On Deception-Based Protection Against Cryptographic Ransomware

Ziya Alper Genç[1][0000−0001−7198−7437], Gabriele Lenzini[1][0000−0001−8229−3270], and Daniele Sgandurra[2][0000−0001−5238−8068]

[1] Interdisciplinary Centre for Security Reliability and Trust (SnT)
University of Luxembourg
ziya.genc@uni.lu, gabriele.lenzini@uni.lu
[2] Information Security Group, Royal Holloway, University of London
daniele.sgandurra@rhul.ac.uk

Abstract. In order to detect malicious file system activity, some commercial and academic anti-ransomware solutions implement deception-based techniques, specifically by placing decoy files among user files. While this approach raises the bar against current ransomware, as any access to a decoy file is a sign of malicious activity, the robustness of decoy strategies has not been formally analyzed and fully tested. In this paper, we analyze existing decoy strategies and discuss how they are effective in countering current ransomware by defining a set of metrics to measure their robustness. To demonstrate how ransomware can identify existing deception-based detection strategies, we have implemented a proof-of-concept anti-decoy ransomware that successfully bypasses decoys by using a decision engine with few rules. Finally, we discuss existing issues in decoy-based strategies and propose practical solutions to mitigate them.

Keywords: Ransomware · Cryptographic · Malware · Deception · Decoy

1 Introduction

In the last few years cryptographic ransomware (in short, crypto-ransomware) attacks have dominated the cyber-threats landscape [26]. They target the most valuable asset of today's computer users and companies: *data*. Crypto-ransomware encrypts these data which become inaccessible to their owners. If, during this task, the crypto-ransomware uses strong cryptography, it is unfeasible that the legitimate file-owner can recover the files' contents without the decryption key. Victims, users and companies alike, are then forced to pay a ransom if they want to regain access to their data[3].

The quick return in revenue together with the practical difficulties in the accurate tracking of cryptocurrencies, used to perform the ransomware payment by victims, have made ransomware a preferred tool for cyber-criminals. In particular,

[3] Some statistics show that nearly 50% of those companies who paid the ransom were actually able to recover their data back, e.g., see [7].

exploiting few zero-day vulnerabilities found in Windows operating system (OS), the most-widely used OS on desktop computers, has enabled ransomware to extend its threat and damage at world-wide level. For instance, WannaCry and NotPetya have affected almost all countries, impacted organizations, and the latter alone caused damage which costs more than $10 billion [11].

The huge and wide-spread impact of crypto-ransomware has quickly gained the attention of cyber-security researchers. In the last few years, several anti-ransomware strategies have been proposed, each implement several protection and detection strategies, such as: (i) deception-based protection [19,17]; (ii) controlling secure random number generator [10]; (iii) behavioral analysis of applications [15,5,23]; (iv) key escrow [16]; (v) network level defense [4]; and (vi) machine-learning detection [24]. While for all of the above approaches there are pros and cons, *deception-based protection* deserves a special attention for several reasons. In the context of ransomware, deception is primarily achieved by crafting artificial files, *decoys*, that the user is supposed not to see nor access. In particular, by placing these files among real files of user, a minefield-like area is established on the file-system. Whenever a process writes to a decoy file, it is immediately considered as a suspicious activity as any legitimate application would not access any of these files, and a predetermined response is taken.

One noteworthy aspect of deception-based ransomware detection is the lack of false positives (*e.g.,* typically decoys are hidden from users to prevent user's mistakes). Another outstanding property of this approach is that it provides real-time detection with minimal overhead as no additional computation is involved, such as those performed by behavioral detection. However, the main issue of using decoys for ransomware detection is that if the strategy used to create them is weak, then ransomware can detect the presence of decoys and skip them while building the list of target files to encrypt. Therefore, to be effective, decoy files should mimic as closely as possible real-user files to deceive ransomware. The problem of building a robust decoy strategy shares similar properties with that of creating a realistic sandbox environment to perform dynamic analysis of malware [3]. In this scenario, the ransomware needs to be deceived that it is running on a real host while it is actually being run and monitored in an artificial environment prepared by the malware analyst. In fact, some ransomware try to fingerprint the execution environment and look for traces of typical test systems, *e.g.,* vendor ID of device drivers, and act as benign programs if they suspect of being monitored.

While decoy-aware ransomware is not an emerging threat yet, cyber-criminals might turn their experience in fingerprinting sandbox environments for detecting decoy files. We envision this potential development and ask the following question to ourselves: *how secure are the current deception-based anti-ransomware systems?* It is crucial to find the answer of this question before such a development happens, as the damage of ransomware might be irreversible. Therefore, we take the task of analyzing the security of current decoy file strategies used to stop ransomware.

