Threat Models for Analyzing Plausible Deniability of Deniable File Systems

Michal Kedziora[1,*], Yang-Wai Chow[2] and Willy Susilo[2]

[1]*Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wroclaw, Poland*
[2]*Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong, Australia*
E-mail: michal.kedziora@pwr.edu.pl; {caseyc,wsusilo}@uow.edu.au
**Corresponding Author*

Received 15 August 2017; Accepted 22 October 2017;
Publication 27 November 2017

Abstract

Plausible deniability is a property of Deniable File System (DFS), which are encrypted using a Plausibly Deniable Encryption (PDE) scheme, where one cannot prove the existence of a hidden file system within it. This paper investigates widely used security models that are commonly employed for analyzing DFSs. We contend that these models are no longer adequate considering the changing technological landscape that now encompass platforms like mobile and cloud computing as a part of everyday life. This necessitates a shift in digital forensic analysis paradigms, as new forensic models are required to detect and analyze DFSs. As such, it is vital to develop new contemporary threat models that cater for the current computing environment that incorporates the increasing use of mobile and cloud technology. In this paper, we present improved threat models for PDE, against which DFS hidden volumes and hidden operating systems can be analyzed. In addition, we demonstrate how these contemporary threat models can be adopted to attack and defeat the plausible deniability property of a widely used DFS software.

Journal of Software Networking, 241–264.
doi: 10.13052/jsn2445-9739.2017.012

Keywords: Deniable file system, Hidden operating system, Plausibly deniable encryption, Veracrypt.

1 Introduction

The underlying notion of deniable encryption is to be able to decrypt a cipher text into two different plaintexts depending on the key that is provided [1]. The purpose of this is to protect against adversaries who can force a user to provide a password to decrypt the cipher text, as the password that is provided will only reveal the decoy message while the true message remains hidden. An additional requirement in Plausibly Deniable Encryption (PDE) is to guarantee that the adversary cannot detect the presence of a hidden message in the cipher text. This property is known as plausible deniability, as the existence of the hidden message cannot be proven.

A Deniable File System (DFS) is developed using a PDE scheme, as a file system where the existence of a portion of it can be hidden from view [2]. An example of such a system is where a person creates a regular (non-deniable) encrypted file system, which is protected by a password. Within this file system, the person can also create a DFS that is protected by a second password. This inner, DFS is referred to as a hidden volume, which is deniable because unless the person reveals the second password to an adversary, it should be impossible for that adversary to determine whether the regular encrypted file system contains an encrypted hidden volume [2].

DFSs can be used for a variety of different purposes. For example, a professional journalist or human rights worker operating in a region of conflict or oppression may need to hide sensitive data in a hidden file system. This is to protect the individual from severe punishment or retribution if the human rights violators were to discover that the individual has evidence of the atrocities or other sensitive data in their possession [2, 3]. On the other hand, DFSs can be a double-edged sword as it can be used by criminals or terrorists to hide secret data from the police or authorities, who may not be able prosecute the criminals due to being unable to prove the existence of the hidden data.

One of the currently used security models against which DFSs can potentially be secured was described in Czeskis et al. [2]. According to them, threat models against which hidden encrypted volumes can potentially be secured are based on three situations: one-time access, intermittent access and regular access. However, these models were proposed a number of years ago and there are several issues with them when applied to the modern day computing landscape, in which platforms like mobile and cloud computing

are a part of everyday life. This necessitates a shift in digital forensic analysis paradigms, as new forensic models are required to detect and analyze DFSs.

In practice, the security threat models of DFSs should closely relate to the digital forensic process. There are number of guidelines and procedures used to describe this process [4–6]. The emergence of ubiquitous mobile devices and operating systems with integrated backup functions, on-the-fly encryption, mobile and cloud integration, etc. has resulted in the traditional forensic model becoming increasingly obsolete. The live forensic approach was introduced as an alternative approach by adding live analysis to forensic procedures [7]. In addition, PDE on mobile devices is a growing area of research [3, 8, 9]. Considering the growing mobile and wireless environment, the previous threat models do not address the requirements of this emerging landscape and thus should be revisited with improvements.

This paper presents improved threat models for PDE, against which DFS hidden volumes and hidden operating systems can be analyzed. This is based on our work in Kedziora et al. [10]. In addition, we demonstrate how these contemporary threat models can be adopted to attack and defeat the plausible deniability property of VeraCrypt, which is a widely used DFS software [11, 12].

2 Background

2.1 Deniable File Systems

A Deniable File System (DFS) is one where the existence of a portion of the file system can be hidden from view [2]. DFSs are based on deniable encryption, which was first introduced by Canetti et al. [1]. Canetti et al. [1] proposed a shared-key encryption scenario where the sender and receiver share a random secret key to encrypt message, along with a fake shared key. This allows the encrypted message to be decrypted into two different plaintexts depending on which key is used. Plausible deniability is the property where one cannot prove the existence of a hidden message in the cipher text without being provided with the second key. In Plausible Deniable Encryption (PDE), the assumption is that it should be impossible to prove that a hidden message exists.

Since its inception, PDE have been adopted in a variety of encrypted file storage schemes. Oler and Fray [13] discussed a number of concepts of DFSs, including their advantages, drawbacks and use as a file system for storing sensitive data. Deniable cryptography has been used for cloud storage. The concept of deniable cloud storage includes the privacy of data even when

one's communication and storage can be opened by an adversary. This was introduced by Gasti et al. [14], in which they designed a sender-and-receiver deniable public-key encryption scheme and provided an implementation of a deniable shared file system for cloud storage. In addition, PDE schemes have also been devised to provide deniable storage encryption for mobile devices. Examples from the research community on implementing PDE on mobile platforms include Mobiflage [3], MobiHydra [8] and MobiPluto [9].

One of the most common DFS software that is used in practice is based on the TrueCrypt implementation [2]. TrueCrypt is an on-the-fly encryption application, which implements DFS as hidden volumes that reside within an encrypted volume. While the TrueCypt project was discontinued in 2014, alternatives exist. For example, VeraCrypt is the most popular DFS software to date. VeraCrypt is an open source software used for on-the-fly encryption [11, 15]. It implements PDE in the form of hidden volumes and hidden operating systems. Its process is user transparent in that data is encrypted right before it is saved, and decrypted right after it is being loaded, without any user intervention [11]. PDE software for encrypted and hidden volumes are also available on mobile devices using mobile applications such as Disk Decipher [16] and Crypto Disks [17].

