Steganographic information hiding that exploits a novel file system vulnerability

Avinash Srinivasan* and Satish Kolli

Volgenau School of Engineering, George Mason University, Fairfax, VA 22030, USA Email: asriniv5@gmu.edu Email: skolli@gmu.edu *Corresponding author

Jie Wu

Computer and Information Sciences Department, Temple University, Philadelphia, PA 19122, USA Email: jiewu@temple.edu

Abstract: In this paper, we present *DupeFile*, a simple yet critical security vulnerability in numerous file systems. By exploiting *DupeFile*, adversary can store two or more files with the same name/path, with different contents, inside the same volume. Consequently, data-exfiltration exploiting *DupeFile* vulnerability, hereafter called *DupeFile Hiding*, becomes simple and easy to execute. In *DupeFile Hiding*, a known good file is chosen, whose name serves as the cover for hiding the malicious file. Hence we classify *DupeFile Hiding* as a steganography technique. This vulnerability can also be exploited for legitimate applications- hiding product license, DRM, etc. *DupeFile* was first uncovered on a FAT12-formatted disk on Win-98 VM. Nonetheless, the vulnerability exists in numerous file systems, including NTFS, HFS+, and HFS+ Journaled. We have developed two tools: *DupeFile Detector* and *DupeFile Extractor* for hiding files in legitimate applications.

Keywords: data hiding; file systems; integrity; security; steganography; vulnerability.

Reference to this paper should be made as follows: Srinivasan, A., Kolli, S. and Wu, J. (2013) 'Steganographic information hiding that exploits a novel file system vulnerability', *Int. J. Security and Networks*, Vol. 8, No. 2, pp.82–93.

Biographical notes: Avinash Srinivasan is currently a Faculty member in the Computer Science Department at George Mason University. His research interests include information and network security and forensics, forensic analysis of file systems, forensic file carving, and security in WSNs and MANETs. He has published 30+ papers in scholarly conferences and journals including *IEEE INFOCOM* and *ACM SAC*.

Satish Kolli is a PhD student in Information Security and Assurance at George Mason University. He received his MS in Computer Science from Johns Hopkins University. His research interests include information security and protocol analysis.

Jie Wu is the Chair and Laura H. Carnell Professor in the Department of Computer and Information Sciences at Temple University. His research interests include wireless networks, mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. His publications include over 600 papers in scholarly journals, conference proceedings, and books. He has served on several editorial boards, including *IEEE Transactions on Computers and IEEE Transactions on Service Computing*. He was General Chair for *IEEE MASS*-2006, *IEEE IPDPS*-2008, and *IEEE ICDCS*-2013, and was the Program Co-Chair for *IEEE INFOCOMM*-2011. Currently, he is an *ACM* Distinguished Speaker and a Fellow of the IEEE.

This paper is a revised and expanded version of a paper entitled 'Duplicate file names – a novel steganographic data hiding technique' presented at the 'International Workshop on Identity - Security, Management and Applications (ID' 2011)', Kochi, India, 22–24 July 2011.

1 Introduction

Steganography comes from the Greek word steganos meaning covered writing. It is the art and science of writing hidden messages in such a way that no one, apart from the sender and intended recipient, suspects the existence of the message (Petitcolas et al., 1999). This is also referred to as Security through obscurity¹. The idea and practice of hiding information exchanges – aka steganography– has a long history. Traditional techniques of steganography ranged from tattooing the shaved head of a trusted messenger² to using 'invisible ink' and 'microdot' during the two world wars.

Steganography includes information hiding within computer files, such as an image file, audio file, or a video file. It uses a simple and seemingly harmless file as the cover file, hiding the malicious data underneath. The hiding process does not alter the content of the cover medium to an extent that is easily recognisable. More advanced techniques hide with such effectiveness that even statistical methods of detection can be evaded seemingly easily. Several techniques have been developed to detect information hiding; these are accomplished by various steganographic tools that employ a limited number of steganographic algorithms. However, the adversary has been consistently successful in developing new techniques to achieve evasion. Figure 1 presents the taxonomy of information hiding techniques, while Figure 2 presents the taxonomy of steganographic techniques.

Modern steganography employs digital media content as camouflage, powerful computers and signal-processing techniques to hide secret data, and methods to distribute stegomedia throughout cyberspace, thus posing a serious challenge to scientists and professionals alike in the field of information security (Wang and Wang, 2004). Especially for the digital forensic community, steganography has been a great challenge from the very beginning. Nonetheless, one has to be prudent and unbiased to recognise the good side of steganography, such as digital copyrighting and watermarking.

It is well know that one of the most widely used benchmarks for evaluation of information systems' security



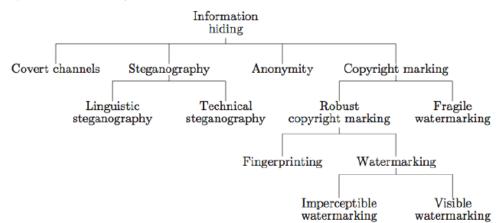
focuses on the three core goals– *Confidentiality, Integrity* and *Availability* of information. These three core coals are often collectively referred to as the 'CIA of security'. While all the three core goals are equally important for the security of a system, depending on the nature of the information and the corresponding domain, one or more of these three core goals can weigh in more than the other(s). In a well designed and implemented file system, which is the primary focus of this paper, all the three core goals of security have to be met. However, it is the integrity component of a file system that ensures all files and folders have unique names and/or paths, a key requirement for information storage and retrieval.

