Open Crypto Audit Project
TrueCrypt

Security Assessment

iSECpartners
part of nccgroup

Prepared for:

Open Crypto Audit Project

Prepared by:

Andreas Junestam –Security Engineer

Nicolas Guigo – Security Engineer

Document Change Log		
Version	**Date**	**Change**
0.5	2014-02-12	First draft of document
0.9	2014-02-13	Peer reviewed by Josh Yavor, Mike Ryan, Javed Samuel, and Russ Sevinsky
1.0	2014-02-14	Delivered to Open Crypto Audit Project
1.1	2014-03-04	Finalized for publication

Table of Contents

1 Executive Summary

Application Summary	
Application Name	TrueCrypt
Application Version	7.1a
Application Type	Disk encryption software
Platform	Windows, C / C++

Engagement Summary	
Engineers Engaged	Two (2)
Engagement Type	Security Assessment
Testing Methodology	Source code aided security assessment

Vulnerability Summary	
Total High severity issues	Zero (0)
Total Medium severity issues	Four (4)
Total Low severity issues	Four (4)
Total vulnerabilities identified	Eleven (11) (incl. three (3) Informational)

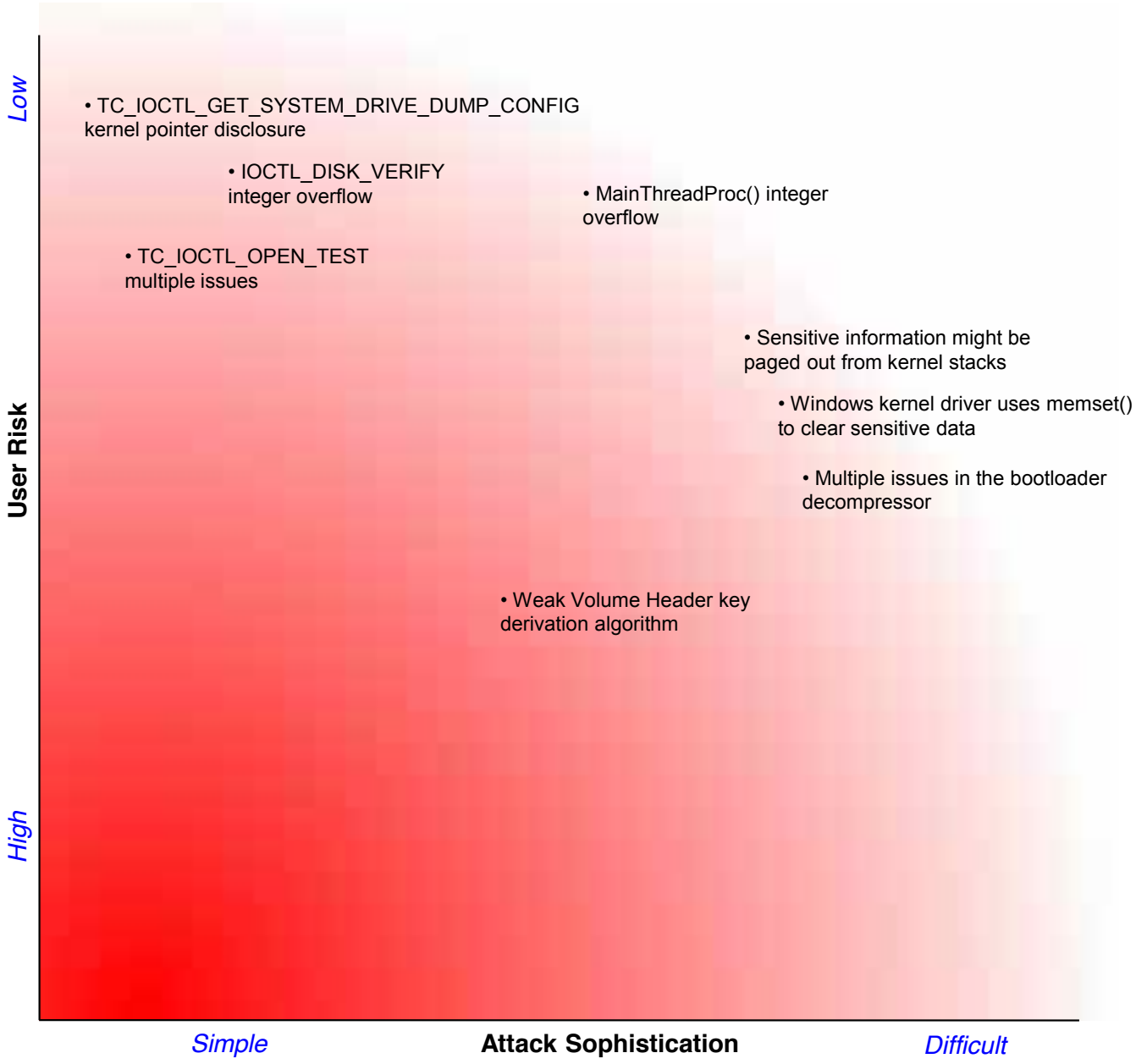
See Section 3.1 for descriptions of these classifications.

Category Breakdown:

Access Controls	0
Auditing and Logging	0
Authentication	0
Configuration	0
Cryptography	1 ■
Data Exposure	4 ■■■■
Data Validation	3 ■■■
Denial of Service	2 ■■
Error Reporting	1 ■
Patching	0
Session Management	0
Timing	0

1.1 iSEC Risk Summary

The iSEC Partners Threat Matrix chart evaluates discovered vulnerabilities according to estimated user risk. The impact of the vulnerability increases towards the bottom of the chart. The sophistication required for an attacker to find and exploit the flaw decreases towards the left of the chart. The closer a vulnerability is to the chart origin, the greater the risk.