2.2 Threat Models for DFSs

Threat models for DFSs were described in Czeskis et al. [2]. In their work, they proposed threat models against which hidden encrypted volumes can potentially be secured. These are based on one of three situations:

- The first scenario is the *One-Time Access* scenario. This is when the attacker has only one copy of the disk image containing a DFS volume. This is the worst-case scenario. An example of this is when the police seize a device and make a binary copy of its data.
- Their second model is *Intermittent Access.* According to Czeskis et al. [2], this is when an attacker has several copies of the evidence volume, taken at different times. An example is when border guards make a copy of a person's device every time the person enters or leaves the country.
- The third model is *Regular Access.* This is when an attacker has several copies of the evidence data made at short intervals. For example, when the police secretly enter a person's apartment every day while the person is away and make a copy of the device's contents each time.

There are several issues with these models when applied to the modern day computing environment. The purpose of One-Time Access was to focus on

a situation where there is a chance to discover information about a DFS via analysis of its algorithm and implementation. In addition, it was meant to deal with a situation when a single binary copy of a hard disk containing a DFS was seized and analyzed. However, this situation rarely occurs as investigators nowadays can often take a snapshot of the device's RAM before it is shut down. In addition, current operating systems have features such as automatic backup functions for important files. As such, a copy of a hard disk typically contains multiple archived copies of DFS volumes. In common forensic investigations, the One-Time Access model is severely affected if several copies of the DFS volume exists, as this encroaches on the Intermittent or Regular Access models.

Furthermore, the Intermittent and Regular Access models both rely on access to multiple copies of the data. The difference between them is based on the number of copies and the intervals at which these copies were made. The purpose of these models is based on the ability to analyze changes both in the DFS and any side channel leaks that these changes may create. However, the interval in which copies are seized versus the number of copies, does not play a significant role in investigations to distinguish between these models. In addition, with the increasing number of copies and automatic backups, which is characteristic of the modern computing environment, this severely muddles the distinction between the Intermittent and Regular Access models.

Therefore, the inconsistencies with these traditional threat models result in an inability to practically employ these models in the current increasingly diverse computing environment. Moreover, part of the deficiency also lies in the fact that the traditional models fail to capture the live forensic approach, which has become the commonplace when handling live access to cloud and mobile data. The current forensic shift is to analyzes live running systems remotely without shutting them down. This is not captured in the current threat models and therefore misses important attack vectors on DFSs.

3 Improved Threat Model

This paper presents improved threat models for the security of DFSs, which addresses the weaknesses in the models described in Czeskis et al. [2]. This section discusses purpose of the proposed threat models and presents analysis on their significance to accommodate the current increasingly diverse computing environment, comprising of ubiquitous systems like mobile and cloud computing with their associated synchronization and automatic backup features. The main drawback of the traditional models is that the One-Time Access model often encroaches on the Intermittent or Regular Access

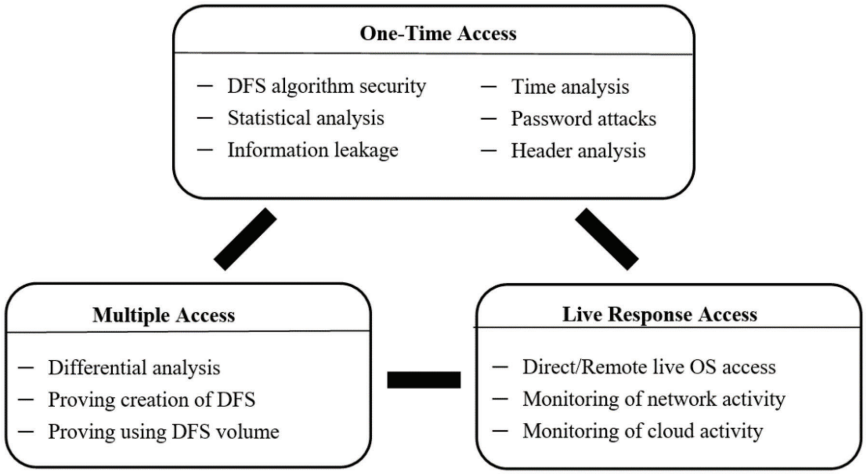


Figure 1 Threat models and attack vectors on Deniable File Systems (DFSs).

models. Furthermore, there is little to distinguish between the forensic analysis methods and attack vectors that can be used for the Intermittent and Regular Access models.

As such, the proposed approach amalgamates the One-Time Access model with aspects of the Intermittent and Regular Access models, to form a baseline for single system access. This is separate from the Multiple Access model which incorporates techniques like differential analysis. A third model is proposed based on the live forensic approach, which we call Live Response Access. This not only addresses live forensics, but is also associated with new types of DFS attacks based on cloud and network integrity of today's computer systems. Figure 1 depicts the proposed threat models. The proposed model incorporates the One-Time Access, Multiple Access and Live Response Access models along with their associated attack vectors respectively.

3.1 One-Time Access

The One-Time Access scenario in the proposed model is where an investigator is able to access one, or more copies, of a device containing only one copy of the DFS encrypted container. The most conservative variant of this model is when an investigator is able to seize and analyze forensic evidence of a binary image of an encrypted DFS volume. Two common situations are, for example, obtaining a binary copy of a hard drive encrypted with a DFS implementation like TrueCrypt/VeraCrypt, and retrieving a logical copy of the DFS encrypted

container from a backup system. In either of these situations, the investigator's options are limited to analyzing the cover volume or the encrypted container itself.

