

Defeating Plausible Deniability of VeraCrypt Hidden Operating Systems

Michał Kedziora¹(✉), Yang-Wai Chow², and Willy Susilo²

¹ Faculty of Computer Science and Management,
Wrocław University of Science and Technology, Wrocław, Poland
`michal.kedziora@pwr.edu.pl`

² School of Computing and Information Technology,
Institute of Cybersecurity and Cryptology, University of Wollongong,
Wollongong, Australia
`{caseyc,wsusilo}@uow.edu.au`

Abstract. This paper analyzes the security of VeraCrypt hidden operating systems. We present attacks on the plausible deniability attribute of hidden Operating Systems (OSs) created using VeraCrypt. We demonstrate that the encrypted outer volume can contain information that compromises the existence of a hidden OS, and the fact that it was running, even if only one copy of the encrypted drive is examined. To further investigate this, we show that cross drive analysis, previously used to analyze deniable file systems, can also be applied to prove the presence of a hidden OS volume and to estimate its size. In addition, we discuss other attack vectors that can be exploited in relation to cloud and network information leaks. This paper also examines the security requirements of a threat model in which the attacker has direct access to a running hidden OS.

Keywords: Deniable file system · Hidden operation system · Plausible deniability · TrueCrypt · VeraCrypt

1 Introduction

A hidden Operating System (OS) is an operating system installed in an encrypted hidden volume, using software such as VeraCrypt. The assumption is that it should be impossible to prove that a hidden volume exists, and therefore impossible to prove that a hidden operating system exists. This concept is known as plausible deniability, as the existence of the hidden volume cannot be proven. This feature was implemented in TrueCrypt/VeraCrypt software as an extension of Deniable File Systems (DFSs) [10], and is based on deniable encryption which was introduced by Canetti [2, 11].

One notion of deniable encryption is the ability to decrypt a ciphertext into two different plaintexts depending on the key that is provided. An additional property is to ensure that the adversary cannot detect that a hidden message is present in the ciphertext. The purpose of this is to protect against adversaries

who are able to force the user to provide a password to decrypt the content, as the password that is provided will only reveal the decoy message/data while keeping the true message/data hidden.

Plausible deniability is implemented in TrueCrypt/VeraCrypt via its ability to create hidden volumes and hidden operating systems. VeraCrypt was developed based on the original TrueCrypt project. VeraCrypt uses XTS mode for encrypting partitions, drives and virtual volumes [11]. This mode of operation is described by Eq. 1; where \otimes denotes multiplication of two polynomials over the binary field $\text{GF}(2)$ modulo $x^{128} + x^7 + x^2 + 1$; $K1$ is the encryption key; $K2$ 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 $K1$; 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.

$$C_i = E_{K1}(P_i \wedge (E_{K2}(n) \otimes a^i)) \wedge (E_{K2}(n) \otimes a^i) \quad (1)$$

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 Fig. 1. 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.

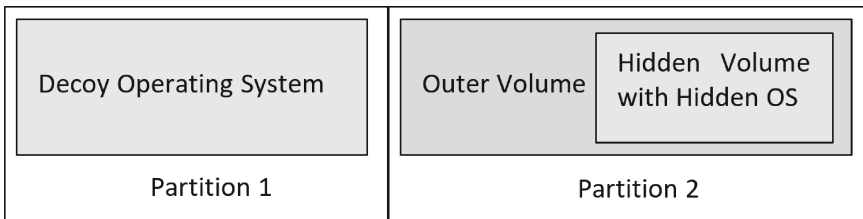


Fig. 1. Layout of a drive containing a hidden operating system.

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 (described in Sect. 2). 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, 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]. In this paper, we demonstrate that the above statement is not entirely true, as the presence of the hidden OS can in fact be revealed.

Our Contribution. In this paper, we analyze the security of VeraCrypt hidden OSs. While this software allow for plausible deniability via the creation of hidden OSs, we demonstrate that the encrypted outer volume can contain information that compromises the existence of a hidden OS. Our results are presented from the point of view of a new threat model incorporating One-Time Access, Multiple Access and Live Response Access scenarios. This paper presents experiment results showing that the VeraCrypt hidden OS implementation has faults that can be exploited to compromise the hidden OS even if an attacker only possess one binary copy of the drive. In addition, we show that it is vulnerable to cross drive analysis, which can be used to estimate the size of the hidden OS. Furthermore, this paper discusses other types of attacks that can be conducted to reveal the existence of a hidden OS on a device based on the Live Response Access scenario.

