt_block(*((uint64*)in), (uint64*)out, ks);
    *((gst_udword*)(out + 8)) = *((gst_udword*)(in + 8)) ^ *((gst_udword*)(out));
    *((gst_udword*)(out + 12)) = *((gst_udword*)(in + 12)) ^ *((gst_udword*)(out + 4));
    gost_encrypt_block(*((uint64*)(out + 8)), (uint64*)(out + 8), ks);
    count--;
    in += 16;
```

[1] It is also specified in *[SP800-38E]* where an additional compulsory requirement is the limit on the number of blocks in a data unit encrypted under the same key, set to 2^{20}.

```
    out += 16;
  }
#endif
}
```

Listing 6.2: **VeraCrypt/src/Crypto/GostCipher.c:251**

```
void gost_decrypt(const byte *in, byte *out, gost_kds *ks, int count)
[...]
     // decrypt two blocks in CBC mode
```

These functions are used in XTS through the call to **EncipherBlock()** or **DecipherBlock()** respectively.

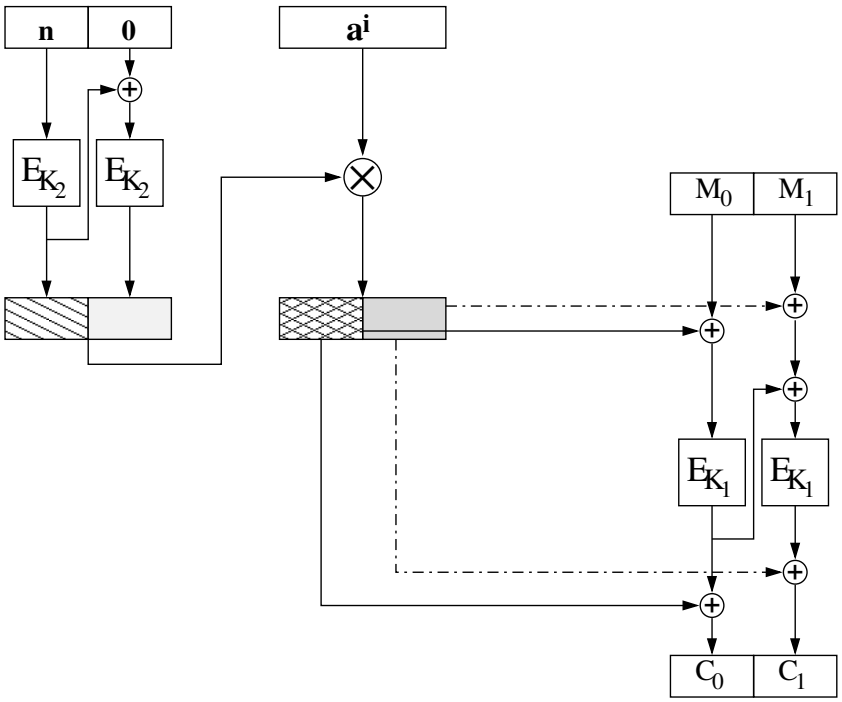

Fig. 6.1: XTS with GOST-CBC-IV-NULL on the ith 128-bit block of a data unit.

There are security proofs which state that the advantage an attacker has in distinguishing an XTS mode instantiated with a block cipher from a perfect tweakable permutation is upper-bounded by a value proportional to $q^2/2^N$, where q is the number of queries an attacker makes and N is the block size, plus the advantage in distinguishing in a CCA (Chosen Ciphertext Attack) setting the block cipher from a random permutation (see for example *[LM08]*).

Unfortunately, when using GOST-CBC-IV-NULL as a replacement for a 128-bit block cipher, one loses the applicability of security proofs on 128 bits. Indeed the use of a 64-bit block cipher in CBC-IV-NULL mode makes it straightforward to distinguish it from a 128-bit random permutation. If we denote by $M = (M_1 \| M_2)$ a 128-bit plaintext as the concatenation of two 64-bit halves, then the resulting ciphertext is $C = (E_K^{\text{64-bits}}(M_1) \| E_K^{\text{64-bits}}(M_2))$. If we consider a second plaintext $M' = (M_1 \| M_2')$ with the same first half and different second one, then $C' = (E_K^{\text{64-bits}}(M_1) \| E_K^{\text{64-bits}}(M_2'))$ also shares the exact same first half as C. This is not the proper behavior of a 128-bit random permutation, where the whole ciphertext should change.

This property could have had a strong impact on the use of GOST-CBC-IV-NULL in XTS if the index n of the data unit were not encoded in little endian before being encrypted. Indeed, the

index of a data unit being smaller than 2^{64}, the most significant bits would have constantly been zero throughout the entire use of the mode for every first block of each data unit whose index denoted i is zero. For these blocks, the behavior of the mode would have almost exactly been the one of ECB, including the bad properties of distinguishability in case of repeating plaintext blocks, as it is the case with the filling of free space with encryption of zero blocks in VeraCrypt. The composition is immune from this danger thanks to the encoding of the standard but the encoding is not *per se* a security measure and the example shows that the security problem is avoided rather by chance.

A second general problem arises from the use of the CBC mode: it is possible for an attacker to apply selective bit flips on the second half of any 128-bit plaintext block by bit flipping the corresponding bits in the first half of a 128-bit ciphertext block. This attack is not possible with the usual XTS mode with a 128-bit block cipher.

Then a third problem is brought by the use of a 64-bit block cipher. It is difficult to avoid any appearance of a *birthday paradox* bound when using a block cipher mode. For a 128-bit block cipher, the birthday bound is around 2^{64}, which means that to remain secure, one has to call the underlying block cipher significantly less than 2^{64} times. Examples of limitations on the number of queries to the underlying block cipher with the same key are given in the standard, with the corresponding success probability for an attack. For 2^{36} calls, *i.e.* 2^{36} 128-bit blocks encrypted or 1 terabyte of data, the success probability is 2^{-53}.