We begin with adapting an existing threat model of deception to ransomware. In §2, we define the measures of *quality* and *confoundedness*, which are particu-

larly applicable to ransomware, both (i) theoretically, *e.g.,* when the ransomware strategy is given; and (ii) practically, *e.g.,* for post-mortem examination of a monitored execution. Furthermore, the theoretical bounds of decoy-based ransomware defenses are also discussed. Next, we review anti-ransomware strategies employing decoy files in §3. Using our observations, in §4 we anticipate some anti-decoy techniques that ransomware might utilize to evade detection, and in §5, we discuss their mitigation. To support our arguments, we report the results of our experiments in §6. The related work is reviewed in §7. We state our view on ethics and explain our commitment to it for this research in §8. In §9, finally, we share the direction of our future research and conclude the paper.

2 Decoy Files: The Theory

Decoy documents, sometimes called *honey files*, are fictitious files first introduced as a deception mechanism to detect unauthorized accesses to computer systems (see [27]). Their goal is twofold: (a) to attract the attention intruders eager to steal information and lure them to access the files so revealing their presence *ex-ante*, and (b) to infiltrate bogus information that an insider entered in possession of the files may use eventually and somewhere. This signals an intruder's presence *ex-post*.

This second function, *i.e.,* serving as a beacon of an intrusion activity after the intruder has left the system, seems less relevant to ransomware. Cryptographic ransomware operates without exfiltrating information and with the goal to block access to files in house. The use of decoy documents as alluring bait, instead, can be pivotal in revealing a ransomware's activity and in enabling strategies to minimize the damage.

Decoy files with this purpose have been proposed in academic literature (*e.g.,* [18]) and also used in commercial products such as CryptoStopper [25], an anti-ransomware. In [17], the authors provide a solution that is pragmatically tailored to nullify the search and sorting strategies of eleven ransomware analyzed.

Beyond the specific decoy strategies implemented by a few anti-ransomware, what are the qualities that a decoy file has to enjoy to be effective? Not being recognized as a decoy is surely one, but others may be relevant too. Other works have addressed the question in other domains, mainly in intrusion detection. In particular, in [2], Bowen *et al.* identifies seven properties that a decoy file in its function of insider-trap has to enjoy, namely being: *believable*, that is recognized as if it were an authentic file[4]; *enticing*, that is alluring for the insider; *conspicuous*, that is, easily visible so to minimize the effort of the insider; *detectable* that is, serving well in detecting a malicious activity; *variability*, that is, not easily identifiable as bogus; *non-interference*, that is, not making hard for honest users to recognize the non-decoy files; and *differentiable*, that is, easily discernible by honest users.

[4] Juels and Rivest, who propose honeywords to detect a password leak, call it *flatness* [14].

Such properties, that in [2] come with probabilistic measures to quantify them, need of a little reinterpretation in the context of ransomware where the adversary is not a human masquerader aiming at finding and exfiltrating sensitive data. Properties such as "detectability", that in [2] is interpreted as hiding beacons (called decoy tokens) inside the files to trace them outside the system or injecting bogus information in a sort of counter-intelligence action, do not apply as is. Others, instead, do make perfect sense, like that of "being believable".

2.1 Quality Measures for a Decoy Strategy

Although understanding which properties a decoy file should have *per se*, in this paper we prefer a pragmatic approach. We define two measures that are directly observable. The first is about the quality level of "deceptfullness" of a decoy strategy against a chosen ransomware. The second is a measure of its usability which directly links to the rate of false positives due to the activities of honest users.

We assume that the set of decoy files, D, is generated following a strategy $\mathsf{g}(\neg D, k)$ that can, but not necessarily, depend on some non-decoy files $\neg D$ (those which have to be protected) and some secret parameter k. We have no requirements on g but when talking of a "file" we mean content, name, and meta-data (*e.g.,* be a hidden file, date of creation, last access, and so on), and also the directory structure that includes them. Function g can be static, that is defining D once and for all, or dynamic, that is changing D with time. It can be deterministic or randomized. We do not enter into the details of this procedure but a "good" g should make it hard for an adversary, A, to decide whether a given file f does belong to D.

