Bypassing information leakage protection with trusted applications

Jorge Blasco[a], Julio Cesar Hernandez-Castro[b], Juan E. Tapiador[c], Arturo Ribagorda[a]

[a]*Computer Science Department, Carlos III University of Madrid, Av. de la Universidad 30, 28911 Leganés*
[b]*School of Computing, University of Portsmouth, Buckingham Building, Lion Terrace, Portsmouth PO1 3HE, UK*
[c]*Department of Computer Science, University of York, Heslington, York, YO10 5DD, UK*

Abstract

Insider threats are an increasing concern for most modern organizations. Information leakage is one of the most important insider threats, particularly according to its potential financial impact. Data Leakage Protection (DLP) systems have been developed to tackle this issue and they constitute the main solution to protect information systems against leaks. They work by tracking sensitive information flows and monitoring executed applications to ensure that sensitive information is not leaving the organization. However, current DLP systems do not fully consider that trusted applications represent a threat to sensitive information confidentiality. In this paper, we demonstrate how to use common trusted applications to evade current DLP systems. Thanks to its wide range, trusted applications such as Microsoft Excel can be transformed into standardized block ciphers. Information can thus be encrypted in such a way that current DLP techniques cannot detect that sensitive information is being leaked. This method could be used by non-skilled malicious insiders and leaves almost no traces. We have successfully tested our method against a well-known DLP solution from a commercial provider (TrendMicro LeakProof). Finally, we also analyze the proposed evasion technique from the malicious insider point of view and discuss some

Email addresses: `jbalis@inf.uc3m.es` (Jorge Blasco), `Julio.Hernandez-Castro@port.ac.uk` (Julio Cesar Hernandez-Castro), `jet@cs.york.ac.uk` (Juan E. Tapiador), `arturo@inf.uc3m.es` (Arturo Ribagorda)

possible countermeasures to mitigate its use to steal information.

Keywords: malicious insiders, information leakage, sensitive information, evasion, trusted applications, data leakage

1. Introduction

An insider can be defined as a user with legitimate access to the organization's computers and networks (Pfleeger et al., 2010). Although insiders usually work on achieving the organization's goals, malicious insiders are becoming an increasing threat. A malicious insider is an insider who intentionally misuses an authorized level of access to affect the confidentiality, integrity or availability of the organizations' assets, including both data and systems (Walker, 2008). Unlike attackers from outside the organization, malicious insiders do not have to bypass security mechanisms deployed on the organization perimeter (IDSs, firewalls, etc.) to attempt a successful attack (Chivers et al., 2009). Additionally, they usually know the computer infrastructure, the way data is stored across it and, more significantly, its value (Moore et al., 2009).

Information leakage is a threat that comprises the unauthorized disclosure of confidential information. This includes any kind of information that is essential to achieve the organization's goals, such as blueprints, source code, financial or investments plans, etc. Client's private records are also considered sensitive, as they enable the organization to access its clients and have to be protected because of privacy regulations. Although information leakage can be originated by both insiders (accidentally or maliciously) and outsiders, information leakage incidents originated by users from within the organization are often much more severe than incidents caused by outsiders (CSO, 2010).

Information leakage events usually result in loss of competitiveness, economic fees imposed by governments and loss of reputation (Rantala, 2008). In fact, the economic losses produced by information theft related events have been studied extensively in several reports and surveys. One, conducted over US banking and financial institutions, stated that 91% of the organizations that suffered from these threats experienced financial losses, which on 30% of the cases exceeded 500.000 US dollars (Randazzo et al., 2005). As an example, a T-Mobile employee sold hundreds of thousands of personal data records of UK customers to rival firms (Wray, 2009). Those records were used to offer new contracts to T-Mobile customers, causing not only a major economic impact due to reputation loss and privacy concerns, but also because of clients that actually switched to another provider. Other reports analyzing the risks derived from information leakage (Ponemon Institute, 2008) estimate that the average economic cost per incident was 2.2 million Euros during 2008.

In order to mitigate the risks posed by information leakage incidents, a few security firms have developed a new kind of security tools usually known as DLP (Data Leakage Protection) systems. Vendors often claim that these tools are able to prevent both accidental and malicious information leakage (McCormick, 2008). DLP systems analyze network packets, files stored in computers and data in use to ensure sensitive data is treated according to the organization's policies and regulations. A major drawback of the current generation of DLP systems is that they do not offer protection against sophisticated insiders attempting to evade detection. In fact, such evasion attacks can be executed by means of well-known techniques, including obfuscating data (e.g., through encryption) or hiding it by using some steganographic scheme (Fisk et al., 2003; Zander et al., 2007). It may be argued that in a tightly protected environment the set of applications that an employee can execute (i.e. those that are trusted) is severely restricted. Thus, no user will have enough privileges to download and execute a program implementing a cipher or a covert channel over the network. On the other hand, trusted applications used to perform daily tasks are generally executed without any restrictions. However, some of these applications offer a rich functionality and can be programmed to help an insider in carrying out an information leakage attack.

In this paper we analyze how trusted applications could be used to bypass current DLP systems. We describe a simple but effective method that could enable malicious insiders to steal information from a supposedly protected organization. The presented evasion method is based on modifying the original purpose of an application using its built-in functionality. Our technique does not require high technical skills or gaining administrative privileges. We focus on common applications in corporative environments (i.e. Microsoft Excel) to transform sensitive information into encrypted data. Information transformed in this way can be transmitted outside the organization premises without being detected by current systems that would not allow the download or usage of any encryption software. We analyze the security and performance of this evasion method and report results obtained against a well known commercial state-of-the-art DLP solution. We finally propose new techniques and mechanisms to improve current DLP systems in order to reduce the risks associated with this kind of evasion attacks.

The rest of this paper is organized as follows. Section 2 provides and overview of current techniques and systems to prevent information leakage. Related work on evasion techniques in the context of DLP is given in Section 3. Section 4 describes our proposed evasion technique. An evaluation and security analysis of our attack is described in Section 5. In Section 6, we discuss some countermeasures that could be used to prevent this sort of evasion attacks. Finally, Section 7 presents our main conclusions and discuss some avenues for future work.

2. Data Leakage Protection

Organizations that do not implement mechanisms to protect their information from being leaked are an easy target for malicious insiders. In fact, simple mechanisms such as email messages, printers or removable devices can be used to transmit sensitive information to unauthorized parties. The steady increase of information leakage threats has driven organizations to implement mechanisms to protect their information assets.

DLP solutions, also known as Information Leakage Protection or Content Monitoring and Filtering solutions, are an emerging category of commercial products made up of a set of components (sensors, information discovery agents, content filtering agents, etc.) deployed in the organization's infrastructure to avoid information leakage (McCormick, 2008). DLP solutions use content analysis techniques to automatically discover what sensitive data an organization holds. Once identified, these solutions monitor and control its usage across the whole organization infrastructure. DLP solutions have been widely analyzed by Gartner (Quellet and Proctor, 2008) and Forrester (Raschke, 2006). Usually, DLP solutions consider information in three different states:

- Information at rest: Information assets that are stored, but not currently in use, are considered information at rest. Information assets in this state are usually stored on hard drives, memory cards, solid state disks or any other physical support for digital data. On servers, this information is usually stored on databases, file repositories, data warehouses, etc. On desktops and mobile devices, it is usually directly stored on the file system as documents, designs, text files, binaries, etc.

- Information in use: Any information asset being used at any workstation or server is considered information in use. Although information being used is in most cases also sitting on a storage device, information being used by a computer program is also stored in the system memory.
- Information in motion: As business processes are performed, employees need to access information located in remote locations and to share them with other team mates. Any information asset that travels through the organization network is considered information in motion. Protocols controled by DLP solutions include HTTP, HTTPS, SMTP, P2P protocols, FTP, etc.