1.2 Project Summary

The Open Crypto Audit Project[1] engaged iSEC Partners to review select parts of the TrueCrypt 7.1a disk encryption software. This included reviewing the bootloader and Windows kernel driver for any system backdoors as well as any other security related issues.

iSEC performed a source code assisted security assessment of the TrueCrypt bootloader and Windows kernel driver. A breakdown of what areas were included, as well as excluded, can be found in Section 2.2. The assessment included reviewing source code for the bootloader and kernel driver as well as hands-on testing. The team also compiled both components and performed limited fuzzing of select interfaces.

The iSEC team reviewed the TrueCrypt 7.1a source code, which is publicly available as a zip archive ("truecrypt 7.1a source.zip") at http://www.truecrypt.org/downloads2. The SHA1 hash of the reviewed zip archive is `4baa4660bf9369d6eeaeb63426768b74f77afdf2`. iSEC did not attempt to create a reproducible build; instead, the team performed hands-on testing against binaries available from http://www.truecrypt.org/downloads and binaries compiled from the source code.

Two engineers performed this work in an engagement concluding on February 14th, 2014. All work was carried out at iSEC's facilities in Seattle, WA.

1.3 Findings Summary

During this engagement, the iSEC team identified eleven (11) issues in the assessed areas. Most issues were of severity Medium (four (4) found) or Low (four (4) found), with an additional three (3) issues having severity Informational (pertaining to Defense in Depth).

Overall, the source code for both the bootloader and the Windows kernel driver did not meet expected standards for secure code. This includes issues such as lack of comments, use of insecure or deprecated functions, inconsistent variable types, and so forth. A more in-depth discussion on the quality issues identified can be found in Appendix B. In contrast to the TrueCrypt source code, the online documentation available at http://www.truecrypt.org/docs/ does a very good job at both describing TrueCrypt functionality and educating users on how to use TrueCrypt correctly. This includes recommendations to enable full disk encryption that protects the system disk, to help guard against swap, paging, and hibernation-based data leaks.

The team also found a potential weakness in the Volume Header integrity checks. Currently, integrity is provided using a string ("TRUE") and two (2) CRC32s. The current version of TrueCrypt utilizes XTS[2] as the block cipher mode of operation, which lacks protection against modification; however, it is insufficiently malleable to be reliably attacked. The integrity protection can be bypassed, but XTS prevents a reliable attack, so it does not currently appear to be an issue. Nonetheless, it is not clear why a cryptographic hash or HMAC was not used instead.

Finally, iSEC found no evidence of backdoors or otherwise intentionally malicious code in the assessed areas. The vulnerabilities described later in this document all appear to be unintentional, introduced as the result of bugs rather than malice.

[1] http://opencryptoaudit.org/
[2] http://www.truecrypt.org/docs/modes-of-operation

1.4 Recommendations Summary

Outside of the specific short and long term recommendations detailed in Section 3 of this document, iSEC Partners also makes the following high level recommendations.

Update the Windows build environment. The current required Windows build environment depends on outdated build tools and software packages that are hard to get from trustworthy sources. For example, following the reproducible build instructions at https://madiba.encs.concordia.ca/~x_decarn/truecrypt-binaries-analysis/ requires access to VC++ 1.52 (released in 1993), in addition to various Windows ports of GNU tools downloadable from wherever they can be found. Using antiquated and unsupported build tools introduces multiple risks including: unsigned tools that could be maliciously modified, unknown or unpatched security vulnerabilities in the tools themselves, and weaker or missing implementations of modern protection mechanisms such as DEP[3] and ASLR[4]. Once the build environment has been updated, the team should consider rebuilding all binaries with all security features fully enabled. For the purpose of auditing, TrueCrypt should release instructions for how to create reproducible builds.

Improve code quality. Due to lax quality standards, TrueCrypt source is difficult to review and maintain. This will make future bugs harder to find and correct. It also makes the learning curve steeper for those who wish to join the TrueCrypt project.

[3] https://en.wikipedia.org/wiki/Data_Execution_Prevention
[4] https://en.wikipedia.org/wiki/Address_space_layout_randomization

2 Engagement Structure

2.1 Internal and External Teams

The iSEC team has the following primary members:

- Andreas Junestam – iSEC Technical Lead
- Nicolas Guigo – iSEC Security Engineer
- Tom Ritter – iSEC Account Contact
- Deanna Bjorkquist – iSEC Project Manager

The Open Crypto Audit Project team has the following primary members:

- Kenneth White – Open Crypto Audit Project Contact
- Matthew Green – Open Crypto Audit Project Contact

2.2 Project Goals and Scope

The goal of this engagement was to review the TrueCrypt bootloader and Windows kernel driver for security issues that could lead to information disclosure, elevation of privilege, or similar concerns. The assessment included a review of the following areas:

- TrueCrypt Bootloader
- Setup process
- Windows kernel driver specifically including:
  - Elevation of Privileges from local user to kernel
  - Information Disclosure during disk operations
  - Volume parsing as it relates to system and drive partitions
  - Rescue Disks code paths that do not have the private key
  - Data Leakage

The assessment explicitly excluded the following areas:

- Volume parsing as it relates to a file container
- Rescue Disks code paths activated when the disk does contain the private key
- Cryptographic Analysis, including
  - RNG analysis
  - Algorithm implementation
  - Security tokens
  - Keyfile derivation
- Hidden Containers
- Linux and Mac Components
- All other components not explicitly included

This review used a combination of proprietary and public automated tools, manual test techniques, and source code review to audit the application.