When using a 64-bit block cipher in the CBC-IV-NULL setting, after 2^{36} calls, the birthday bound for 64-bit blocks is reached and the success probability for an attack is 1! This can be seen when considering the first halves of 128-bit ciphertexts which follow a kind of XTS mode on 64 bits. To reach the same level of security as its 128-bit counterpart, the amount of data to be processed should be around 512 bytes which is too small to be considered for a data at rest encryption system. Examples of attacks are provided in *[IEEE07]*.

More generally, 64-bit block ciphers are less and less adapted for the amount of data processed on a usual basis nowadays. TrueCrypt switched to 128-bit block ciphers only several years ago, keeping the support for 64-bit ciphers (Blowfish, Triple DES, CAST5) for compatibility with older volumes. Recent works have shown that what was once thought of as theoretical attacks could in fact be carried out practically, see *e.g.* the attack called *Sweet32 [BL16]* against TLS connections using 64-bit block ciphers. Therefore we recommend removing GOST 28147-89 from the available ciphers to be used with XTS in VeraCrypt.

As we recommend to remove GOST from the set of available ciphers, we did not proceed with the cryptographic evaluation of the intrinsic strength of the variant implemented with the so-called *dynamic S-boxes*.

6.2 Lack of Test Vectors for Newly Added Algorithms

Class	Severity	Difficulty
Cryptography	Informational	Undetermined

No test vectors are provided for Kuznyechik and GOST 28147-89. If test vectors are not sufficient to verify the correctness of an implementation, they can be used to detect some problems, such as bugs that occur only on some architectures or operating systems (errors on big-endian architectures, on 64-bit architectures, etc.)

The use of test vectors seems necessary here, especially not to induce the user in error. Indeed, an “Auto-Test All” button displays “Self-tests on all algorithms passed” in case of success.

User believes all the algorithms have been tested. This is wrong. Moreover, each implemented algorithm can have several implementations. In the AES case, a classic C version, a x86 and a x64 version, a AES-NI and two versions with a small memory footprint (one in C, one in assembly) exist in the project.

As shown below, a problem that could be easily detected with test vectors has been identified in VeraCrypt. This result alone justifies the necessity to include test vectors for all the cryptographic algorithms available in the software.

6.3 Input and Output Parameters Are Swapped in GOST Magma

Class	Severity	Difficulty
Cryptography	Informational	Undetermined

GOST Magma is an encryption algorithm whose block size is 64 bits, contrarily to all the other algorithms in VeraCrypt which process blocks of 128 bits. In order to benefit from the XTS implementation used by the other algorithms, VeraCrypt emulates a 128-bit block cipher by encrypting not one but two 64-bit blocks simultaneously using CBC mode and a null initialization vector. As seen previously, this is not a good idea.

Two implementations are possible:

- For 32-bit code, the **gost_encrypt** and **gost_decrypt** process two 64-bit blocks in CBC mode and call the C functions **gost_encrypt_block** and **gost_decrypt_block**. This code is also used for 64 bit code under Linux and OS X.
- Under Windows x64, the two assembly functions **gost_encrypt_128_CBC_asm** and **gost_decrypt_128_CBC_asm** are used.

The inputs and outputs of the **gost_encrypt_128_CBC_asm** and the **gost_decrypt_128_CBC_asm** are interchanged: the order of the parameters is not the same in the declaration and in the definition of the function. The C prototype in the function definition is:

Listing 6.3: **src/Crypto/GostCipher.c:230**