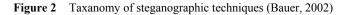
In this paper, we present and discuss *DupeFile*, a simple yet critical security vulnerability that exists in numerous file systems. More specifically, DupeFile is a file system integrity vulnerability. This vulnerability was first discovered on a FAT12 formatted disk on a Windows 98 machine. the vulnerability virtual Precisely, was encountered while recovering deleted files, in the aforementioned environment, using DiskEdit³ (http://wiki. osdev.org/Norton_Diskedit), a Hexeditor⁴ developed by Norton Utilities. However, the vulnerability exists across Microsoft's proprietary File Allocation Table (aka FAT) file system family, which includes FAT12, FAT16, and FAT32. It also exists on other Microsoft NTFS and Apple's HFS+, HFS+ Journaled, to name a few.

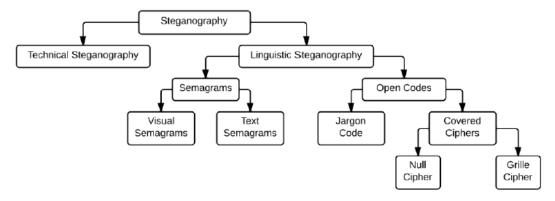
1.1 Problem statement

The discovered file system vulnerability, can be formally stated as follows:

"DupeFile is a file system 'integrity' vulnerability that can be exploited to hide a malicious file bearing the same exact name and extension of another file – a known good file that serves as the cover file – on the same media, at the same hierarchical level (path), without overwriting the contents of the cover file."







This vulnerability, though it appears to be simple, is quite severe in nature. An average computer user with basic knowledge of the underlying file systems' structure can easily exfiltrate important files in and out of a room, building, or even the country. To accomplish this, all he needs is a simple Hexeditor/Diskeditor such as DiskEdit or HxD⁵. The adversary can also directly write to the disk without the use of a Hexeditor/Diskeditor using simple computer programs and/or scripts.

From an adversarial perspective, files hidden employing *DupeFile Hiding* can range anywhere from simple and not so critical data, like a co-worker's salary and bonus package, to important business data, such as design blueprints and intellectual property. From a national security perspective, this could be a document containing classified information, or a terrorist plot. Nonetheless, the hidden files can also be potentially dangerous viruses, malware, or even child pornography image and/or video files. On the other hand, from a legitimate application perspective, *DupeFile Hiding* can be used for hiding password files, manufacturing blue prints, DRM, Copyright, and EULA to name a few. Such files can be accessed, on the fly, using tools that we have developed to counter *DupeFile Hiding*, details of which are presented in later sections.

Now, an important question that arises and needs to be answered is as follows:

"Is this the most sophisticated and stealthy data hiding technique?"

The answer is 'NO'. However, not being the most sophisticated and stealthy data hiding technique neither mitigates the risk, nor eliminates the threat presented by this vulnerability. On the contrary, this seemingly harmless vulnerability presents the adversary a simple and easy to execute data hiding technique with strong 'security through obscurity'. The fact that it is not very complex does indeed work in favour of the adversary and can be easily overlooked, which is what we suspect has been happening so far.

In this paper, we will discuss the vulnerability in details. However, we shall restrict our discussions to a FAT file system. For further simplicity in conveying the criticality of the discovered vulnerability, we limit our discussions to FAT12 file system. The same applies to the other two file systems in the FAT family, namely FAT16 and FAT32. Also discussed in this paper are the steps by which malicious files can be hidden by exploiting *DupeFile*, and easily evade detection. Also presented in this paper are solutions to counter *DupeFile Hiding* and we have developed customised tools for this.

1.2 Research objectives

In summary, the objectives of our research presented in this paper can be summarised as follows:

- 1 Present and discuss the discovered file system 'integrity' vulnerability *DupeFile* in detail.
- 2 Develop a simple and easy to use tool, *DupeFile Creator* that can be used for *DupeFile Hiding*.
- 3 Develop a simple and easy to use tool, *DupeFile Detector* that can be used to detect files hidden by exploiting *DupeFile* vulnerability.
- 4 Develop a simple and easy to use tool, *DupeFile Extractor* that can be used to extract hidden files detected by *DupeFile Detector*.
- 5 Confirm that *DupeFile Hiding* meets the requirements of security and capacity of a good data hiding technique as presented by Provos and Honeyman (2001).

1.3 Contributions

1

Our contributions in this paper can be summarised as follows:

- This is, to the best of our knowledge, the first work:
 - that proposes a perfectly reversible file hiding technique in the user space of a disk;
 - that proposes a steganographic technique, which, unlike contemporary steganographic techniques, does not use techniques such as compression, substitution, or embedding;
 - that proposes a steganographic technique in which, unlike contemporary steganographic techniques, the cover medium is the name of a file and not an actual file;

- to uncover a potential vulnerability in FAT file systems that can be exploited to compromise the file system integrity by hiding files with duplicate names;
- to identify all potential file systems that are susceptible to the discovered file system vulnerability *DupeFile*.
- 2 We are the first to develop tools to specifically thwart data hiding that exploits *DupeFile* vulnerability we call it *DupeFile* Detector and *DupeFile Extractor*.
- 3 We have also developed a tool that can hide genuine data, when needed, by exploiting the *DupeFile* vulnerability; we call it *DupeFile Creator*.
- 4 Our tools, *DupeFile Detector* and *DupeFile Extractor*, can be easily extended and linked to *Windows Explorer* and/or *MAC Finder*.

1.4 Paper organisation

The reminder of this paper is organised as follows. In Section 2, we will review some of the important works in the field of steganography that are relevant to our work. We then present example application scenarios in Sections 3. In Section 4, we discuss important concepts that are in alignment with the research presented in this paper including FAT internals in Section 4.1, duplicate files' scenarios in Section 4.2, and requirements for DupeFile Hiding in Section 4.3. This is followed by a brief discussion on our adversary threat model in Section 5 and discussion on novelty of DupeFile Hiding and its adherence to Provos and Honeyman's security and capacity requirements of a good data hiding technique in Section 6. Then, in Section 7 we will discuss the details on how files can be hidden in plain sight, by exploiting the DupeFile vulnerability. We present the methods of detection and recovery in Section 8. Later in Section 9, we discuss issues prevalent in MAC OS X systems in light of the discovered file system vulnerability in Section 9.1, followed by other possible solutions from Manufacturers and/or Vendors in Section 9.2 and then some important observations that we have made in Section 9.3. Finally, in Section 10, we conclude our work with directions for future research.