Regardless of the state a piece of information is in, leakage is prevented by triggering certain actions when some conditions are met. Such actions include blocking file transfers, encrypting data or filtering out the executed command and generating an alarm. Current DLP solutions still face many open issues, notably reducing the number of false positives (Caputo et al., 2009; Lawton, 2008).

Some recent works have presented several approaches that can be included under the scope of DLP systems. Schear et al. (2007), proposed a system to control information leakage through HTTP connections. Their proposal, named *Glavlit*, is based on the usage of a warden. Any user inside the organization who wants to place documents on public web servers must first obtain permission from the warden, which may be a machine or a human. The warden analyzes the file in order to detect whether it contains sensitive information or not. If a warden has not approved a file exiting the organization, a gateway on the edge of the network will stop the HTTP request. A similar approach is presented in (Liu et al., 2009). Authors propose the usage of a framework that inspects network packets and looks for matches against a critical data repository. The framework, which is called SIDD, uses signature matching to detect sensitive content. Additionally, SIDD allows the detection of covert channels (Zander et al., 2007) that may be used to silently extract sensitive information. Evaluation is performed by the authors on a video redistribution scenario, which cannot be fully compared against real scenarios wherein many more different kinds of content can be transmitted.

A different kind of work has focused on techniques to detecting general malicious behavior from insiders, including information theft attempts. Schonlau et al. (2001) proposed to model users' behavior thorugh different classification techniques. Such models can be later used to distinguish normal behavior from malicious insiders. A similar approach, but using also network related events has been proposed by the MITRE Corporation (Caputo et al., 2008, 2009). Using events generated by network traffic and organization contextual information, they built a Bayesian network to detect IT misuse by malicious insiders.

Multilevel security models allow the classification of information into different levels in such a way that only users in possession of the necessary clearance level can get access. The Bell-LaPadula model (Bell and LaPadula, 1973), which was originally designed for military environments, establishes a security level to every subject and document. The BLP model establishes two mandatory premises for access control. First, no process may read information from a higher security level. Second, no process may write information to a lower security level. Taking this into account, if a process is able to read a sensitive information file, it means that the subject has clearance to read sensitive information files. In such a case, and following the BLP model, he would not be able to downgrade the security level of the file using that process. This, along with mechanisms to avoid extraction of sensitive files could be used to stop information leakages.

Although models such as BLP can serve to fight against information leakage, they also create a series of problems that make its use in organizations infeasible (Anderson, 2001). First, considering that the security level of a process is upgraded according to the files it accesses and the security level of the subject, if a sensitive file is opened, the next opened files will have to be written at the same level, independently of the level they came from. This leads to an overall increase of the security level of all files that are opened with that process. Therefore, besides stopping potential data leaks, this model can lead to employees being stopped from doing their jobs. Additionally, in order to work properly, applications and systems have to pass through major (and often costly) modifications (Rubinovitz, 1994).

Process coloring is a technique devised to detect and trace the propagation of worms across a system (Jiang et al., 2006). Each process susceptible of being exploited is uniquely colored. Process interaction (read, write, etc.) with other objects such as files or processes results in the propagation of the color. In this way, if a worm infects a process, all interactions made by the infected process can be traced back to the original source. Although the presented technique is focused on processes, it could be adapted to track sensitive information. Assume that, apart from assigning colors to processes, each sensitive file is given a unique color. Using process coloring, each process will get colored by the colors of the sensitive files they interact with (colors can be accumulated). If a colored process writes into another file, that file will also gain the process colors. To avoid leaks of sensitive information, the security policy could define a set of colors that cannot be transmitted outside the organization by any means.

The usage of this kind of technique involves certain drawbacks. First, process coloring, as presented by Jiang et al., is a technique that works over processes. A color is assigned to each process, but documents and other operating system elements are colorless until a process interacts with them. In order to track sensitive information, additional colors should be initially assigned to sensitive files. Besides, even after assigning colors to sensitive files it is unclear if once an application is closed, colors should remain on its processes. For example, an employee could use a text editor to modify a sensitive file. Later on, the same user could open again the same application and use it to create a personal document. Additionally, depending on the amount of sensitive information to manage, the numbers of colors can saturate the system, hindering the sensitive information tracing process (Jiang et al., 2008).

3. Evasion Techniques against Information Leakage Protection

When DLP systems are deployed in an organization, the risk of information leakage is reduced drastically, as they are capable of properly handling most accidental information leakages. Nevertheless, the perspective of high economic benefit and the sense of entitlement to information of some employees can motivate them to steal data, even when they perceive a risk of being caught (Moore et al., 2009; Farahmand, 2009). In these cases, the malicious insider will have to bypass the DLP mechanisms deployed.

Depending on the position of the malicious insider inside the organization, it may be easier for him to steal information. For example, system administrators and other IT-related staff usually have direct access to most systems inside the organization. This allows them to steal information more easily. As an example, during 2006 and 2007 an HSBC employee was able to steal 24.000 records from private banking clients in Switzerland (Simonian and Goff, 2010). Client records were delivered to French authorities to prosecute tax dodgers. On the other hand, users with limited privileges may be able to steal only the information they can access. In this case, in order to pass unnoticed to implemented protection systems they will have to use deception or evasion techniques.

Current DLP solutions use simple pattern recognition techniques and keywords to automatically detect sensitive information. Therefore, transforming sensitive information in such a way that patterns do not match the modified information can be used to bypass current DLP solutions. In fact, simple transformations (such as character or word replacing or simple mathematical operations) produce this effect. However, these transformations can be reversed easily, not being the most suitable method to be uses by a malicious insider.

Cryptography and steganography perform transformations on information in such a way that it is necessary to know the value of a secret key to recover the original information, provided that the used algorithm is secure enough. Cryptographic suites such as GnuPG (Skala et al., 2010) or OpenSSL (Cox et al., 2001) could be used to encrypt sensitive information files. In the same way, steganographic programs such as JPHS (Latham, 1999) and MP3Stego (Petitcolas, 1998) could hide sensitive information files into innocuously looking images or audio files. Such applications are free and easy to find on the Internet and do not require high technical skills.

To avoid the execution of such applications, operating systems and DLP solutions allow system administrators to restrict application execution. These mechanisms allow system administrators to define the applications that users can run inside the organization's computing environment. This forbids the usage of the aforementioned tools and limits the allowed ones to a very small number of trusted applications. Moreover, forensic evidence left by these tools could be used to trace back the malicious insider (Zax and Adelstein, 2009).

Nevertheless, such restrictions are not applied to programs approved for use by employees. Depending on the organization's activity and the user role, a wide variety of programs can be approved for execution, including text processors, CAD programs, spreadsheets, etc. These applications are extremely complex systems that are constantly used for the creation and manipulation of information. DLP solutions do not consider the possibility of using these applications for malicious purposes. However, the information manipulation features included in these programs are so advanced that it is possible to transform these into cryptographic applications. Thus, users could have access to modern cryptographic algorithms that are presumably restricted. We prove this point with an Excel implementation of two widely known block ciphers: TEA and AES.

4. Evading Information Leakage Protection

The goal of this section is to describe a method to bypass information leakage protection systems through the usage of common trusted applications. Our method uses a commonly trusted application, Microsoft Excel, to transform sensitive information in such a way that it will no longer be recognized as sensitive. Other trusted applications could be used for the same purpose. Nevertheless, spreadsheet applications allow to implement standard ciphers as transformations, increasing the security (from the malicious insider point of view) and hindering the detection of the sensitive information. Transformations can only be reversed with the knowledge of the secret key used to encrypt the information. This enables a successful extraction of the data from the organization without detection by current DLP technologies.

