Journal of Network and Computer Applications 145 (2019) 102405

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

Using malware for the greater good: Mitigating data leakage

Mordechai Guri [*], Rami Puzis, Kim-Kwang Raymond Choo, Sergey Rubinshtein, Gabi Kedma, Yuval Elovici

Cyber-Security Research Center, Ben Gurion University of the Negev, Israel

ARTICLE INFO

Keywords:
Data leakage
Insider threat
Malware signature
Data exfiltration

ABSTRACT

Accidental (i.e., non-malicious) data leakage can occur through emails, storage media, file-sharing services, social networks, and so on, and are one of the most commonly reported threats. We present *DocGuard*, a novel method designed to counter accidental data leakage. Unlike existing solutions, *DocGuard* is effective even when a file has *already leaked out* of the organization's network. However, our approach does not require additional installation or software update, outside the organizational network, and it supports virtually any type of file (e.g., binaries, source-code, documents and media). Specifically, the key idea is to let existing anti-malware/anti-virus (AV) products (at the user PCs, cloud services, ISPs and e-mail gateways) identify the leaked file and block access to the identified file, in the same manner the AV product stops the propagation of an identified malware. DocGuard injects a hidden signature associated with a known malware to sensitive files. If the files are somehow leaked out of the organization's boundaries, an AV, either on the user's PC or at the network, will detect it as a real threat and immediately delete or quarantine it before it can be accessed and shared further. We implement DocGuard and evaluate it on various file types including documents, spreadsheets, presentations, images, executable binaries and textual source code. Our evaluations include different leakage paths such as e-mails, file-sharing and cloud services, social networks and physical media. The evaluation results have demonstrated almost 100% effectiveness in stopping the leakage at its initial phases. In order to evaluate DocGuard at a larger scale, we simulate a leakage scenario over the topology of real social networks. Our results show that DocGuard is highly effective not only for stopping the initial leak but also in preventing the propagation of leaked files over the Internet and though social networks.

1. Introduction

Information leakage is a concern that has been raised by scholars, governments and organizations, and has been extensively studied. Most of the existing information leakage studies have focused on threats from both internal and external attackers, rather than non-malicious/accidental leakage (Hoang et al., 2017; Hu et al., 2017). In the latter category, sensitive data can be leaked due to a number of reasons (Cheng et al., 2017a; Guri et al., 2014; Asaf et al., 2012; Papadimitriou and Garcia-Molina, 2010). For example, the employee is not aware of the sensitivity of the data, and/or sensitive data (e.g., classified materials) are somehow mixed with non-sensitive data during the preparation of a report. For example in the incident involving Boeing, a spreadsheet sent by one of the employees included hidden columns, which contained “sensitive, personally identifiable information of 36,000 Boeing employees, including names, places of birth, BEMSID, or employee ID numbers, and accounting department codes” (Paganini, 2017).

There are many avenues where sensitive data can be leaked accidently, such as online (e.g., email, file sharing services, mobile devices, cloud services, and social networks) (Paganini, 2017; Lab, 2017; Shane et al., 2017; Fagundes, 2014; WikiLeaks, 2016; SafeSend, 2016; Quick and Choo, 2013), or offline (e.g., removable media and printed media) (Cisco, 2014; Vasciannie, 2015; Romer, 2014) – see Fig. 1.

While it may not be possible to entirely prevent data leakage, we seek to minimize the consequence of such leakages. Specifically, in the case of accidental leakage, when the unintended recipient receives some sensitive data, he/she will be dissuaded from opening or accessing the data. However, we should also assume that most users are honest-but-curious and this is also a typical assumption used in the design of cryptographic protocols where the researchers assume that there exists an honest-but-

[*] Corresponding author. Cyber-Security Research Center, Ben Gurion University of the Negev, Israel.
E-mail addresses: gurim@post.bgu.ac.il (M. Guri), puzis@bgu.ac.il (R. Puzis), raymond.choo@fulbrightmail.org (K.-K.R. Choo), sergeyru@post.bgu.ac.il (S. Rubinshtein), gabik@post.bgu.ac.il (G. Kedma), elovici@bgu.ac.il (Y. Elovici).

https://doi.org/10.1016/j.jnca.2019.07.006
Received 22 September 2018; Received in revised form 26 March 2019; Accepted 10 July 2019
Available online 14 July 2019

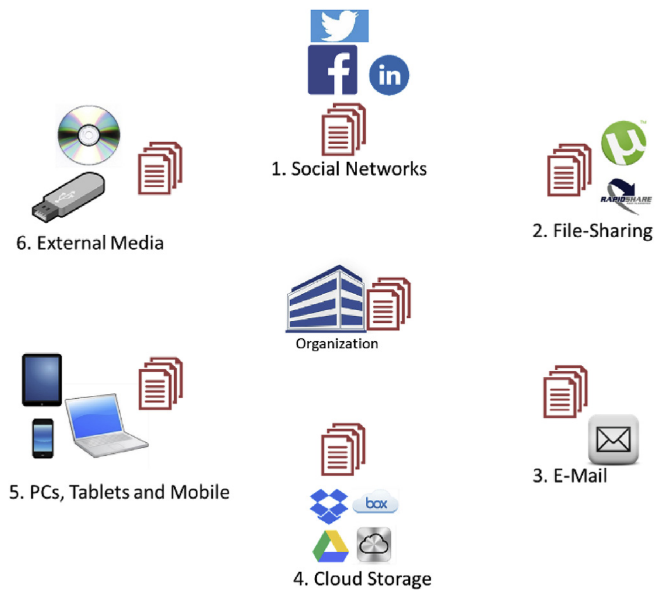


Fig. 1. Common leakage paths: social networks, file-sharing, e-mails, public clouds, mobile devices and external media.

curious adversary and/or third-party (including the cloud service provider that hosts the user's data) (Liu et al., 2016, 2018).

Therefore, in this paper, we propose embedding a hidden signature of a known malware to each sensitive document within the organization's boundaries. Hence, if the message is leaked outside the organization, it will be flagged by most anti-malware/anti-virus (AV) programs as malicious. Hence, the recipient of a message (e.g., file) that was accidently leaked will not likely open the message since it will be quarantined or deleted and the user will be alerted about the blocked content. In other words, the leaked message would be stopped by the host AV, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) in email gateways, online services (e.g., file-sharing and social networks) and Internet Service Providers (ISPs).

In the next section, we will review related literature.

2. Related work

At the time of this research, there is no effective data leakage prevention (DLP) approaches to prevent or limit the spread of a document *outside* the organizational boundaries. DLP (also known as Information Leakage Prevention – ILP) has been extensively researched by researchers (Cheng et al., 2017a, 2017b; Asaf et al., 2012; Kongsgard et al., 2017; Kongsgarrd et al., 2017; Peng et al., 2016) and there are also a number of existing commercial products (Alneyadi et al., 2016). For example, to mitigate data leakage due to a malicious or corrupted insider, there have been a number of solutions in the literature, such as those based on honeypots or honey-tokens (Jogdand and Padiya, 2016), and so on (Guevara et al., 2017).

Data that have been leaked can potentially be traced using 'watermarking' or by unobtrusive techniques (Papadimitriou and Garcia-Molina, 2010; Govinda and Joseph, 2017). With the increased usage of cloud to store and process data (cloud computing), existing DLP approaches have also been extended to support/meet the technological requirements of data stored on public and private cloud servers (Govinda and Joseph, 2017; Choo, 2010). For example, some approaches suggested encrypting virtual machine (VM) image in the cloud, in order to prevent internal attackers from violating the data access policies (Narayana and Pasupuleti, 2018). A trusted third party monitors the VMs in the cloud to maintain the integrity of VMs. Priebe et al. presented CloudSafetyNet, a method designed to detect accidental data leakage between cloud tenants, especially in Platform-as-a-Service (PaaS) clouds (Priebe et al., 2014). With the emerging usage of mobile devices (smartphones, tablets, etc.) and the Bring Your Own Device (BYOD) trend, some solutions focus on data leakage of sensitive data from mobile devices. Examples of solutions include using Trusted Execution Environments (TEEs) to store confidential data, and ARM TrustZone to maintain isolation between trusted and non-trusted operation systems (OS) (Anon). Geolyang et al. proposed a DLP solution for mobile devices using hypervisors on ARM architecture to secure data between the Secure Domain (SD) and Normal Domain (ND) compartments (Kim et al., 2017).