The security of this is based on the cryptographic algorithm and the assumption that it can be formally and mathematically proven. However, in practice, DFSs are usually seized as a container file from a complex operating system. This results in the possibility of new attack vectors, in addition to the problem of detecting the DFS itself, as DFS implementations use encryption to hide deniable data together with encrypted cover data. Hence, all encrypted data found on an evidence device should be treated by the investigator as containing a DFS unless proven otherwise. While this problem is not commonly addressed in DFS related papers, it is very important from a forensic investigator's point of view. This was not only presented in previous work [18, 19], but also confirmed in Davies [20], where initial detection techniques are based on statistical detection of volumes by randomness testing. Statistical tests based on entropy, chi-square, arithmetic mean, Monte Carlo for Pi and serial correlation coefficient can be used.

The main threat vector for DFS security in the One-Time Access model is information leakage, which can compromise covert and hidden volumes. Information leakage through the operating system was introduced in Czeskis et al. [2], where they gave an example of shortcut files that can point to data on the hidden volume, or copies of hidden volume files saved in unencrypted area of disk, thus compromising its presence. The second main vector is in locating keys and password attacks against DFS. DFSs based on TrueCrypt/VeraCrypt are only as strong as its password, which is a practical problem when many users do not comply with secure password usage policies. Furthermore, there are methods of obtaining passwords from the memory of a running DFS volume.

In situations where an investigator can access more than one copy of a DFS volume [21], and in situations where an investigator can interact with a running system to find cryptographic keys should be excluded from the One-Time Access threat model. This is because the former scenario is captured in the Multiple Access model, while in the later is modelled in the Live Response Access model, which are discussed in the sections to follow.

3.2 Multiple Access

A Multiple Access scenario is where an investigator has multiple device images containing multiple hidden encrypted containers. The main threat to

DFSs in this case is differential analysis of hidden volumes, which can result in the ability to attack the plausible deniability attribute. This issue was first raised by Czeskis et al. [2], where they highlighted that if disk snapshots can be obtained at close enough intervals, the existence of any deniable files will be obvious, since seemingly random bytes on the hard drive will change. A practical implication of this was presented in Hargreaves and Chivers [21], where they described how hidden encrypted volumes can be detected and how their sizes can be estimated. In addition, research on detecting the creation of a DFS inside an encrypted container was presented in Jozwiak et al. [15].

The Multiple Access model also involves the situation where more than one copy of a hidden volume can be retrieved from only one seized disk image. An example of this was presented by Hargreaves and Chivers [21], where they managed to obtain multiple copies of an encrypted container using the Shadow Copy function in the Windows Vista, Windows 7 and Windows 10 operating systems. Shadow Copy extends the Restore Point feature of Windows XP. The Shadow Copy feature is important for finding forensic artifacts during investigations as demonstrated by Purcell and Lang [22]. This situation is common in forensic investigations due to the standard usage of automatic backup functions integrated in modern operation systems including Shadow Copy for Microsoft Windows and Time Machine for MacOS. The emergence of mobile and cloud computing with integrated backup also produces a source for obtaining multiple copies of DFS containers.

3.3 Live Response Access

We define a new model for capturing the scenario where investigators have live access to data through a network. We refer to this as the Live Response Access model. Three main example scenarios for this model are where an investigator/attacker has:

- Direct/remote live access to the hosting Operating System (OS) running a DFS volume
- Direct/remote live access to DFS based hidden OS
- Access to the network environment within which a hidden OS is running, or has access to the cloud application in which the hidden OS is connected

When Czeskis et al. [2] introduced their threat models against which a DFS could potentially be secured, forensics procedures typically involved the switching-off of computers and making a binary copy of the hard drive. Nowadays, much more effort is directed and focused towards live forensics,

whereby the main idea is to preserve volatile data that is mostly lost if a computer or mobile device were to be switched off [23]. Live response and memory analysis tools have the capabilities of collecting information from a variety of sources including network connections, opened ports and sockets, running processes, terminated processes, loaded Dynamic-Linked Libraries (DLLs), opened files, OS kernel modules, process dumps, and strings or user logs [24]. Each of these information sources can lead to compromising the presence of a DFS by identifying a hidden volume disk area.

Although most of these techniques can also be used in the One-Time Access model, as volatile forensic artifacts related to hidden DFS volumes can be found in temporary system files as swap or hibernation files, it is more appropriate to extend these to the Live Response Access model. This is because it can lead to the scenario where an investigator has access to the host system, a common situation nowadays, which can generate new approaches and threats to DFS security.

A scenario that was ignored in previous models is securing a DFS when an investigator or an attacker has access to the hidden volume or the hidden OS while it is running. The reason why this scenario was ignored is because a DFS is assumed to have secure encryption. However, this situation has changed with the hidden OS option when using an implementation like TrueCrypt/VeraCrypt. As such, the Live Response Access model embraces the scenario where investigators can remotely use live response tools to directly access a running DFS OS. In practice, this can be achieved using remote access via software like Team Viewer, VNC, Windows Remote Desktop or just physical access to the device. Another scenario is the running of the hidden OS in a networked environment with the need to connect to third party mobile and cloud applications. This results in new possibilities for detecting the presence of a DFS based on live access to the DFS that is currently in use.

4 Defeating Plausible Deniability

In this section, we demonstrate how the proposed threat models can be used in practice to defeat plausible deniability of a VeraCrypt hidden Operating System (OS) [12]. A hidden OS is an operating system installed in an encrypted hidden volume, using software such as VeraCrypt. This feature was implemented in TrueCrypt/VeraCrypt software as an extension of DFSs [25].

4.1 VeraCrypt Hidden Operating System

VeraCrypt uses XTS mode for encrypting partitions, drives and virtual volumes [11]. This mode of operation is described by Equation (1), where $\otimes$ denotes multiplication of two polynomials over the binary field GF(2) modulo $x^{128}+x^7+x^2+1$; $\wedge$ denotes an XOR operation; K_1 is the encryption key; K_2 is the secondary encryption key; *i* is the cipher block index within a data unit; *n* is the data unit index within the scope of K_1; and *a* is the primitive element of Galois Field (2^{128}) that corresponds to polynomial *x* [11]. This implies that a change in one bit of the plaintext will result in a change to the entire 8-bytes (128 bits) data block of the encrypted volume.

$$C_i = E_{K1}(p_i^{\wedge}(E_{K2}(n) \otimes a^i)) \,^{\wedge}(E_{K2}(n) \otimes a^i) \tag{1}$$

The VeraCrypt documentation provides a guide on how to encrypt a hidden OS [11]. A practical implementation consists of two partitions and a boot loader residing in the first track of a system drive (or a VeraCrypt RescueDisk). However, this is not a smart solution as the unencrypted boot loader will indicate that the drive is encrypted by VeraCrypt. To overcome this issue there is an option to create a VeraCrypt rescue disk containing the boot loader, as depicted in Figure 2. This will provide plausible deniability as a decoy OS can be created. Obviously, the system installed on the first partition must not contain any sensitive files.