4.1. Implementing a Spreadsheet Cipher

Spreadsheet applications allow the usage of cells to perform data manipulation, calculations through formulas and data representation for different purposes: mathematics, statistics, management, etc. The presented evasion method uses spreadsheet cells to implement modern ciphers. Our proposal does not rely on the use of Visual Basic Scripts, as its execution can be easily blocked by system administrators. Sensitive information can be encrypted through the usage of these "encrypting" cells, being able to export the encrypted information and use it with no restrictions. This DLP evasion technique hinders the prosecution of malicious insiders, as a secret key is needed to reveal the real nature of the extracted information.

Apart from this trivial use, we believe that this techinique could be used in other environments. For example, cipher spreadsheets may be used to export certain cryptographic primitives to countries with export restrictions. As the spreadsheet cells do not constitute a program itself, export restriction laws could not be applied to it. Additionally, they could be used to encrypt information without leaving any evidence of having an actual encryption program in the computer. This could mislead computer forensic investigators and hinder criminal investigations.

We have implemented the Tiny Encryption Algorithm (TEA (Wheeler and Needham, 1995)) and the Advanced Encryption Standard[1] (AES (National Institute of Standards and Technology, 2001)). We believe both algorithms are good candidates for a proof of concept implementation because of their simplicity. Nevertheless, we emphasize that TEA has been proven to be insecure (Kelsey et al., 1997; Hong et al., 2004; Ko et al., 2004). The resulting spreadsheets are freely available, so other researchers can analyze these constructions and update them with new cryptographic algorithms.[2]. In the following we describe in detail our method: We first explain how to build encrypting workbooks and then discuss how to use the spreadsheet to leak information.

4.1.1. Basic operations

Spreadsheets applications such as Microsoft Excel operate with decimal numbers and do not include binary operations such as shift, rotations, bitwise XOR, etc. Standard ciphers extensively use these kind of basic operations. Thus, it is necessary to define such operations inside a spreadsheet environment. We next show how to implement some basic bitwise transformations using standard operations (addition, multiplication, modulo, etc.) over decimal numbers.

Shifts. A Shift operation is equivalent to a multiplication or division by 2^b, with b being the number of positions to be shifted. If A is the number to be shifted, a right shift operation can be described as follows:

$$A >> b = \frac{A}{2^b} \tag{1}$$

This can be implemented in Excel through the following formula:

```
=QUOTIENT(A;POWER(2;b))
```

Similarly, the left shift operation is given by:

$$A << b = (A * 2^b)\ mod\ 2^{32} \tag{2}$$

which can be implemented as:

```
=MOD(A*(POWER(2;b));POWER(2;32))
```

[1]Our AES implementation is restricted to 128 bits key length.

[2]Available at `http://www.seg.inf.uc3m.es/excelCiphers/ExcelCiphers.zip`

Rotations. Rotations are quite similar to shift operations, but bits shifted out are inserted at the other end of the number. Implementing this in decimal format requires to add the shifted bits to the result previously obtained from the shift operation. The right rotation would then look like:

$$A >>> b = \frac{A}{2^b} + (A \ mod \ 2^b) * 2^{32-b} \tag{3}$$

This could be implemented with the following formula:

```
=QUOTIENT(A;POWER(2;b))+(MOD(A;POWER(2;b))*POWER(2;32-b))
```

Similarly, left rotation is given by:

$$A <<< b = (A * 2^b) \ mod \ 2^{32} + \frac{A}{2^{32-b}} \tag{4}$$

and can be implemented as:

```
=MOD(A*(POWER(2;b));POWER(2;32))+QUOTIENT(A;POWER(2;32-b))
```

Exclusive OR (XOR). Bitwise operations require to convert numbers stored in cells into vectors, so bit-to-bit comparisons can be performed. Using an auxiliary vector (P), a decimal number can be converted into a bit vector to perform the XOR operation. Equation 5 describes the implementation of the XOR operation in a spreadsheet environment. Let $P = \{2^1, 2^2, \ldots, 2^{31}\}$.

$$A \oplus B = \Sigma_{i=0}^{32}((((\lceil \frac{A}{P_i} \rceil - 2 * \lceil \frac{\lceil \frac{A}{P_i} \rceil}{2} \rceil) + (\lceil \frac{B}{P_i} \rceil - 2 * \lceil \frac{\lceil \frac{B}{P_i} \rceil}{2} \rceil)) \ mod \ 2) * P_i \tag{5}$$

This can be implemented in a spreadsheet cell by:

```
=SUMPRODUCT(MOD(INT((A/P)-2*INT(INT(A/P)/2))
 +(INT(B/P)-2*INT(INT(B/P)/2));2);P)
```

For the sake of clarity P has been defined as a reference in the formula presented in this paper. Anyway, in any spreadsheet, the vector P can obtained by replacing its reference with the following:

```
P = 2^(32-ROW(INDIRECT("A1:A32")))
```

These three operators suffice to implement both TEA and AES. Different bitwise operations present in other algorithms can be implemented accordingly. In the following we describe the implementation process in more detail.

4.1.2. Defining Input and Constants

While TEA works with 32-bit integers, AES works with 8-bit integers. This means that in our implementation, a cell used to implement TEA will hold 32-bit integers and a cell used to implement AES will hold 8-bit integers. Therefore, depending on the algorithm to be implemented a different number of cells will be required to define its input and constants. In our implementation, 2 cells are required to define each plaintext block for TEA and 16 cells are required for AES.

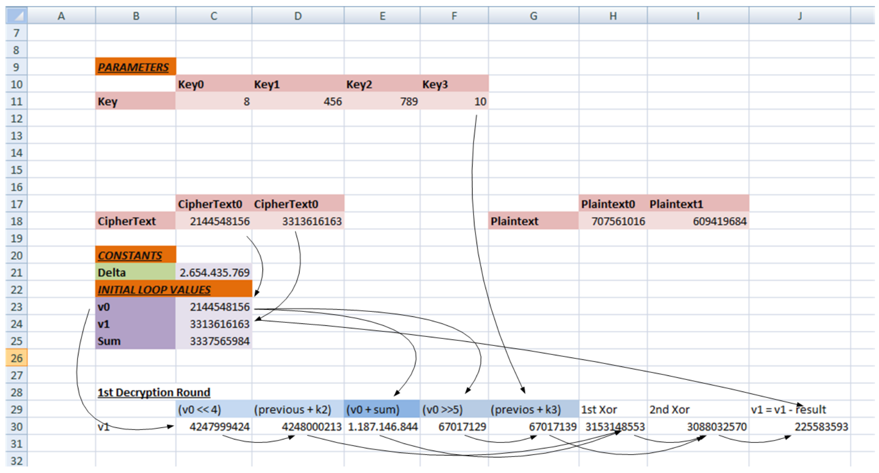
As most ciphers, our cipher implementation receives its input as integer numbers, producing also numbers as output. Depending on the data to encrypt, it may be necessary to preprocess the input before passing it to the cipher. This process can also be performed using the spreadsheet application. Text to be encrypted can be transformed to integers using the CODE() function, which converts characters to its ASCII value. To get back the original text, the CAR() function can be used. By using these functions, our approach can encrypt any sensitive information field in numeric or textual form. Furthermore, if data is not be disposed in blocks (cells) of the required length (32 bits for TEA and 8 bit for AES), the MOD() function can be used to divide the input integer in the required n bit blocks. This approach has served us well to leak several fictional (but valid) social security numbers from a system protected by a modern DLP solution (see Section 5.3).

4.1.3. Building Lines of Code

In a computer program, a line of code usually performs one or more basic operation and assigns the result to a variable in memory. Similarly, in our case each basic operation can be implemented using one cell and the corresponding spreadsheet formula. Note, however, that complex operations can be also written in just one cell by replacing references to other cells with the formula included in that cell. Thus for example, Figure 1 shows the implementation of a line of code that calculates the value of the first 32 bit of round i (v_0) in TEA.