Other approaches in the literature have focused on mitigating data leakage at three stages, namely: data-at-rest, data-at-motion and data-in-use. Endpoint solutions enforce access policies in desktop machines, and thus can prevent confidential documents from leaving the organization boundaries. Guevara, Santos and Lopez (Guevara et al., 2014) proposed leveraging the user's historical data access (e.g., typical hour of access, duration, day of the week; and user's typical operation) to flag suspicious user behavior. Network traffic monitors can also be used to continuously analyze the network communication to identify whether a sensitive file was sent, which is in violation of the organization's security policies (Priebe et al., 2014). Chea et al. proposed a private data DLP method for peer-to-peer (P2P) networks using the file sharing hardening approach (Chae et al., 2016). Another network level approach is to use file hashes groups and tags to track data leakage and document propagation within a network (Gula and Ranum, 2018). File-level systems can also embed security-related information as metadata in sensitive files (Asaf et al., 2012; Zhang et al., 2009; Phua, 2009). In this approach, traffic scanners at the network level and file scanners at the computer level are blocking the leakage of sensitive files.

Note that the aforementioned solutions are only effective within the organization's perimeter. Digital right management (DRM) solutions (Microsoft, 2015a) can also prevent unauthorized users, within or outside the organization's boundaries, from reading document, which was accidently sent to them. However, DRM solution is limited to certain type of documents and it does not prevent the file from spreading further on the external network once it is leaked. It also does not prevent a skilled hacker from attempting to decipher the contents of the file (D'Orazio and Choo, 2016). Another common solution is to encrypt sensitive files, preventing it from being opened in a readable form in a non-authorized environment (Zhang et al., 2009). However, leaked documents can be subject to offline brute-force attacks and decrypted.

Our method, on the other hand, may halt the initial leakage, and limit the propagation of the leaked document, by preventing access to it. The presented solution can be used along with encryption-based methods to limit the harmful effects of accidental data leakage. Here, we consider another goal: "Assuming that a sensitive digital document has already leaked, how do we contain the leakage to minimize the damage?". Our concept is designed to work for a wide range of documents, and aims to impede and limit the spreading of the leaked file beyond the organization's boundaries. It also aims to prevent access to the leaked file, preferably by having it deleted before unauthorized users have the opportunity to access it.

Table 1 shows the applicability of various solutions to (a) the initial leak, and (b) further spreading of the leaked document. Our method is the only one which is applicable against further spreading, beyond the boundaries of the organization.

In our preliminary work (Guri et al., 2014), we proposed that attaching (or 'injecting') a virus signature that would be detected by common AV tools. We conducted thorough experiments with various injection methods, virus signatures, and AV programs in order to determine the ideal marking signature. However, our previous work did not provide a thorough testing and evaluation of the method in of leaked files through different leaking paths.

3. DocGuard design and architecture

In this section, we describe the basic design considerations, which

Table 1
Applicability-range of various approaches.

Solution	Techniques	Inside Organization	Outside Organization
End-point solutions	Access-controls (Cheng et al., 2017a), Behavioral monitors, Mobile device security (Anon), trusted execution environments (TEEs) (Kim et al., 2017), Encryption (Zhang et al., 2009)	Applicable	Not applicable
Network monitoring	Traffic scanners, Packet inspection, cloud monitors (Liu et al., 2018; Priebe et al., 2014), service hardening	Applicable	Not applicable
File System Monitoring	File scanners (Gula and Ranum, 2018), activity scanners (Asaf et al., 2012), behavioral monitors (Guevara et al., 2014)	Applicable	Not applicable
Watermarking	File watermarking (Papadimitriou and Garcia-Molina, 2010; Govinda and Joseph, 2017)	Applicable	Not applicable
Right Management & Access Control	Access-control, Behavioral monitors (Guevara et al., 2014), Mobile device security,	Applicable	Not applicable
DocGuard (this work)	Embedded malicious signature	Applicable	Applicable

must be met in order to provide a practical solution.

Encounter the leakage via different paths: There are several ways data (e.g., confidential documents) can be leaked from the organization's secure perimeter, for example to a user's BYOD device or the user's external cloud storage account that are outside the organization's reach. Thus, there is a need to develop approaches that will limit the spread of the leaked data. Our goal is to limit different leaking paths, such as social networks, file-sharing services, e-mails, cloud, personal computers (PCs), tablets and smartphones, and external media.

Coverage of Prevalent Document Types: The goal is to design a method that will prevent the spread of a large percentage of information used within a typical organization. Thus, in the case we have examined, the method was required to cover at least (a) Microsoft Office documents, (b) Adobe PDF documents, (c) common media files (images and movies) and (d) source-code files.

Detectability Using Prevalent AV Programs: Assume that the new method is based on injecting a hidden virus signature to each sensitive document within the organization's boundaries. The signature should be detectable by major prevalent AV programs. We have decided to focus on the relevant products of the 'top ten' AV vendors as listed by (AV vendors, 2018), which account for a large percentage of the existing platforms and market share.

Effective Action Taken by the AV Program: Being detectable by AV program is not sufficient. The injected signature should also cause the AV program to take effective action. An effective action would be either deleting the entire document or putting it into quarantine. In the case of gateways such as mail servers with traffic filtering, the transmission of the document should be prevented. Preferably the user should not be given a permissive choice.

Transparent Deployment: Injecting the signature to the sensitive file within the organization should be done automatically, and in a manner that is transparent to the user. This process should be applied to all endpoint computers within the organization's boundaries. The system should be able to attach the signature to any document, which meets a certain level of classification. The system should be able to notice whenever such a document is saved to disk or 'printed' to a file (as in the case of a PDF when created from a Microsoft Office application). All these tasks should be carried on in computers, which are monitored by an active AV program and without affecting or disturbing the standard work of the user.

3.1. *AV signature-based detection*

A typical AV program employs an updatable database of signatures of known malware, and a monitor, which scans new or modified files for the presence of such signatures. A signature may be as simple as the hash value or 'checksum' of a whole file, but modern viruses use polymorphic or metamorphic methods (Kirat et al., 2014), so the signature should consist of fixed sequences within the file, which indicate malicious codes. Often the signature is like a regular expression, where some parts of the sequence may be fixed while others may vary (Christodorescu and Somesh, 2004). The AV program scans executable files and certain types of data files, which may contain an executable code. Modern binary executable files such as PE on Microsoft Windows (Microsoft, 2010) or ELF on Linux (The Linux man-pages project) have a well-documented format, which is available to attackers and defenders alike. Malicious code may reside in various sections of the file, and a malicious executable may be packed or embedded within another executable (either as a 'resource' or otherwise). Malicious code may be prepended, inserted or appended to innocuous files. Thus, the filename extension is not enough to determine the true nature of a given file. Furthermore, certain kinds of data files, like Microsoft Office documents and Adobe PDF files, may contain an executable code in the form of macros, embedded binary executables, embedded vulnerable files like Flash SWF, and shellcode (e.g., binary code sequences, which are not packaged as a standard binary executable). Other kinds of data files may contain shellcode, which is designed to execute upon a buffer overflow or a similar vulnerability's exploit. Malicious code may be packed, encoded or obfuscated to hinder its detection by malware analysts and AV programs (Miramirkhani et al., 2017; Honig, 2014).

Modern AVs maintains a kernel driver (e.g., File-system filter driver in Microsoft Windows OS), which monitor every file creation and modification. The monitored file might be a file locally created, downloaded from the Internet or retrieved via external media. Each file is then identified by its format and passed through a set of file *parsers*, *uncompressors*, and *extractors*. The File format identification is the process of figuring out the format of a sequence of bytes and is done using simple rules such as header signatures or heuristically. The current top-10 AVs are capable of identifying hundreds to thousands of file formats. The uncompressors, parsers and extractors are the recursively uncompress the file, and extract internal files, if exists. Note that document files and executable might contains an embedded files as well. For example, office documents (.docx, pptx, xlsx, etc.) might include an embedded executable file as an attachment and Windows PE may include DLL files hidden in its.rsrc (resource) section. All he extracted files are then passed through a set of signature and heuristic scanners to grade the probability of the file to be malicious. The signature scanners try to locate a series of bytes in the file. It could also be a hash of the file or its sections. Many of the signature scanners are using YARA rules (or similar regular expression framework) to define complex signatures of malicious files. For example, a complex rule may include three sequences of bytes in the file, where the location of each sequence is unknown in advance. Because many of the modern malware instances are encrypted and obfuscated, the signatures are usually taken from the invariant part of the file, e.g., the loader or the unpacker. Heuristic scanners aims at generically detecting malware by statically analyzing files for malicious or suspicious characteristics without an exact signature match. For instance, by looking for the presence of suspicious APIs in the import table, or by locating special instructions or obfuscated code in the examined file.

3.2. *Injected virus signature*

DocGuard based on the capability of AV to detect virus signature.