The second partition is also encrypted and can be mounted by the user upon supplying the second password. The outer volume contains an integrated hidden volume within which the hidden OS is installed. Existence of the hidden volume, which is a DFS, cannot be proven via One-Time Access methods (previously described in Section 3.1). To access the hidden OS, the user must provide the valid password that is different from the decoy OS volume's password. The boot loader will first try to decrypt the decoy OS's header,

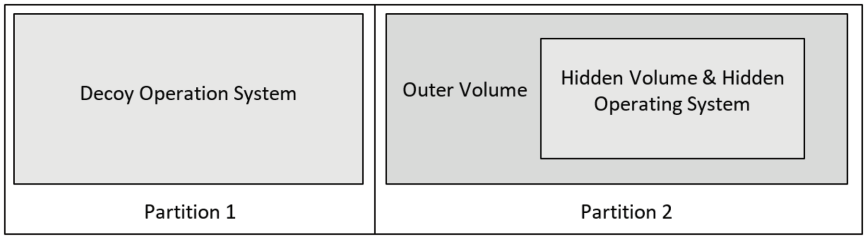


Figure 2 Example layout of a drive containing a VeraCrypt hidden OS.

and after it is unsuccessful, it will then attempt to decrypt the hidden OS's header. What is important is that when running, the hidden OS will appear to be installed on the same partition as the decoy OS. All read/write operations will be transparently redirected from the system partition to the hidden volume inside the outer volume. The VeraCrypt documentation asserts that neither the OS nor any application programs will know that all data is essentially written to and read from the hidden volume [11]. We demonstrate that the above statement is not entirely true, as the presence of the hidden OS can in fact be revealed.

4.2 Test Environment

A test environment was created using Oracle Virtual Box version 5.1.12. A hard drive image size of 50 GB was created. However, since the virtual box operates using the Virtual Disk Image (VDI) file format with included metadata, its image had to be converted to a binary RAW format before analysis using computer forensic tools. Both the decoy and hidden OS (MS Windows 10) where installed using VeraCrypt 1.19. The designed layout of the partitions is depicted in Table 1.

The first partition, /dev/sda1, was for the Windows Recovery Environment (WinRE) and was unencrypted. The second partition, /dev/sda2, was the one on which the decoy operating system was installed; the whole partition was encrypted. /dev/sda3 was the extended partition that hosts the /dev/sda5/ partition, which was the completely encrypted outer volume; the hidden OS was installed within this partition. As the hidden OS was contained within the encrypted hidden volume, which was located inside the encrypted outer volume, plausible deniability necessitates that it should be impossible to prove the existence of this hidden OS. However, in the next section, we show that plausible deniability of the VeraCrypt hidden OS is not met even in the simplest threat model scenario.

Table 1 Layout of the partitions in the test environment

Partition	Starting Sector	Last Sector	Size (MB)
/dev/sda1	2048	1026047	500
/dev/sda2	1026047	43530239	20270
/dev/sda3	43532225	104855551	29240
/dev/sda5	43532288	104855355	29240
Unallocated	104855552	104857599	1

4.3 Encrypted Drive Analysis

First, we investigated the possibility of defeating plausible deniability of a VeraCrypt hidden OS under the most basic thread scenario, i.e. the One-Time Access scenario. An example of such a scenario is when Alice's computer is seized by police, who force Alice to reveal the password of the encrypted partitions. Alice reveals the password for the decoy OS and for the outer volume. According to the plausible deniability attribute of the VeraCrypt hidden OS, the police should not be able to prove that Alice has a hidden OS installed on the computer, as it is stored in an encrypted hidden volume inside the encrypted outer volume.

A VeraCrypt hidden OS requires a special uncommon disk layout consisting of at least two partitions that are both completely encrypted. This information, in conjunction with the fact that VeraCrypt is installed on the computer under investigation, can potentially raise the suspicion of the police to the presence of a hidden OS. Nevertheless, this can reasonably be explained by Alice as the need to separate the system and documents into separate partitions. However, any solid indication that a hidden OS is installed on the computer under investigation is sufficient to defeat plausible deniability.

We conducted randomness testing to check for artifacts in the outer volume. The reason for this is because if a hidden OS is running inside a completely encrypted hidden volume that is located within an outer volume, which is also completely encrypted, no pseudorandom anomalies should be found. When we performed entropy analysis on the outer volume, it showed that most of the examined data had values between 7.9978 and 7.9986, which represent expected values from correctly encrypted cipher text data. However, we were able to observe some unexpected values in specific sectors that were occupied by the outer volume. In particular, there were two areas which clearly showed significantly lower entropy values of 7.9966 and 7.997, as can be seen in the plot provided in Figure 3.

The first of these observed areas was located in sector number 61345696, and the second was located 45928448 bytes later in sector number 61435400. Both of these sectors are located within the /dev/sda5 partition, which was within the completely encrypted outer volume. The hidden volume hosting the hidden OS had a size of 42504191 sectors. This could infer that the lower entropy areas indicate the beginning and end of the hidden volume hosting the hidden OS. The presence of these lower entropy areas violates the plausible deniability of the existence of a VeraCrypt hidden OS.