```
void gost_encrypt_128_CBC_asm(const byte *in, byte *out, gost_kds *ks, uint64 count);
void gost_decrypt_128_CBC_asm(const byte *in, byte *out, gost_kds *ks, uint64 count);
```

Here is the beginning of the assembler implementation:

Listing 6.4: **src/Crypto/gost89_x64.asm:294**

```
global gost_encrypt_128_CBC_asm
; gost_encrypt_128_CBC_asm(uint64* out, uint64* in, gost_kds* kds, uint64 count);
; rcx - &out
; rdx - &in
; r8  - &gost_kds
; r9  - count
gost_encrypt_128_CBC_asm:
```

The **in** and **out** parameters have been swapped. The same inversion is made in **gost_encrypt_128_CBC_asm** :

Listing 6.5: **src/Crypto/gost89_x64.asm:396**

```
global gost_decrypt_128_CBC_asm
; gost_decrypt_128_CBC_asm(uint64* out, uint64* in, const gost_kds* kds, uint64 count);
; rcx - &out
; rdx - &in
; r8  - &gost_kds
; r9  - count
gost_decrypt_128_CBC_asm:
SaveRegs                                ; Saving

sub rsp, 32
mov [rsp], rdx                          ; Save out addr
mov [rsp+8], rcx                        ; Save in addr
mov [rsp+16], r8                        ; key addr
```

However, in this function, **in** and **out** parameters are swapped a second time: contrary to what is written in the comments, **rcx** points to **in** and **rdx** points to **out**. This double inversion actually makes the **gost_decrypt_128_CBC_asm** correct.

The code of **gost_encrypt_128_CBC_asm** remains invalid: the input and the output are indeed swapped. However, as all the encryption operations are performed in-place, as shown below (code has been simplified to make it more readable), the resulting code is functionally correct.

Listing 6.6: **src/Common/Crypto.c:177**

```
void EncipherBlock(int cipher, void *data, void *ks)
{
   switch (cipher)
   {
      case AES:               aes_encrypt (data, data, ks); break;
      case TWOFISH:           twofish_encrypt (ks, data, data); break;
      case SERPENT:           serpent_encrypt (data, data, ks); break;
      case CAMELLIA:          camellia_encrypt (data, data, ks); break;
      case GOST89:            gost_encrypt(data, data, ks, 1); break;
      case KUZNYECHIK:        kuznyechik_encrypt_block(data, data, ks); break;
      default:                TC_THROW_FATAL_EXCEPTION;       // Unknown/wrong ID
   }
}
```

Even if the resulting code works, we think this problem should be immediately fixed. The insertion of a new encryption operation that is not performed in-place might have serious consequences.

Such a problem would have been quickly detected if test vectors for all the encryption primitives were present. In fact, we spotted it by checking if our test vectors were verified.

6.4 Notes on the PBKDF2 Implementation

Class	Severity	Difficulty
Cryptography	Informational	Undetermined

The VeraCrypt volume header keys are derived from the user password with PBKDF2. The iteration count of the pseudo-random function has been greatly increased according to the

recommendations in the iSec report. A problem occurred: the key derivation was slow, and mounting the volume took too much time.

The PBKDF2 implementation has been rewritten and optimized [2], making the key derivation twice faster. A few minor problems have been identified in this implementation. One of them was already in TrueCrypt, the other ones are specific to VeraCrypt. These are actually bad practices rather than real problems.

6.4.1 The PBKDF2 Implementation Does Not Fully Comply With the Standard

During the key generation, a 32-bit "block index" is stored in big-endian [RFC2898]. In the VeraCrypt implementation, only the least significant byte of this index is used, all the other ones being set to zero:

Listing 6.7: **src/Common/Pkcs5.c:175**

```
   /* big-endian block number */
   memset (&k[salt_len], 0, 3);
   k[salt_len + 3] = (char) b;
```

Cycles can be spotted in the data generated by this implementation whenever the output is longer than 256 times the underlying hash function used by PKBDF2:

- 5120 bytes for RIPEMD-160 ;
- 8192 bytes for SHA-256 ;
- 16394 bytes for SHA-512, Whirlpool and Streebog.

This has absolutely no impact on the security of the product: all the data coming from the PBKDF2 implementation is at most 192 bytes long (6 keys of 256 bits). This behavior could be fixed in order to avoid a bad usage of these functions in the future, if longer key material might be generated for some reason.

6.4.2 Bad Coding Practice in the HMAC-SHA512 Computation

During HMAC computations, a "context", which is a structure containing information about the current state of the computation, is used. This context is passed to the functions performing the hashing operations:

Listing 6.8: **src/Common/Pkcs5.c:278**

```
typedef struct hmac_sha512_ctx_struct
{
   sha512_ctx ctx;
   sha512_ctx inner_digest_ctx; /*pre-computed inner digest context */
   sha512_ctx outer_digest_ctx; /*pre-computed outer digest context */
   char k[PKCS5_SALT_SIZE + 4];
   /* enough to hold (salt_len + 4) and also the SHA512 hash */
```

[2] Commit on the VeraCrypt repository: Cryptography: Divide mount and boot times by 2 using a pre-computation of values used in PRF HMac calculation (thanks to Xavier de Carné de Carnavalet for finding this optimization). https://github.com/veracrypt/VeraCrypt/commit/59afc2c4d9704476bdaf8c4c8b45684a80781a06

```
   char u[SHA512_DIGESTSIZE];
} hmac_sha512_ctx;
```

When the derivation functions have been rewritten, a stack buffer containing temporary data has been suppressed. The new, faster implementation uses the **k** field of the HMAC context to store intermediate values. This field is followed by the **u** field, which contains the final HMAC value.

The **hmac_sha512** writes **SHA512_BLOCKSIZE** = 128 bytes in **k**, whose size is only 68 bytes. Hence, 60 bytes are written in **u**:

Listing 6.9: **src/Common/Pkcs5.c:325**