3 Detailed Findings

3.1 Classifications

The following section describes the classes, severities, and exploitation difficulty rating assigned to each identified issue by iSEC.

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking or order of operations

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Undetermined	The extent of the risk was not determined during this engagement
Low	The risk is relatively small or is not a risk the customer has indicated is important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possible legal implications for client
High	Large numbers of users, very bad for client's reputation, or serious legal or financial implications

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploit was not determined during this engagement
Low	Commonly exploited, public tools exist or can be scripted that exploit this flaw
Medium	Attackers must write an exploit, or need an in-depth knowledge of a complex system
High	The attacker must have privileged insider access to the system, may need to know extremely complex technical details or must discover other weaknesses in order to exploit this issue

3.2 Vulnerability Overview

The following table is a summary of the vulnerabilities identified during testing by iSEC. Subsequent pages of this report detail each of the vulnerabilities, along with short and long term remediation advice.

Vulnerability	Class	Severity
1. Weak Volume Header key derivation algorithm	Cryptography	Medium
2. Sensitive information might be paged out from kernel stacks	Data Exposure	Medium
3. Multiple issues in the bootloader decompressor	Data Validation	Medium
4. Windows kernel driver uses memset() to clear sensitive data	Data Exposure	Medium
5. TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG kernel pointer disclosure	Data Exposure	Low
6. IOCTL_DISK_VERIFY integer overflow	Data Validation	Low
7. TC_IOCTL_OPEN_TEST multiple issues	Data Exposure	Low
8. MainThreadProc() integer overflow	Denial of Service	Low
9. MountVolume() device check bypass	Data Validation	Informational
10. GetWipePassCount() / WipeBuffer() can cause BSOD	Denial of Service	Informational
11. EncryptDataUnits() lacks error handling	Error Reporting	Informational

3.3 Detailed Vulnerability List

1. Weak Volume Header key derivation algorithm

Class: Cryptography **Severity:** Medium **Difficulty:** Medium

FINDING ID: iSEC-OCAP-11

TARGETS: Encrypted Volume Header

DESCRIPTION: The key used to encrypt the TrueCrypt Volume Header is derived using PBKDF2, a standard key derivation algorithm[5]. Developers are responsible for specifying an iteration count that influences the computational cost of deriving a key from a password. The iteration count used by TrueCrypt is either 1000 or 2000, depending on the hash function and use case.

In both cases, this iteration count is too small to prevent password guessing attacks for even moderately complex passwords. The paper that introduces scrypt[6], an alternate key derivation function, demonstrates the challenge of using PBKDF2 even with a very high iteration count – brute-forcing key derivation is easily parallelized and becomes more efficient each year with advances in CPU performance. The use of a small iteration count in TrueCrypt permits efficient brute-force attacks against its header key.

EXPLOIT SCENARIO: An attacker captures an encrypted TrueCrypt volume and performs an offline brute-force and / or dictionary attack to identify the key used to encrypt the Volume Header. They use the recovered key to decrypt the volume.

SHORT TERM SOLUTION: Support the use of configurable iteration counts for PBKDF2 to keep pace with advances in CPU and GPU speed. If the current volume format does not include reserved space to store such a value, and if changes to the Volume Header cannot be made, this value might be derived from a portion of the salt, so long as it is guaranteed to exceed a certain minimum value.

LONG TERM SOLUTION: Consider supporting the use of additional key derivation functions. Scrypt, in particular, requires the use of large amounts of memory and requires more expensive hardware to brute-force.

[5] http://www.truecrypt.org/docs/header-key-derivation
[6] http://www.tarsnap.com/scrypt/scrypt.pdf

2. Sensitive information might be paged out from kernel stacks

Class: Data Exposure **Severity:** Medium **Difficulty:** High

FINDING ID: iSEC-OCAP-6

TARGETS: TrueCrypt Windows kernel driver

DESCRIPTION: The TrueCrypt Windows driver code makes some effort to prevent sensitive information from being paged out during a low memory situation by allocating memory from the non-paged pool. However, sensitive information, such as key material, may still leak to various places during execution, among those kernel stack pages, which can be paged out under certain conditions.

If the stack for the system thread created during volume mounting were to be paged out, there is a risk that key information in `ReadVolumeHeader()` could end up on disk.

It should be noted that for this to be a threat, the user must be running a configuration in which the main Windows system disk is not encrypted, something the TrueCrypt documentation explicitly recommends against[7].

EXPLOIT SCENARIO: A user has a system with a TrueCrypt-encrypted partition on it, in which they save sensitive information. An attacker creates a low memory situation on the user's machine, forcing key information to be paged out to the unencrypted system disk. The attacker later gains access to the disk and can extract the key from the page file.

SHORT TERM SOLUTION: Consolidate all sensitive information to one single location. The data can then be locked into memory with the help of functions such as `MmLockPagableDataSection()` or `KeSetKernelStackSwapEnable()` to prevent it from being paged out to disk.

LONG TERM SOLUTION: The short term solution is sufficient to correct this issue. The TrueCrypt team already has documentation discouraging users from using a setup that could be exposed to this.

