Evading Disk Investigation and Forensics using a Cluster-Based Covert Channel[*]

Hassan Khan, Mobin Javed, Fauzan Mirza and Syed Ali Khayam
School of Electrical Engineering & Computer Science (SEECS)
National University of Science & Technology (NUST), Islamabad 44000, Pakistan
{hassan.khan, mobin.javed, fauzan.mirza, ali.khayam}@seecs.edu.pk

ABSTRACT

Contemporary storage-based information hiding methods support plausible deniability by embedding encrypted information among bulk random content. Since the presence of random data is easily detected, these schemes facilitate plausible deniability by enabling disclosure of less sensitive information whilst concealing the existance of some other information. We propose a covert channel on storage media in which information is embedded by modifying the fragmentation patterns in the cluster distribution of an existing file. As opposed to existing schemes, the proposed covert channel does not require storage of *any* additional information on the filesystem. Since fragmentation also occurs through normal usage of a filesystem, our proposed channel allows one to conceal the very existence of hidden data, and consequently is the first storage-based covert channel to support an additional layer of *two-fold plausible deniability.*

1. INTRODUCTION

Concealment of sensitive information on digital transmission and storage media is becoming increasingly difficult with the growing sophistication of network and disk forensics tools. As data encryption using traditional cryptographic techniques is easily detected during forensic evaluations information hiding provides an additional layer of security over encryption by hiding sensitive information inside an innocuous medium. To evade information disclosure in such adversarial scenarios, covert channels—a subclass of information hiding techniques—hide sensitive information in media that are neither designed for nor intended to transfer information.

A practical covert channel must allow the defenders to deny the presence of hidden data. Existing covert channels, however, require storage of easily detectable encrypted/random content on the disk. Since the owner can be forced to release decryption keys in adversarial scenarios (e.g., by law or force,) existing techniques support plausible deniability by allowing the communicating parties to successfully deny that further hidden data exists at the expense of revealing some amount of data [2, 3, 4].

To provide an additional layer of evasion, we introduce a new concept of *two-fold plausible deniability* in which the covertly communicating parties get two chances to deny the presence of hidden information. In the first attempt, the parties simply deny the very existence of hidden data. If the first approach fails, they still have the option to only reveal the less sensitive information and deny the presence of any other hidden data on the medium as in [2, 3, 4].

We propose a novel information hiding approach using the above property to evade disk forensics. The proposed approach uses fragmentation patterns of an existing file in a filesystem to hide sensitive information. Under the proposed approach, the distribution of innocuous storage units (clusters in a filesystem) represents hidden information. Two-fold plausible deniability is provided because the proposed channel does not require storage of *any* additional (random or non-random) information on the storage medium, with the fragmentation being explained as a consequence of regular disk usage.

2. THREAT MODEL

A mass storage covert channel is a modified and considerably relaxed (warden-friendly) version of the prisoners' problem posed in [5]. Alice and Bob want to covertly exchange some information by evading Wendy, a passive warden, who monitors their communication. Alice can give any mass storage device to Wendy, who will inspect the device for clandestine (including encrypted) data. If the contents appear benign, the device is allowed to be transferred to Bob. Wendy prevents covert communication only through means of detection and does not tamper with the data on the storage medium.

3. PROPOSED APPROACH

We propose a novel covert channel which evades disk forensics by hiding sensitive information in the cluster patterns of existing files. While the proposed channel can be embedded into most of the contemporary block-based filesystems (NTFS, Ext2, UFS, HPSF, etc.) with minor modifications, throughout this paper we use the FAT filesystem as a proof-of-concept example.

In the FAT filesystem, a cluster is a group of consecutive sectors which is used to store data. If a file's size exceeds the size of one cluster, multiple clusters are allocated to it. If a file consists of multiple clusters and consecutive unallo-

[*]An extended version of this paper is available at [1].