Similarly, A's decision making can be based on a simple or complex selection strategy (*i.e.,* it can be a deterministic search and sort, or can be a randomized search), however, what counts is that we can observe it. In other words, we can design experiments where A operates in an environment where it can access all files and where it is possible to find out what files A selects and encrypts and in which order. So, let $X_A^{\mathsf{g}}(f)$ be the random variable that returns the number of files that A encrypts before selecting and encrypting f, when A runs in a file system with files $F = \neg D \cup D$, and where D, the set of decoy files, is generated using g. For $S \subseteq F$, we write $X_A^{\mathsf{g}}(S)$ to refer to the event $\min_{\forall f \in S}\{X_A^{\mathsf{g}}(f)\}$.

Definition 1 (Measure of Quality of a Decoy Strategy). *Let be A a ransomware, D the set of files generated by a decoy strategy g, $F = D \cup \neg D$, $S \subseteq F$ a set of files, and n a natural number. The quality of a decoy strategy g is defined as follows:*

$$\Pr\left[X_A^{\mathsf{g}}(S) = n\right] \tag{1}$$

It is the probability that A encrypts n other files before encrypting one in S.

When $S = D$, Eq. (1) tells us the probability that A encrypts exactly n non-decoy files before encrypting a decoy file.

We can use Def. 1 in many ways. Intuitively, $\Pr[\![X_A^{\mathtt{g}}(D) = 0]\!]$ indicates the quality of a decoy generation strategy in fooling immediately A. A good decoy strategy should minimize $\Pr[\![X_A^{\mathtt{g}}(D) > 0]\!]$ that is the probability that a ransomware encrypts some good files before encrypting a decoy. It can be desirable to have a $\mathtt{g}$ that works more steadily against A: $\Pr[\![X_A^{\mathtt{g}}(\neg D) = n]\!]$ tells us the quality of a decoy strategy to keep ransomware busy for n decoy files before it eventually starts encrypting a good file. Intuitively, a good decoy strategy should maximize probability $\Pr[\![X_A^{\mathtt{g}}(\neg D) \geq n_0]\!]$, if n_0 is the minimal number of files that a certain anti-ransomware strategy needs before detecting that there is an illegitimate encryption in place. Often it can be sufficient to have $n_0 = 1$, but it may depend on the false positive rate of the anti-ransomware.

A strategy $\mathtt{g}$ that is also *usable* should non-interfere with the ability of a user U to recognize a non-decoy file. The experimental setting we propose to measure this quality assumes a random variable $Y_U^{\mathtt{g}}(S)$ that returns $\mathbf{1}$ when user U accesses to one of the files in a set S over a period of time in a working session (*e.g.,* the time in-between two lock-screens); it returns $\mathbf{0}$ if U does not access to any file in S. We are interested in the following measure, that we call *confoundedness*, in which it indicates whether the user can get confused about his/her accessing non-decoy files.

Definition 2 (Measure of Confoundedness). *Let be U a user and D the set of decoy file generated according to a strategy $\mathtt{g}$, $F = D \cup \neg D$, and $S \subseteq F$ a set of files. Confoundedness is defined as follows:*

$$\Pr[\![Y_U^{\mathtt{g}}(S) = \mathbf{1}]\!] \tag{2}$$

It is the probability that U accesses a file in S within a working session.

Def. 2 is useful when we instantiate $S = D$. Intuitively, $\Pr[\![Y_U^{\mathtt{g}}(D) = \mathbf{1}]\!] = 0$ means that U never gets confused. Therefore, a usable decoy strategy should be able to keep $\Pr[\![Y_U^{\mathtt{g}}(D) = \mathbf{1}]\!]$ small, where small should be set according to empirical measure of the user experience, a value beyond which there is evidence that the user may switch off the decoy defence [1].

2.2 On the Theoretical Limits of Anti-Ransomware Decoy Strategies

The two measures we have defined in the previous section can be effectively computed once a decoy strategy has been defined and when either a threat model for A is set or when we have the possibility to observe A in execution.

We discuss here those measures in respect of a particular threat model for A. Bowen *et al.* [2] consider an adversary that is an insider, and define a "highly privileged" adversary as one having almost full control of the system, including knowing of the existence of a decoy strategy but without knowing it (*i.e., A* ignores $\mathtt{g}$). We assume at least the same "highly privileged" adversary. Under this condition, the adversary follows its own strategy to come out with a list of target files T from which to pick.