[7] http://www.truecrypt.org/docs/paging-file#Y3l1

3. Multiple issues in the bootloader decompressor

Class: Data Validation **Severity:** Medium **Difficulty:** High

FINDING ID: iSEC-OCAP-5

TARGETS: `Decompressor.c`

DESCRIPTION: The code to decompress the main bootloader suffers from several implementation weaknesses. Throughout the source code, signed and unsigned integer types are mixed, arrays are accessed without checking if the index is within bounds, and so forth. In several cases, the lack of array bounds checking results out-of-bound accesses actually being performed. Three (3) examples can be found in Appendix A.

It should be noted that in order to exploit this, an attacker would need access to the disk on which the TrueCrypt-encrypted system resides. An attacker with this level of access could instead perform a more effective evil maid attack[8].

EXPLOIT SCENARIO: An attacker modifies the compressed bootloader on the disk to exploit one of the issues in the decompressor. Successful exploitation allows the attacker to modify the TrueCrypt code to record and save the password while the user enters it.

SHORT TERM SOLUTION: Fix the issues identified in the decompressor. Alter the input buffer handling to take an input size argument and verify that the code does not attempt to read past the end of the buffer while decompressing.

LONG TERM SOLUTION: Make integer types more consistent throughout the TrueCrypt source code, favoring well-defined unsigned types wherever possible. Perform extensive fuzzing of the bootloader decompressor as well as a more in-depth review of the functionality.

[8] http://theinvisiblethings.blogspot.com/2009/10/evil-maid-goes-after-truecrypt.html

4. Windows kernel driver uses memset() to clear sensitive data

Class: Data Exposure **Severity:** Medium **Difficulty:** High

FINDING ID: iSEC-OCAP-8

TARGETS: TrueCrypt Windows kernel driver

DESCRIPTION: The function `burn()` is used to clear sensitive data throughout most of the TrueCrypt Windows kernel driver. In the Windows version, `burn()` wraps `RtlSecureZeroMemory()`, which is guaranteed to securely erase memory and will not be optimized out. However, in a handful of places, `memset()` is used to clear potentially sensitive data. Calls to `memset()` run the risk of being optimized out by the compiler.

One such location identified is in `DriveFilter.c`, line 104:

```
BootArgs = *bootArguments;
BootArgsValid = TRUE;
memset (bootArguments, 0, sizeof (*bootArguments));

if (BootArgs.BootLoaderVersion < 0x600)
```

With a second one later in the same file, at line 335:

```
if (mappedCryptoInfo)
{
    Dump ("Wiping memory %x %d\n", cryptoInfoAddress.LowPart,
        BootArgs.CryptoInfoLength);
    memset (mappedCryptoInfo, 0, BootArgs.CryptoInfoLength);
    MmUnmapIoSpace (mappedCryptoInfo, BootArgs.CryptoInfoLength);
}
```

EXPLOIT SCENARIO: A user has a system with a TrueCrypt-encrypted partition on it, in which they save sensitive information. An attacker creates a low memory situation on the user's machine, forcing key information that should have been securely wiped to be paged out to the unencrypted system disk. The attacker later gains access to the disk and extracts the key from the paging file.

SHORT TERM SOLUTION: Alter the above code to call `burn()` instead of `memset()`.

LONG TERM SOLUTION: Audit the code for other instances of `memset()` calls that should be replaced with calls to `burn()` to prevent potential information leakage.

5. TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG kernel pointer disclosure

Class: Data Exposure **Severity:** Low **Difficulty:** Low

FINDING ID: iSEC-OCAP-3

TARGETS: `ProcessVolumeDeviceControlIrp()` in `Ntdriver.c`

DESCRIPTION: The above function contains the following code:

```
case TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG:
    if (ValidateIOBufferSize (Irp, sizeof (GetSystemDriveDumpConfigRequest),
        ValidateOutput))
    {
        GetSystemDriveDumpConfigRequest *request = (GetSystemDriveDumpConfigRequest
            *) Irp->AssociatedIrp.SystemBuffer;
        request->BootDriveFilterExtension = GetBootDriveFilterExtension();
```

`GetBootDriveFilterExtension()` is implemented as follows:

```
DriveFilterExtension *GetBootDriveFilterExtension ()
{
        return BootDriveFilterExtension;
}
```

`BootDriveFilterExtension` is a pointer to the boot drive's extension object, residing in the kernel address space. Calling the function identified above will return this pointer to the caller:

```
>tc_test.exe \\.\truecrypt
0x852a4640
```

This issue discloses a kernel pointer to an unauthenticated userland program, which can be used to help bypass kernel ALSR.

EXPLOIT SCENARIO: An attacker utilizes the kernel pointer disclosure from `TC_IOCTL_GET_SYSTEM_DRIVE_DUMP_CONFIG` to learn where in the kernel address space `BootDriveFilterExtension` resides.

SHORT TERM SOLUTION: Change the code so this functionality cannot be reached from user-space.

LONG TERM SOLUTION: The short term solution is sufficient to correct this issue.

6. IOCTL_DISK_VERIFY integer overflow

Class: Data Validation **Severity:** Low **Difficulty:** Low

FINDING ID: iSEC-OCAP-1

TARGETS: `ProcessVolumeDeviceControlIrp()` in `Ntdriver.c`

DESCRIPTION: The above function contains the following code:

```
case IOCTL_DISK_VERIFY:
…
if (pVerifyInformation->StartingOffset.QuadPart + pVerifyInformation->Length >
Extension->DiskLength) Irp->IoStatus.Status = STATUS_INVALID_PARAMETER;
else
{
    IO_STATUS_BLOCK ioStatus;
    PVOID buffer = TCalloc (max (pVerifyInformation->Length, PAGE_SIZE));
```

By setting a large value for `StartingOffset`, `Length` values larger than `Extension->DiskLength` can bypass the above check. For example, by setting `StartingOffset` to 0xffffffffdaffffff and `Length` to 0x25000002 (~592Mb) the sum of these values will overflow to 1, bypassing the check. The largest value of `pVerityInformation->Length` and `PAGE_SIZE` will later be used in the call to `TCalloc()` to allocate memory from the non-paged pool.