4.1.4. Rounds

Modern ciphers extensively use the concept of rounds. While rounds can be implemented using iterative clauses provided by computer languages (i.e. *for*), in our domain, to add a round, it is just necessary to copy and paste the cells containing the operations from the previous round. If references between cells are correctly defined, it will not be necessary to perform any changes in



	A	B	C	D	E	F	G	H	I	J
7										
8										
9		*PARAMETERS*								
10			Key0	Key1	Key2	Key3				
11		Key	8	456	789	10				
12										
13										
14										
15										
16										
17			CipherText0	CipherText0				Plaintext0	Plaintext1	
18		CipherText	2144548156	3313616163			Plaintext	707561016	609419684	
19										
20		*CONSTANTS*								
21		Delta	2.654.435.769							
22		*INITIAL LOOP VALUES*								
23		v0	2144548156							
24		v1	3313616163							
25		Sum	3337565984							
26										
27										
28		1st Decryption Round								
29			(v0 << 4)	(previous + k2)	(v0 + sum)	(v0 >>5)	(previos + k3)	1st Xor	2nd Xor	v1 = v1 - result
30		v1	4247999424	4248000213	1.187.146.844	67017129	67017139	3153148553	3088032570	225583593
31										
32										

Figure 1: A detailed view of the implementation of the first round of TEA.

the copied cells. Once a full version of the cipher has been created, copying the whole algorithm (i.e. the cells) to other locations of the spreadsheet will allow to encrypt another block of information (64 bits in TEA and 128 in AES). This can be used to encrypt several blocks of information in the same spreadsheet. Later on Section 5.2 we analyze the amount of information that can be encrypted using a single spreadsheet.

4.2. Stealing Information With a Spreadsheet Cipher

If the spreadsheet application is trusted, which is one of our assumptions, it will mean that its use is required to perform business-related activities. Then, the required knowledge of the application to carry out the attack can be taken for granted by all employees enabled to execute it. Employees that want to steal information through this method will only require temporary access to the information. As shown by Moore et al. (2009), most employees (74% of analyzed cases) steal information they have direct access to. Therefore, that access can also be taken as granted. In the following, we outline the steps (Algorithm 1) to be performed by a malicious employee to steal information using a cipher spreadsheet.

To decrypt the stolen information, the malicious employee would have to copy the encrypted cell values to the corresponding cells in the decryption sheet, using the same password used to encrypt the information. If the information was transformed (text to number) prior to encryption, the reverse transformation should be applied on the obtained cell values.

Algorithm 1 Steps to steal sensitive information using a cipher spreadsheet

```
1. **Open** or **import** the file with sensitive information.
2. **Disable** the version control mechanism.
3. **Download** and **open** the cipher workbook file using the trusted application (i.e.
   Microsoft Excel).
4. *If* the clipboard is being controlled by the DLP solution
   (a) **Copy** the cipher cells into the sensitive information file.
   (b) **Write** the password in the corresponding cells.
   (c) **Reference** the cells whose values are to be encrypted in the cipher plaintext
       cells.
5. *If* the clipboard is not being monitored
   (a) **Write** the password in its corresponding cells.
   (b) **Copy** the values from the sensitive cells to the cells referencing the plaintext
       cells values.
6. **Copy** the ciphertext values or export them to a new file. As the information being
   copied is meaningless, no alert will be raised.
7. **Transfer** the file outside the organization through any of the available means (net-
   work, removable drive, etc.).
8. **Delete** the cipher workbook file and the cipher cells from the sensitive workbook
   (if they were copied).
```

5. Evaluation

We next discuss some aspects relative to the security and suitability of the proposed evasion technique from the point of view of the malicious insider. To obtain experimental proof of our evasion technique, we have tested our approach against a commercial DLP solution.

5.1. Security Analysis

In order to analyze the security of the proposed DLP evasion technique, we propose the following scenario. A disgruntled employee decides to steal sensitive information from an organization. The security policy of the organization states that sensitive files cannot leave the organization's boundaries by any means (printer, removable drives, network, etc.). To implement the security policy, the organization deploys a DLP solution that execute mechanisms to automatically detect sensitive information files and logs all attempts to extract sensitive information by any of the means defined in the security policy. Additionally, the DLP solution allows to define a set of trusted applications. Only those applications can be executed on workstations with access to sensitive information. Common applications such as the Microsoft Office suite are included in the trusted application list. For them, the installed DLP solution only controls actions that can directly produce an information leak, such as for example attempting to print a file or emailing a document).

Let $W = w_1, w_2, \ldots, w_n$ be a set of sensitive information, where w_i can be described with the regular expression exp_w. Let $C = c_1, c_2, \ldots, c_m$ be the implementation of a cipher in spreadsheet form where c_i codifies part of the cipher $Cipher(x, k)$. Both w_i and c_i are composed by a set of symbols that are accepted as content for a spreadsheet cell. Let $Sensitive_w(x)$ be a function that returns $true$ if x conforms to exp_w and $false$ otherwise. To simplify we can assume that $Sensitive_w(x)$ is called each time a spreadsheet cell is going to be copied into the clipboard. If $Sensitive_w(x)$ returns $true$, x can not be copied into the clipboard and the incident is logged.

As all operations are performed within a trusted application, we do not consider the existence of several trusted applications being used at the same time. If a malicious insider copies C into the clipboard, $Sensitive_w(x)$ will be called m times returning $false$ for all of them. When C is pasted into the spreadsheet containing W, operations defined in C will be performed using W as input. This will result in $W' = w'_1, w'_2, \ldots, w'_n$, where $w'_i = Cipher(w_i, k)$. Depending on the specific transformations performed by $Cipher(x, k)$, the result of $Sensitive_w(w'_i)$ will be $false$, enabling the malicious insider to copy W out and send it outside the organization. In the case of most commercial DLP solutions, $Sensitive_w(x)$ checks x against a set of defined regular expressions and keywords. Therefore, $Cipher(x, k)$ transformations have two requirements.

First, k must be required to obtain back x from w. If $Cipher(x, k)$ can be easily reversed, he might be able to evade the restrictions imposed by the DLP solution, but transformations could be reversed easily and new definitions of $Sensitive_w(x)$ will be able to detect the transformed information. In our case, $Cipher(x, k)$ implements both TEA and AES. The difficulty to obtain back the original information is (for both algorithms) orders of magnitude higher than any other simple transformation that could render $Sensitive_w(x)$ unusable. To the best of our knowledge there is no publicly available attack on the full version of AES 128 that reduces the complexity of finding the encryption key to significantly less than a brute force attack (Dobbertin et al., 2005). In the case of TEA, Kelsey et al. (1997) discovered that TEA suffers from equivalent keys. This means that each TEA key has 3 equivalent ones, reducing its complexity to 2^{126}. Other attacks exist, but still their practical implications in our setting are very limited. Nevertheless, as the key has to be introduced in the same spreadsheet, implementation attacks could be used against our proposal. However, to be able to implement such attacks, it would be necessary to detect that this kind of evasion is being used. This is addressed in Section 6.2.1.

Second, it should generate a w that does not match with any of the defined regular expressions or keywords. This is done by adding an additional transformation at the end of the ciphering process. This transformation ensures that the encrypted data will not match any sensitive data pattern or keyword. In the AES and TEA spreadsheet implementations, dots are introduced between each 3 digits to avoid any possible n digit pattern.