2 Related work

Steganography can be used to insert plain or encrypted data in a cover file to avoid detection. The sole purpose of steganography is to conceal the very fact that something exists, as opposed to cryptography, which aims at rendering the contents uninterpretable. Data embedding has also been found to be useful in covert communication, or steganography. The goal was, and still is, to convey messages undercover, concealing the very existence of an information exchange (Petitcolas et al., 1999).

McDonald and Kuhn (1999). According to cryptographic file systems provide little protection against legal or illegal instruments that force the owner of the data to release decryption keys for stored data once the presence of encrypted data on an inspected computer has been established. Their proposed steganographic file system, StegFS, hides encrypted data in the unused blocks of a Linux ext2 file system. Consequently, it makes the data look like a partition in which unused blocks have recently been overwritten. The proposed method of overwriting with random bytes mimics a disk-wiping tool.

The Metasploit Anti-Forensics Project (http://www. metasploit.com/research/projects/antiforensics/) seeks to develop tools and techniques for removing forensic evidence from computer systems. This project includes a number of tools, including Timestomp, Slacker, and SAM Juicer, many of which have been integrated in the Metasploit framework. Metasploit's Slacker hides data within the slack space of the FAT or NTFS file system.

FragFS (Thompson and Monroe, 2004) hides data within the NTFS master file table. It scans the MFT table for suitable MFT entries that have not been modified within the last year. It then calculates how much free space is available and divides it into 16-byte chunks for hiding data.

RuneFS (Grugq, 2005) hides files in blocks that are assigned to the inode of bad blocks, which happens to be inode #1 in ext2. Forensic programs are not specifically designed to look at the bad blocks inode. Newer versions of RuneFS also encrypt files before hiding them, making it a twofold problem.

Special areas, such as the Host Protected Area (HPA), on modern ATA hard drives, can be used for hiding information that is neither visible to the BIOS nor the operating system (Garfinkel and Malan, 2006). However, it can be extracted with special tools. As with the HPA, all of these techniques for data hiding can be detected with tools that understand the typical format of the file system or the application structures.

Khan et al. (2011) have applied steganography to hard drives. Their method hides data so well that it is 'unreasonably complex' to detect. They have already managed to encode a 20-megabyte message on a 160gigabyte portable hard drive. Their technique relies on the way hard drives store file data in numerous small chunks, called clusters. The drive controller stores these clusters all over the disc wherever there is free space, and keeps track of the positions of the clusters by using a special database on the disk. The software overrides the disk controller chip and positions the clusters according to a code. In order to read the data, the person needs to know this code.

Alternate data streams is the closest to our work but are still not quite the same. There are easier ways to detect an ADS, as compared to detecting files that were hidden using our method. However, there are some interesting facts about ADS. Prior to Windows XP, the ADS did not even appear in the process listing. Had the ADS been hidden behind something innocuous like *cmd.exe* or *notepad.exe*, the execution of the ADS would be undetected. Anything digital can become an ADS. The hiding of the function behind an innocuous-looking executable is similar to internet scams where web sites harvest personal or private information, a technique called 'phishing' (Berghel and Brajkovska, 2004).

Ni et al. (2006) have proposed a novel reversible data hiding algorithm, which can recover the original image without any distortion from the marked image, after the hidden data has been extracted. This algorithm utilises the zero or the minimum points of the histogram of an image, and slightly modifies the pixel greyscale values to embed data into the image. It can embed more data than many of the existing reversible data hiding algorithms. It is proven analytically and shown experimentally that the Peak Signalto-Noise Ratio (PSNR) of the marked image generated by this method versus the original image is guaranteed to be above 48 dB. This lower bound of PSNR is much higher than that of all reported reversible data hiding techniques in literature. They have successfully applied the algorithm to a wide range of images, including commonly used images, medical images, texture images, aerial images, and all of the 1,096 images in the CorelDraw database.

Celik et Al. (2002) have presented a novel reversible data hiding technique, which enables the exact recovery of the original host signal upon extraction of the embedded information. A generalisation of the well-known LSB (Least Significant Bit) modification is proposed as the data embedding method, which introduces additional operating points on the capacity-distortion curve. Lossless recovery of the original is achieved by compressing portions of the signal that are susceptible to embedding distortion, and transmitting these compressed descriptions as a part of the embedded payload. A prediction-based conditional entropy coder, which utilises static portions of the host as sideinformation, improves the compression efficiency, and thus the lossless data embedding capacity.

Tian (2003), on the other hand, has also proposed a revisable data embedding method for digital images. He has explored the redundancy in digital images to achieve a very high embedding capacity, while keeping the distortion low.

According to Pang et al. (2003), while user access control and encryption can protect valuable data from passive observers, those techniques leave visible ciphertexts that are likely to alert an active adversary to the existence of the data, who can then compel an authorised user to disclose it. To address this problem, they propose StegFS, a steganographic file system that aims to overcome that weakness by offering plausible deniability to owners of protected files.

StegFS securely hides user-selected files in a file system so that, without the corresponding access keys, an attacker would not be able to deduce their existence, even if the attacker is thoroughly familiar with the implementation of the file system, and has gained full access to it. Unlike previous steganographic schemes, this construction satisfies the prerequisites of a practical file system in ensuring the integrity of the files, and maintaining efficient space utilisation. Note that this StegFS is different from the StegFS in (McDonald and Kuhn, 1999), but it comes from the same concept.

3 Application scenario

In this section, we present a couple of scenarios in different domains to emphasise the potential threat that can be posed by *DuprFile* vulnerability that can be exploited for data hiding. Here, we are assuming manual hiding rather than using our tool, *DupeFile Creator*.