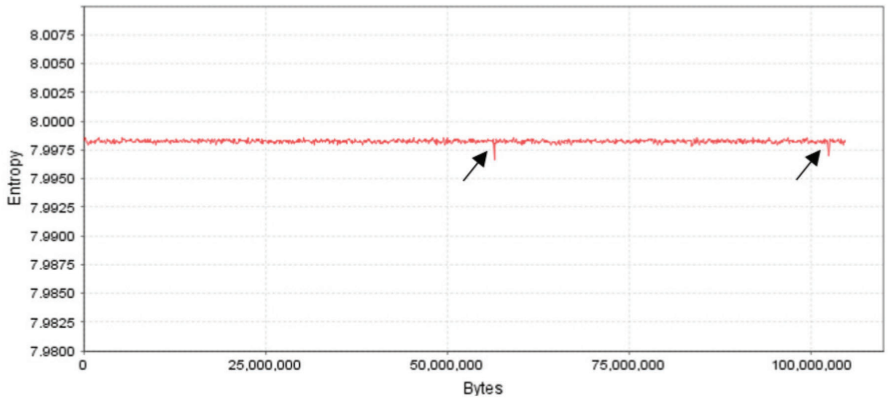


Figure 3 Areas with significantly lower entropy inside the outer encrypted volume.

Both areas are exactly 512 bytes in length and consist of "00" bytes and strings, and the path to the "\windows\system32\winload.exe" file, refer to Figure 4. Cross drive analysis showed that the second area correlates to the running of the hidden OS. Three bytes at offset 61435400 are altered every time the hidden OS is started. This is highlighted in Figure 4, the bytes 90 90 00 change to CD 1E 01 whenever the hidden OS is started. A VeraCrypt ciphertext block size is 16 bytes (128 bits), this indicates that this area is not overwritten by the VeraCrypt encryption algorithm.

```
3E5C0FF8 AC 9E E9 D2 3D 5E 89 F4  03 00 00 00 10 00 00 00  00 00 01 00 40 01 00 00  ¬žéÒ=^‰ô............@...
3E5C1010 A7 D5 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  §Õ..............................
3E5C1028 40 00 00 00 01 00 00 00  02 00 00 00 01 00 00 00  E1 07 01 00 1F 00 0F 00  @...............á.......
3E5C1040 0B 00 10 00 00 00 07 00  01 00 00 00 00 00 00 00  A7 D5 00 00 00 00 00 00  ................§Õ......
3E5C1058 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  78 00 00 00 01 00 00 00  ................x.......
3E5C1070 02 00 00 00 11 00 00 00  46 C8 8A D7 BB E1 E6 11  82 E9 C7 85 AF 96 8B 25  ........FÈŠ×»áæ.‚éÇ…¯–‹%
3E5C1088 00 00 00 00 5C 00 57 00  69 00 6E 00 64 00 6F 00  77 00 73 00 5C 00 73 00  ....\.W.i.n.d.o.w.s.\.s.
3E5C10A0 79 00 73 00 74 00 65 00  6D 00 33 00 32 00 5C 00  77 00 69 00 6E 00 6C 00  y.s.t.e.m.3.2.\.w.i.n.l.
3E5C10B8 6F 00 61 00 64 00 2E 00  65 00 78 00 65 00 00 00  90 90 00 00 00 00 00 00  o.a.d...e.x.e...........
3E5C10D0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  78 00 00 00 01 00 00 00  ................x.......
3E5C10E8 02 00 00 00 11 00 00 00  46 C8 8A D7 BB E1 E6 11  82 E9 C7 85 AF 96 8B 25  ........FÈŠ×»áæ.‚éÇ…¯–‹%
3E5C1100 00 00 00 00 5C 00 57 00  69 00 6E 00 64 00 6F 00  77 00 73 00 5C 00 73 00  ....\.W.i.n.d.o.w.s.\.s.
3E5C1118 79 00 73 00 74 00 65 00  6D 00 33 00 32 00 5C 00  77 00 69 00 6E 00 6C 00  y.s.t.e.m.3.2.\.w.i.n.l.
3E5C1130 6F 00 61 00 64 00 2E 00  65 00 78 00 65 00 00 00  00 00 00 00 00 00 00 00  o.a.d...e.x.e...........
3E5C1148 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C1160 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C1178 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C1190 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C11A8 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C11C0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C11D8 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ........................
3E5C11F0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  AF 08 40 D8 66 3A 37 E5  ..............¯.@Øf:7å
```

Figure 4 Lower entropy areas.

In summary, an investigator can easily find these areas in a One-Time Access threat model scenario. The presence of these areas is correlated with the existence of a hidden OS, and thus violates the plausible deniability attribute of a VeraCrypt hidden OS. Furthermore, if an investigator is able to compare this area with binary snapshots taken over an interval of time (i.e. in the case of a Multiple Access model), this can provide strong evidence as to the running of a hidden OS on the computer.

4.4 Cross Drive Analysis

In this section, we demonstrate a method of defeating plausible deniability of a VeraCrypt hidden OS in the case of a Multiple Access threat model. This scenario assumes that an investigator is in possession of multiple binary copies of Alice's computer hard drive that were taken over several time intervals during which Alice was using either the decoy OS or the hidden OS. This method has previously been used in DFSs for detecting the existence of TrueCrypt hidden volumes on a drive under investigation [21]. Our research adopts this method for detecting the presence of a VeraCrypt hidden OS.

First, we split the binary images of the investigated drives into 1000 MB blocks. Then the SHA-1 cryptographic hash of each block was computed. This was done under the assumption that this will help narrow down the analysis from a 50 GB image to smaller parts of the drive where data actually changes, which was true in the case of analyzing TrueCrypt hidden volumes [19]. It turns out that running a VeraCrypt OS's "on the fly" encryption (even when the OS is idle) writes large amounts of data, which distributes changes over the whole system partition. VeraCrypt statistics estimate that 17, 33, and 520 MBs of data written on an encrypted volume correspond to 1 minute, 2 minute and 5 minute intervals [11]. Analysis of the cryptographic hash function values clearly showed that mismatched blocks in the case of running the decoy OS are placed in the first half of the investigated drive image. This is in contrast to the running of the hidden OS, which changes only the second half of the drive image.

We performed a detailed comparison of changes in each corresponding data block, and a visual depiction of this is presented in Figure 5. In Figure 5, every rectangle represents a 1000 MB block of the binary image from the investigated drive (except for the last block which is 200 MB in size). The first block is on the upper left, while the last block is on the lower right. The data that changed during the running of the decoy and hidden OSs are depicted as the horizontal gray lines.