EXPLOIT SCENARIO: An attacker repeatedly calls the IOCTL_DISK_VERIFY with malicious values in order to starve the kernel of memory and make other allocations fail. This in turn can result in either a Denial of Service or issues in other parts of the code.

SHORT TERM SOLUTION: Verify that the addition will not overflow before the sum is checked against `Extension->DiskLength`.

LONG TERM SOLUTION: Deploy proper integer overflow / underflow checks throughout the TrueCrypt source code. For Windows, the IntSafe[9] library is a convenient way to do this.

[9] http://msdn.microsoft.com/en-us/library/windows/desktop/ff521693(v=vs.85).aspx

7. TC_IOCTL_OPEN_TEST multiple issues

Class: Data Exposure **Severity:** Low **Difficulty:** Low

FINDING ID: iSEC-OCAP-2

TARGETS: `ProcessVolumeDeviceControlIrp()` in `Ntdriver.c`, IOCTLs `TC_IOCTL_OPEN_TEST` and `TC_IOCTL_GET_SYSTEM_DRIVE_CONFIG`

DESCRIPTION: The code handling the `TC_IOCTL_OPEN_TEST` IOCTL contains multiple issues, all of which result in information leakage at a minimum. This is due to the fact that the code uses `ZwCreateFile()`, which does not perform an ACL check, to open arbitrary files and performs various operations on them.

With this, an attacker can:

- Deduce the presence of files they do not have access to
- Deduce if said files are smaller than `TC_MAX_VOLUME_SECTOR_SIZE`
- Deduce if said files start with the string "TrueCrypt" or one of four magic markers

It should also be noted that the same issues apply to the code handling the IOCTL `TC_IOCTL_GET_SYSTEM_DRIVE_CONFIG`.

EXPLOIT SCENARIO: An attacker exploits this vulnerability to gain information about files they would normally not have access to.

SHORT TERM SOLUTION: TrueCrypt should investigate if the call to `ZwCreateFile()` can be replaced with a call to `NtCreateFile()`. `NtCreateFile()` will carry out an ACL check before opening the file, only allowing the caller to open files they have access to. If not, stringent validation must be performed on the supplied filename.

LONG TERM SOLUTION: Assess all calls made to `Zw` functions from within the TrueCrypt Windows kernel driver. TrueCrypt should replace all those which can be replaced with a call to the corresponding `Nt` function. For all others, filter arguments that are supplied from requests originating in user-mode.

8. MainThreadProc() integer overflow

Class: Denial of Service **Severity:** Low **Difficulty:** Medium

FINDING ID: iSEC-OCAP-7

TARGETS: `MainThreadProc()` in `EncryptedIoQueue.c`

DESCRIPTION: The above function contains the following code:

```
case IRP_MJ_READ:
    item->Write = FALSE;
    item->OriginalOffset = irpSp->Parameters.Read.ByteOffset;
    item->OriginalLength = irpSp->Parameters.Read.Length;
    break;
```

Both `irpSp->Parameters.Read.ByteOffset` and `irpSp->Parameters.Read.Length` originate from user-space. These values are used later in the same function:

```
if (queue->IsFilterDevice
    && !item->Write
    && item->OriginalLength > 0
    && (item->OriginalLength & (ENCRYPTION_DATA_UNIT_SIZE - 1)) == 0
    && (item->OriginalOffset.QuadPart & (ENCRYPTION_DATA_UNIT_SIZE - 1)) != 0)
{
    byte *buffer;
    ULONG alignedLength = item->OriginalLength + ENCRYPTION_DATA_UNIT_SIZE;
    LARGE_INTEGER alignedOffset;
    alignedOffset.QuadPart = item->OriginalOffset.QuadPart & ~((LONGLONG)
        ENCRYPTION_DATA_UNIT_SIZE - 1);

    buffer = TCalloc (alignedLength);
```

If `item->OriginalLength == 0xFFFFFE00`, the addition with `ENCRYPTION_DATA_UNIT_SIZE` will overflow and `alignedLength` will become zero (0) and an empty chunk will be allocated by the `TCalloc()` call. The null-sized read request is then passed down the stack. Assuming the lower-level driver graciously handles zero-size reads, the code will then read outside the buffer when it tries to copy `item->OriginalLength` bytes from the attacker-controlled buffer:

```
memcpy (dataBuffer, buffer + (item->OriginalOffset.LowPart &
(ENCRYPTION_DATA_UNIT_SIZE - 1)), item->OriginalLength);
```

Since the destination `dataBuffer` is also mapped in user-space (via MDL), this could result in information disclosure.

EXPLOIT SCENARIO: An attacker submits a read request to the TrueCrypt Windows kernel driver that overflows the unsigned integer.

SHORT TERM SOLUTION: Ensure all values submitted from user-space are sanitized before using them.

LONG TERM SOLUTION: Deploy proper integer overflow / underflow checks throughout the TrueCrypt source code. For Windows, the IntSafe library is a convenient way to do this.