```
   char* buf = hmac.k; /* there is enough space to hold SHA512_BLOCKSIZE (128) bytes
                        * because k is followed by u in hmac_sha512_ctx
                                                                      */
   ...
   /* Pad the key for inner digest */
   for (b = 0; b < lk; ++b)
      buf[b] = (char) (k[b] ^ 0x36);
   memset (&buf[lk], 0x36, SHA512_BLOCKSIZE - lk);
```

The **u** buffer is not used at that time, and is only written later, at the end of the function. Nevertheless, such a code is not acceptable in such a critical function of the software.

6.4.3 Unused Parameters in Derivation Sub-Functions

A last fix should be applied. The derivation function which uses SHA-256 as a PRF, **derive_key_sha256**, calls another function, **derive_u_sha256**, which in turn calls **hmac_sha256_internal**.

The last two functions take **k**, the HMAC secret key, and **lk**, its length, as first parameters. It is actually the password supplied by the user. However, these two parameters are never used: only the **derive_key_sha256** processes them. This makes reading quite disturbing, and should be fixed.

This scheme is identical for all the hash functions.

6.5 Random Byte Generators in DCS Should Be Improved

Class	Severity	Difficulty
Cryptography	Informational	Undetermined

DCS, the UEFI bootloader, contains three random byte generators. They are used to create random data on startup. Currently, these generators cannot be used without a custom configuration step performed by DcsCfg, a diagnostic tool for the bootloader. The consequences of their use is beyond the scope of this audit. However, we would like to give an opinion about them.

Currently, the random data that can be generated at startup is not very sensitive: the random generators are used only to create new salts when a user decides to change his password and has used DcsCfg to enable the use of the generators. A new 64-byte salt is generated during the creation of the new volume header.

Three random byte generators are present in DCS :

- **RndDtrmHmacSha512**, a simplified implementation of HMAC_DRBG specified in *[SP800-90A]*, Section 10.1.2, with SHA-512.
- **RndRDRand**, which returns the output of RDRAND.
- **RndFile**, which returns the content of a given file.

The last generator has probably been included for test purposes, in order to return known data. It should be included only in debug versions of the project.

We have reservations about the direct use of the **RndRDRand** output. It seems desirable to use the output of RDRAND (or the output of RDSEED on newer processors) as an additional source of entropy for a software-based random generator, but its use without post-processing is undesirable because its implementation, purely hardware, cannot be verified.

The first generator, HMAC_DRBG, is only fed by the content of a file, specified in the configuration, and by the system time. Using RDSEED as an additional source seems relevant, particularly when sensitive data is generated.

The generation of random data at startup is an arduous task. The implementation should be carefully studied. It is difficult to gather "good" sources of entropy when the computer starts. We strongly recommend using such mechanisms just in case of absolute necessity.

Lastly, a comment on the code of **RndDtrmHmacSha512**, *a simplified implementation of HMAC DRBG with SHA-512*. We could argue on the *simplified* qualifier. Indeed, the *Reseed* function of the NIST's standard is not implemented, which is supposedly the reason for the *simplified*. However, for a code which takes care to put in the comments the parallel with the standard, it is unfortunate that the logical bounds of the different functions, especially *Update* and *Instantiate* is not followed, and that the *reseed* parameter used in the code has nothing to do with the meaning of *Reseed* in the standard. This makes the code more difficult to read/check against the standard it implements. There is probably an off-by-one in the value of *reseed_counter*/**ReseedCtr** between the code and the standard. The pieces of notation used in **DcsCfgLib.h** for the DRBG state follows the one for Hash_DRBG (**V/C**) and not the one of HMAC_DRBG (**V/K**).

7. UEFI Support Assessment

As explained previously, one of the important features brought by VeraCrypt 1.18 is the UEFI support. It is actually one of the most important features added by VeraCrypt since the beginning of the project.

All the code specific to UEFI is in a separate repository, named VeraCrypt-DCS. DCS means "Disk Cryptography Services". This new module is considered much less mature than the rest of the project by M. Idrassi, VeraCrypt's lead developer. No documentation concerning the compilation of the project, its architecture and its features is available.

Some parts are incomplete, or not implemented at all. This is the case of the "Picture Passwords" or the TPM support. We focused on the parts that can be run without having to modify the volume information accessible only after authentication.

DCS is composed of the following elements:

- DcsBoot : DCS bootloader application
- DcsInt : DCS block R/W interceptor
- DcsBml : DCS boot menu lock application
- DcsRe : DCS recovery loader application
- DcsCfg : DCS configuration tool

DcsInt is the main application. It handles user authentication and installs the driver that manages the transparent encryption and decryption of the disk sectors. It is loaded by DcsBoot, whose role is, then, to load the Windows bootloader once the user has been authenticated and the driver has been installed.

DcsRe is the recovery application. It is installed on the rescue disk, and allows to handle various problems. In particular, it restores the VeraCrypt bootloader if it has been erased.

DcsCfg is a maintenance and diagnostic tool. It is not intended for end users. Its study is out of the scope of the audit. Nevertheless, some features of DcsInt and DcsRe need to be configured first with DcsCfg. The problems in DcsCfg have been transmitted to IDRIX but are not detailed in this report.

A major part of DCS relies on the VeraCrypt code. In this part, only the problems specifically related to DCS are listed.

7.1 Keystrokes Are Not Erased After Authentication

Class	Severity	Difficulty
Data Exposure	High	High

As explained in *The Length of the Password Can Be Computed When Encryption Is Activated*, on startup, keystrokes are stored in a specific buffer of the BIOS Data Area. A parallel can be drawn to UEFI: each driver has its own buffer containing the keystrokes. The address of this buffer is not known, and fully depends on the implementation.

The password supplied by the user is read character per character with the `GetKey` function of the VeraCrypt bootloader:

Listing 7.1: **Library/CommonLib/EfiConsole.c:87**