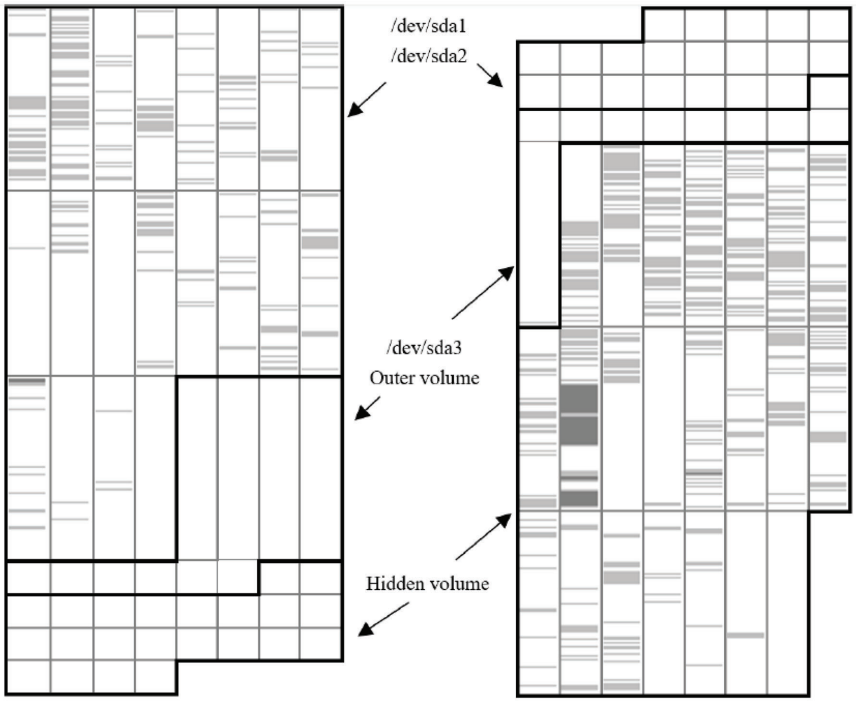


Figure 5 A visual depiction of the changes that VeraCrypt made to the volume while running the decoy OS (on the *left*) and the hidden OS (on the *right*).

The experiment started with the creation of the binary images of the investigated drive containing both the installed decoy and hidden OSs. Then, virtual machines were cloned, switched on and immediately turned off for the decoy OS and a second clone for the hidden OS. While running the decoy OS, only data on the second portion changed, whereas running the hidden OS only resulted in changes in the outer volume, which is located in the third partition. Analyzing the first change sector offset (62351360) and the last sector (103601344) allows for an estimation of the hidden OS partition size. In the case of the experiment, it was estimated as 19.7 GB, which compares favorably with the actual hidden OS partition size of 20.26 GB. It is assumed that a more accurate estimation can be made if we allowed the OSs to operate for some time, rather than simply switching it on and off.

In summary, this demonstrates that cross drive analysis can uncover evidence that a hidden OS is running on a drive under investigation, based on an analysis of changes made to the encrypted drive.

4.5 Other Attack Vectors

Hidden OSs are by design intended to ensure plausible deniability, especially in the case of a One-Time Access model. In the previous section, we demonstrated that they are vulnerable to Multiple Access attacks. In this section, we discuss attack vectors based on the Live Response Access scenario. This is based on the situation where an investigator has live access to the running hidden OS or to the network/cloud environment within which the hidden OS is operating. Our purpose is to reveal any information that can lead to proving that either a decoy or a hidden OS is running. Despite information provided in the VeraCrypt documentation that asserts that neither the OS nor any application programs will know that all data is essentially written to and read from the hidden volume [11], we discovered that even non-privilege level applications can reveal some information that can be used to detect a hidden OS.

Right after logging into the hidden OS, a pop-up message informing the user that "for security reasons, when a hidden operating system is running, local unencrypted file systems and non-hidden VeraCrypt volumes are mounted as read-only", which gives away the fact that the system is running a hidden OS. This pop-up message is shown in Figure 6. In addition, when

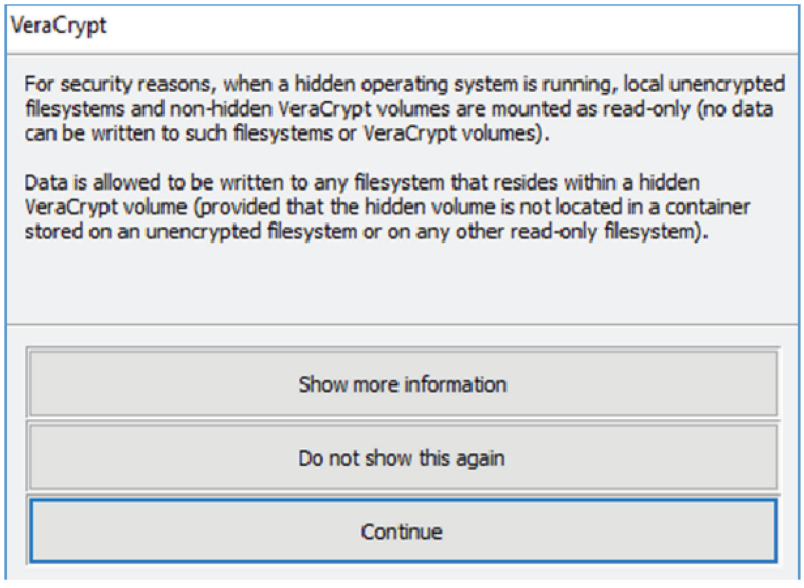
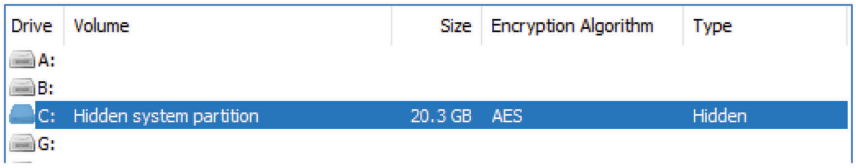


Figure 6 Pop-up message displayed while launching a VeraCrypt hidden OS.