9. MountVolume() device check bypass

Class: Data Validation **Severity:** Informational **Difficulty:** Low

FINDING ID: iSEC-OCAP-9

TARGETS: `VolumeThreadProc()` in `Ntdriver.c`

DESCRIPTION: The above function contains the following code:

```
if (memcmp (pThreadBlock->mount->wszVolume, WIDE ("\\Device"), 14) != 0)
{
    wcscpy (pThreadBlock->wszMountVolume, WIDE ("\\??\\"));
    wcsncat (pThreadBlock->wszMountVolume, pThreadBlock->mount->wszVolume,
        sizeof (pThreadBlock->wszMountVolume) / 2 - 5);
    bDevice = FALSE;
}
else
{
    pThreadBlock->wszMountVolume[0] = 0;
    wcsncat (pThreadBlock->wszMountVolume, pThreadBlock->mount->wszVolume,
        sizeof (pThreadBlock->wszMountVolume) / 2 - 1);
    bDevice = TRUE;
}
```

`pThreadBlock->mount->wszVolume` is directly supplied by the user-mode caller through the IRP[10]. If this string starts with "`\\device\\`" (in lower case), the code will not treat this as a device and `bDevice` will be `FALSE`. This will result in an unintended code path being followed in the function `TCOpenVolume()`.

SHORT TERM SOLUTION: Alter this check to be more robust. For example, it could use a case insensitive string comparison to prevent the above bypass.

LONG TERM SOLUTION: The short term solution is sufficient to correct this issue.

[10] http://msdn.microsoft.com/en-us/library/windows/hardware/ff550694(v=vs.85).aspx

10. GetWipePassCount() / WipeBuffer() can cause BSOD

Class: Denial of Service **Severity:** Informational **Difficulty:** Medium

FINDING ID: iSEC-OCAP-4

TARGETS: `GetWipePassCount()` and `WipeBuffer()` in `Wipe.c`

DESCRIPTION: The above functions contain the following code:

```
default:
    TC_THROW_FATAL_EXCEPTION;
```

Where `TC_THROW_FATAL_EXCEPTION` is defined as:

```
#elif defined (TC_WINDOWS_DRIVER)
#      define TC_THROW_FATAL_EXCEPTION KeBugCheckEx (SECURITY_SYSTEM, __LINE__, 0,
0, 'TC')
```

If a user-mode caller submits a wipe algorithm to the driver which is not among the defined cases, the code will execute `TC_THROW_FATAL_EXCEPTION` and produce a Blue Screen of Death. This behavior can be triggered by submitting the IOCTL `TC_IOCTL_BOOT_ENCRYPTION_SETUP` or `TC_IOCTL_START_DECOY_SYSTEM_WIPE` to the TrueCrypt driver. It should be noted that both these IOCTLS require administrative privileges to call.

SHORT TERM SOLUTION: Alter the error handling to produce a more appropriate and actionable error message.

LONG TERM SOLUTION: TC_THROW_FATAL_EXCEPTION is used in several places in the driver. Investigate if better error handling would be prudent in some places.

11. EncryptDataUnits() lacks error handling

Class: Error Reporting **Severity:** Informational **Difficulty:** High

FINDING ID: iSEC-OCAP-10

TARGETS: `EncryptDataUnits()` in `Crypto.c`

DESCRIPTION: The `EncryptDataUnits()` function is called from both the bootloader and the Windows kernel driver to encrypt data. This function is always expected to succeed and therefore does not return any kind of status value. If the encryption fails for any reason, the caller will still write the data, resulting in unencrypted data being written to disk.

Exploit Scenario: Unexpected operating system or hardware conditions, such as failing RAM or low-memory situations, cause the `EncryptDataUnits()` function to fail, resulting in unencrypted data being written to disk.

SHORT TERM SOLUTION: Bear this risk.

LONG TERM SOLUTION: Consider re-designing and re-implementing this functionality to be more robust. For example, more security conscious users could be allowed to set a flag that forces each encrypted block to be verified, with appropriate error values returned in case of failure.

4 Appendices

A Example issues in the bootloader decompressor

iSEC identified several issues in the bootloader decompressor. For an overview of these issues, refer to iSEC-OCAP-5.

A.1 Out of bounds read in `stored()`

The `stored()` function operates on a byte stream read from disk. The amount of bytes to operate on is read from the input byte stream. Once read, this amount is verified against the size of the output buffer and is then used to copy bytes from input to output:

```
len = s->in[s->incnt++];
len |= s->in[s->incnt++] << 8;
…
if (s->out != NIL) {
    if (s->outcnt + len > s->outlen)
        return 1;                           /* not enough output space */
    while (len--)
        s->out[s->outcnt++] = s->in[s->incnt++];
}
```

If `len` is smaller than the size of the output buffer but larger than the input buffer, the line copying the bytes will read outside the `s->in` array.

This issue can be triggered with the input `\xf8\x7f\x00\x80\xff`, which requires the output buffer to be at least 0x7f bytes.

A.2 Out of bounds write in `construct()`

The following code can be found in the function `Dynamic()`:

```
symbol = decode(s, &lencode);
if (symbol < 16)                /* length in 0..15 */
    lengths[index++] = symbol;
…
/* build huffman table for literal/length codes */
err = construct(&lencode, lengths, nlen);
```

The function `decode()` can return -9 as an error value. This error condition is never checked for, and the value -9 is instead used as a symbol value. This symbol value is later used in two different places in the function `construct()`:

```
for (symbol = 0; symbol < n; symbol++)
    (h->count[length[symbol]])++;
…
for (symbol = 0; symbol < n; symbol++)
    if (length[symbol] != 0)
        h->symbol[offs[length[symbol]]++] = symbol;
```

The first use results in memory at `h->symbol[-9]` being increased `n` times, while the second use results in using whatever is at `offs[-9]` as index into `h->symbol[]`.