```
EFI_INPUT_KEY
GetKey(void)
{
   EFI_INPUT_KEY key;
   UINTN EventIndex;

   gBS->WaitForEvent(1, &gST->ConIn->WaitForKey, &EventIndex);
   gST->ConIn->ReadKeyStroke(gST->ConIn, &key);
   return key;
}
```

It is difficult to make sure the driver implementation will erase the buffer containing the keystrokes. For example, the file **IntelFrameworkModulePkg/Csm/BiosThunk/KeyboardDxe/BiosKeyboard.c** in EDK II shows that strokes are retrieved from the BIOS keyboard buffer through INT 16h. The module never directly accesses the BIOS Data Area. Hence, it will never be erased.

Our recommendation is, whatever the driver used, to always call the **Reset()** method of **gST->ConIn** to reset the buffers manipulated by the keyboard module. One has to remember that there is no guarantee that they will be correctly erased.

7.2 Sensitive Data Is Not Correctly Erased

Class	Severity	Difficulty
Data Exposure	High	High

The data handled by the boot loader are rarely erased. The user password is properly cleared at startup. However, when a user changes his password, the **Password** structures containing the new password will not be erased (see the **SecRegionChangePwd** function in **DcsInt / DcsInt.c**).

TrueCrypt's developers and VeraCrypt's have carefully checked if sensitive data was correctly cleared in memory. This level of care has not been taken into DCS yet.

7.3 Memory Corruption Can Occur When the Recovery Disk Is Read

Class	Severity	Difficulty
Data Validation	High	High

VeraCrypt proposes to create a rescue disk able to recover a volume in case of crash. This disk restores the EFI loader settings, restores the loader itself, or boots the system with its own copy of the loader.

This "Recovery Disk" is actually a Zip archive containing the recovery application, related modules, and a backup of the VeraCrypt system volume header.

All the data added to this archive can be considered as trusted:

- The EFI application and modules are extracted from the VeraCrypt.exe resources.
- Reading the system volume header requires administrator privileges.
- Reading the configuration requires administrator privileges.

- Configuration is created by the application.

The vulnerabilities identified in **XZip** and **XUnzip** could therefore not be triggered. However, it is possible to verify the created image, through the "System" → "Verify Rescue Disk Image" menu. In that case, VeraCrypt will open and parse the Zip. It will then be possible to trigger the vulnerabilities identified in **inflate** and **XUnzip**.

Just before reading this Zip, VeraCrypt must perform a privileged action. To perform it, it launches an elevated instance of VeraCrypt.exe with administrator privileges, which acts as a COM server. This server exposes functions able to copy or delete files on the entire disk with administrator rights, or to rewrite the EFI loader. The severity was ranked "High" in part because of it.

The operating conditions are quite unrealistic: an attacker alters the victim's rescue disk, who has to verify it to be compromised. Note that the rescue disk is not a secret data (it does not have the encryption keys).

7.4 Mistakes in the DCS Code

7.4.1 A Null Pointer Can Be Dereferenced When Encrypted Blocks Are Written

Class	Severity	Difficulty
Data Validation	Low	Undetermined

The function responsible for the on-the-fly encryption of data during disk write operations does not return correctly in case of errors. This can lead to a null pointer dereference.

IntBlockIO_Write is the write function registered by the driver installed by DcsInt. It takes the size of the data to encrypt as a parameter. A buffer of the corresponding size is allocated. If the allocation fails, **EFI_BAD_BUFFER_SIZE** is assigned to the return value **Status**. The function might then return. However, the **return Status;** line seems to have been deleted inadvertently from the source code.

Listing 7.2: **DcsInt/DcsInt.c:262**

```
        writeCrypted = MEM_ALLOC(BufferSize);
        if (writeCrypted == NULL) {
           Status = EFI_BAD_BUFFER_SIZE;

        }
        CopyMem(writeCrypted, Buffer, BufferSize);
        //      Print(L"*");
        UpdateDataBuffer(writeCrypted, (UINT32)BufferSize, startSector);
        EncryptDataUnits(writeCrypted, (UINT64_STRUCT*)&startSector,
                         (UINT32)(BufferSize >> 9),
                         DcsIntBlockIo->CryptInfo);
        Status = DcsIntBlockIo->LowWrite(This, MediaId, startSector, BufferSize,
                                         writeCrypted);
```

If the allocation fails, the content of **Buffer** will be copied at address `0`.

7.4.2 Dead Code in DcsInt

Class	Severity	Difficulty
Undetermined	Informational	Undetermined

The function **IntBlockIo_Hook** in **DcsInt/DcsInt.c** contains dead code. This code seems to come from a rewriting of the function.

Listing 7.3: **DcsInt/DcsInt.c:345**