- 1 Scenario-1: *Trafficking child pornography*: A child pornographer can hide child porn images and videos by using the same name as that of an innocuous-looking image and video file, respectively. The child pornographer can be doing this at his work place or at home. Since the two files have the same name, clicking on either will always open the known good-cover file. The technical details behind this are discussed later in section 4.3.
- 2 Scenario-2: *Information Theft*: An employee can easily steal confidential and proprietary data from the workplace. He first copies the cover file onto his external media, and then copies the file that he is stealing with the same name as the cover file, and walks out. Even if the security personnel were to check employees and their storage media before leaving, it would only reveal two files with the same name at best, and most people would overlook this situation.

4 Background

In this section we discuss topics of relevance and importance in order to understand the presented vulnerability and its exploitation.

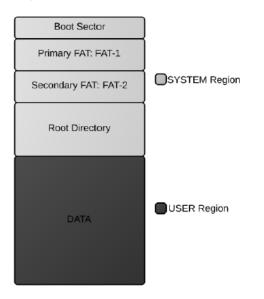
4.1 FAT12 file system internals

For simplicity, we consider the example of hiding a malicious file on a disk image formatted with the FAT12 file system. Additionally, to enable the reader to appreciate and understand the file hiding technique presented in this paper, we will briefly discuss the layout and important data structures of a FAT12 file system, as shown in Figure 3.

A FAT12 formatted volume can be divided into two main regions – system region and data region. The system region consists of important areas and data structures: boot area; file allocation table (two copies – primary and secondary); and root directory.

For file recovery, the two most critical areas are the file allocation table and the root directory. The standard, default size of a root directory entry is 32 bytes, and is consistent across the three FAT file systems, namely FAT12, FAT16, and FAT32.

Figure 3 Layout of a FAT12 formatted disk



The 32-byte directory entry of a file stored on a FAT formatted volume contains important information, which is listed below, that can be useful in detecting files that are hidden by exploiting *DupeFile*: *File Name and Extension*, *File Attribute(s)*, *Creation Date and Time*, *Modification Date and Time*, *Last Accessed Date*, *Start Cluster Number*, *and File Size*.

In particular, for files that are hidden by exploiting *DupeFile*, though they have the same name despite storing different contents, the start cluster numbers will be unique. The file size, in almost all cases, should be different as well; however, it cannot alone serve as evidence to trigger 'suspicion'. Nonetheless, when combined with 'start cluster number' information, the 'file size' information can make the case stronger by reinforcing the fact that they are indeed different files.

4.2 Duplicate file – name, content or both?

Note that the following two situations have to be clearly differentiated to understand and appreciate the discovered file system vulnerability. For any given file, there can be one or more duplicates of that file, either with the same name or a different name. In both cases, we are assuming that the file and its duplicates have the same content. A duplicate file, with the same name and with the same or different content, has to reside at a location different from the original file. On the other hand, a duplicate file with a different name and the same content can reside at any location, including that of the original file.

1 *Duplicate files with the same name*: If two or more copies of a file have the same name, and are inside the same volume on a drive, then they each should have a unique path, i.e. be at different hierarchical levels. Each copy still has a unique root directory entry. As long as duplicate copies are inside the same volume, copies with the same name will have consistent data, as long as they are duplicates.

2 *Duplicate files with different names*: If two or more copies of a file each have a different name, then there will be a separate root directory entry for each copy of the file, irrespective of the hierarchical level they reside at. Modifying a file will not update the duplicate copies with different file names.

Furthermore, for any two given files that are stored on the same volume, four possibilities exist. The idea has been captured more intuitively in Table 1.

- They both have the same name but hold different contents

 DupeFile Files. The two files cannot exist in the same location, i.e. cannot have the same path, but we show that this is possible because of the DupeFile vulnerability.
- 2 They both have different names but hold the same content Duplicate Files. The two files can exist in the same location, i.e. have the same path.
- 3 They both have the same name and they hold the same content Duplicate Files. The two files cannot exist in the same location, i.e. cannot have the same path.
- 4 They both have different names and they hold different contents Distinct Files. The two files can exist anywhere inside the volume irrespective of the other's location.

 Table 1
 Possible combinations for two given files with respect to their name and content

	FILE-2	File-2
File-1	TYPE-I: SAME NAME AND SAME CONTENT	TYPE-II: SAME NAME AND DIFFERENT CONTENT
File-1	Type-III: different name and same content	Type-IV: different name and different content

In this paper, we are addressing the first scenario. There are commercially available tools to handle the second and third scenarios. The fourth scenario is benign, and poses no threat as such.

4.3 Requirements

To exploit the *DupeFile* vulnerability, the following requirements have to be met.

- 1 The cover file must have a lower 'start cluster' number, compared to the file to be hidden. This is because the OS, when you access a file, will open the file with the lower starting cluster number – first hit. This is true for both MAC and Windows OSs.
- 2 The cover file and the hidden file have to be at the same hierarchical level in the directory structure. In light of this statement, we have to answer the following question:

"Is it possible to have two different files with the same name, but different contents, at the same hierarchical level, i.e. on the same drive, inside the same partition, and inside the same folder (aka directory)?" The answer to this question is 'no', and if it is possible, then it is a clear violation of file system integrity. This is one of the reasons why hiding in plain sight will not easily raise a flag. Trivially, there are two ways to attempt to create two files with the same name:

- 1 *Renaming an existing file*: Two files already exist inside a folder with different names. Try to rename one of them so that they both have the same name. An error message will pop up.
- 2 *Creating a new file*: A file already exists. Try to create a new file and save it in the same folder as the existing one with the same name. This is the same as opening an existing file and using the 'save as' option. Once again, you will see an error message pop up.