This issue can be triggered with the input `\x14\x00\x00\x00`, as long as the input buffer is large enough to not trigger the following issue.

A.3 Out of bounds read in `decode()`

With a legal input stream, `decode()` can run out of buffer before it finds a symbol to decode. This will trigger an out of bounds read in the below line:

```
bitbuf = s->in[s->incnt++];
```

Since `s->incnt` is increased in each iteration (without any check to make sure it is in bounds) it will result in an out of bounds read once the index is larger than the input buffer.

This issue can be triggered with the input `\x14\x00\x00\x00`.

B Code quality issues in TrueCrypt

During the review of the TrueCrypt bootloader and driver for Windows, iSEC identified a number of issues in the source that affected overall quality and maintainability of the code base without directly leading to security problems. There is a risk that some of these issues could result in security vulnerabilities if the code is altered. These should therefore be addressed as a Defense in Depth measure.

B.1 Signed / unsigned mismatches

Description

In both the bootloader and driver source code, there is a mix between signed and unsigned integer types. This does not directly result in any security issues, but it can have unintended consequences. We include two (2) examples of signed / unsigned mixing in TrueCrypt.

`ProcessMainDeviceControlIrp()` in `NtDriver.c`:

```
typedef struct
{
        …
        unsigned __int64 diskLength;

typedef struct EXTENSION
{
        …
        __int64 DiskLength;

case TC_IOCTL_GET_VOLUME_PROPERTIES:
…
prop->diskLength = ListExtension->DiskLength;
```

`ProcessMainDeviceControlIrp()` in `NtDriver.c`:

```
case TC_IOCTL_IS_DRIVER_UNLOAD_DISABLED:
    if (ValidateIOBufferSize (Irp, sizeof (int), ValidateOutput))
    {
        LONG deviceObjectCount = 0;

        *(int *) Irp->AssociatedIrp.SystemBuffer = DriverUnloadDisabled;

        if (IoEnumerateDeviceObjectList (TCDriverObject, NULL, 0,
            &deviceObjectCount) == STATUS_BUFFER_TOO_SMALL && deviceObjectCount >
            1)
        *(int *) Irp->AssociatedIrp.SystemBuffer = TRUE;
```

A telling example of the risks of mixing signed and unsigned integer types is the chunked encoding integer overflow in nginx[11]. In this example, attackers are able to gain arbitrary code execution. Please see http://www.vnsecurity.net/2013/05/analysis-of-nginx-cve-2013-2028/ for an in-depth discussion of this issue.

[11] http://nginx.org/

Recommendation

Consider making integer variable types consistent wherever possible, with unsigned types typically being the safer choice. If a specification forces the use of a specific type for certain values, make sure these are explicitly converted (if needed) in a safe manner before being operated on.

B.2 Inconsistent integer variable types

Description

In both the bootloader and driver source code, integers of different sizes are used and assigned between one another without giving care to potential conversion issues. Again, the examples below have not directly resulted in any security issues but could have unintended consequences.

`AskPassword()` in `BootMain.cpp`:

```
typedef struct
{
    unsigned __int32 Length;
…
size_t pos = 0;
…
password.Length = pos;
```

`ProcessMainDeviceControlIrp()` in `Ntdriver.c`:

```
typedef struct _DEVICE_OBJECT {
  …
  LONG                        ReferenceCount;

case TC_IOCTL_GET_DEVICE_REFCOUNT:
    if (ValidateIOBufferSize (Irp, sizeof (int), ValidateOutput))
    {
        *(int *) Irp->AssociatedIrp.SystemBuffer = DeviceObject->ReferenceCount;
```

Recommendation

Consider making integer variable types consistent wherever possible. For Windows driver code, this is usually types such as ULONG, LONG etc.

B.3 Lack of integer overflow protections / checks

Description

In both the bootloader and driver source code, arithmetic operations are performed on untrusted data that has not been fully verified. Additionally, no care is taken to prevent integers from overflowing or underflowing when these operations are performed. This results in several arithmetic issues throughout the source code, most of which are harmless. However, if the code is changed or someone identifies a technique to exert more control over the behavior, this could result in a security issue. Below follows an example of arithmetic issues in TrueCrypt.

`DispatchControl()` in `VolumeFilter.c`:

```
switch (irpSp->Parameters.DeviceIoControl.IoControlCode)
{
    case IOCTL_DISK_IS_WRITABLE:
    {
        …
        ++HiddenSysLeakProtectionCount;
```

By calling this functionality multiple times, `HiddenSysLeakProtectionCount` can be made to overflow and become zero (0) again.

Recommendation

Deploy a library that helps perform arithmetic operations in a secure manner, such as IntSafe. This will help preventing any integer overflows / underflows from causing issues.

B.4 Use of deprecated, insecure string APIs

Description

The code in `Ntdriver.c`, as well as supporting files, makes heavy use of APIs considered insecure and which are now deprecated. This includes functions such as:

```
wcscpy()
wcscat() / wcsncat()
sprintf()
…
```

Recommendation

The Visual Studio CRT contains a new family of string handling functions (`StringCch`) that are much more secure. These secure functions have effectively become the replacement for the more insecure string handling functions

B.5 Suppression of compiler warnings

Description

The Microsoft compilers used to build TrueCrypt will warn against some of the issues mentioned above. However, in both the bootloader and Windows kernel driver, some of these warnings have been suppressed. Some are suppressed with `#pragma` in the code, while others are suppressed in the build scripts. This results in the code compiling without warnings, even though it contains issues that should be corrected.