```
   if (!EFI_ERROR(Status)) {
      // Check is this protocol already hooked
      DcsIntBlockIo = (DCSINT_BLOCK_IO *)MEM_ALLOC(sizeof(DCSINT_BLOCK_IO));
      if (DcsIntBlockIo == NULL) {
          return EFI_OUT_OF_RESOURCES;
      }

      // construct new DcsIntBlockIo
      DcsIntBlockIo->Sign = DCSINT_BLOCK_IO_SIGN;
      DcsIntBlockIo->Controller = DeviceHandle;
      DcsIntBlockIo->BlockIo = BlockIo;
      DcsIntBlockIo->IsReinstalled = 0;

      if (EFI_ERROR(Status)) {
          gBS->CloseProtocol(
                  DeviceHandle,
                  &gEfiBlockIoProtocolGuid,
                  This->DriverBindingHandle,
                  DeviceHandle
                  );
          MEM_FREE(DcsIntBlockIo);
          return EFI_UNSUPPORTED;
      }
```

The first condition checks whether **Status** is not an error code, while it checks just below if it is an error code. The **CloseProtocol** method will never be called. Incidentally, the first comment is misleading because no check is done on the hook here.

7.4.3 The Function Reading the Configuration May Read Inconsistent Data

Class	Severity	Difficulty
Data Validation	Informational	Undetermined

The configuration file of the loader, **DcsProp**, is read in the **ConfigRead** function. This function calls **FileLoad** to load it, but does not check the value returned by **FileLoad**:

Listing 7.4: **Library/VeraCryptLib/DcsVeraCrypt.c:36**

```
BOOL ConfigRead(char *configKey, char *configValue, int maxValueSize)
{
   char *xml;

   if (ConfigBuffer == NULL)
      FileLoad(NULL, L"\\EFI\\VeraCrypt\\DcsProp", &ConfigBuffer, &ConfigBufferSize);
```

```
   xml = ConfigBuffer;
   if (xml != NULL)
   {
      xml = XmlFindElementByAttributeValue(xml, "config", "key", configKey);
      ...
}
```

The configuration will be parsed if **ConfigBuffer** is not null. However, **FileLoad** may well return an error while returning a non-null **ConfigBuffer** filled with zeros:

Listing 7.5: **Library/CommonLib/EfiFile.c:200**

```
EFI_STATUS
FileLoad(
   IN    EFI_FILE*   root,
   IN    CHAR16*     name,
   OUT   VOID**      data,
   OUT   UINTN*      size
   )
...
   *data = MEM_ALLOC(sz);
   if (*data == NULL) {
   ...
   }
   res = FileRead(file, *data, &sz, NULL);
   if (EFI_ERROR(res)) {
      FileClose(file);
      MEM_FREE(*data);
      return res;
   }
```

This does not lead to a security issue, but might be fixed.

Incidentally, the memory allocated to read the configuration file is never freed.

7.4.4 Bad Pointer Check in `EfiGetHandles`

Class	Severity	Difficulty
Data Validation	Informational	Undetermined

EfiGetHandles calls the **LocateHandle** service to retrieve all the handles that implement a given protocol. If several handles are present, the allocated space to retrieve this list will be too small. Hence **EfiGetHandles** allocates more memory. The pointer returned by the allocator is not correctly checked. The consequence is that the function can dereference a null pointer. The cause of this bug is a typo error, as one can see below.

Listing 7.6: **Library/CommonLib/EfiBio.c:76**

```
   *Buffer = (EFI_HANDLE*) MEM_ALLOC(sizeof(EFI_HANDLE));
   if (*Buffer) {
      BufferSize = sizeof(EFI_HANDLE);
      res = gBS->LocateHandle(SearchType, Protocol, SearchKey, &BufferSize, *Buffer);
      if (res == RETURN_BUFFER_TOO_SMALL) {
          MEM_FREE(*Buffer);
          *Buffer = (EFI_HANDLE*)MEM_ALLOC(BufferSize);
```

```
          if (!Buffer) {  // Typo error: Buffer is checked instead of *Buffer
             return EFI_OUT_OF_RESOURCES;
          }
```

7.4.5 Potential Dereference of a Null Pointer in the Graphic Library

Class	Severity	Difficulty
Data Validation	Informational	Undetermined

Two functions of the graphic library, **BltLine** and **BltCircle**, which respectively print a line and a circle, take a graphical context **draw** as a parameter. The type of **draw** is **PDRAW_CONTEXT**. This variable is checked at the beginning of both functions in order to assign a variable to the **mask** value:

Listing 7.7: **Library/GraphLib/EfiGraph.c:300**