5.2. Performance Analysis

Implementing bitwise operations through standard decimal arithmetic functions increases the computational complexity of the cipher implementation. While in usual software implementations basic bitwise operations can be translated into a single line of assembly code, in our implementation it requires multiple operations. We have compared the spreadsheet implementations of TEA and AES with standard C implementations. In our test we encrypted up to 1 Megabit of information with each implementation. During tests, the machine was not executing any other task. To obtain average values, we repeated each encryption procedure 24 times. Average results show that the spreadsheet implementations are significantly (orders of magnitude) slower than C implementations (Table 1). In the case of AES, the Excel implementation is very inefficient, mostly due to the *MixColumns* step. Therefore, a spreadsheet application does not seem the optimal way to implement a cipher, and it is suitable only for small amounts of information or a proof of concept implementaton.

In terms of memory space, our TEA implementation requires 70 cells to encrypt 64 bits of information: 2 as plaintext input, 4 as password input and 64 to implement TEA. With AES we require 704 to encrypt 128 bits: 16 as plaintext input, 16 as password input, 512 to implement constants and 160 to implement AES operations. To encrypt more blocks, the plaintext and implementation cells must be copied. Therefore, with our current approach it is theoretically possible to encrypt up to $2^{40} \div 66 = 16659267087$ bits for each available sheet (up to 1.93 GB) with TEA and up to $2^{40} \div 176 = 6247225157$

Data	**Excel AES**	**C AES**	**Excel TEA**	**C TEA**
128 bits	0.23 sec.	$<$ 1 ms.	0.29 sec.	$<$ 1 ms.
256 bits	0.38 sec.	$<$ 1 ms.	0.33 sec.	$<$ 1 ms.
512 bits	0.66 sec.	$<$ 1 ms.	0.39 sec.	$<$ 1 ms.
1 Kb	1.35 sec.	$<$ 1 ms.	0.47 sec.	$<$ 1 ms.
2 Kb	2.70 sec.	$<$ 1 ms.	0.70 sec.	$<$ 1 ms.
4 Kb	5.28 sec.	$<$ 1 ms.	1.34 sec.	$<$ 1 ms.
8 Kb	10.53 sec.	1.6 ms.	2.68 sec.	$<$ 1 ms.
16 Kb	20.95 sec.	3 ms.	5.34 sec.	$<$ 1 ms.
32 Kb	42.23 sec.	6 ms.	10.61 sec.	$<$ 1 ms.
64 Kb	83.79 sec.	12 ms.	21.09 sec.	$<$ 1 ms.
128 Kb	171.19 sec.	25 ms.	42.22 sec.	$<$ 1 ms.
256 Kb	333.22 sec.	50 ms.	84.47 sec.	1.6 ms.
512 Kb	673.75 sec.	0.1 sec.	168.97 sec.	3.4 ms.
1 Mb	1343.26 sec.	0.2 sec.	335.72 sec.	6.6 ms.

Table 1: Performance comparison of spreadsheet and C implementations to encrypt up to 1 Megabit of information (average of 24 runs)

bits (up to 0.72 GB)[3] with AES.

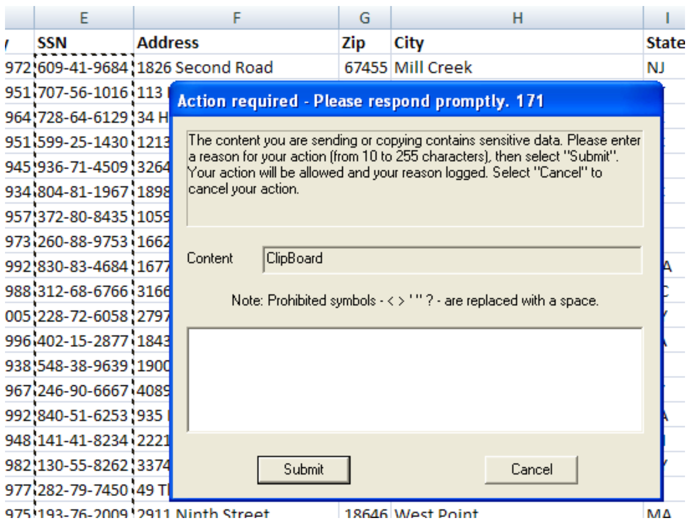
5.3. Bypassing a Commercial DLP Solution

We have tested our method against a commercial DLP solution from TrendMicro (i.e. LeakProof v5.0[4]). LeakProof is composed by a server and a workstation agent. The LeakProof server is used by administrators to scan workstations for sensitive information, define security policies, analyze alerts sent by LeakProof agents and generate reports. The LeakProof Agents crawl the system for sensitive information. Additionally they monitor and control all leakage vectors (removable drives, network, etc.). The current version of LeakProof is only able to control Windows workstations.

We built a simulated organization with several workstations controlled by a LeakProof Server. LeakProof uses the concepts of *digital asset* and *company policy* to establish a DLP configuration. Digital assets are defined through the use of keywords and regular expressions. The tested version of LeakProof includes 21 keywords and 36 regular expressions that allow to identify possible sensitive files such as source code files, social security numbers,

[3]Excel 2007 allows a maximum $2^{20} \times 2^{20}$ cell spreadsheets

[4]TrendMicro LeakProof v5. `http://us.trendmicro.com/us/products/enterprise/data-loss-prevention`



	E	F	G	H	I
	SSN	Address	Zip	City	State
972	609-41-9684	1826 Second Road	67455	Mill Creek	NJ
951	707-56-1016	113			
964	728-64-6129	34 H			
951	599-25-1430	1213			
945	936-71-4509	3264			
934	804-81-1967	1898			
957	372-80-8435	1059			
973	260-88-9753	1662			
992	830-83-4684	1677			A
988	312-68-6766	3166			C
005	228-72-6058	2797			Y
996	402-15-2877	1843			A
938	548-38-9639	1900			
967	246-90-6667	4089			
992	840-51-6253	935			A
948	141-41-8234	2221			
982	130-55-8262	3374			Y
977	282-79-7450	49 T			
975	193-76-2009	2911 Ninth Street	18646	West Point	MA

Action required - Please respond promptly. 171

The content you are sending or copying contains sensitive data. Please enter a reason for your action (from 10 to 255 characters), then select "Submit". Your action will be allowed and your reason logged. Select "Cancel" to cancel your action.

Content: ClipBoard

Note: Prohibited symbols - < > ' " ? - are replaced with a space.

Figure 2: LeakProof v5.0 alerts when trying to copy sensitive cells into the clipboard.

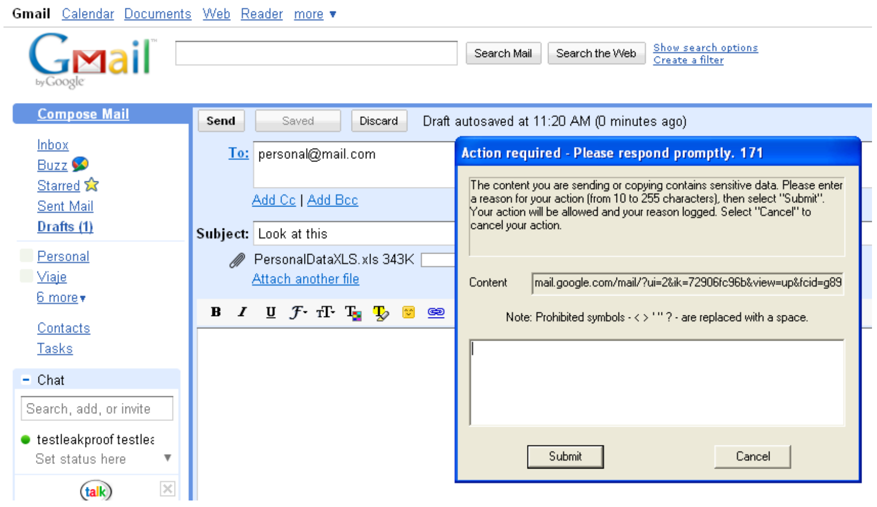
etc. We generated personal data records files[5], which LeakProof correctly identified as sensitive. Company policies are built to control digital assets that are transmitted through specific channels. The server company policy was configured with compliance templates that are built in the LeakProof Server. Specifically, templates to comply with PCI-DSS (PCI Securty Standards Council, 2010), California Senate Bill 1386 of 2002 (SB 1386) and the Gramm-Leach-Bliley Act of 1999 (GLBA) were enabled. Additionally, a compliance template to avoid source code extraction and US personally identifiable information (including social security numbers) were active.