Algorithm 1: Data Hiding

```
Input: An $n$ chunk message $M = [B_1, \dots, B_n]$, where each
       chunk is $\ell - 1$-bit, to be encoded; a coverfile $F$ with
       a filename string $t$; a padding marker; number of
       collision resolution (CR) bits ($\delta$) and the cluster
       distribution, $D$, of the coverfile $F = [c_1, \dots, c_L]$
       where $c_1, \dots, c_L$ are the file clusters
Output: Modified cluster distribution $D'$ of $F$.

begin
    if $L > n$ then
        Append $L - n$ padding chunks to $M$ such that
        $M' = [M||B_{n+1}, \dots, B_L]$, where $B_{n+1}, \dots, B_L$
        correspond to padding chunks;
    end
    Initialize MSB for each chunk as $CF$ (Collision Flag) bit
    with value set to 0;
    Write $c_1$ to arbitrary unallocated cluster location $k$ on
    the filesystem;
    for $i \leftarrow 2$ to $L$ do
        if cluster $k + B_i$ is free then
            Write $c_i$ to cluster location $k + B_i$;
            $k = k + B_i$;
        end
        else if cluster $k + B_i$ is allocated to a coverfile then
            Set CF bit = 1 ;
            Find $B'_i$ by modifying CR bits of $B_i$ such that
            $k + B'_i$ is free;
            Reformulate chunks from $B_{i+1}$ to $B_L$;
            Write $c_i$ to cluster location $k + B'_i$;
            $k = k + B'_i$;
        end
        else
            Free up $k + B_i$;
            Write $c_i$ to cluster location $k + B_i$;
            $k = k + B_i$;
        end
    end
end
```

cated clusters are available, those clusters are allocated to the file. If no consecutive unallocated clusters are available, the contents of the file are scattered or *fragmented* across the filesystem, thus forming a cluster chain. A File Allocation Table (FAT) containing an entry for each cluster keeps track of the cluster chains.

In a FAT32 filesystem, each cluster entry in the FAT uses 32 bits, of which 4 bits are reserved, leaving 28 bits for the cluster value. This gives an available address range of `0x0 - 0x10000000`, which can be used to hide data by embedding it in a cluster chain. The idea here is to use the distance between the fragmented clusters of a file to embed the hidden data. We first illustrate the idea with a simple data embedding example. Suppose that we need to hide 8 bytes of data given by `0x0123456789ABCDEF`. We divide it into 3 chunks of 3 bytes each and append a null byte at the start of the first byte. After performing this operation, we get the chunks `0x000123`, `0x456789` and `0xABCDEF`. Now in order to encode 3 chunks, we need a minimum of 4 clusters on a FAT filesystem. We choose an existing file of 4 clusters. Assume that cluster # `0x4BFA60` is allocated to the first cluster. Now to hide `0x000123`, we write the contents of the second cluster of the file at cluster # `0x4BFB83`, so that the distance between cluster numbers of the two clusters is `0x000123`; i.e, `0x4BFA60`$-$`0x4BFB83` $=$ `0x000123`. Similarly, in order to hide `0x456789` and `0xABCDEF`, we write the next two clusters at cluster # `0x91630C` and cluster # `0x13D30FB`, respectively.

Algorithm 2: Retrieving Hidden Data

```
Input: A key $P$; number of CR bits assigned for collision
       resolution ($\delta$) and the padding marker.
Output: A decoded $n$-chunk message $M$.

begin
    Set $t \leftarrow$ filename string of coverfile from the key $P$;
    Determine the location of the first cluster and size $L$ of
    $F$ from directory entry table by using $t$;
    $n \leftarrow L$;
    Initialize $n$ chunks $M$, $M = [B_1, \dots, B_L]$, to be empty;
    for $i \leftarrow 2$ to $n$ do
        Using FAT, determine the cluster location $j$ and $k$ of
        $c_i$ and $c_{i+1}$ respectively;
        $B_i \leftarrow k - j$ (less CF and CR bits);
    end
    Discard the padding chunks from $M$;
end
```

A *collision* may occur when the user is trying to write the contents of a cluster c to a cluster location k, where c is a cluster in coverfile F, but k already contains contents of d where d is a cluster in file G. We classify such collisions into soft and hard collisions. If G is not a coverfile, the collision is *soft*, otherwise it is a *hard collision*. To find out whether the collision is hard, the file to which the contentious cluster belongs is tested against the possible keys to check if it is a coverfile. In the case of a soft collision, the contents of d can be moved to an arbitrary unallocated cluster location l so that k becomes unallocated for F. In the case of a hard collision, the contents of d cannot be moved since G is also a coverfile, so a collision resolution technique is required.