```
   mask = draw ? draw->DashLine : gDrawContext.DashLine;
   dmask = mask;
   cmask = 32;
   for (;;) {
        /* loop */
           // Dash
           if ((dmask & 1) == 1) {
              // always true if draw is NULL, as gDrawContext.DashLine is 0xffffffff
              BltPoint(blt, draw, x0, y0);
           }
```

Hence the **draw** value is potentially null. In that case, the global context **gDrawContext** is used instead of **draw->DashLine**. The circle or the line are printed point after point with the **BltPoint** function. This function also takes a parameter of type **PDRAW_CONTEXT**. However, in both cases, the **draw** variable is passed as a parameter to **BltPoint** instead of the global context **gDrawContext**.

BltPoint dereferences this pointer without checking it:

Listing 7.8: **Library/GraphLib/EfiGraph.c:231**

```
EFI_STATUS
BltPoint(
        IN BLT_HEADER* blt,
        IN PDRAW_CONTEXT draw,
        IN UINTN x,
        IN UINTN y
        ) {
        if (draw->Brush == NULL) return BltPointSingle(blt, draw, x, y);
        else
```

An analysis of the calling functions of **BltLine** and **BltCircle** shows that the **draw** parameter can actually never be null. The bug must be fixed, however.

8. Recommendations

In this section, we sum up all vulnerabilities and related recommendations.

8.1 Unfixed or Partially Fixed Vulnerabilities from Former Audits

8.1.1 OCAP Phase 1 Audit (iSec Partners, NCC Group)

Sensitive information might be paged out from kernel stacks					
Make sure the user follow VeraCrypt's documentation by encrypting the system partition/drive and making sure that all paging files are located on partitions within the key scope of the system encryption.					
Class	Data Exposure	Severity	Medium	Status	Not Fixed

Windows kernel driver uses `memset()` to clear sensitive data					
The structure **`mappedCryptoInfo`** must be erased with **`burn()`** before raising the exception **`TC_THROW_FATAL_EXCEPTION`** in **`src/Driver/DriveFilter.c`**.					
Class	Data Exposure	Severity	Medium	Status	Partially fixed

`TC_IOCTL_OPEN_TEST` multiple issues					
To be found in iSec's report. Warning: a similar issue can be found in the **`TC_IOCTL_GET_SYSTEM_DRIVE_CONFIG`** ioctl.					
Class	Data Exposure	Severity	Low	Status	Not fixed

`EncryptDataUnits()` lacks error handling					
To be found in iSec's report.					
Class	Error Reporting	Severity	Informational	Status	Not fixed

8.1.2 OCAP Phase 2 Audit (Cryptography Services, NCC Group)

AES implementation susceptible to cache-timing attacks					
Fix **`Aes_x86.asm`** and **`Aes_x64.asm`** first.					
Class	Cryptography	Severity	High	Status	Not fixed

Keyfile mixing is not cryptographically sound					
To be found in NCC Group's report. We emphasize the need to implement the recommendation.					
Class	Cryptography	Severity	Low	Status	Not fixed

Unauthenticated ciphertext in volume headers					
Implement a cryptographic authentication mechanism.					
Class	Cryptography	Severity	Undetermined	Status	Not fixed

8.2 VeraCrypt's Modifications Assessment

The length of the password can be computed when encryption is activated					
Erase pointers to last and next password character position in the keystroke buffer.					
Class	Data Exposure	Severity	Low	Difficulty	Medium

Out-of-date `inflate` and `deflate`					
Add a dependency on **`zlib`** to benefit from an up-to-date code base.					
Class	Patching	Severity	High	Difficulty	High

`XZip` and `XUnzip` need to be completely re-written					
Use another component to handle Zip files.					
Class	Data Validation	Severity	High	Difficulty	High

Integer overflow when computing the number of iterations for PBKDF2 when PIM is used					
Unify the application behavior so that the checks on the PIM will be performed in the core functions of the program.					
Class	Data Validation	Severity	High	Difficulty	High

PIN code on command line					
Remove the feature or at least attach a clear security warning to it.					
Class	Data Exposure	Severity	Informational	Difficulty	Low

8.3 New Cryptographic Mechanisms Assessment

GOST 28147-89 Must Be Removed from VeraCrypt					
Remove GOST 28147-89 and more generally any 64-bit block cipher from the list of available block ciphers.					
Class	Cryptography	Severity	High	Difficulty	High

Lack of test vectors for newly added algorithms					
Add relevant test vectors.					
Class	Cryptography	Severity	Informational	Difficulty	Undetermined

Input and output parameters are swapped in GOST Magma					
Fix the implementation.					
Class	Cryptography	Severity	Informational	Difficulty	Undetermined

The PBKDF2 implementation does not fully comply with the standard					
Make the implementation compliant with the standard.					
Class	Cryptography	Severity	Informational	Difficulty	Undetermined

Bad coding practice in the HMAC-SHA512 Computation					
Fix the implementation.					
Class	Cryptography	Severity	Informational	Difficulty	Undetermined

Unused parameters in key derivation sub-functions					
Fix the implementation.					
Class	Cryptography	Severity	Informational	Difficulty	Undetermined

Random Byte Generators in DCS Should Be Improved					