In summary, one cannot save two files with the same name inside the same directory without overwriting. Once overwritten, the original file content will be lost forever. However, this is easily accomplished with the use of any freely available HeX editor. It also requires some knowledge about the underlying file system. By using a HeX editor, the adversary can change the names of files to read the same. Since, with a HexEditor, we work below the the OS on the raw data, the OS will not complain and say 'file already exists'. The OS also does not overwrite the contents of the original file, thereby preserving the original file whose deletion will defeat the purpose of our hiding. Hence, there can be several files with the same name inside the same directory. This has been illustrated in Figures 8 and 9 and is clearly a violation of the file system integrity.

4.4 Common areas for data hiding

There are several common areas on the disk that are either unused or reserved, and can serve the purpose of hiding data without interfering with the intended operations of the partition. Following are areas, some of which occur only on the OS partition, while some others occur only on a non-OS partition – slack space, boot sector of non-bootable partition, unallocated space, volume slack, file system slack, partition slack, inter-partition gap, fake bad clusters, HPA, Device Configuration Overlay (DCO), encrypted data in unused blocks (ext2), alternate data streams (NTFS only), MFT slack (NTFS only), \$Boot file (NTFS only), and \$DATA Attribute (NTFS only) to name a few.

5 The threat model

The adversary is motivated to hide data in plain sight. He is aware of the discovered file system vulnerability and has the appropriate means to exploit it. His intention is to keep the process simple and utilise the user space for hiding his data, unlike most existing data hiding techniques that try to hide data in reserved areas, system areas, unallocated space, slack space, etc. However, the constraint that most existing data hiding techniques suffer are since such special/reserved areas are not intended to hold user data, it can quickly trigger suspicion of a 'malicious activity'. On the other hand, data hiding with the vulnerability that we have presented overcomes this constraint and in addition there are a few more notable advantages. The advantages are as follows:

- 1 Unlike hiding in unallocated space, slack space, etc., there is no fear of the hidden data being over-written, in this case.
- 2 Since the hidden file is part of the user space, there is less of a chance of an alert for deviation from the normal behaviour.
- 3 Since the proposed technique does not employ compression, substitution, or embedding techniques, it will go undetected by conventional steganography detection tools.
- 4 Since the proposed technique uses a file name as its cover instead of an actual file, it will go undetected by statistical steganography detection tools that measure deviation in the payload or other statistical measures.

6 DupeFile Hiding

6.1 Novelty

As can be seen from the discussion above, there have been several proposals on data hiding (embedding), based on various aspects of files, transmission types, and mediums, etc. However, our approach is the first to exploit a vulnerability – which was to the best of our knowledge, still unknown at the time of this writing – at the file system level to achieve perfectly reversible file hiding. Additionally, we have proposed a technique that does not break the file into bits and pieces to hide it. This is the first such approach to hide data in plain sight in the user-accessed region of a disk. Our work does not exploit the file system data structure to hide information (Huebnera et al., 2006). It does not hide data in various slack spaces and unallocated space (Huebnera et al., 2006).

Finally, *DupeFile Hiding* is not about hiding data in protected and hidden areas like DCO and HPA. Our adversary simply intends to hide files within the user space. The HPA is a reserved area on a Hard Disk Drive (HDD). It was designed to store information in such a way that it cannot be easily modified, changed, or accessed by the user, BIOS, or the OS. This area can contain information ranging from HDD utilities, to diagnostic tools, as well as to the boot sector code. On the other hand, the DCO allows system vendors to purchase HDDs from different manufacturers with potentially different sizes, and then configure all HDDs to have the same number of sectors. For instance, a vendor can use DCO to make an 80 GB HDD appear as a 60 GB HDD to both the OS and the BIOS (Technical committee T13, 2001).

6.2 Evaluation against standards

According to Provos and Honeyman (2001), from the suspect's view point, a good data hiding technique in the

NTFS file system should meet the following goals of security and capacity.

- 1 Normal system check with utility, such as chkdsk, does not return error: DupeFile Hiding does pass chkdsk without any errors. In our case, this is true even on all three FAT file systems.
- 2 Possibility of hidden data being overwritten is low, or even none: This is the biggest advantages of DupeFile Hiding: As previously noted, DupeFile Hiding is neither a filecompression technique nor a data-embedding technique. Consequently, the contents of the hidden file does not get overwritten if the cover file grows in size. Additionally, with duplicate files, accessing either file will always open the cover file, as long as the requirements is Section 4.3 are met. Hence there is no possibility of the hidden file being even accidentally overwritten.
- 3 *A normal user would not notice the hidden data*: Throughout the paper we have substantiated facts that an ordinary use is almost certain not to notice the hidden file(s).
- 4 *The technique can store a reasonable amount of hidden data*: This is the another advantage of *DupeFile Hiding*. Since *DupeFile Hiding* is neither a file-compression technique nor a data-embedding technique, there is no constraint on the amount of data that can be hidden without visually or statistically distorting the cover file. The user can store as much information as he wants as long as the requirements for *DupeFile Hiding* as presented in Section 4.3 are satisfied.

7 The process of hiding

In this section, we will discuss the method of hiding files with duplicate names, exploiting the *DupeFile* vulnerability. We are restricting our discussion to a FAT12 formatted volume. However, the same process applies to other file systems. While, on one hand, this can be accomplished by using a simple Hexeditor tool, HxD, a more sophisticated and savvy adversary can directly write to the disk at the low level to accomplish file hiding. We have developed a customised tool, *DupeFile Creator*, for hiding files by exploiting the *DupeFile* vulnerability – *DupeFile Hiding*.

To begin the process, the cover file has to be chosen. While the cover file can be chosen randomly, choosing it consciously is preferable, since it has to be innocuous in nature, both in name and content. While choosing a cover file, the type of the cover file (its extension) is not a key concern, since an extension can be easily modified for the malicious file to match that of the innocuous cover file. Henceforth, without loss of generality, we will use the term 'good file' to refer to the innocuous cover file being used, whose name will not cause any suspicion or raise flags, and use 'bad file' to refer to the malicious file being hidden, which can be proprietary information of a corporation, a child pornography image or video, product blue print, etc. Note that when used for legitimate application, the hidden file will not be malicious in nature.