Injecting a virus signature to a file practically means adding sequence of bytes to the file at some location. This operation take place during the file creation and file save events. The file itself is saved on disk with the embedded virus signature, which is also referred to as a ‘tag’. If the file itself is leaked somehow outside the organization, it will contains the embedded signature. A dedicated kernel module is responsible to eliminate the embedded signature when the file is opened by an application (include the organization AV), to enables its functionality. The preliminary work at (Guri et al., 2014) broadly discuss the type of malware signatures and their detection rates along with the best injecting location. We refer the interested reader to (Guri et al., 2014) for better understanding the technical consideration regard malware signature and injection place.

3.3. *AV prevalence on end-points*

The widespread use of AV software among Internet users can play a major role in DocGuard effectivity and architecture. According to (AV vendors, 2018), the percentage of computers globally that have an active AV program installed is approximately 85%. The top AV vendors in 2017–2018 are: Avast, Microsoft, AVG, Avira, Symantec, McAfee, ESET, Kaspersky Lab, Sophos and Bitdefender.vTogether, the top ten AV vendors hold 85% of the market, leaving only the remaining 15% to others. Note that the global percentage of AV usage is expected to increase over the next few years due to growing awareness of malware threats. This is demonstrated by Microsoft Windows 10 OS which include malware protection activated by default and AMSI (Antimalware Scan Interface) support (Microsoft, 2015b).

3.4. *DocGuard marker*

When a file is leaked to the Internet, it contains the DocGuard signature (the ‘tag’). The tag might be used by adversaries to scan the Internet for sensitive files and collect them. E.g., by scanning mail severs for attachments that include DocGuard tags. Note that in order to achieve such scanning capabilities, the adversary must have a cooperation in the services (cloud, storage, mail servers, etc.). However, the adversary must know *in advance* the exact DocGuard signature(s) used to tag these sensitive files. It is important to remember, that the DocGuard signatures are meant to be kept secretly in the internal networks, and not be known to attackers outside the organization. Without knowing the exact signature in advance, the file is seen by the online systems as any type of malware spread over the Internet. Moreover, in most of the cases, the file itself - once scanned and detected as a malware – is removed from the online service and cannot be accessed further (e.g., Google drive and Dropbox). This reduces the possibility of accessing and crawling the file, even when the adversary know the DocGuard signature in advance.

4. Implementation and evaluation

4.1. *Prototype*

We implemented a prototype of DocGuard which handle the signature injection within the organization's perimeter (by ‘injecting’ we mean attaching a virus signature to a file). In general the DocGuard engine ensure that sensitive files on every computer within the organization are injected with a virus signature. Applications reading the sensitive document receive the clean content of the file, without the virus signature. Applications such as internet browsers, email clients and their like, which receive the whole file include the virus signature. When an application writes to a sensitive file, the new or altered information should not overwrite the signature. DocGuard handles variety of file types, including Microsoft Office documents (e.g.: DOC, DOCX, XLS, XLSX), Adobe PDF files, and binary executable files (EXE, DLL). The injection mechanism is coordinated with the AV programs which are used within the organization, to avoid false alarms.

We developed appropriate kernel driver for Microsoft Windows 7, 8 and 10 as the target platforms. We built a kernel-mode *Filesystem Minifilter driver* and adding our specific functionalities. We used the Windows Filter Manager architecture (Microsoft) to extend the file system and hook some of its functionality. The architecture of the kernel mode filter is illustrated in Fig. 2. Mini-filters (4, 5) are managed by the filter manager (3) and are placed at a given ‘altitude’ (the numerical level of a filter, relative to other drivers). Our filter (5) is placed lower than the AV filter (4). Fig. 2 shows the response of our filter driver to a read or write request on a tagged file, where the request was made by a proper process and the requested operation is performed on the untagged contents. Please note that there are processes that will be provided with the full content (including the virus signature) since they be used unintentionally to leak the file. For example, processes such as Web browsers and e-mail clients, are blacklisted. During *save* and *write* operation, our filter driver check whether the virus signature is already injected into the file and inject it accordingly. During the *open* and *read* operations, the filter driver eliminate the virus signature form the file and pass it to the next filter

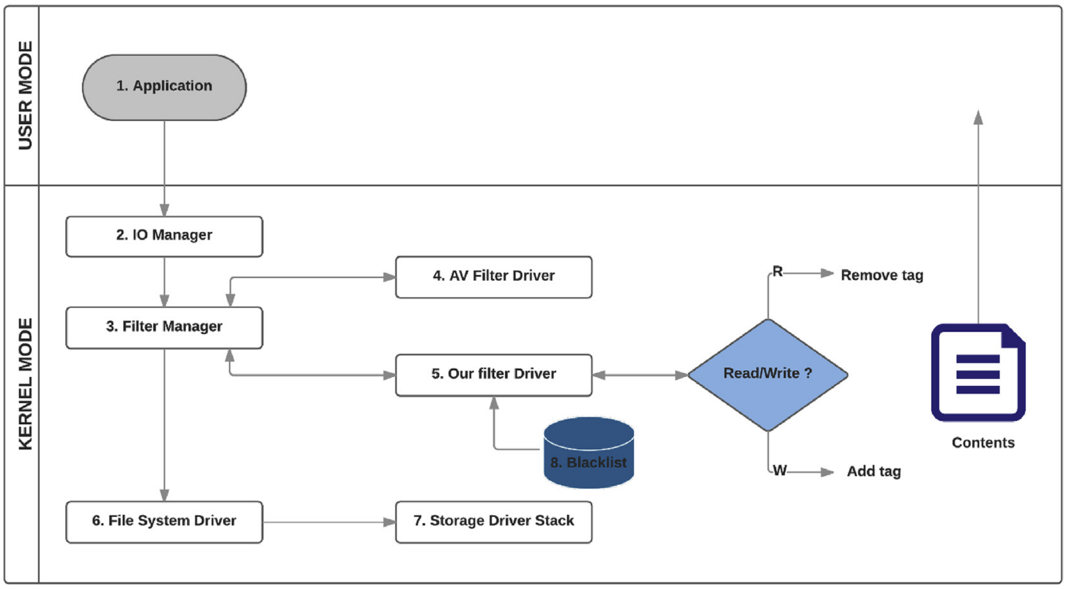


Fig. 2. The filter's conceptual architecture, showing response to a read or write request on a ‘tagged’ file (where the requesting process is not black-listed).

driver (3,4). This way, the injected virus signature (a 'tag') is transparent to the legitimate process at kernel and user level (1). Our filter driver was successfully tested on various AV and applications, including Microsoft Office applications (Word, Excel, and PowerPoint), Adobe Acrobat reader and Windows Media player.

The kernel Driver works in three modes.

- **List mode.** Only files in a closed list are included in the injection process. Only administrator can manage the content of the list.
- **Automatic mode.** All files of certain types are automatically considered to be sensitive and handled by the driver. The type of files are loaded from XML file which is managed by the administrator.
- **Rules mode.** Only files that meets given criteria are handled by DocGuard. E.g., documents which certain title or metadata. The rules are loaded from XML file which is managed by the administrator.

Millions of malware samples and malware signatures are available online, such as (Seltzer, 2019; VirusTotal, 2018). Our kernel driver implements different types of signature injection:

1. A mixture of malware signatures are included as byte-arrays within the file. The signatures strings are not harmful, and several hundreds of them can be included within a file, to raise the probability of detection.
2. Benign version of a generic attack pattern (e.g. heap-spray (Stevens, 2011)). The source code of the program actually includes some well-known exploitation code which is detected by AVs, but does not use it for malicious purposes.
3. Sterilized versions of malicious files. The binary code or the header of some well-known malware is modified to avoid malicious operations or overall execution.
4. EICAR (European Expert Group for IT-Security), a benign signature used to test AV programs compliance.
5. Modern malware signature. These signatures generally have highest detection rates.
6. Nonfunctional malware are still detected by most AV programs, although in general they do fails to run on modern operating systems, due to differences in memory management, executable files structure, and low-level behavior (Symantec; Ször, 2005). Among thousands of samples we choose old virus (Securelist) for testing. Similar options are viruses which are unable to run on modern 32/64 bits OS.

The implicit assumption behind this research is that the DMS alone should not cause any harm and should not bring about any predictably harmful side effects. This assumption is a mandatory requirement due to security considerations. Within the corporate's boundaries, all active AV programs should be configured to exclude the given DMS from their detection lists or to override their default behavior when detecting the given DMS. Thus, the corporate's computers are not protected from the real threat mimicked by the DMS. It is therefore, essential that the chosen DMS does not affect the operating systems and applications, which are used within the corporate's boundaries. For example, if we use a DMS, which mimics an ancient virus (unable to function under modern Windows OS), its exclusion would be considered quite safe, assuming that the organization does not use any ancient OS operated computers (or exclusion must not be applied to those specific machines). If a modern malware signature is used, it has to be attached in harmless way as discussed later. Note that when choosing modern malware signature as DMS, the exclusion rule will be applied only to the harmless form of the DMS, while the regular signature will be detected as malicious. In that case, harmless DMS will be avoided by AV, still the organization is protected against the real threat.