Our tests involved three different methods to try to leak information. First, we tried copying sensitive information from one file to another through the clipboard. Secondly, we tried to mail a sensitive file using a webmail account accessed though HTTPS. Finally, we used our evasion technique to transform the sensitive information. Finally, we tried to send the transformed information using the same webmail account.

Copying sensitive information into the clipboard was detected by the LeakProof Agent, as shown in Figure 2. During our second test, sending a sensitive file using a webmail account, LeakProof did not allow us to at-

[5]To generate fake personal records, we used an Excel plugin available at `http://www.codeforexcelandoutlook.com/wkbks/RandomDataGenerator\_unlocked2007.xlam`

tach the sensitive file (Figure 3). All these operations were logged into the LeakProof server and also reported to the security administrator through his email account. Other operations such as copying to a removable drive and printing the document were also forbidden and logged.



Draft autosaved at 11:20 AM (0 minutes ago)

To: personal@mail.com

Subject: Look at this

PersonalDataXLS.xls 343K

Action required - Please respond promptly. 171

The content you are sending or copying contains sensitive data. Please enter a reason for your action (from 10 to 255 characters), then select "Submit". Your action will be allowed and your reason logged. Select "Cancel" to cancel your action.

Content: mail.google.com/mail/?ui=2&ik=72906fc96b&view=up&fcid=g89

Note: Prohibited symbols - < > ' " ? - are replaced with a space.

Figure 3: LeakProof v5.0 alerts when trying to send a sensitive file by email.

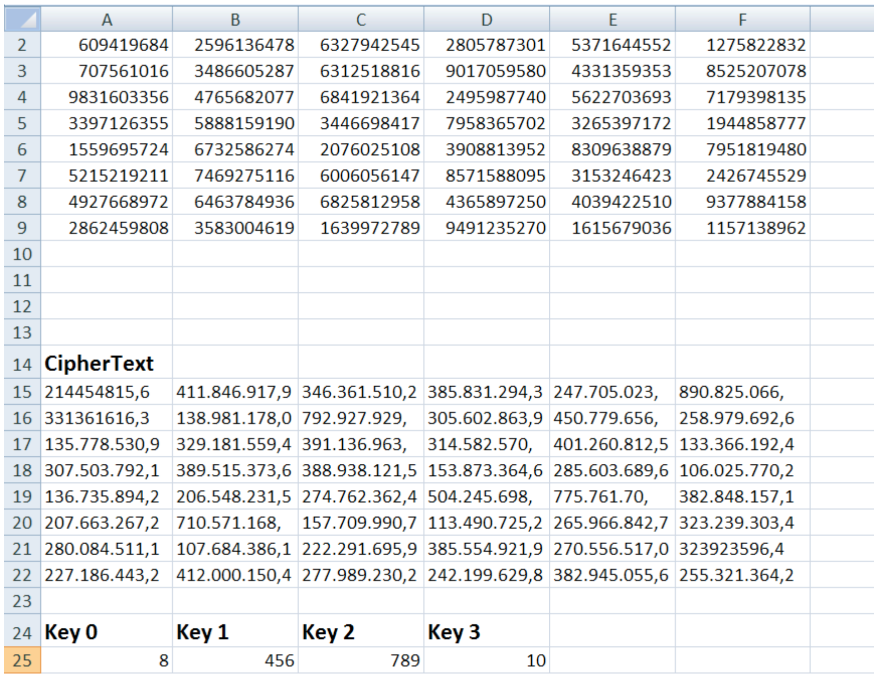
Using one of our spreadsheet ciphers and the previously defined Algorithm 1, we were able to steal social security numbers that we were not able to leak by other means. Figure 4 shows the resulting file, encrypted using the TEA spreadsheet implementation, which we were able to transfer outside the organization using the previously forbidden leakage procedures.

In our case, as the clipboard is actively monitored by LeakProof, we copied the cipher cells into the sensitive file spreadsheet. The encrypted information did not match any of the patterns defined into LeakProof policies (see Table 2).

SSN Pair	LeakProof Search Pattern	ExcelTEA Result
609-41-9684, 707-56-1016	[^ \d-](\d{9}|\d{3}-\d{2}-\d{4})[^ \d-]	2144548156, 3313616163

Table 2: Example of the transformation performed on a social security number to bypass LeakProof detection mechanism.

To leak other sensitive information such as names, address, etc., their values would need to be translated into numeric format using the CODE() function.



	A	B	C	D	E	F
2	609419684	2596136478	6327942545	2805787301	5371644552	1275822832
3	707561016	3486605287	6312518816	9017059580	4331359353	8525207078
4	9831603356	4765682077	6841921364	2495987740	5622703693	7179398135
5	3397126355	5888159190	3446698417	7958365702	3265397172	1944858777
6	1559695724	6732586274	2076025108	3908813952	8309638879	7951819480
7	5215219211	7469275116	6006056147	8571588095	3153246423	2426745529
8	4927668972	6463784936	6825812958	4365897250	4039422510	9377884158
9	2862459808	3583004619	1639972789	9491235270	1615679036	1157138962
10						
11						
12						
13						
14	**CipherText**					
15	214454815,6	411.846.917,9	346.361.510,2	385.831.294,3	247.705.023,	890.825.066,
16	331361616,3	138.981.178,0	792.927.929,	305.602.863,9	450.779.656,	258.979.692,6
17	135.778.530,9	329.181.559,4	391.136.963,	314.582.570,	401.260.812,5	133.366.192,4
18	307.503.792,1	389.515.373,6	388.938.121,5	153.873.364,6	285.603.689,6	106.025.770,2
19	136.735.894,2	206.548.231,5	274.762.362,4	504.245.698,	775.761.70,	382.848.157,1
20	207.663.267,2	710.571.168,	157.709.990,7	113.490.725,2	265.966.842,7	323.239.303,4
21	280.084.511,1	107.684.386,1	222.291.695,9	385.554.921,9	270.556.517,0	323923596,4
22	227.186.443,2	412.000.150,4	277.989.230,2	242.199.629,8	382.945.055,6	255.321.364,2
23						
24	**Key 0**	**Key 1**	**Key 2**	**Key 3**		
25	8	456	789	10		

Figure 4: Social Security Numbers encrypted using the TEA spreadsheet implementation.

6. Discussion

The presented attack allows malicious insiders to bypass DLP solutions and leak information from an organization. Nevertheless, a spreadsheet implementation of a cryptographic algorithm involves some restrictions that do not apply over specific purpose cryptographic applications. This section analyzes how these restrictions affect the feasibility of this method to evade information. Additionally, we propose several countermeasures that could be implemented on current DLP software to reduce the risk associated with the presented attack.

6.1. Feasibility of current DLP evasion technique

Executing encryption algorithms over a spreadsheet application enables the evasion of restrictions imposed by DLP solutions. Nevertheless, this kind of implementation has two drawbacks that may affect its feasibility under a real scenario.