The tool, *DupeFile Creator*, scans the entire root directory and returns the top five files, which is a customisable parameter, that very closely match the given file in size and attributes. Then the user can choose the cover file, file whose name and extension will be used as the cover for hiding the malicious file. Once the user makes his choice, the remaining process is simple and straightforward, which is taken care of by *DupeFile Creator*. Below are the steps for hiding files using our interactive *DupeFile Creator*.

- 1 User saves the malicious file to be hidden on the storage device.
- 2 *DupeFile Creator* scans the root directory for a suitable cover file and returns the top five picks.
- 3 *User* chooses the cover file from the files returned in step-2.
- 4 *DupeFile Creator* overwrites the name and extension of the file to be hidden with the name and extension of the cover file chosen by the user in step-3.
- 5 *DupeFile Creator* saves the change made and gives a confirmation message to the user.

Now, when the volume, containing the duplicate files, is opened on any system, there will be two files with the same exact name and extension, at the same hierarchical level. The matrix in Figure 4 presents the behaviour of various OSs to *DupeFile Hiding*.

OS /File Sur	FA	ΔT	NT	FS	HFS+			ŝ+J
OS/File Sys	Т	Е	Т	E	Т	F	Т	F
Win 95	Y	Y	NA	NA	NA	NA	NA	NA
Win 98	Y	Y	NA	NA	NA	NA	NA	NA
Win NT	Y	Y	NA	NA	NA	NA	NA	NA
Win XP	Y	Y	Y	N	NA	NA	NA	NA
Win Vista	Y	Y	Y	N	NA	NA	NA	NA
Win 7	Y	Y	Y	N	NA	NA	NA	NA
MAC OS X	Y	N	Y	N	Y	N	Y	N

Figure 4 Matrix with behaviour of each OS to various file system formats when hiding using DupeFile

 $T \rightarrow Terminal; E \rightarrow Explorer; F \rightarrow Finder; Y \rightarrow Displays duplicate file;$

 $N \rightarrow$ Does not display duplicate file; $NA \rightarrow$ Not applicable

8 The process of detection and recovery

We are restricting our discussion to a FAT12 formatted volume. However, the same process applies to other file systems.

8.1 Detecting hidden files

Detecting files with duplicate names but with different content can be challenging. We have developed customised tools - (a) *DupeFile Detector* for detecting files that are hidden exploiting *DupeFile* and (b) *DupeFile Extractor* for recovering detected hidden files. The following steps are performed by our tools to detect and extract the hidden files:

- 1 Loads the disk.
- 2 Scans the root directory entries on the entire disk recursively, including subdirectories, for duplicate file names along with the extension.
- 3 If there is more than one file, with the same name and extension, and at the same hierarchical level, then they will be marked for examination. (Note: two files with the same name and extension, if they are exactly the same in content, should have the exact same start cluster number and size.)
- 4 Marked files will be examined for their start cluster number and size.
- 5 If the marked files, with duplicate names, have the same start cluster number, then it can be ignored since it represents duplicate files w.r.t. content Type-I according to Table 1.
- 6 If the marked files, with duplicate names, have different start cluster numbers, then they are indeed different files w.r.t. content Type-II according to Table 1.
- 7 Files from step-6 are then subject to one of the recovery methods discussed later in this section.

8.2 Recovering hidden files

Once two or more files are detected to have the same name but different content, by using the detection mechanism delineated above, then they have to be separated before the hidden file's original content can be extracted. We have proposed two methods for the same-named files: (a) renaming method of recovery and (b) hash method of recovery. The latter is forensically sound and the evidence can be ensured to be admissible in the court of law when needed. Both of these methods are discussed in detail below.

8.2.1 Renaming method of recovery

This is a simple method for recovering duplicate file(s) that are hidden by exploiting the *DupeFile* vulnerability. The renaming method of recovery works as follows:

- 1 All detected malicious files are retrieved to a different location, such that the original content is unaltered.
- 2 In step-1, rename the files under investigation as SUSPECT-1.EXT, SUSPECT-2.EXT, etc., where EXT is the original extension of the file.
- 3 Note that there is a possibility that the original extension of the malicious file has been modified to suit that of the cover file extension. In such cases, retaining the original extension does not help much, and more needs to be done to access such files. One method would be to use a signature analysis to identify the file-type of such files accurately.
- 4 Now, open both files. Having named them differently, the malicious file will not be protected under the guise of the innocuous file any longer, and accessing it will reveal the actual content.

In Figures 5 to 9, we show screenshots of a diskette with two saved files with distinct names (DiskEdit and Explorer views), a diskette with BADFILE.PNG deleted (DiskEdit and Explorer views), and a diskette with BADFILE.PNG recovered with the name GOODFILE.JPG (DiskEdit and Explorer views), respectively.