2 Threat Model

This work is based on our previously improved threat model for the security analysis of Deniable File Systems (DFSs) and hidden Operating Systems (OSs) [9]. This new model is an improvement on the model proposed by Czeskis et al. [3], as it addresses the flaws and inconsistencies in the previous model. The improved threat model is depicted in Fig. 2, in which the attack vectors are defined by One-Time Access, Multiple Access and Live Response Access scenarios. Compare with the previous model, this new model is much more practical and suitable for assessing the security of hidden OSs.

The One-Time Access scenario is a situation where an investigator has managed to obtain one or more copies of a device containing only a single copy of the drive containing a hidden OS [3]. Attack vectors based on this model have been presented in related work [4, 6, 7]. However, most of these findings are based on detecting DFSs, but cannot be applied to detecting hidden OSs. This is because in the case of hidden OSs, the entire drive is encrypted, thus, reducing the potential sources of information leaks that can compromise the hidden volume.

In a Multiple Access scenario, an investigator has access to multiple device images containing multiple hidden encrypted containers. The main threat to DFSs in this scenario lies in possibility of differential analysis for detecting hidden volumes, as this results in the ability to attack the plausible deniability attribute. This issue was raised in Czeskis et al. [3], where they highlighted that if disk snapshots could be obtained at close enough intervals, then the existence of any

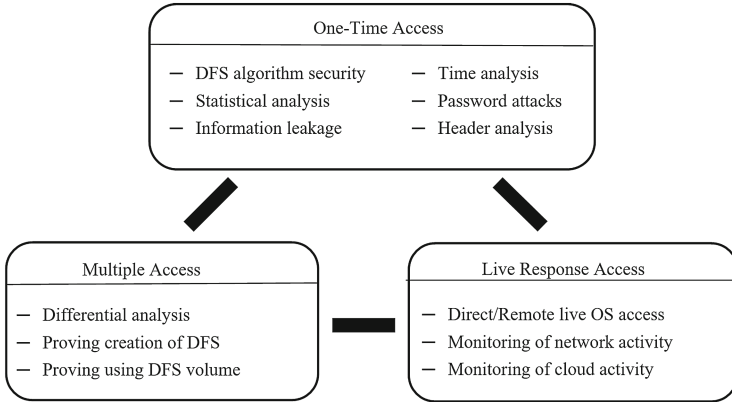


Fig. 2. Threat model and attack vectors on deniable file systems and hidden operating systems.

deniable files would become obvious. This is due to the fact that examination using differential analysis can reveal that seemingly random bytes on the hard drive will change in a non-random fashion. This was practically demonstrated by Hargreaves and Chivers [6], and research on detecting the creation of DFSs inside an encrypted container have been presented in Jozwiak [8].

The Live Response Access model is the model that is most suitable for detecting a hidden OS. Examples of such a scenario is when an investigator has direct live access to a DFS based hidden OS, or has access to the network environment within which a hidden OS is operating, or has access to cloud applications in which a hidden OS is connected to. A typically situation will involve an investigator remotely logging into a system containing a hidden OS using live response tools or just using standard remote access software like Team Viewer or VNC. Live response and memory analysis tools have the capabilities of collecting information from network connections, open ports and sockets, running processes, terminated processes, loaded DLLs, open files, OS kernel modules, process dumps, strings or user logs [12].

3 Defeating Deniability of Hidden Operating Systems

In this section, we present practical attacks on the deniability of hidden Operating Systems (OSs). For this, a test environment was created using Oracle Virtual Box version 5.1.12. A hard drive image size of 50GB was created. However, since the virtual box operates using the 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) were installed using VeraCrypt 1.19. The designed layout of partitions is depicted in Table 1.

Table 1. Layout of 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	1048553551	29240
Unallocated	104855552	104857599	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.

3.1 Encrypted Drive Analysis

First, we investigated the possibility of defeating plausible deniability of a VeraCrypt hidden OS under the most basic threat 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 pseudo-random 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 Fig. 3.

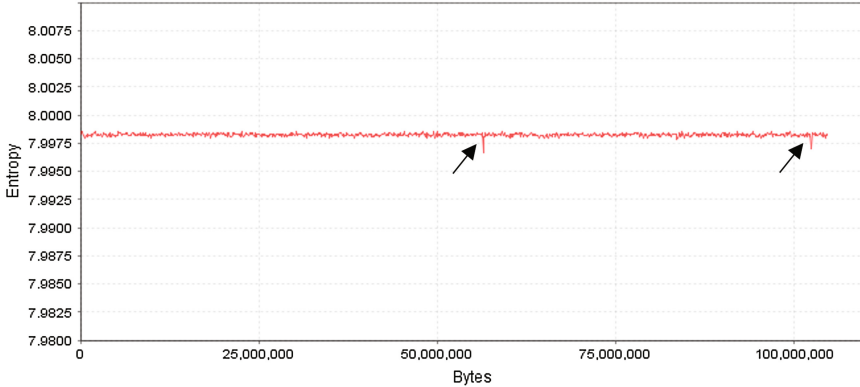


Fig. 3. Areas with significantly lower entropy inside the outer encrypted volume.