The generation of random data at startup is an arduous task. The implementation should be carefully studied. It is difficult to gather "good" sources of entropy when the computer starts. We strongly recommend using such mechanisms just in case of absolute necessity.					
Class	Cryptography	Severity	Informational	Difficulty	Undetermined

8.4 UEFI Support Assessment

Keystrokes are not erased after authentication					
Always call the `Reset()` method of `gST->ConIn` to reset the buffers manipulated by the keyboard module.					
Class	Data Exposure	Severity	High	Difficulty	High

Sensitive data is not correctly erased					
Securely clear sensitive data from memory.					
Class	Data Exposure	Severity	High	Difficulty	High

Memory corruption can occur when the recovery disk is read					
Use another component to handle Zip files.					
Class	Data Validation	Severity	High	Difficulty	High

A null pointer can be dereferenced when encrypted blocks are written					
Fix the implementation.					
Class	Data Validation	Severity	Low	Difficulty	Undetermined

Dead code in `DcsInt`					
Fix the implementation.					
Class	Undetermined	Severity	Informational	Difficulty	Undetermined

The function reading the configuration may read inconsistent data					
Fix the implementation.					
Class	Data Validation	Severity	Informational	Difficulty	Undetermined

Bad pointer check in `EfiGetHandles`					
Fix the implementation.					
Class	Data Validation	Severity	Informational	Difficulty	Undetermined

Potential dereference of a null pointer in the graphic library					
Fix the implementation.					
Class	Data Validation	Severity	Informational	Difficulty	Undetermined

9. Conclusion

This audit, funded by OSTIF, required 32 man-days of study. It shows that this follow-up of TrueCrypt is very much alive and evolves with new functionalities like the support of UEFI.

The results shows that evaluations at regular intervals of such difficult security projects are not an option. When well received by the project's developers, they provide useful feedbacks to help the project mature. The openess of the evaluation results help build confidence in the product for the final users.

9. Bibliography

[OCAP1] iSec Partners, part of NCC Group. *Open Crypto Audit Project - TrueCrypt, Security Assessment.* 2014. Available at https://opencryptoaudit.org/reports/iSec_Final_Open_Crypto_Audit_Project_TrueCrypt_Security_A

[OCAP2] Cryptography Services of NCC Group. *Open Crypto Audit Project - TrueCrypt, Cryptographic Review.* 2015 Available at https://opencryptoaudit.org/reports/TrueCrypt_Phase_II_NCC_OCAP_final.pdf

[FSIT] Fraunhofer Institute for Secure Information Technology. *Security Analysis of TrueCrypt.* 2015. Available at https://www.bsi.bund.de/DE/Publikationen/Studien/TrueCrypt/truecrypt.html

[SP800-132] NIST Special Publication 800-132. *Recommendation for Password-Based Key Derivation.* December 2010. Available at http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf

[SOGETI] Sogeti ESEC Lab. *Security assessment of TrueCrypt.* 2008. Available at http://esec-lab.sogeti.com/posts/2008/12/08/security-assessment-of-truecrypt-english.html

[UPR] Ubuntu Privacy Remix Team. *Security Analysis of TrueCrypt 7.0a with an Attack on the Keyfile Algorithm.* 2011. Available at https://www.privacy-cd.org/en/tutorials/analysis-of-truecrypt

[P0-537] Google Project Zero. *Truecrypt 7 Derived Code/Windows: Incorrect Impersonation Token Handling EoP.* Available at https://bugs.chromium.org/p/project-zero/issues/detail?id=537

[P0-538] Google Project Zero. *Truecrypt 7 Derived Code/Windows: Drive Letter Symbolic Link Creation EoP.* Available at https://bugs.chromium.org/p/project-zero/issues/detail?id=538

[P0-BLOG] Google Project Zero. *Windows Drivers are True'ly Tricky.* Available on Google Project Zero's Blog at https://googleprojectzero.blogspot.fr/2015/10/windows-drivers-are-truely-tricky.html

[GOST89] Government Committee of the USSR for Standards. *Cryptographic Protection for Data Processing System*, GOST 28147-89, Gosudarstvennyi Standard of USSR, 1989. (In Russian)

[IEEE07] IEEE P1619/D16. *Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices.* 2007.

[SP800-38E] NIST Special Publication 800-38E. *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices.* January 2010.

[Ro04] Phillip Rogaway. *Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC.* Asiacrypt 2004. LNCS vol. 3329. Springer, 2004. Available at http://web.cs.ucdavis.edu/~rogaway/papers/offsets.pdf

[LM08] Moses Liskov and Kazuhiko Minematsu. *Comments on XTS-AES* , in response to NIST Public Request for Comments on XTS. 2008. Available at

http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/comments/XTS/XTS_comments-Liskov_Minematsu.pdf

[BL16] Karthikeyan Bhargavan and Gaëtan Leurent. On the Practical (In-)Security of 64-bit Block Ciphers — Collision Attacks on HTTP over TLS and OpenVPN. To be published at ACM CCS 2016.

[RFC2898] PKCS #5: Password-Based Cryptography Specification Version 2.0. Available at https://tools.ietf.org/html/rfc2898#section-5.2

[SP800-90A] NIST Special Publication 800-90A Revision 1. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. June 2015. Available at http://dx.doi.org/10.6028/NIST.SP.800-131Ar1