Figure 5 Diskette with two distinct files – explorer view (see online version for colours)



Figure 6 Diskette with two distinct files – DiskEdit view (see online version for colours)

Action Edit	CD	Flopp	y Help				
-		ana ana sin		masaana	- second -	Disk Ed	itor
Ob jec	t E	dit	Link	View	Info	Tools	Help
Name	.Ex	t	Size		Date	Tim	e
Sector 1	9						
GOODFILE			15	36 11	-13-09	3:35	pm
GOODFILE	JPG		28	18 11	-30-09	1:23	pm

Figure 7 DiskEdit view – after deleting BADFILE.PNG (see online version for colours)

Action Edit	CD Flopp	y Help				
					Disk Ed	itor
Ob ject	: Edit	Link	View	Info	Tools	Help
Name		Size		Date	Time	e
Sector 19						
GOODFILE		15	36 11	-13-09		pm
σADF ILE	PNG	28	18 11	-30-09	1:23	pm

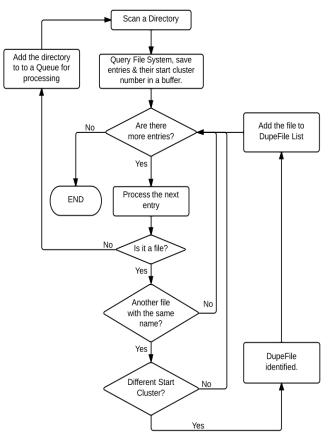
Figure 8 After recovering BADFILE.PNG by naming it as GOODFILE.JPG – DiskEdit view (see online version for colours)

Action Edit	CD Flopp	iy Help				
		and the second	0.000		Disk Ed	itor
Object	: Edit	Link	View	Info	lools	Help
Name	.Ext	Size	r /	Date	Tim	e
Sector 19)					
GOODFILE	JPG	15	36 11	-13-09	3:35	pm
GOODFILE	JPG	28	18 11	-30-09	1:23	pm

Figure 9 After recovering BADFILE.PNG by naming it as GOODFILE.JPG – Explorer view (see online version for colours)



Figure 10 Flow chart depicting the process of detecting files that are hidden by exploiting the *DupeFile* vulnerability



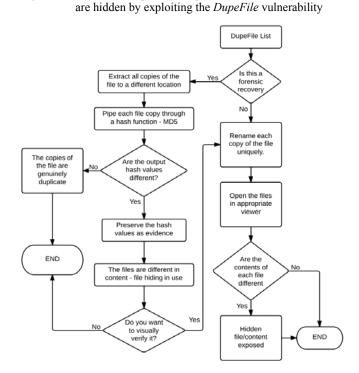


Figure 11 Flow chart depicting the process of extracting files that

8.2.2 Hash-method of recovery

For a forensically-sound process of detecting and recovering files that were hidden exploiting *DupeFile*, such that the evidence is admissible in legal proceedings, we propose the hash method of recovery. It is well-known that the hash value for a file is computed using the file's content and not its metadata. Consequently, the hash values will serve as evidence supporting the fact that the two files bearing the same name indeed have different contents. The recovery process is as follows:

- 1 All marked files are hashed using a hash function such as MD5 or SHA-256.
- 2 If the hash values for the marked files are different, it is certain that they hold different content.
- 3 Once the evidence is established using the hash, we can switch to the 'renaming method' for recovering the file(s).

9 Discussions

9.1 MAC OS X

Just like on a Windows machine, both in the terminal and the Explorer, MAC OS X Lion has different behaviours toward *DupeFile Hiding* for different file systems – both terminal and finder. This is shown in Figure 4. As can be seen, MAC Finder is more sensitive to duplicate files, because of which it filters the results and displays only one file. This is shown in Figure 12. On the other hand, the MAC terminal does not filter duplicate file names, i.e. distinct files with duplicate names, and displays all files. This can be seen in Figure 13. Figure 12 MAC OS X – finder view (see online version for colours)

it View Go Window Help		A 1
PDRIVE		
	Q.	
Name	 Date Modified 	Size
FILE1.TXT	Today 9:16 AM	14 bytes
	PDRIVE	PDRIVE PDRIVE PORIVE PORIVE O Arr Date Modified

Figure 13 MAC OS X – terminal view (see online version for colours)

🗯 Term	inal Shell	Edit Viev	v Window He	p
00			— bash — 80×24	
	IVE PDRIVE skoll: 1 skolli sta 1 skolli sta	i\$ ls -lart aff 14 Feb aff 14 Feb	*.TXT 28 09:16 FILE1.T 28 09:16 FILE1.T	

9.2 More solutions

In this section, besides our proposed solution, we discuss other possible solutions that can counter the exploitation of *DupeFile* vulnerability. In is the file system protection level that influences the overall computer security. In most cases, malware save their code within the computer file system. Hence, its critical to have proper file system protection in order to defends the computer from most malware.

- 1 *Fix the file system*: This vulnerability can be fixed by making chafes to the file system itself. This, however, is a more complex solution but one that can fix the problem inside out.
- 2 *Anti virus*: Anti virus is the most widely used approach to detect viruses, worms, trojans, etc. The discovered vulnerability, *DupeFile*, though not a virus, is a serious bug in the file system. Hence, our proposed solution can be implemented into existing Anti virus softwares such that it can be delivered to end users as a comprehensive security solution.

9.3 Important observations

- 1 Though the terminal displays all files, be it on a Windows machine or a Macintosh, it is important to realise that only someone who is aware of this vulnerability will be suspicious of *DupeFile Hiding*.
- 2 From a previous research, we have the following statistics on the number of files on a Windows Vista machine and MAC Snow Leopard Machine. We have more information for OS X 10.6 Snow Leopard 10.6.0 to 10.6.8 in Table 2. The file counts were generated using hashing techniques.
 - A fresh Windows Vista installation without any applications or updates have about 40 K to 50 K files.
 - A fresh MAC Snow Leopard installation (10.6.0 to 10.6.8), without installing any application or updates, has approximately 300 K to 500 K files depending on specific version and build.

- 3 Identifying duplicate files that are hidden exploiting *DupeFile* is that of a 'needle in the haystack' problem.
- 4 When accessed, it is always the innocuous cover file that is presented to the user. Consequently, even if someone tried to access both the files, out of suspicion that they both have the same name, they will always see the cover file. This is equivalent to Type-I according to Table 1.