Our work at (Guri et al., 2014) discuss the type of malware signatures and their detection rates along with the best injecting location. We have evaluated the alternatives according to several criteria, including their detectability by prevalent AV programs, the possibility of safe exclusion and the best location of injection. Detailed evaluation of each alternative given in Guri et al. (2014).

Table 2
Effectiveness of DocGuard use in different social networks.

Social network	Sharing options	Status
Facebook (facebook.com)	Share a link, Profile photo, Share in group, Chat	100% Blocked (upload/download)
LinkedIn (linkedin.com)	Share a link, Profile photo, Share a file	100% Blocked (upload/download)
Twitter (twitter.com)	Share a link, Profile photo, Messages	100% Blocked (upload/download)
Google+ (plus.google.com)	Share a link, Profile photo, Hangouts	100% Blocked (upload/download)
Pinterest (pinterest.com)	Share a link, Profile photo	100% Blocked (upload/download)
Instagram (instagram.com)	Share a link, Profile photo	100% Blocked (upload/download)
Tumblr (tumblr.com)	Post/share a link, Upload photo	100% Blocked (upload/download)
Flickr (flickr.com)	100% Upload a file	Blocked (upload)

4.2. Evaluation

In this section we present the test result of DocGuard in different leakage paths. Our tests include leakage via Social Networks, leakage via emails and file-sharing services, and leakage via removable media. We also discuss the system's storage and run-time overhead.

4.2.1. Social networks

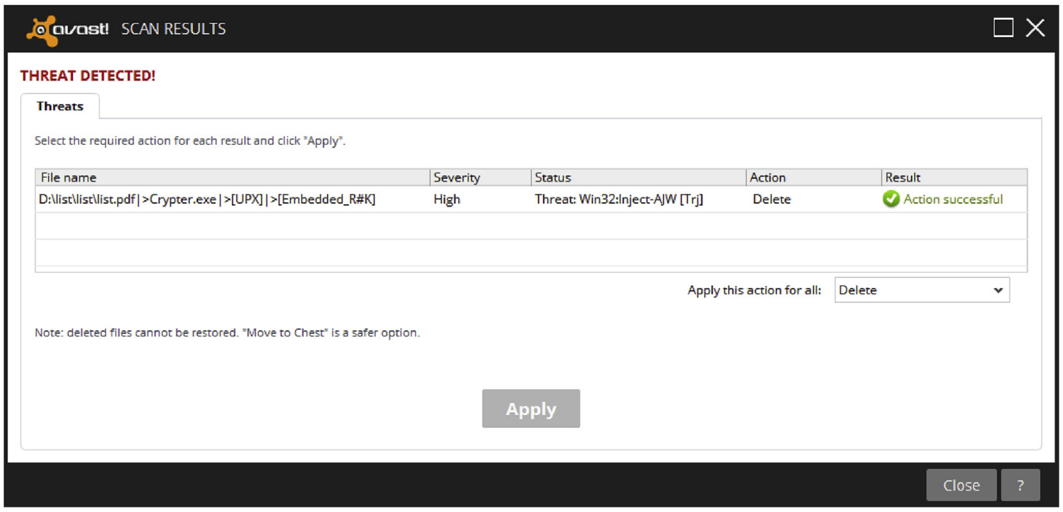
We evaluated the effectiveness of DocGuard in preventing data leakage in popular social networks. The chosen signature identified by most of the current AVs, including all of the top ten AV vendors. The types of files included documents and images (PDF, DOCX, and JPEG). During the experiments, a user (leakage initiator) tried to share a file in various ways through the most popular social networks used today. The capability of a DocGuard in limiting the file propagation is summarized in Table 2. As can be seen, in all cases AVs successfully blocked the leaked files before they were accessed, preventing the user from opening or sharing them (Fig. 3). Notably, in some cases (e.g., Facebook) files were detected and blocked by the service providers themselves before the files even arrived at the user's PC.

4.2.2. E-mails, cloud and file sharing services

In addition to sharing the file via the conventional social network sharing mechanisms (e.g., image, URL link, etc.), users in social networks may choose to distribute the files via email or file sharing services. The major cloud providers employ advanced protection mechanisms to detect and prevent malware from infecting host machines and spreading in the cloud. For example, the Windows Defender Antivirus cloud service is a defense suit used for delivering updated protection to network and endpoints threats. These types of AVs are continuously updated with the malware signatures for static analysis and heuristic signatures for run-time protection. Our tests on the three big cloud providers (Microsoft, Amazon and Google) shows that their malware scanners were able to detect all the documents signed with the DocGuard. We also observed that online document viewers for Adobe PDF and office documents (DOCX, PPTX XLSX) are not capable of viewing the leaked document due to the corrupted format. Table 3 shows the effectiveness of the use of a DocGuard when using these file sharing methods.

4.2.3. Removable media (USB, CD, DVD)

We test DocGuard effectively in stopping a leakage occur via removable media. During the tests, we inserted the media with 'leaked' file to user workstation. Our tests include the top 10 AV. At can be seen in Table 4, the file was blocked once the file was copied of opened. Notably, some AV start scanning the removable media once inserted to the



Select the required action for each result and click "Apply".

File name	Severity	Status	Action	Result
D:\list\list\list.pdf|>Crypter.exe|>[UPX]|>[Embedded_R#K]	High	Threat: Win32:Inject-AJW [Trj]	Delete	Action successful

Apply this action for all: Delete

Note: deleted files cannot be restored. "Move to Chest" is a safer option.

Fig. 3. Deletion upon detection. The Antivirus program (Avast! in this case) notifies the user that the whole document (list.pdf) was deleted.

Table 3
Effectiveness of DocGuard use with different sharing services.

Shared via	File types	Block
Gmail, Yahoo Mail, Hotmail	Documents, Images, Binaries	100% blocked (Server side, Block send and receive)
Dropbox, Google Drive	Documents, Images, Binaries	100% blocked (Users' PCs)
Cloud storage (Microsoft Azure, Amazon AWS, Google Cloud)	Documents, Images, Binaries	100% blocked (Host machine), server side
Torrents shares	Documents, Images, Binaries	100% blocked (Users' PCs)
File-sharing web-sites	Documents, Images, Binaries	100% blocked (Users' PCs)

Table 4
Effectiveness of DocGuard use with Removable Media.

File Type	Blocked	Blocked at	Top 10 AV
PDF (Acrobat)	100% Blocked	Copy/Open/Scan	10/10
DOCX (Microsoft Word)	100% Blocked	Copy/Open/Scan	10/10
XLSX (Microsoft Excel)	100% Blocked	Copy/Open/Scan	10/10
PPTX (Microsoft Power point)	100% Blocked	Copy/Open/Scan	10/10
Text file	100% Blocked	Copy/Open/Scan	10/10
.C,.CPP (source code)	100% Blocked	Copy/Open/Scan	10/10
DLL	100% Blocked	Copy/Open/Scan	10/10
EXE	100% Blocked	Copy/Open/Scan	10/10

computer, hence blocked the file immediately.

4.2.4. *Smartphones and tablets*

We test DocGuard effectively in stopping a leakage occur via smartphone and tablets. The main threats with the emerging trend of Bring Your Own Device (BYOD) are when an employee uploads a sensitive document into his device and then spread it through other vectors (Email, file-sharing, social network and so on). In a common case, the documents saved in the device are automatically uploaded to the cloud and stored there. In the case of Android and IOS devices the documents are scanned by the cloud provider and blocked. As shown in the previous section, we achieved %100 detection rate in the Azure, AWS and Google cloud services. If the documents were sent to the device through e-mail, they will be blocked at the mail provider and will not be delivered to the device. Note that in addition to the cloud malware scanning services, there are malware scanning apps for files stored on the local device which blocks the file locally, even before uploaded to the cloud.