Even this "highly previledged" adversary may be however not as powerful as a ransomware can. In fact, due to its being able to run in a victim machine, ransomware may have a further weapon that instead is not available to a human insider: observing what files U accesses during a working session. Let us call this set of files $[F]_U$ and let us assume here that this is the set of files which U cares about: s/he would be willing to pay a ransom to have them back. Under this threat model we encounter a serious limitation of using decoy files as a general protection against ransomware. If g is perfectly usable, then its confoundedness is null, that is $\Pr[\![Y_U^{\mathsf{g}}(D) = \mathbf{1}]\!] = 0$. If A observed $[F]_U$, then A could simply choose among the files in $[F]_U$ to have a perfect strategy to avoid picking decoy files even without knowing how g works.

In general, however, $\Pr[\![Y_U^{\mathsf{g}}(D) = \mathbf{1}]\!] = p > 0$, which means that $[F]_U \cap D \neq \emptyset$. Assuming that A picks a target file in $[F]_U$ at random, it still has $\frac{|[F]_U \cap \neg D|}{|[F]_U|} \cdot p + (1-p)$ chances to pick up a good file. Although a precise statistics can be computed only if all the parameters are set, if p is negligible it still gives a good chance of success to A. Instead if p is significant, that is, there is a high probability that U accesses a decoy file, it seems that it is better that U accesses as many decoy files as possible, which seems going against usability. Besides, we have to consider that A can also couple its random picking in $[F]_U$ with its own strategy to select files that are not decoy. This strategy is based on some intrinsic quality of the files, such as their names, extension, location, and this combination of strategies leads to an interesting theoretical question about what is the best game for A and for g.

As far as we know such an A does not exists, and other strategies can be put in place to detect the presence of such an intrusive and curious ransomware, but our argument should be considered to raise awareness of what limits do exist when designing any g.

3 Anti-Ransomware Systems with Decoy Files

In this section, we give a brief description of some current anti-ransomware systems that uses deception-based strategy by implementing decoy files.

3.1 CryptoStopper

CryptoStopper is a commercial anti-ransomware solution developed by Watch-Point Data [25] and is advertised as "software to detect and stop ransomware". It places "randomly generated watcher files" in file system to detect ransomware. According to WatchPoint Data, the average time for CryptoStopper to detect a ransomware is 9 seconds.

In the case that malicious activity is detected, *i.e.,* a process tries to write to a decoy file, CryptoStopper alerts the system administrator and the infected host is shut down. Furthermore, other computers at the network are notified so that (if they are running CryptoStopper) they drop packages coming from the infected host, isolating that machine from the network. In this regard, CryptoStopper can

also be viewed as a local threat intelligence system that can protect the network from a zero-day ransomware with minimal loss.

3.2 RWGuard

RWGuard [18] unifies techniques from previous proposals in a single tool to mitigate cryptographic ransomware. To detect ransomware, RWGuard comprises dedicated modules to (i) check if a decoy file is modified; (ii) monitor process behaviours; (iii) identify abnormal file changes; (iv) classify user's cryptographic activity; and (v) control built-in cryptographic Application Programming Interface (API).

Decoy generator tool in RWGuard uses the original files of the user. The authors state that the names of decoy files are generated similar to the genuine user files and in a way that decoys can be clearly identified, though, the exact procedure is not described. The number of decoy files is determined by the user for each directory. The extension list of decoy files are static (`.txt`, `.doc`, `.pdf`, `.ppt`, and `.xls`) and their contents are created by copying from user's genuine files. The sizes of decoy files are randomly chosen from a range based on the sizes of user's genuine files while the total size of decoy files is limited to 5% of user's genuine files.

3.3 R-Locker

To detect ransomware, R-Locker [12] employs decoy files but in a slightly different manner. The proposed approach is to create a central decoy file in user's home directory, which is actually a first-in first-out (FIFO) special file, *i.e.,* a named pipe. Next, R-Locker writes a few bytes to this FIFO file, which will not be read until a process accesses to the pipe. Consequently, any process trying to read this pipe will trigger the protection module and will be detected. Authors suggest to place multiple symbolic links pointing to the central decoy file in various locations on the file system to decrease the time to detect ransomware.

In contrast to the other anti-ransomware systems, R-Locker interprets any read access to decoy files as ransomware activity. The false-positive rate of this approach, *i.e.,* the frequency of occurrences of read operations initiated by a legitimate process like background service or a system daemon, is not evaluated, though.

3.4 Decoy Generation Strategy of Lee et al.

In [17], Lee *et al.* reverse engineered 11 cryptographic ransomware samples from different families and analyzed their file system traverse patterns. Based on this analysis, the authors developed a method which generates decoy files in directories that the samples they analyze starts to traverse from. The authors also found that these samples sort files alphabetically. Based on this observation, the proposed method creates two decoys: one with a file name which comes first

in normal ordering, and another decoy that comes first in the reverse order, to nullify both ordering strategies.