Firstly, the spreadsheet implementation is computationally inefficient. As shown in Section 5.2, the spreadsheet implementation of a cipher is orders of magnitude slower than a standard C implementation of the same cipher. Additionally, as opposite to other cipher implementations, the memory requirements increase with the amount of data to encrypt: a block of cipher implementation cells is required for each block the user wants to encrypt in the same algorithm run. Encrypting large amounts of information causes an overhead on the spreadsheet application that could be easily identified as abnormal behaviour (see Section 6.2.3).

Secondly, the cipher spreadsheet is only able to encrypt information stored in spreadsheet cells. As these cells only allow to introduce textual information, binary information such as images or other file formats that can not be easily represented by readable characters cannot be encrypted. Encrypting such information would require the usage of additional applications to encode the binary information into cell allowed characters. This fact, together with the computational restrictions of the cipher spreadsheet, makes the presented evasion technique suitable only for small amounts of data such as personal data records or small text fragments.

6.2. Countermeasures to the proposed DLP evasion technique

Although the proposed attack enables bypassing current DLP solutions, there exist several techniques and mechanisms that could be used to detect the usage of the proposed technique. The following subsections discuss some of them.

6.2.1. Detecting the presence of cipher spreadsheets

Both the Windows registry and UNIX system logs store system status information and application messages sent to the operating system. (Zax and Adelstein, 2009; Baek et al., 2008) proposed the use of registry entries and system logs to detect malicious insiders and suspicious applications affecting information leakage threats. Microsoft Excel generates entries on the Windows registry keys that could be used as forensic artifacts. We used Systracer[6] to obtain the changes made to the Windows registry after successfully leaking sensitive information. The only change observed was on the following key:

```
HKEY_CURRENT_USER\\Software\\Microsoft\\Office\\12.0\\Excel\\File MRU
```

[6]Systracer 2.0 available at `http://www.blueproject.ro/systracer`

This registry key, which stores the name of the recent opened files, changed to add the name of the file where the cipher was placed. That name is not enough to detect a malicious or suspicious behavior, as it can be changed easily by the malicious insider.

Another approach to detection would be to consider the cipher spreadsheets as some sort of malware. After all, the main purpose of the cipher workbook file is arguably to steal information. Antivirus and Intrusion Detection Systems may be adapted to detect the transmission or usage these kinds of spreadsheets on a computer system. Regarding the AES implementation, its S-Boxes could be used as a signature to detect the presence of the Excel cipher implementation. A new IDS rule could be written to detect such content into incoming network traffic.

Nevertheless, our TEA and AES implementations are just an example. Virtually thousands of different files with the same purpose can be created and used easily, as well as other completely different implementations of other ciphers. This implies that it will be quite easy to develop new alternatives unknown to antimalware software that could be employed without any risk of being detected. However, these kinds of spreadsheets share some common characteristics that could facilitate their detection, such as the usage of a short set of functions and the high amounts of nested formulas required to steal practical amounts of information.

6.2.2. Encrypted data detection

DLP solutions inspect contents and file metadata as well as network packets to detect sensitive information leaving the organization (Lawton, 2008; Baek et al., 2008). One central property of modern ciphers is the randomness of the ciphertext they produce (Soto, 1999). Randomness tests could be used to detect encrypted content being transferred out of the organization. However, spreadsheet encrypted information could be transformed to reduce its apparent redundance or transmitted wrapped under file formats such as PDF, DOC, etc. This should be taken into account when designing encryption data detectors.

Using the software "Glavlit" (Schear et al., 2007), any file to be sent outside the organization must pass through a warden. A specific warden could be created to detect files produced with our cipher spreadsheets, usually Microsoft Excel or other format files with random contents. However, other innocent-looking information could be easily introduced into the file to confuse the warden.

6.2.3. Detection of abnormal behavior

The use of our cipher spreadsheet requires the malicious insider to follow a specific procedure. All these actions are done at the application level: opening a sensitive file, performing some actions on it and storing the information in a new file. The generation of fine grained in-application log data could help to detect these kinds of malicious insider attacks. Events such as "user opens sensitive file", "sensitive cells are used as input for certain cells", etc. give accurate information about the user behavior. Data mining techniques could be used over this data to build information leakage detectors. A similar approach was proposed by Schonlau et al. to detect malicious insiders using running processes (Schonlau et al., 2001).

Additionally, the overhead that the cipher spreadsheet implementations produce on the system when encrypting (Section 5.2) could be used as a hint to detect a malicious insider. Nevertheless, this depends on the amount of information to encrypt. Depending on the organization's activities, these variations can also be attributed to other legitimate Excel operations or other user activities, such as web navigation, etc.

6.2.4. Security models to avoid data leaks

Implementing application level restrictions on specific user actions concerning sensitive information may help to avoid these evasion technique, i.e. forbid to reference a cell whose value is a piece of sensitive information. In this regard, the usage of Enterprise Digital Rights Management Solutions (Yu and Chiueh, 2004) would likely defeat our attack. These solutions should be correctly configured through the security policy. Additionally, authors should not have control of their own documents, as they may simply reduce the security level of the document to steal it.

Finally, the usage of multilevel security models (Section 2) could be useful to forbid information leakage using our evasion technique. In the Bell-LaPadula model, when the malicious insider opens the sensitive information document he is no longer able to create non sensitive documents. Nevertheless, some work is still needed in order to adapt these kind of models to commercial environments (Rubinovitz, 1994).

Other techniques such as process coloring, could be useful to trace back the origin of the leak (Jiang et al., 2008). If an application opens a sensitive information file, it propagates its color (a unique identifier) to other opened documents. DLP solutions could be configured to deny transmission of certain colors outside the organization.

7. Conclusions and future work

Information leakage is one of the more rapidly growing threats that organizations face nowadays. The work presented in this paper demonstrates that trusted applications may entail a threat to the confidentiality of sensitive information. This threat has not been addressed by current DLP systems and could allow malicious insiders to leak sensitive information by using trusted applications. Specifically, we show how to implement a standard cipher in a simple and widely used spreadsheet application. As actions performed within the trusted application environment are not controlled, information can be encrypted and sent outside the organization without being detected.

Our approach could be used by malicious insiders to steal information from organizations already protected by malicious insider detection or DLP systems. We have tested our approach against a commercial DLP solution and demonstrated that it is able to extract sensitive information from a simulated organization without being detected. Additionally, we have also pointed out some of the mechanisms that could be implemented on current DLP software to detect and avoid our attack.

Our future work moves in various directions. We are working towards implementing more cryptographic algorithms to validate our approach. Additionally, we are also exploring the use of steganographic techniques by means of trusted applications, in such a way that the transformed information can be part of an innocuous-looking message. This will allow it to pass unnoticed by a warden who is supossed to vet all information exiting an organization, as in (Schear et al., 2007). In terms of countermeasures, we are looking forward to further develop and implement the necessary mechanisms to take into account actions performed inside trusted applications. Advances in this direction would enable to protect against this kind of DLP evasion technique.

Acknowledgements

We would like to specially thank the anonymous reviewers for providing us suggestions that have helped us to improve the content and quality of this paper.

2010 E-Crime Watch Survey. Technical Report; CSO Magazine in cooperation with U.S. Secret Service, Carnegie Mellon University Software Engineering Institute's CERT Coordination Center and Deloitte; 2010.

http://www.cert.org/archive/pdf/ecrimesummary10.pdf Accessed on January 2011.

Anderson, R.. Security Engineering: A Guide to Building Dependable Distributed Systems. Wiley, 2001.

Baek, E., Kim, Y., Sung, J., Lee, S.. The Design of Framework for Detecting an Insider's Leak of Confidential Information. In: Proceedings of the 1st international conference on Forensic applications and techniques in telecommunications, information, and multimedia and workshop. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering); 2008. p. 1–4.

Bell, D., LaPadula, L.. Secure Computer Systems: Mathematical Foundations and Model. Technical Report ESD-TR-73-278; Mitre Corporation; 1973.