OS X 10.6.X	XCODE	Number of files
10.6.0	No	331,047
10.6.1	No	330,858
10.6.2	No	331,502
10.6.3	YES	457,241
10.6.4	No	468,285
10.6.5	No	468,927
10.6.6	No	469,531
10.6.7	No	471,061
10.6.8	No	470,998

Table 2Number of files on different versions of MAC OS X10.6 – Snow Leopard

10 Conclusion and future work

In this paper, we have presented a subtle, yet serious file system vulnerability that we discovered on a FAT12 formatted volume on a Windows 98 virtual machine. The vulnerability was encountered when recovering deleted files from a FAT12 formatted volume during which time accidentally two files were named the same and the file system failed to complain and instead saved two both the files, without overwriting the previously existing file on the volume. We have named this vulnerability *DupeFile* and it is the integrity component of file system security that is compromised by this vulnerability. DupeFile can be exploited to hide files in plain sight by using duplicate file names and easily evading detection. We call this file hiding DupeFile Hiding, one that exploits DupeFile. Since an existing file's name is used as the cover for DupeFile Hiding, we have classified it as a steganography technique.

It is important to note that *DupeFile Hiding* is neither a file-compression technique nor a data-embedding technique, unlike contemporary Steganographic techniques. Hence, it is perfectly reversible. We have provided strong reasons, through example application scenarios and detailed discussions, why *DupeFile* can have a big payoff for the adversary with minimum risk. With further investigation, we found out that *DupeFile* exist in numerous file systems including FAT (12/16/32), NTFS, HFS+, HFSJ+, etc.

We have also proposed solutions by means of developing customised tools – (a) *DupeFile Detector* that can detect file that are hidden exploiting *DupeFile* and (b) *DupeFile Extractor* that can extract the actual content of the hidden file(s). We have also developed a customised tool – *DupeFile Creator* for *DupeFile Hiding* in legitimate applications. This tool has been developed solely for education and research purposes.

As part of our future work, we wish to investigate other file systems such as exFAT, ext2, ext3, and UFS. We also intend to research into more powerful data hiding techniques, anti-forensics. Finally, we are developing a prototype file system, using the FUSE framework that has the proposed fix for the *DupeFile* vulnerability presented in this paper.

References

- Bauer, F.L. (2002) Decrypted Secrets: Methods and Maxims of Cryptology, Springer-Verlag, 3rd ed., New York.
- Berghel, H. and Brajkovska, N. (2004) 'Wading into alternate data streams', *Communications of the ACM*, Vol. 47, No. 4, pp.21–27.
- Celik, M.U., Sharma, G., Tekalp, A.M. and Saber, E. (2002) 'Reversible data hiding', *Proceedings of the IEEE International Conference on Image Processing*, Vol. 2, pp.II-157–II-160.
- Garfinkel, S.L. and Malan, D.J. (2006) 'One big file is not enough: a critical evaluation of the dominant free-space sanitization technique', *The 6th Workshop on Privacy Enhancing Technologies*, UK, Vol. 4258, pp.135–151.
- Grugq (2005) 'The art of defiling', *Black Hat*. Available online at: http://www.blackhat.com/presentations/bh-usa-05/bh-us-05-grugq.pdf
- Huebnera, E., Bema, D. and Wee, C.K. (2006) 'Data hiding in the NTFS file system', *Digital Investigation*, Vol. 3, No. 4, pp.211–226.

- Khan, H., Javed, M., Khayam, S.A. and Mirza, F. (2011) 'Designing a cluster-based covert channel to evade disk investigation and forensics', *Computers & Security*, Vol. 30, No. 1, pp.35–49.
- McDonald, A. and Kuhn, M. (1999). 'StegFS: a steganographic file system for linux', in Pfitzmann, A. (Ed.): *Information Hiding*, LNCS, Vol. 1768, pp.463–477.
- Ni, Z., Shi, Y-Q., Ansari, N. and Su, W. (2006) 'Reversible data hiding', IEEE Transactions on Circuits and Systems for Video Technology, Vol. 16, No. 3, pp.354–362.
- Pang, H., Tan, K. and Zhou, X. (2003) 'StegFS: a steganographic file system', *Proceedings of the 19th International Conference on Data Engineering (ICDE '03)*, pp.657–667.
- Petitcolas, F.A.P., Anderson, R.J. and Kuhn, M.G. (1999) 'Information hiding-a survey', *Proceedings of the IEEE, Special Issue on Protection of Multimedia Content*, Vol. 87, No. 7, pp.1062–1078.
- Provos, N. and Honeyman, P. (2001) 'Detecting steganographic content on the internet', *Proceedings of ISOC NDSS*.
- Roch, A. and Goldenstein, S. (2008) 'Information hiding: types and applications', *First IEEE International Workitorial on Vision of the Unseen*, Anchorage, Alaska, USA.
- Technical committee T13 (2001) AT-Attachment with Packet Interface-6. Available online at: http://pdos.csail.mit.edu/ 6.828/2005/readings/hardware/ATA-d1410r3a.pdf
- Thompson, I. and Monroe, M. (2004) 'FragFS: an advanced data hiding technique, BlackHat Federal.
- Tian, J. (2003) 'Reversible data embedding using a difference expansion', *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 8, pp.890–896.
- Wang, H. and Wang, S. (2004) 'Cyber warfare: steganography vs. steganalysis', *Communications of the ACM*, Vol. 47, No. 10, pp.76–82.

Note

- Security through obscurity is a security engineering principle that attempts to use secrecy of design/implementation to provide security.
- 2 As reported by the 5th century Greek historian Herodotus.
- 3 DiskEdit is a Hexeditor, developed by Norton Utilities, for logical and physical disk drives on all Windows file systems. It is an undocumented utility that comes along with the standard Norton utilities package for Windows.
- 4 A Hexeditor is a type of computer program that allows a user to manipulate the fundamental binary data that makes up computer files including file metadata 5HxD is a Hexeditor and Diskeditor for Windows and is proprietary freeware.
- 5 HxD is a Hexeditor and Diskeditor for Windows and is proprietary freeware.