`Platform.h` used by the bootloader contains the following line:

```
#pragma warning (disable: 4018 4102 4704 4769)
```

`BuildDriver.cmd` used to build the driver contains the following line:

```
set TC_C_DISABLED_WARNINGS=-wd4057 -wd4100 -wd4127 -wd4152 -wd4201 -wd4701 -wd4702
-wd4706
```

Refer to http://msdn.microsoft.com/en-us/library/8x5x43k7.aspx for a reference of Microsoft compiler warning values.

Recommendation

Consider removing the suppressions from both the code and the build environment. Instead, alter the source code to ensure it compiles without warnings.

B.6 Use of Zw APIs

Description

The TrueCrypt Windows kernel driver makes heavy use of the `Zw` family of system calls. When a `Zw` function is called, the originating caller is set to kernel, which causes the system to bypass all ACL checks. If a user-mode caller can supply arbitrary arguments (via the IRP) to any of the `Zw` calls, there is a major risk that a security issue will follow.

Recommendation

Assess all calls made to `Zw` functions from within the TrueCrypt Windows kernel driver. Replace all those that can be replaced with a call to the corresponding `Nt` function. For all others, filter arguments originating from user-mode.

B.7 Other minor issues

Description

The TrueCrypt Windows driver code contains a handful of other minor non-security issues that do not fit under any of the other categories.

`Ntdriver.c`, line 817:

```
if (ValidateIOBufferSize (Irp, sizeof (LONG), ValidateOutput))
{
    LONG tmp = VERSION_NUM;
    memcpy (Irp->AssociatedIrp.SystemBuffer, &tmp, 4);
```

The call to `memcpy()` should be:

```
memcpy (Irp->AssociatedIrp.SystemBuffer, &tmp, sizeof(LONG));
```

`EnsureNullTerminatedString()` in `Ntdriver.c`:

```
void EnsureNullTerminatedString (wchar_t *str, size_t maxSizeInBytes)
{
    ASSERT ((maxSizeInBytes & 1) == 0);
    str[maxSizeInBytes / sizeof (wchar_t) - 1] = 0;
}
```

`maxSizeInBytes` is never verified to be larger than zero (0), and since the `ASSERT()` is only compiled in for debug builds, it is never verified to be an even number. All calls to this function are currently well controlled.

`MountDrive()` in `DriveFilter.c`:

```
if (NT_SUCCESS (status) && BootArgs.BootDriveSignature != *(uint32 *) (mbr +
    0x1b8))
```

The value `0x1b8` should be replaced with the help of a `#define` with a descriptive name.

`MOUNT_LIST_STRUCT` in `Apidrvr.h`:

```
typedef struct
{
    unsigned __int32 ulMountedDrives;   /* Bitfield of all mounted drive letters */
    wchar_t wszVolume[26][TC_MAX_PATH]; /* Volume names of mounted volumes */
    unsigned __int64 diskLength[26];
    int ea[26];
    int volumeType[26]; /* Volume type (e.g. PROP_VOL_TYPE_OUTER,
        PROP_VOL_TYPE_OUTER_VOL_WRITE_PREVENTED, etc.) */
} MOUNT_LIST_STRUCT;
```

The value `26` should be replaced with the help of a `#define` with a descriptive name. Additionally, this should be synchronized with the `#define MAX_MOUNTED_VOLUME_DRIVE_NUMBER` in `Common.h`

B.8 General readability issues

Description

In addition to the implementation issues above, the code is difficult to read and follow in some places. This makes it hard to assess the source code for bugs and perform general maintenance. The following is a list of a few things found in the TrueCrypt source code that decrease readability and ease of understanding, together with examples.

- **"Compressed" code constructs**
  A common trait among C developers is to write so called one-liners:
  `IoGetNextIrpStackLocation (irp)->FileObject = fileObject;`
  While constructs like these are very size efficient, they do decrease readability and make maintenance and error handling harder. Additionally, the extra space between the function name and the parenthesizes will throw off some readers.

- **Mixing of code and variables**
  Mixing code into variable declarations can make the code harder to spot:
  ```
  BiosResult WriteEncryptedSectors (uint16 sourceSegment, uint16 sourceOffset,
  byte drive, uint64 sector, uint16 sectorCount)
  {
      BiosResult result;
      AcquireSectorBuffer();
      uint64 dataUnitNo;
  ```
  Having a clear separation between variable declarations and the code using the variables increases readability.
- **Lack of comments**
  One of the better ways of increase understanding, readability, and maintainability of a large source code project is to have helpful comments. This is especially true for a project being worked on by a multi-person team. The TrueCrypt source code largely lacks comments, leaving the reader to work out the detailed functionality intended by the author.
- **Mixing of user-mode and kernel-mode functions with the same name**
  In part of the TrueCrypt source code that is shared among components, multiple functions with the same name exist and are differentiated using `#ifdef`. For example, there are two (2) `ReadVolumeHeader()` functions, one that is called from the kernel driver and one that is called from the UI component. Functions such as these should be separated and named differently to avoid risk of confusion.
- **Overly long functions**
  While most functions in the TrueCrypt Windows kernel driver are designed to perform an atomic task, a few such as `TCOpenVolume()` (~700 lines), can be hard to follow. Refactoring such monolithic functions would greatly help readability.