4.3. *System's overhead*

4.3.1. *Storage overhead*

Through signature attachment, each document is inflated by a constant delta such that $V_T = V_O + V_D$ where V_T denotes the size of the tagged file, V_O denotes the size of the original file, and V_D denotes the size of the signature. The effect of V_D on the overall size of the tagged files essentially depends on the file-size distribution of the original files which are subject to tagging. Several distribution-types have been proposed for the overall population of files on a user's computer, including lognormal distribution and lambda distribution (Evans and Kuenning, 2002). If we hypothetically apply Pareto principle to the relevant original documents size, then the largest 20% hold approximately 80% of the total size of the original files which we denote as $\sum V_O$. Hence the effect of V_D on $\sum V_T$ (the total size of the tagged files) would be practically higher than what would be expected with normal distribution. However, if we assume that $V_D \ll \overline{V_O}$ where $\overline{V_O}$ denotes the average size of relevant original documents, regardless of the distribution, then the total volume required by the tagged files should not significantly exceed the total original volume of relevant files.

4.3.2. *Performance overhead*

The additional operations performed by our driver when reading and writing files are negligible ($O(1)$) regarding the size of the manipulated file as well as the number of manipulated files. We made several tests to corroborate this assumption, where files were created, written, read and deleted; the files' contents were randomized, to eliminate possible effects of caching. With 10,000 files the performance with the driver was practically equal to the performance without the driver.

5. Wide-scale simulation

In this section we evaluate the effectivity of DocGuard not only for stopping the initial leak but also in preventing the *propagation* of leaked files over the Internet. To that end, we analyze and evaluate the expected impact of our method on the spread of a leaked file using the common model of independent cascades (Goldenberg et al., 2001). In the model of independent cascades, an individual exposed to a document shared by one of his/her peers has a chance of being “infected” (a.k.a. activated, adopted) by the idea expressed in the document and sharing it. Note that homogeneous models set the probability p of sharing a document (a.k.a. infectivity) to be equal for all nodes in a social network. Other models set these probabilities for each pair of nodes or distinguish between the probability of being infected by the document and the probability of sharing it. An extensive review of viral spreading models in social networks is beyond the scope of this paper.

5.1. *Information diffusion in social networks*

When sensitive information goes public through any one of the currently prospering online social networking services, such as Facebook, Twitter, Flickr, etc., it may reach a huge audience in a number of hours. The information spread in social networks is viral in its nature. A message, link, or file posted by a network participant is immediately visible to friends who can, in turn, share (a.k.a. favor, like, retweet) the post and make it visible to their friends and friends of friends, etc. This social cascade phenomenon has been extensively studied in recent years (Moreno et al., 2004; Chierichetti et al., 2009; Bordia and DiFonzo, 2005), and we refer the reader to a recent survey on information diffusion in social networks (Guille, 2013).

Predictive models can be used to estimate the reach of a specific diffusion process in a given network. Common models focus either on the receiver or the sender of the information. Threshold models are receiver-centric and categorize nodes as either active (a.k.a. adopter, familiar with, actively spreading the information or the behavior) or inactive (a.k.a. non-adopter). Every active node influences its neighbors and may persuade them to adopt the new behavior or share the information. Every inactive node has a certain influence threshold, and when the cumulative influence of its active friends exceeds this threshold, the node becomes active as well. Among the many threshold models proposed, the linear threshold model (Granovetter, 1978) is probably the most popular.

The independent cascades model (Goldenberg et al., 2001) is an example of a receiver-centric model, where every inactive node has a given probability of being activated by one of its active friends. An active node has only a single chance to influence each one of its neighbors. If unsuccessful, the node will have no additional trials. Both the linear threshold and the independent cascade models have been shown to be strong predictors of the extent of a diffusive process in a social network, but they do not capture the time dynamics. This aspect is tackled by further extensions of these models, e.g., the time-based asynchronous independent cascades (Hacid, 2012) and epidemic propagation models.

The spread of an interesting piece of information in a social network environment is somewhat similar to the propagation of a contagious disease and can be modeled in much the same way. Common models of epidemic propagation (Hethcote, 2000) categorize the population into three defined states: (1) Susceptible individuals do not have the disease in question but can catch it if exposed to someone who does; (2) Infective individuals have the disease and can pass it on; and (3) Removed individuals have been disabled by the disease and cannot be infected or infect others. In the context of information diffusion in social networks these individuals correspond to those that can adopt the information and share it, those that have shared the information and can trigger interest among others, and those that can no longer disseminate the information.

Different epidemic models define different transitions between the states. The seminal SIR (susceptible, infected, removed) model assumes that any susceptible individual has a probability p of catching the disease, in a unit of time, from any infective entity. Infected entities are removed with a probability γ in a unit of time (The Linux man-pages project). Models of this type assume that the population is fully mixed. As a result, the fractions of entities in the susceptible, infective, and removed states can be expressed analytically by differential equations. In reality, diseases can only spread between entities that have actual physical contact. The nature of the contact influences the disease propagation (Miramirkhani et al., 2017). Researchers usually study epidemic propagation in networks using stochastic simulations.

It has been shown that the network topology has a strong effect on the dynamics of contagious spreading (Honig, 2014). In homogeneous networks an epidemic occurs only if the rate of infection of susceptible nodes connected to infected nodes exceeds the so-called epidemic threshold. If the effective spreading rate is below an epidemic threshold, the diffusion cannot persist for a long period. The nonlinear dynamical system (NLDS) accurately models viral propagation in any arbitrary network, including real and synthesized network graphs (Honig, 2014). The authors proved that the epidemic threshold for a network is exactly the inverse of the largest eigenvalue of its adjacency matrix. Below that epidemic threshold, infections die out at an exponential rate.

In heterogeneous networks, on the other hand, it is well-known that the epidemic threshold is negatively correlated with the standard deviation of the connectivity distribution. When the connectivity of nodes varies significantly, nodes with high connectivity act as "boosting hubs," accelerating the diffusion. This feature is paradoxically amplified in networks that possess diverging connectivity fluctuations (Pastor-Satorras and Vespignani, 2001).

In the following section we use models of information diffusion in social networks in order to evaluate the effectiveness of the proposed method to contain and limit the spread of leaked files through social networks.

5.2. *Model and representation*

In the rest of this section we present the problem, describe the set of simulations and tests, and discuss the results.

Let $G = (V, E)$ represent an online social network where the set of nodes (V) represents the profiles, and the set of edges (E) represents the friend (or follower) relationships. For the sake of simplicity we assume undirected edges, but the conclusions derived in this paper apply to directed networks (e.g., Twitter) as well. Let D denote the leaked file and $\mathrm{A} \subseteq \mathrm{V}$ denote a set of nodes protected by an AV that can detect the malware signature attached to D. Assume for example, a small social network depicted in Fig. 4 with $V = \{a, b, c, d\}$ and $E = \{(a,b), (a,c), (b,c), (c,d)\}$. The node a has accidently leaked a file, and c is the only node with an appropriate AV program $A = \{c\}$. Assume a file with infectivity $p = 0.5$. Once a has leaked the file, b is exposed to it and has equal probability of sharing it or not sharing it. The node c might have been exposed to the file from both its friends, a and b, but its AV will delete the file upon download and the links will appear broken. Thus c will never be exposed to the content, and as a result, it won't share it. Therefore, d will not be exposed either. This example demonstrates that an appropriate AV program installed on the computers of social network users can help in pruning the social cascades.

5.3. *File propagation simulations*

In order to evaluate the extent to which a certain level of AV prevalence helps in reducing the exposure of leaked files, we performed a series of simulations on three social network topologies. In this experiment we have used datasets from the friend relationships in Facebook (Gummadi et al., 2009), the trust network of Epinions (Domingos and Agrawal, 2003), and the friend/foe relationships in Slashdot (Leskovec et al., 2009). All links were treated as undirected. It can be assumed that if a and b have an edge between them, a can share a file with b, and b can share a file with a. Since the subject of the current study is data leakage

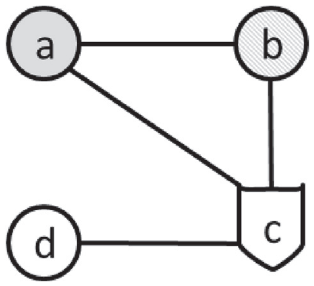


Fig. 4. Representation of the spread of a file over a social network. 'a' is the leaking node, and 'c' is the only node with an appropriate AV program.

Table 5
Social network datasets.

	Vertices	Edges	Source
Facebook	63,392	816,886	AV vendors (2018)
Epinions	75,879	405,740	Kirat et al. (2014)
Slashdot	77,360	469,180	Christodorescu and Somesh (2004)

and not opinions, we disregard the link signs in the Epinions dataset. Table 5 summarizes the social networks used.

We ran a series of simulations for each of these datasets aimed at mimicking the propagation of the leaked file in the social networks. In each simulation the first time unit represents the initial leakage, when a specific node leaks the file for the first time. The next time unit represents the next phase of propagation, and so on.

At the beginning of each simulation the following three parameters were established.