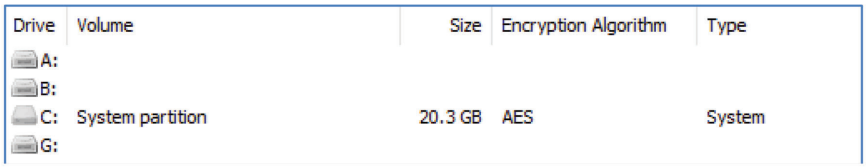
configuration files located in the %APPDATA%\VeraCrypt\folder from both the decoy and hidden OSs were compared, there is a configuration key named "HiddenSystemLeakProtNotifStatus" that was initially set to "1", while no such key exists in the decoy OS's configuration file. There is an option to disable the pop-up message. However, upon disabling this message, the configuration key value will change to "2". This is simple proof that the hidden OS is running. Moreover, when comparing the configurations files, there are clear differences. The hidden OS's configuration file has 58 lines, whereas by default, the decoy OS's configuration file only has 10. While this by itself cannot be treated as hard evidence, it potentially leaks information.

Another indication that a hidden OS is running can be obtained from mounted volume information that the user can retrieve from the VeraCrypt GUI. By default, a decoy OS runs from an encrypted volume named "System partition" with type "System", whereas a hidden OS runs from a volume mounted with the name "Hidden system partition" with type "Hidden". This is shown in Figure 7(a) and (b) respectively. Even a standard user account is able to obtain this information. If an investigator has administrative rights, it is highly likely that additional information can be obtained by analyzing processes and drives on the kernel.

Another class of attack is based on network/cloud environment information leaks. Modern operating systems are enhanced by default in cloud based mechanisms to make work easier for the user. An example of this is the Microsoft account that involves signing into one account for all devices. This information and the number of login attempts are recorded and stored on



(a)



(b)

Figure 7 VeraCrypt GUI when working on (a) the hidden OS; (b) the decoy OS.

user account information which can easily be accessed. In our tests, we also checked the Apple ID, which is used to log into Apple's iCloud as well as Google's single sign on account.

The use of both the decoy and hidden OSs is visible in the account logs and this can be an easy way to prove that another OS is installed on the device by simply observing that two OSs are registered and used concurrently on the same device. Combining this information with forensic analysis indicating that only one OS is present on the device and that the drive structure is capable of running a DFS hidden OS, can be used to prove the existence of a hidden OS. Similar attacks can be performed by comparing browser fingerprints. These types of web tracking techniques are described in Acar et al. [26] and Fifield et al. [27]. We conducted a series of tests which confirm that this method can indeed be used to reveal the presence of a hidden OS.

Information that can compromise the existence of a hidden OS can also be obtained from monitoring device network traffic. An attacker can use both passive and active OS identification techniques. As with cloud based information leaks, these techniques can easily reveal the existence of a hidden OS if the user runs different OS types. Techniques for detecting hidden OSs can also include forensic analysis of decoy OSs by indexing application versions and network services and comparing these with intercepted network traffic. Any unusual traffic from the same IP and MAC, but with applications and services not present in the decoy OS can lead to the conclusion that a hidden OS must be installed on the device.

5 Conclusion

This paper describes commonly used threat models against which Deniable File Systems (DFSs) can potentially be secured. However, with the advancements and progress of modern computer systems that include the integration of mobile and cloud solutions, the existing threat models are increasingly becoming obsolete. New threat models, namely, the One-Time Access, Multiple Access and Live Response Access models were analyzed and discussed. These improved threat models should supersede previous models as they provide greater coverage of security issues faced by DFSs and hidden operating systems. In view of the increasing likelihood of investigators being able to access several copies of DFS volumes during investigations, this issue should be addressed by adopting new precautions or improving encryption algorithms to make it harder to perform cross data analysis, which has emerged as a major threat to the security of DFSs. In addition, we also introduce a model

to cater for the increasingly common scenario where investigators have live access to the user's device through a network or other means. This paper also demonstrates practical examples of how these new models can be used to defeat plausible deniability of DFSs with a hidden operating system.

Acknowledgement

This work was undertaken with the financial support of a Thelxinoe grant in the context of the THELXINOE: Erasmus Euro-Oceanian Smart City Network project.

References

[1] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky (1997). "Deniable encryption," in *Proceedings of the 17th Annual International Cryptology Conference (CRYPTO'97)*, Lecture Notes in Computer Science, Vol. 1294, pp. 90–104, Santa Barbara, California, USA.

[2] A. Czeskis, D. J. S. Hilaire, K. Koscher, S. D. Gribble, T. Kohno, and B. Schneier (2008). "Defeating encrypted and deniable file systems: TrueCrypt v5.1a and the case of the tattling OS and applications," in *Proceeding of the 3rd USENIX Workshop on Hot Topics in Security (HotSec'08)*, pp. 1–7, San Jose, CA, USA.

[3] A. Skillen, and M. Mannan (2014). Mobiflage: Deniable storage encryption for mobile devices. *IEEE Transactions on Dependable and Secure Computing*, 11, 224–237.

[4] M. V. Baryamureeba, and F. Tushabe (2004). "The enhanced digital investigation process," in *Proceedings of the 4th Digital Forensic Research Workshop*, pp. 1–9.

[5] B. D. Carrier, and E. H. Spafford (2003). "Getting physical with the digital investigation process," *International Journal of Digital Evidence*, 2.

[6] National Institute of Justice (U.S.) (2004). 'Forensic examination of digital evidence: a guide for law enforcement,' NIJ special report. U.S. Dept. of Justice, Office of Justice Programs, National Institute of Justice, 2004.

[7] B. Hay, M. Bishop, and K. Nance. (2009). Live analysis: Progress and challenges. *IEEE Security Privacy*, 7, 30–37.

[8] X. Yu, B. Chen, Z.Wang, B. Chang, W. T. Zhu, and J. Jing (2014). "MobiHydra: Pragmatic and multi-level plausibly deniable encryption

storage for mobile devices," in *International Conference on Information Security*, Lecture Notes in Computer Science, Vol. 8783, pp. 555–567.