We resolve hard collisions by inserting some dummy bits between message bits. Our proposed collision resolution technique works by reserving one bit in each ℓ-bit chunk of hidden message to identify a hard collision; we call this the Collision Flag (CF) bit. The flag bit is set if a hard collision occurs, in which case the next δ (where $\delta \leq \ell/2$) bits are used as Collision Resolution (CR) bits. The value of the CR bits is chosen to point to an unoccupied cluster location. The first $2^\delta - 1$ permutations of CR bits are used to select an unoccupied cluster location. In the unlikely case that all the $2^\delta - 1$ cluster locations are already occupied, the last permutation of the CR bits is used as a cluster skip flag which means that this chunk contains no message bits because the hard collision could not be resolved.

Details of data embedding and retrieval are given in Algorithms 1 and 2. Readers are referred to [1] for details.

4. PERFORMANCE EVALUATION

We evaluate the proposed covert channel on four performance metrics: capacity, disk access time, secrecy and volatility.

4.1 Storage Capacity

The capacity of the proposed channel depends on the total number of clusters in a filesystem and the fraction of clusters occupied by the coverfiles and is given by [1]:

$$C \geq z(\ell-1) - \left(\sum_{i=m}^{m+z-1} \frac{N}{N-\alpha i} - z\right)\log_2\left(\frac{\ln(\epsilon)}{\ln\left(\frac{\alpha(m+z-1)}{N}\right)} - 1\right),$$

where z is the number of clusters per which capacity is computed, ℓ is the number of embedded message bits per cluster, N is the total number of clusters in the file system, m

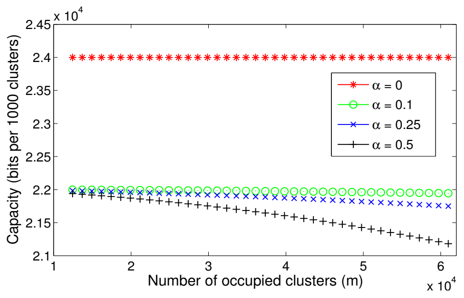

Figure 1: Upper and lower bounds on the capacity for a 256MB disk; $N = 124430$, $z = 1000$, $\epsilon = 0.0001$, & $\ell = 24$.

is the number of initially occupied clusters, $0 \leq \alpha \leq 1$ is the fraction of coverfile clusters and ϵ is an arbitrarily small non-negative value. Fig. 1 plots the above equation for capacity per $z = 1,000$ clusters for varying number of occupied clusters. We use 24 bits from the address to hide data and plot capacity for the case when 0%, 10%, 25% and 50% of the occupied clusters belong to coverfiles. As expected, capacity is a decaying function of the number of allocated clusters except for the case where no collisions have been assumed. The maximum achievable capacity (in case of no hard collisions) of our approach is $24,000$ hidden bits per $1,000$ clusters.

4.2 Disk Access Time

The proposed covert channel offers a significant advantage over existing approaches because it does not require storage of any additional (random or non-random) data on disk. Moreover, retrieval of hidden data only requires reading FAT entries corresponding to the coverfile, which can be performed in a few disk accesses (only the relevant sectors in the FAT need to be read). On the other hand, the retrieval of hidden information in contemporary approaches requires determining the location of hidden blocks from the maintained tables on the filesystem and then finding out whether a hidden data block has been overwritten or not. In case of overwriting, another copy must be read from another location and should be tested for validity. Thus data retrieval using these approaches requires several disk accesses. The write time is also expected to be higher in contemporary approaches as they must write each block to multiple locations and create corresponding entries in the block allocation tables that these approaches maintain on the filesystem. Our approach does not require multiple writes or maintenance of an allocation table.