- N as a single random node was chosen to leak a file.
- An "infection" probability p ranging from 0.1 to 1.0 (with a step of 0.1) was assigned. This represents the probability that the user share the file further, e.g., documents that raise interest among readers are more likely to be shared with friends.
- $x = \frac{|A|}{|V|}$ (ranging from 0.0 to 1.0) is the fraction of users having an AV program that will successfully detect the signature and prevent download and view of the content. In Guri et al. (2014) it was found that currently $x = \sim 0.81$ which means that 81% of the Internet users that will try to open the file will fail to do so, because their AV will detect the signature, block access to the file, and quarantine it.

The file propagation simulator, developed in JAVA, loads the social network dataset files and builds the appropriate graph representation in the memory. Simulation consists of three main phases:

(1) A simulation starts by invoking the *StartSimulation* (N, p, x) method. N, p, *and* x represent the initial leaker node, the infection probability, and the fraction of users having an appropriate AV program, respectively.
(2) At each iteration (time unit), when a node receives the file, it determines whether the file was blocked by the local AV (randomized due to the x parameter). A node that receives the file then 'decides' whether it will share the file with its friends (randomized due to the p parameter).
(3) The file propagation concludes when there are no more nodes that can be infected.

To demonstrate the effectiveness of the use of DocGuard we compare between propagation of a leaked file, with and without an injected signature, over the topology of Facebook, Epinion, and Slashdot. Fig. 5 shows the number of nodes exposed to a leaked file as a function of time units for $p = 0.5$ where $x = 0.8$ and $x = 0$. $x = 0$ represents the leakage of a file with no injected signature. The results are averaged over 1000 different simulations for each social network. The simulations were randomized such tat in each simulation a different node in the graph was set to be the initial leaking node (N). As can be seen in Fig. 5, without the use of an injected signature ($x = 0$) the number of nodes exposed to the leaked file grows exponentially. After 7–9 time units, the leaked file reached 22K–28K nodes for Facebook, Slashdot, and Epinions. Note that the Y axis is of logarithmic scale.

Utilizing the DocGuard ($x = 0.8$), in Facebook and Epinions the propagation stopped after less than ten time units with about 350 and 240 nodes exposed to the leaked file (only 1.25% of the originally exposed nodes). In Slashdot it took 17 time units for propagation to stop with about 235 nodes (less than 1% of the originally exposed nodes). The difference between fractions of nodes exposed to a leaked file in the

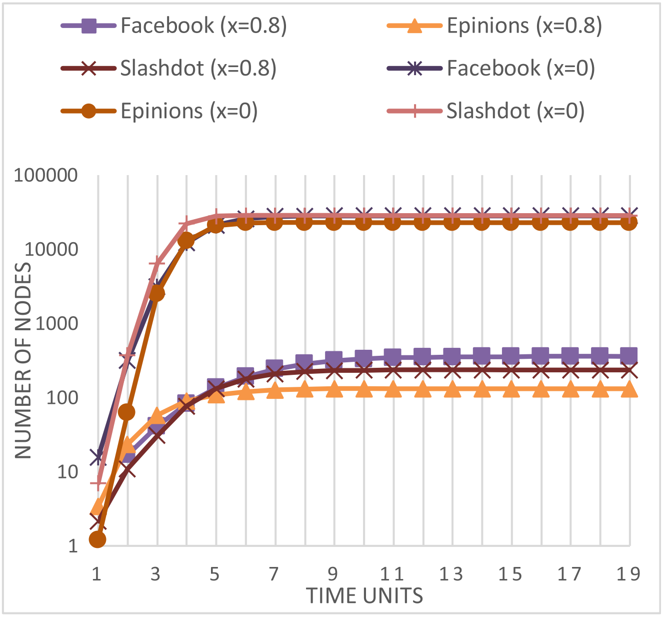


Fig. 5. Propagation of a leaked file in Facebook, Epinions, and Slashdot social networks (p = 0.5, x = 0.8, x = 0), averaged over 1000 different simulations for each social network.

different social networks datasets is attributed to their density, with Facebook being the densest and Epinions the most sparse (Gummadi et al., 2009; Leskovec et al., 2009). In dense networks it is harder to block all of the leaked file's propagation paths, thus a leaked file will reach a larger population in a densely connected social network than in a sparse network, assuming the same infectivity rate and AV prevalence.

5.4. Saved nodes

In order to provide thorough evaluation of our solution, we examine file exposure with various values of x and p. In practice, x values might change over time as AV prevalence changes. x may also vary between groups of social network users (e.g., users in different countries).

Values of p may also vary due to the content of the leaked files or other factors. For example, when the leaked files are documents that raise interest among readers, they are more likely to be shared, and hence have higher p values. We denote the total number of nodes that have been exposed to a leaked file during its propagation as $I_x(p)$. Given p and x, $I_0(p) - I_x(p)$ is the number of nodes that the DocGuard 'saved'. That is, the nodes that weren't exposed to the leaked file because of the use of the injected signature - without the use` of the signature, these nodes were exposed to it. We refer to these nodes as 'saved nodes.' Table 6 lists the percentage of saved nodes $\frac{I_0(p)-I_x(p)}{I_0(p)}$, for different x and p values.

The results, averaged over 1000 simulations, show that for $x = 0.8$,

Table 6
The percentage of saved nodes $\frac{I_0(p) - I_x(p)}{I_0(p)}$ for different x and p values.

x	$p = 0.2$	$p = 0.4$	$p = 0.6$	$p = 0.8$	$p = 1$
0	0%	0%	0%	0%	0%
0.1	24%	12%	26%	14%	17%
0.2	57%	21%	37%	35%	34%
0.3	50%	51%	50%	44%	54%
0.4	72%	61%	64%	51%	64%
0.5	83%	78%	73%	73%	71%
0.6	95%	86%	86%	82%	81%
0.7	92%	88%	90%	89%	89%
0.8	98%	96%	93%	94%	94%
0.9	99.3%	99.2%	99.1%	99.1%	98.6%

more than 92% of the nodes have been saved (weren't exposed to the leaked file). For $x = 0.9$, more than 99% nodes have been saved, regardless of the value of p. The results also show that the potential interest a file may raise doesn't reduce the effectiveness of DocGuard on saved nodes, even in the presence of social cascades. As noted before, currently $x = \sim 0.81$, and this value will increase with the growing prevalence of AVs (AV vendors, 2018).

5.5. Indirectly saved nodes

Intuitively, we can expect $\frac{I_0(p)-I_x(p)}{I_0(p)}$ to be at least as high as x, e.g., if 80% of the nodes are protected by an appropriate AV, then at least 80% will naturally be saved. However, in addition to nodes that have an appropriate AV, some nodes (with or without an appropriate AV) weren't exposed to the file, because their friends (neighbors) had an AV which prevented their exposure. We refer to these nodes as 'indirectly saved nodes.' We define this fraction of indirectly saved nodes as

$$S(p,x) = \frac{IND_x(p)}{I_0(p)}$$

where $IND_x(p)$ is the number of indirectly saved nodes. In order to measure the fraction of indirectly saved nodes S, we marked these nodes appropriately during the simulations. Fig. 6 shows $S(p,x)$ in Facebook, Epinion, and Slashdot topologies (1000 simulations per social network). Every line in Fig. 6 represents a specific infection probability (p).

Interestingly, we found that the infection rate p has no significant effect on $S(p,x)$. In practice it means that some users are socially saved. They weren't exposed to the file even without AV installed and regardless of the infection rate. This is supported by one-way ANOVA test failed on the simulation results grouped by p with a P value as high as 0.7 (see Table 7).

6. Conclusion

Accidental data leakage is a serious threat, causing substantial financial loss, exposing sensitive information, and harming the reputation of the involved organization. Common technological solutions to this problem essentially are incapable of limiting the spread of a sensitive document once it had somehow leaked. In this work we describe a new method, aiming to reduce this conceptual loophole. Our method involves injecting virus signature to sensitive files. These files are detected as a genuine threat by prevalent Antivirus (AV) programs, along with Intrusion Detection and Prevention systems. Consequently, the existing global infrastructure of installed AV programs outside the organization perimeter is employed to serve as remote sentinels and to halt the leaked document, beyond the boundaries of the organization. We developed a working deployment framework, capable of performing transparent injection on various types of documents. We conducted detailed experiments with various types of file types, under various types of AV program. We implement DocGuard and tested it on various file types such as documents, spreadsheets, presentations, images, executable binaries and textual source code. Our tests includes different leakage paths including E-Mails, file-sharing and cloud services, social networks and physical media. The evaluation results have demonstrated almost 100% effectiveness of stopping the leakage at its initial phases. In order to evaluate the large scale effect of DocGuard, we have simulated a leakage scenario over the topology of real social networks. We also found that DocGuard is effective in preventing social cascades, even when the leaked files are documents that raise interest among readers and are hence likely to be or opened and shared. The results show that DocGuard is highly effective not only for stopping the initial leak but also in preventing the propagation of leaked files over the Internet. An interesting future research direction may address the ability of this method to deliver forensic evidence regarding the source and route of the leaked file, by assessing related AV and security system reports and logs.