[9] B. Chang, Z. Wang, B. Chen, and F. Zhang (2015). "MobiPluto: File system friendly deniable storage for mobile devices," in *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC'15)*, New York, NY, USA, pp. 381–390.

[10] M. Kedziora, Y.W. Chow, and W. Susilo (2017). "Improved threat models for the security of encrypted and deniable file systems," in *Proceedings of the 4th iCatse International Conference on Mobile and Wireless Technologies (ICMWT'07)*, Lecture Notes in Electrical Engineering, Vol. 425, pp. 223–230, Kuala Lumpur, Malaysia, 2017.

[11] VeraCrypt, 'VeraCrypt Documentation,' http://veracrypt.codeplex.com/

[12] M. Kedziora, Y. W. Chow, and W. Susilo, "Defeating plausible deniability of VeraCrypt hidden operating systems," in *Proceedings of the International Conference on Applications and Techniques in Information Security (ATIS'17)*, Communications in Computer and Information Science Series, Vol. 719, pp. 1–11, Auckland, New Zealand, 2017.

[13] B. Oler, and I.E. Fray (2007). "Deniable file system – Application of deniable storage to protection of private keys," in *Proceedings of the 6th International Conference on Computer Information Systems and Industrial Management Application (CISIM'07)*, Minneapolis, MN, USA.

[14] P. Gasti, G. Ateniese, and M. Blanton (2010). "Deniable cloud storage: sharing files via public-key deniability," *in Proceedings of the ACM Workshop on Privacy in the Electronic Society, (WPES'10)*, pp. 31–42, Chicago, Illinois, USA, 2010.

[15] I. Jozwiak, M. Kedziora, and A. Melinska (2013). "Methods for detecting and analyzing hidden FAT32 volumes created with the use of cryptographic tools," in *Proceedings of the 8th International Conference on Dependability and Complex Systems, Advances in Intelligent Systems and Computing*, Vol. 224, pp. 237–244, Brunow, Poland, 2013.

[16] R. Huveneers, 'Disk Decipher', http://disk-decipher.hekkihek.nl/

[17] Y. Zeng, 'Crypto Disks', https://itunes.apple.com/us/app/crypto-disks-disk- encryption/id889549308?mt=8

[18] I. Jozwiak, M. Kedziora, and A. Melinska (2011). Theoretical and practical aspects of encrypted containers detection. *Dependable Computer Systems*, Springer Berlin Heidelberg, pp. 75–85.

[19] M. Piccinelli, and P. Gubian (2013). Detecting hidden encrypted volume files via statistical analysis. *International Journal of Cyber-Security and Digital Forensics*, 3, 30–37.

[20] A. Davies (2014). *A security analysis of TrueCrypt: Detecting hidden volumes and operating systems*, Information Security Group, Royal Holloway, University of London.

[21] C. Hargreaves, and H. Chivers (2010). *Detecting hidden encrypted volumes. Communications and Multimedia Security*, (Springer: Berlin, Heidelberg), pp. 233–244.

[22] D. M. Purcell, and S.D. Lang (2008). Forensic artifacts of Microsoft Windows Vista system. *Intelligence and Security Informatics*, 304–319.

[23] M. Lessing, and B. von Solms (2008). Live forensic acquisition as alternative to traditional forensic process, in *Proceedings of the IT-Incidents Management & IT-Forensics (IMF'08)*, pp. 107–124, Mannheim, Germany, 2008.

[24] C. Waits, J. Akinyele, R. Nolan, and L. Rogers (2008). 'Computer forensics: Results of live response inquiry vs. memory image analysis', Technical Report CMU/SEI-2008-TN-017, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2008.

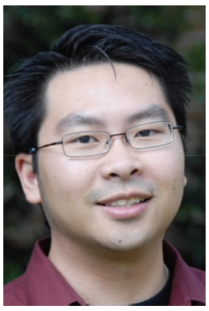
[25] N. Loginova, E. Trofimenko, O. Zadereyko, and R. Chanyshev (2016). "Program-technical aspects of encryption protection of users' data," in *Proceedings of the 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET)*, pp. 443–445.

[26] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz (2014). 'The web never forgets: Persistent tracking mechanisms in the wild', in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*, pp. 674–689, New York, NY, USA, 2014.

[27] D. Fifield, and S. Egelman (2015). "Fingerprinting web users through font metrics," in *International Conference on Financial Cryptography and Data Security*, pp. 107–124, Springer Berlin Heidelberg, 2015.

Biographies

Michal Kedziora received his M.Sc. (Eng.) and Ph.D. degree in Computer Science from Wroclaw University of Science and Technology, Wroclaw, Poland. Between 2016/2017 Dr. Kedziora was recipient of Thelxione postdoctoral research grant in University of Wollongong, New South Wales, Australia. Dr. Kedziora is Certified Information Systems Security Professional (CISSP) and Digital Forensics Investigator with many years working in public and private sector.



Yang-Wai Chow received his B.Sc., B.Eng. (Hons.) and Ph.D. from Monash University, Australia. He is a Senior Lecturer in the School of Computing and Information Technology, at the University of Wollongong, Australia. His research interests include virtual reality, interactive real-time interfaces, multimedia security and cyber security. He is a senior member of IEEE.

Willy Susilo received a Ph.D. degree in Computer Science from University of Wollongong, Australia. He is a Professor and the Head of School of Computing and Information Technology and the director of Institute of Cybersecurity and Cryptology (iC2) at the University of Wollongong. He was previously awarded the prestigious ARC Future Fellow by the Australian Research Council (ARC) and the Researcher of the Year award in 2016 by the University of Wollongong. His main research interests include cybersecurity, cryptography and information security. His work has been cited more than 10,000 times in Google Scholar. He is the Editor-in-Chief of the Information journal. He has served as a program committee member in dozens of international conferences. He is currently serving as an Associate Editors in several international journals, including Elsevier Computer Standards and Interface and International Journal of Information Security (IJIS, Springer). He has published more than 450 research papers in the area of cybersecurity and cryptology. He is a senior member of IEEE.