4.3 Security

We have been assuming that fragmentation is a common phenomenon and raises no suspicions. Even if an investigator is suspicious, it would be difficult for him/her to determine whether the fragmentation is intentional (to hide data) or unintentional (due to addition and deletion of files on the filesystem over a period of time). Two-fold plausible deniability is provided as it is always possible to fabricate an intentional fragmentation which is very similar to an unintentional fragmentation. Thus, at the first level of plausible deniability, the user may simply deny that *any* hidden data exists on the filesystem, and can justify the file fragmentation due to regular filesystem usage. The second level of plausible deniability, similar to the method exercised by [2, 3, 4], allows the user to reveal the less sensitive hidden data (as all the files are fragmented but only a subset of them contain hidden data).

An investigator aware of the embedding strategy may analyze all possible files for hidden information using the known strategy. The maximum number of appended bytes will be less than size of the coverfile. If F comprises of c clusters and the location of the appended bits start at the d-th cluster, where $1 \leq d < c$, and end at the e-th cluster, where $d \leq e < c$, and where the size of appended bits between d and e is dependent on the the number of appended bits/bytes in the message; the hidden message m will be broken into two parts m_1 and m_2. If an attacker knows the algorithm he might be able to decode the $m_1 + (e - d) + m_2$ bits comprising the hidden information and the padding. In order to determine the boundary between m_1 and m_2, he can attempt to recover m_1 and m_2 through an exhaustive search, in which two unknowns less than m have to be recovered. Complexity of an exhaustive search would be $O(m^2)$.

4.4 Volatility of Hidden Data

The hidden information in our proposed approach is invulnerable to normal filesystem usage, i.e., addition or deletion of regular files; hidden data can only be destroyed if a coverfile is reallocated, extensively modified or deleted. On the contrary, existing schemes share hidden blocks with the filesystem, by using unallocated blocks and since accidental overwrites by the filesystem are very likely, it can lead to corruption of hidden information. This is handled by writing these blocks to several locations so that in case of corruption another copy can be used. However, this intentional redundancy reduces storage capacity, increases disk I/O and requires a method for detecting accidental overwrites by using a checksum.

5. LIMITATIONS

This section lists the limitations of the proposed covert channel: 1) The hidden data is difficult to modify once it has been written, because a slight change in the hidden data would require changing many coverfile chunks. 2) Defragmentation or deletion of a coverfile from the filesystem will result in loss of the hidden data. 3) A disk that has never been used would not contain any deleted data and would most likely be filled with some regular pattern. A fragmented file on such a disk will appear unusual. Thus, our technique is most effective on disks that have been used somewhat frequently. 4) The maximum amount of data that can be hidden by one coverfile is constrained by the filesystem limitation. It may be possible to address some of these limitations whereas others are constraints on our general technique imposed by the filesystem.

6. REFERENCES

[1] H. Khan, M Javed, F Mirza, S.A. Khayam, "Evading Disk Investigation and Forensics using a Cluster-based Covert Channel," NUST Technical Report, 2009, `http://wisnet.seecs.edu.pk/publications/fatCC.pdf`.

[2] R. Anderson, R. Needham, and A. Shamir, "The Steganographic Filesystem," in Proc. Information Hiding Workshop (IH98), Springer, 1998.

[3] A. McDonald and M.G. Kuhn, "Stegfs: A Steganographic Filesystem for Linux," in Proc. Information Hiding Workshop (IH99), Springer, 1999.

[4] H. Pang, K.L. Tan, and X. Zhou, "StegFS: A steganographic filesystem", in Proc. International Conference on Data Engineering, 2003.

[5] G.J. Simmons, "The Prisoners' Problem and the Subliminal Channel," in Proc. CRYPTO, 1984.