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. Presence of these areas violates the plausible deniability of the existence of a VeraCrypt hidden OS.

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 Fig. 4. Cross drive analysis showed that the second area correlates to running the hidden OS. Three bytes at offset 61435400 are altered every time the hidden OS is started. This is highlighted in Fig. 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.

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

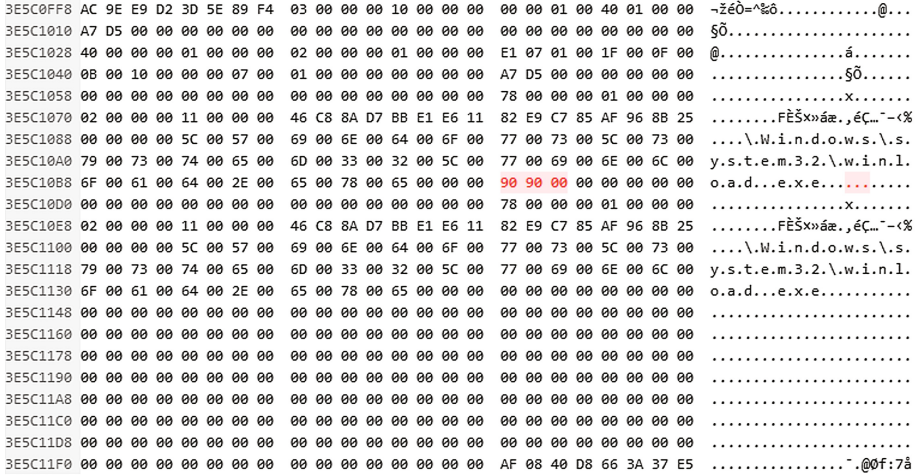


Fig. 4. Lower entropy areas.

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.

3.2 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 [6]. 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 SHA1 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 [6]. 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 min, 2 min and 5 min 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 running 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

Fig. 5. In Fig. 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.

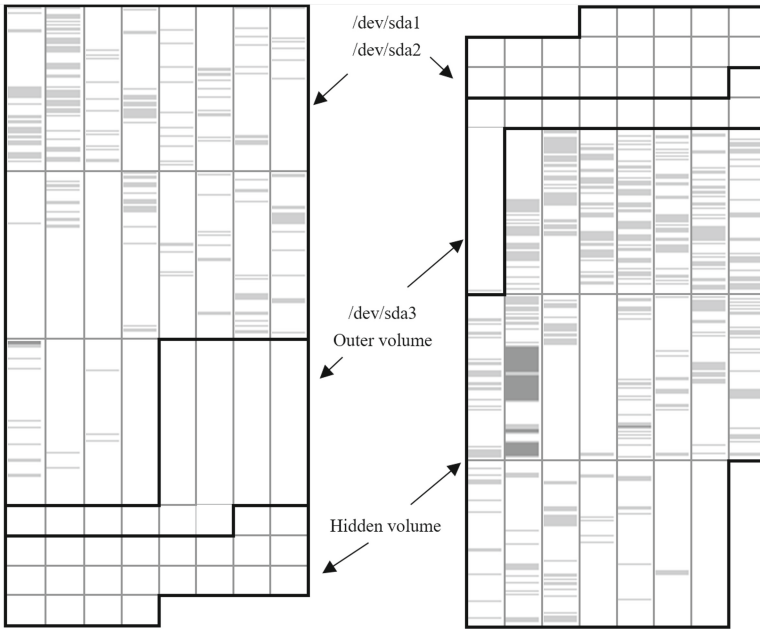


Fig. 5. A visual depiction of changes that were made to the volume while running the decoy OS (left) and hidden OS (right).

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, 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 didn't just turn the OSs on and off, but allowed the OSs to operate for some time.

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

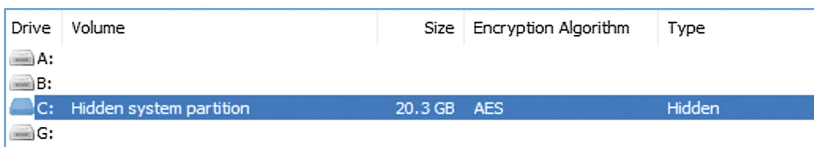
3.3 Other Attack Vectors

Hidden OS by design are 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. In addition, when configuration files located in the %APP-DATA%\VeraCrypt\ folder from both the decoy and hidden OSs are compared, there is a configuration key named “HiddenSystemLeakProtNotifStatus” that is 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 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 Fig. 6. 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



Drive	Volume	Size	Encryption Algorithm	Type
A:				
B:				
C:	Hidden system partition	20.3 GB	AES	Hidden
G:				

Fig. 6. VeraCrypt GUI while running a hidden OS.