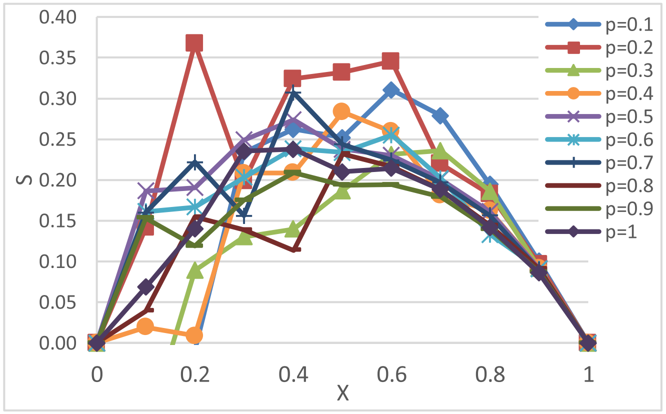


Fig. 6. Fraction of indirectly saved nodes for various propagation probabilities p as a function of x.

Table 7
ANOVA test on $S(p,x)$ values for various infection probabilities.

	SS	Df	MS	F	P	F
Between Groups	0.070	9	0.007	0.708	0.700	1.974
Within Groups	1.110	100	0.011			
Total	1.181	109				

References

Alneyadi, S., Sithirasenan, E., Muthukkumarasamy, V., 2016. A survey on data leakage prevention systems. J. Netw. Comput. Appl. 62, 137–152.
Anon [Online]. Available: https://www.samsung.com/us/business/solutions/samsung-knox/.
Asaf, S., Elovici, Y., Rokach, L., 2012. A Survey of Data Leakage Detection and Prevention Solutions. Springer Science & Business Media.
AV vendors, 2018. Market Share Held by the Leading Windows Anti-malware Application Vendors Worldwide, as of January 2018 [Online]. Available: https://www.statista.com/statistics/271048/market-share-held-by-antivirus-vendors-for-windows-systems/.
Bordia, P., DiFonzo, N., 2005. Psychological motivations," in rumor spread. In: Rumor Mills: The Social Impact of Rumor and Legend, pp. 87–101.
Chae, C.-J., Shin, Y., Choi, K., Kim, K.-B., Choi, K.-N., 2016. A privacy data leakage prevention method in P2P networks. Peer-to-Peer Netw. Appl. 9 (3), 508–519.
Cheng, L., Liu, F., Yao, D., 2017a. Enterprise data breach: causes, challenges, prevention, and future directions. In: Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, vol. 7. Wiley Online Library).
Cheng, L., Liu, F., Yao, D., 2017b. Enterprise data breach: causes, challenges, prevention, and future directions. Wiley Interdiscip. Rev.: Data Min. Knowl. Discov. 7 (6).
Chierichetti, F., Lattanzi, S., Panconesi, A., 2009. Rumor spreading in social networks. In: Automata, Languages and Programming. Springer Berlin Heidelberg, pp. 375–386.
Choo, K.-K.R., 2010. Cloud Computing: Challenges and Future Directions. Trends and Issues in Crime and Criminal justice, no..
Christodorescu, M., Somesh, J., 2004. Testing malware detectors. In: ACM SIGSOFT International Symposium on Software, Boston, Massachusetts, USA.
Cisco, 2014. Data Leakage Worldwide: Common Risks and Mistakes Employees Make [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/enterprise-networks/data-loss-prevention/white_paper_c11-499060.html.
D'Orazio, C., Choo, K.-K.R., 2016. An adversary model to evaluate DRM protection of video contents on iOS devices. Comput. Secur. 56, 94–110.
Domingos, M.R., Agrawal, R., 2003. Trust management for the semantic web. ISWC. In: The Semantic Web. ISWC2003, Sanibel Island, Florida, USA.
European Expert Group for IT-Security, EICAR [Online]. Available: http://www.eicar.org/.
Evans, K.M., Kuenning, G.H., 2002. A study of irregularities in file-size distributions. In: In Proceedings of," in the 2002 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS).
Fagundes, L., 2014. Data Leaks: Your Employees Might Be Sharing Sensitive Info [Online]. Available: http://www.securedocs.com/blog/2014/06/accidental-data-leaks-how-your-employees-might-be-inadvertently-sharing-confidential-information.
Goldenberg, J., Libai, B., Muller, E., 2001. Using complex systems analysis to advance marketing theory development". Acad. Market. Sci. Rev. 9 (3), 1–18.
Govinda, K., Joseph, D., 2017. Dynamic data leakage using guilty agent detection over cloud. In: International Conference on Intelligent Sustainable Systems (ICISS).
Granovetter, M., 1978. Threshold models of collective behavior. Am. J. Sociol. 1420–1443.