Caputo, D., Maloof, M., Stephens, G.. Detecting Insider Theft of Trade Secrets. IEEE Security and Privacy 2009;7:14–21.

Caputo, D., Stephens, G., Stephenson, B., Cormier, M., Kim, M.. An Empirical Approach to Identify Information Misuse by Insiders (Extended Abstract). In: Recent Advances in Intrusion Detection. Springer Berlin / Heidelberg; volume 5230 of *Lecture Notes in Computer Science*; 2008. p. 402–403.

Chivers, H., Nobles, P., Shaikh, S.A., Clark, J.A., Chen, H.. Accumulating Evidence of Insider Attacks. In: 1st International Workshop on Managing Insider Security Threats. Purdue University, West Lafayette, USA; 2009. p. 34–50.

Cox, M., Engelschall, R., Henson, S., Laurie, B., Young, E., Hudson, T.. OpenSSL. 2001. http://www.openssl.org Accessed on January 2010.

Dobbertin, H., Knudsen, L., Robshaw, M.. The Cryptanalysis of the AES - A Brief Survey. In: Advanced Encryption Standard - AES. Springer Berlin / Heidelberg; volume 3373 of *Lecture Notes in Computer Science*; 2005. p. 571–572.

Farahmand, F.. Insider Behavior: An Analysis of Decision under Risk. In: 1st International Workshop on Managing Insider Security Threats. Purdue University, West Lafayette, USA; 2009. p. 22–33.

Fisk, G., Fisk, M., Papadopoulos, C., Neil, J.. Eliminating Steganography in Internet Traffic with Active Wardens. In: Revised Papers from the 5th International Workshop on Information Hiding. London, UK: Springer-Verlag; IH '02; 2003. p. 18–35.

Hong, S., Hong, D., Ko, Y., Chang, D., Lee, W., Lee, S.. Differential Cryptanalysis of TEA and XTEA. Information Security and Cryptology-ICISC 2003 2004;:402–417.

Jiang, X., Buchholz, F., Walters, A., Xu, D., Wang, Y.M., Spafford, E.H.. Tracing Worm Break-In and Contaminations via Process Coloring: A Provenance-Preserving Approach. IEEE Transactions on Parallel and Distributed Systems 2008;19:890–902.

Jiang, X., Walters, A., Xu, D., Spafford, E.H., Buchholz, F., Wang, Y.M.. Provenance-Aware Tracing of Worm Break-in and Contaminations: A Process Coloring Approach. International Conference on Distributed Computing Systems 2006;0:38.

Kelsey, J., Schneier, B., Wagner, D.. Related-key cryptanalysis of 3-way, Biham-des, cast, des-x, newdes, rc2, and tea. Information and Communications Security 1997;:233–246.

Ko, Y., Hong, S., Lee, W., Lee, S., Kang, J.. Related Key Differential Attacks on 27 Rounds of XTEA and Full-round GOST. In: Fast Software Encryption. Springer; 2004. p. 299–316.

Latham, A.. JPHide and JPSeek. 1999. `http://linux01.gwdg.de/~alatham/stego.html` Accessed on January 2010.

Lawton, G.. New Technology Prevents Data Leakage. Computer 2008;41(9):14–17.

Liu, Y., Corbett, C., Chiang, K., Laboratories, S.N., Archibald, R., Ghosal, D.. SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack. In: 42nd Hawaii International Conference on System Sciences HICSS '09. IEEE; 2009. p. 1–10.

McCormick, M.. Data Theft: A Prototypical Insider Threat. Insider Attack and Cyber Security 2008;39:53–68.

Moore, A.P., Cappelli, D.M., Caron, T.C., Shaw, E., Trzeciak, R.F.. Insider Theft of Intellectual Property for Business Advantage : A Preliminary Model. In: 1st International Workshop on Managing Insider Security Threats. Purdue University, West Lafayette, USA; 2009. p. 1–22.

National Institute of Standards and Technology, . FIPS 197: Advanced Encryption Standard. Technical Report; National Institute of Standards and Technology; 2001.

PCI Securty Standards Council, . Payment Card Industry (PCI) Data Security Standard - Requirements and Security Assessment Procedures. Technical Report; PCI Securty Standards Council; 2010.

Petitcolas, F.A.P.. MP3Stego. 1998. `http://www.petitcolas.net/fabien/steganography` Accessed on January 2010.

Pfleeger, S.L., Predd, J.B., Hunker, J., Bulford, C.. Insiders Behaving Badly: Addressing Bad Actors and Their Actions. IEEE Transactions on Information Forensics and Security 2010;5(1):169–179.

Ponemon Institute, . 2008 Annual Study: Germany Cost of a Data Breach. Technical Report; Ponemon Institute; 2008.

Quellet, E., Proctor, P.. Magic Quadrant for Content Monitoring and Filtering and Data Loss Prevention. Technical Report; Gartner Group; 2008.

Randazzo, M., Keeney, M., Kowalski, E., Cappelli, D., Moore, A.. Insider Threat Study: Illicit Cyber Activity in the Banking and Finance Sector. Technical Report CMU/SEI-2004-TR-021; Carnegie Mellon Software Engineering Institute; 2005.

Rantala, R.R.. Cybercrime Against Businesses, 2005. Technical Report; U.S. Department of Justice; 2008.

Raschke, T.. The Forrester Wave: Data Leak Prevention, Q2 2008. Technical Report; Forrester; 2006.

Rubinovitz, H.. Issues Associated with Porting Applications to the Compartmented Mode Workstation. ACM SIGSAC Review 1994;12(4):2–5.

Schear, N., Kintana, C., Zhang, Q., Vahdat, A.. Glavlit: Preventing Exfiltration at Wire Speed. In: Proceedings of 5th Workshop on Hot Topics in Networks (HotNets). 2007. .

Schonlau, M., DuMouchel, W., Ju, W.H., Karr, A.F., Theus, M., Vardi, Y.. Computer Intrusion: Detecting Masquerades. Statistical Science 2001;16(1):58 – 74.

Simonian, H., Goff, S.. Data theft hits 24,000 HSBC clients. Financial Times 2010;:5`http://www.ft.com/cms/s/0/aa0b9e2e-2ced-11df-8025-00144feabdc0.html` Accessed on January 2010.

Skala, M., Roth, M., Hernaeus, N., Guyomarch, R., Koch, W.. The GNU Privacy Guard. 2010. `http://www.gnupg.org` Accessed on January 2010.

Soto, J.. Randomness Testing of the Advanced Encryption Standard Candidate Algorithms. Technical Report; National Institute of Standards and Technology; 1999.

Walker, T.. Practical Management of Malicious Insider Threat - An Enterprise CSIRT perspective. Information Security Technical Report 2008;13(4):225–234.

Wheeler, D., Needham, R.. TEA, a Tiny Encryption Algorithm. In: Fast software encryption: Second International Workshop. Springer Berlin / Heidelberg; 1995. p. 363–366.

Wray, R.. T-Mobile confirms biggest phone customer data breach. The Guardian 2009;`http://www.guardian.co.uk/uk/2009/nov/17/t-mobile-phone-data-privacy` Accessed on January 2010.

Yu, Y., Chiueh, T.c.. Enterprise Digital Rights Management: Solutions against Information Theft by Insiders. Technical Report; Department of Computer Science, Stony Brook University; 2004.

Zander, S., Armitage, G., Branch, P.. Covert Channels and Countermeasures in Computer Network Protocols [Reprinted from IEEE Communications Surveys and Tutorials]. Communications Magazine, IEEE 2007;45(12):136–142.

Zax, R., Adelstein, F.. FAUST: Forensic Artifacts of Uninstalled Steganography Tools. Digital Investigation 2009;6(1-2):25–38.