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 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 [1, 5]. 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.

4 Conclusion

This paper demonstrates that the implementation of the VeraCrypt hidden operating system has faults that can be exploited to compromise the plausible deniability attribute of the hidden OS even if an attacker only possess one binary copy of the drive. This paper also presents experiment results showing that the VeraCrypt hidden OS is vulnerable to cross drive analysis. This is because even if the OS is idle, it still performs large amounts of read/write operations that distribute changes to the entire partition area. Simply turning the hidden OS on and off generates enough changes in the binary image to estimate the size of the hidden OS. In addition, we discuss other types of attacks based on the Live Response Access model that can be used to reveal the existence of a hidden OS. Current hidden OS implementations do not cater for the possibility of cloud and network applications, which result in information leaks that can be exploited to prove that a hidden OS is installed on a device.

Acknowledgment. This work was undertaken with financial support of a Thelxinoe grant in the context of the EMA2/S2 THELXINOE: Erasmus Euro-Oceanian Smart City Network project, grant reference number: 545783-EM-1-2013-1-ES-ERA MUNDUS-EMA22.

References

1. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The web never forgets: persistent tracking mechanisms in the wild. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, CCS 2014, pp. 674–689. ACM, New York (2014)
2. Canetti, R., Dwork, C., Naor, M., Ostrovsky, R.: Deniable encryption. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 90–104. Springer, Heidelberg (1997). doi:[10.1007/BFb0052229](https://doi.org/10.1007/BFb0052229)
3. Czeskis, A., Hilaire, D.J.S., Koscher, K., Gribble, S.D., Kohno, T., Schneier, B.: Defeating encrypted and deniable file systems: TrueCrypt v5.1a and the case of the tattling OS and applications. In: Provos, N. (ed.) 3rd USENIX Workshop on Hot Topics in Security, HotSec 2008, San Jose, CA, USA, 29 July 2008, Proceedings. USENIX Association (2008)
4. Davies, A.: A security analysis of TrueCrypt: detecting hidden volumes and operating systems a security analysis of TrueCrypt. Detecting hidden volumes and operating systems (2014)
5. Fifield, D., Egelman, S.: Fingerprinting web users through font metrics. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 107–124. Springer, Heidelberg (2015). doi:[10.1007/978-3-662-47854-7_7](https://doi.org/10.1007/978-3-662-47854-7_7)
6. Hargreaves, C., Chivers, H.: Detecting hidden encrypted volumes. In: Decker, B., Schaumüller-Bichl, I. (eds.) CMS 2010. LNCS, vol. 6109, pp. 233–244. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-13241-4_21](https://doi.org/10.1007/978-3-642-13241-4_21)
7. Jozwiak, I., Kedziora, M., Melinska, A.: Theoretical and practical aspects of encrypted containers detection - digital forensics approach. In: Zamojski, W., Kacprzyk, J., Mazurkiewicz, J., Sugier, J., Walkowiak, T. (eds.) Dependable Computer Systems. AISC, vol. 97, pp. 75–85. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-21393-9_6](https://doi.org/10.1007/978-3-642-21393-9_6)
8. Jozwiak, I., Kedziora, M., Melinska, A.: Methods for detecting and analyzing hidden FAT32 volumes created with the use of cryptographic tools. In: Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T., Kacprzyk, J. (eds.) New Results in Dependability and Computer Systems. AISC, vol. 224, pp. 237–244. Springer, Heidelberg (2013). doi:[10.1007/978-3-319-00945-2_21](https://doi.org/10.1007/978-3-319-00945-2_21)
9. Kedziora, M., Chow, Y.-W., Susilo, W.: Improved threat models for the security of encrypted and deniable file systems. In: Kim, K., Joukov, N. (eds.) The 4th iCatse International Conference on Mobile and Wireless Technology, ICMWT 2017. LNEE, vol. 425, pp. 223–230, Kuala Lumpur, Malaysia, 26–29 June 2017. Springer (2017). doi:[10.1007/978-981-10-5281-1_24](https://doi.org/10.1007/978-981-10-5281-1_24)
10. Loginova, N., Trofimenko, E., Zadereyko, O., Chanyshev, R.: Program-technical aspects of encryption protection of users' data. In: 2016 13th International Conference on Modern Problems of Radio Engineering, Telecommunications and Computer Science (TCSET), pp. 443–445, February 2016
11. VeraCrypt. VeraCrypt Documentation. <http://veracrypt.codeplex.com/documentation>
12. Waits, C., Akinyele, J., Nolan, R., Rogers, L.: 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)