Guevara, C., Santos, M., Lopez, V., 2014. Data leakage detection algorithm based on sequences of activities. In: Proceedings of the 17th International Symposium Research in Attacks, Intrusions and Defenses RAID.
Guevara, C., Santos, M., Lopez, V., 2017. Data leakage detection algorithm based on task sequences and probabilities. Knowl. Based Syst. 120, 236–246.
Guille, A., 2013. Information diffusion in online social networks: a survey. ACM SIGMOD Record 42 (2), 17–28.
Gula, R., Ranum. M., System and Method for Using File Hashes to Track Data Leakage and Document Propagation in a Network. US Patent US9367707B2, 2018.
Gummadi, B.V., Mislove, A., Cha, M., K. P, 2009. On the evolution of user interaction in Facebook. In: The 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09), Barcelona, Spain.
Guri, M., Kedma, G., Carmeli, B., Alovici, Y., 2014. Limiting unintentionally leaked sensitive documents using malware signatures. In: The ACM Symposium on Access Control Models and Technologies (SACMAT 2014). Canada, London, Ontario.
Hacid, G., 2012. A Predictive Model for the Temporal Dynamics of Information Diffusion in Online Social Networks. WWW '12 Companion, pp. 1145–1152.
Hethcote, H.W., 2000. The mathematics of infectious diseases. SIAM Rev. 42 (4), 599–653.
Hoang, P.V.V., Yu, S., Sood, K., Cui, L., 2017. Privacy issues in social networks and analysis: a comprehensive survey. IET Netw. 7 (2), 74–84.
Honig, M.S., 2014. Practical Malware Analysis. No Starch Press.
Hu, N., Ye, M., Wei, S., 2017. Surviving information leakage hardware trojan attacks using hardware isolation. IEEE Trans. Emerg. Top. Comput.
Jogdand, P., Padiya, P., 2016. Survey of different IDS using honeytoken based techniques to mitigate cyber threats. In: Electrical, Electronics, and Optimization Techniques (ICEEOT), International Conference on.
Kim, G., Lim, J., Kim, J., 2017. Mobile security solution for sensitive data leakage prevention. In: Proceedings of the 5th International Conference on Communications and Broadband Networking.
Kirat, D., Vigna, G., Kruegel, C., 2014. BareCloud: bare-metal analysis-based evasive malware detection. In: USENIX Security Symposium.
Kongsgard, K.W., Nordbotten, N.A., Mancini, F., Haakseth, R., Engelstad, P.E., 2017. Data leakage prevention for secure cross-domain information exchange. IEEE Commun. Mag. 55 (10).
Kongsgarrd, K.W., Nordbotten, N.A., Mancini, F., Engelstad, P.E., 2017. An internal/insider threat score for data loss prevention and detection. In: Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics.
Lab, K., 2017. Top 5 Largest Data Leaks of 2017 — So Far [Online]. Available: https://www.kaspersky.com/blog/data-leaks-2017/19723/.
Leskovec, J., Lang, A.D.K., Mahoney, M., 2009. Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. Internet Math. 6 (1), 29–123.
Liu, X., Choo, R., Deng, R., Lu, R., Weng, J., 2016. Efficient and privacy-preserving outsourced calculation of rational numbers. IEEE Trans. Dependable Secure Comput.
Liu, X., Deng, R., Choo, K.-K.R., Yang, Y., Pang, H., 2018. Privacy-Preserving outsourced calculation toolkit in the cloud. IEEE Trans. Dependable Secure Comput.
Microsoft, Filter Manager Concepts, [Online]. Available: https://msdn.microsoft.com/en-us/windows/hardware/drivers/ifs/filter-manager-concepts.
Microsoft, 2010. Microsoft Portable Executable and Common Object File Format Specification. Microsoft.
Microsoft, 2015a. Plan Information Rights Management [Online]. Available: https://technet.microsoft.com/en-us/library/cc179103.aspx#BKMK_WhatIsIRM. (Accessed 1 July 2016).
Microsoft, 2015b. Microsoft "technet.Com" [Online]. Available: http://blogs.technet.com/b/mmpc/archive/2015/06/09/windows-10-to-offer-application-developers-new-malware-defenses.aspx.
Miramirkhani, N., Appini, M.P., Nikiforakis, N., Polychronakis, M., 2017. Spotless sandboxes: evading malware analysis systems using wear-and-tear artifacts. In: 2017 IEEE Symposium on Security and Privacy (SP).
Moreno, Y., Nekovee, M., Pacheco, A.F., 2004. Dynamics of Rumor Spreading in Complex Networks [Online]. Available: http://arxiv.org/pdf/cond-mat/0312131.pdf.
Narayana, K.S., Pasupuleti, S.K., 2018. Trusted model for virtual machine security in cloud computing. In: Progress in Computing, Analytics and Networking. Springer, pp. 655–665.
Paganini, P., 28 February 2017. Boeing Notified 36,000 Employees Following an Accidental Data Leak [Online]. Available: https://securityaffairs.co/wordpress/56736/data-breach/boeing-data-leak.html.
Papadimitriou, P., Garcia-Molina, H., 2010. Data leakage detection. IEEE Trans. Knowl. Data Eng. 23 (1), 51–63.
Pastor-Satorras, R., Vespignani, A., 2001. Epidemic spreading in scale-free networks. Phys. Rev. Lett. 86 (14), 3200–3203.
Peng, J., Choo, K.-K.R., Ashman, H., 2016. User profiling in intrusion detection: a review. J. Netw. Comput. Appl. 72, 14–27.
Phua, C., 2009. Protecting organisations from personal data breaches. Comput. Fraud Secur. 1 (1), 13–18.
Priebe, C., Muthukumaran, D., O'Keeffe, D., Eyers, D., Shand, B., Kapitza, R., Pietzuch, P., 2014. Cloudsafetynet: detecting data leakage between cloud tenants. In: Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security.
Quick, D., Choo, K.-K.R., 2013. Digital droplets: Microsoft SkyDrive forensic data remnants. Future Gener. Comput. Syst. 29 (6), 1378–1394.
Romer, H., 2014. Healthcare Organizations Suffering Data Breaches from Stolen Devices. Accellion [Online]. Available: http://www.accellion.com/blog/healthcare-organizations-suffering-data-breaches-stolen-devices.
SafeSend, 2016. Flemingdon Health Centre - Case Study [Online]. Available: https://www.safesendsoftware.com/content/FHC-Case-Study.pdf.
Securelist, Virus.DOS.Aids.552, [Online]. Available: http://www.securelist.com/en/descriptions/6880300/Virus.DOS.Aids.552.
Seltzer, L., 2019. Malware Sample Sources for Researchers [Online]. Available: http://zeltser.com/combating-malicious-software/malware-sample-sources.html.
Shane, S., Rosenberg, M., Lehren, A.W., 2017. WikiLeaks Releases Trove of Alleged C.I.A. Hacking Document [Online]. Available: https://www.nytimes.com/2017/03/07/world/europe/wikileaks-cia-hacking.html.
Stevens, D., 2011. Malicious PDF documents explained. IEEE Secur. Priv. 9 (1), 80–82.
Symantec, Understanding Virus Behavior under Windows NT, Symantec Research Center, [Online]. Available: http://www.symantec.com/avcenter/reference/virus.behavior.under.win.nt.pdf.
Ször, P., 2005. The Art of Computer Virus Research and Defense [Online]. Available: http://vxheaven.org/lib/aps00.html.
The Linux man-pages project, Elf - Format of Executable and Linking Format (ELF) Files, [Online]. Available: http://man7.org/linux/man-pages/man5/elf.5.html.
Vasciannie, B., 2015. 7 Ways to Lose Sensitive Data. Digital Guardian [Online]. Available: https://digitalguardian.com/blog/7-ways-lose-sensitive-data.
VirusTotal, 2018. VirusTotal [Online]. Available: https://www.virustotal.com/.
WikiLeaks, 2016. Email Archive [Online]. Available: https://wikileaks.org/clinton-emails/.
Zhang, X., Liu, F., Chen, T., Li, H., 2009. Research and application of the transparent data encpryption in intranet data leakage prevention. In: Computational Intelligence and Security, 2009. CIS '09. International Conference on.

Dr. Mordechai Guri received his B. Sc and M.Sc in computer science from the Hebrew University of Jerusalem. He is finished a Ph.D. it's at BGU in the Department of Information Sytems Engineering, under the supervision of Prof. Yuval Elovici. Mordechai manages a research team in various topics of cyber security. His research interests include advanced malware, rootkits, embedded systems and mobile security. Mordechai recently selected to receive the prestigious 2015–2016 IBM PhD Fellowship Award.

Dr. Rami Puzis Lecturer, BSc Software Engineering, MSc Information Systems Engineering and PhD topic on Deployment of Intrusion Detection Systems. He has worked as a research associate in the Laboratory of Computational Cultural Dynamics, University of Maryland. His primary specialization is in the area of complex networks with applications to cyber security, social and communication network analysis. He has been the principal investigator of a series of research projects funded by Deutsche Telekom AG, Israeli Ministry of Defense, Israeli Ministry of Economy, and several leading cyber security industries.

Prof. Kim-Kwang Raymond Choo holds a Ph.D. in information technology from Queensland University of Technology, Australia. Prior to starting his Cloud Technology Endowed Professorship at UTSA, Professor Choo spent five years working for the University of South Australia, and five years working for the Australian Government Australian Institute of Criminology. He was also a visiting scholar at INTERPOL Global Complex for Innovation between October 2015 and February 2016 and a visiting Fulbright scholar at Rutgers University School of Criminal Justice and Palo Alto Research Center (formerly Xerox PARC) in 2009. Professor Choo's areas of research include big data analytics, cyber security and digital forensics. His research has been widely cited, including in government reports of the Australian Government, United Nations Office on Drugs and Crime (UNODC), U.S. CRS Report for Congress (Prepared for Members and Committees of Congress), International Center for Missing & Exploited Children, International Telecommunication Union (ITU), and UK Home Office. One of his published cryptographic protocols was included in two independent submissions to the IEEE 802.11, the working group setting the standards for wireless LANs, by computer scientists from Fujitsu Labs in America; and the IETF / Network Working Group by a team of computer scientists from Tropos Networks (US), Toshiba, Huawei, and University of Murcia. Another of his published protocol was used in the P2P solution of the Milagro TLS (pairing-based cryptography for Transport Layer Security) presented to the IETF by researchers from MIRACL Ltd in 2016. In April 2017 he was appointed an Honorary Commander, 502nd Air Base Wing, Joint Base San Antonio-Fort Sam Houston, USA. He is also a Fellow of the Australian Computer Society and a Senior Member of IEEE.

Mr. Gabi Kedma received his B.Sc. in civil engineering and B.Arch. in architecture from the Technion, Israel Institute of Technology at Haifa, and his M.A. in science, technology and society from Bar-Ilan University. He is currently a Ph.D. student at BGU in the Department of Information Systems Engineering, under the supervision of Prof. Yuval Elovici. His research interests include evasive malware and runtime detection of malicious code.

Mr. Sergey Rubinstein A software engineer with 4 years of experience. He received his BSc in Information Systems Engineering from Ben Gurion University in 2013. Currently Sergey is a graduate student in the same department working on the systemization of knowledge and modeling of advanced multistage cyber-attacks. He participated in a number of research projects sponsored by the Israeli Ministry of Defense, EMC, Lockheed Martin and now leads a team of students in a research project supported by IBM.

Prof. Yuval Elovici is the director of the Telekom Innovation Laboratories at Ben-Gurion University of the Negev (BGU), head of BGU Cyber Security Research Center, Research Director of iTrust at SUTD, and a Professor in the Department of Information Systems Engineering at BGU. He holds B.Sc. and M.Sc. degrees in Computer and Electrical Engineering from BGU and a Ph.D. in Information Systems from Tel-Aviv University. Prof. Elovici has published articles in leading peer-reviewed journals and in various peer-reviewed conferences. In addition, he has co-authored a book on social network security and a book on information leakage detection and prevention. His primary research interests are computer and network security, cyber security, web intelligence, information warfare, social network analysis, and machine learning.