In addition to the current ransomware, [17] also attempts to anticipate the possible evasion methods that may come up in the future. In this regard, the authors extend their algorithm by taking into account the alternative orderings based on file size and access time. Furthermore, the authors mention the case that ransomware orders files randomly and propose "monitoring random function calls to detect the ransomware which traverse in random order". However, we were not able to understand how exactly the proposed algorithm works.

4 Decoy-aware Ransomware

In the previous section we discussed some key elements of the strategies of a few anti-ransomware systems. They can be considered instances of what we called g. We now imagine a few anti-decoy strategies for a ransomware, which then becomes a *decoy-aware ransomware*. Such strategies can be either to black-list files that the ransomware considers decoy, thus not to be encrypted, or to list files that it labels as user files, thus to be encrypted. In particular, we describe how ransomware can detect decoys by relying on heuristics, for instance the presence of zero-filled files (Section 4.1) and on statistical methods (Section 4.2). Then we describe how ransomware can find out the files accessed by users which are quite likely non-decoy files (Section 4.3).

4.1 Detecting Static Decoys through Heuristics

A ransomware can look for patterns that are indicative of decoy files generated by some (weak) strategy, such as files that are hidden or filled with empty values, or where the creation date and content include a static pattern which can be discovered by the attackers.

Similarly to the case of *anti-analysis* techniques (e.g., anti-sandbox/anti-debugging), ransomware authors can first create a database of fingerprint-based decoy checks to be included in future versions of ransomware. This set of fingerprinting checks can be then performed at run-time when scanning the victim computer to create the list of target files to encrypt. Note that, differently from the case of anti-analysis, in which a malware typically stops performing any malicious actions if it detects signs of an analysis environment, ransomware does not stop performing its malicious operations when decoys files are detected but simply excludes them from the target files.

One fact must be observed. A false positive is less troublesome in the case of a ransomware's anti-decoy detection with respect to anti-analysis. If the ransomware skips a user file mistakenly believed to be a decoy, this has a little impact on the overall strategy of the ransomware. The ransomware still encrypts several other files, and the request for ransom still holds.

In the following, we describe two heuristics for decoys files. For the first (see Alg. 1) decoys are files that are hidden and empty. For the second (see Alg. 2) decoys are non-regular files, such as symbolic links or named-pipes.

Algorithm 1 Collect files that are not hidden or filled with zero value.

```
1:  function COLLECT(path)                                  ▷ Directory of files to scan.
2:      FileList ← EnumerateFiles(path)
3:      GenuineList ← ∅
4:      for all f ∈ FileList do
5:          if IsHidden(f) then
6:              allNull ← True
7:              while not EOF do
8:                  b ← f.ReadByte()
9:                  if b ≠ 0 then
10:                     allNull ← False
11:                     break                    ▷ f might not be decoy, try next file.
12:             if allNull = True then
13:                 GenuineList ← GenuineList ∪ {f}
14:         else
15:             GenuineList ← GenuineList ∪ {f}
16:     return GenuineList
```

Algorithm 2 Collect files on Ext4 FS that are not a FIFO or a symbolic link.

```
1:  function COLLECT(path)                                  ▷ Directory of files to scan.
2:      FileList ← EnumerateFiles(path)
3:      GenuineList ← ∅
4:      for all f ∈ FileList do
5:          if IsPipe(f) then
6:              continue                                    ▷ Skip pipes,
7:          else if IsSymbolicLink(f) then
8:              continue                                    ▷ and symbolic links.
9:          else
10:             GenuineList ← GenuineList ∪ {f}
11:     return GenuineList
```

With the exception of [12], all deception-based anti-ransomware systems that we know trigger protection when a decoy file is modified or deleted but not when the file is only read. Thus Alg. 1 ad Alg. 2 can work mostly undetected. And, since modern file systems store a file's contents and metadata separately (see §4.2), Alg. 2 might even be able to work without reading the actual files but only their metadata, reaching full stealthiness. For example, on Linux OS, Alg. 2 can obtain all the required information by calling `readdir()`[5] function.

[5] See the manual page at http://man7.org/linux/man-pages/man3/readdir.3.html.

4.2 Distinguishing Decoys Using Statistical Methods

Let us first briefly recall some technical details of the file storage on Windows OS to understand file attributes and metadata.

On modern versions of Windows platform, New Technology File System (NTFS) is the default file system for controlling the storage and retrieve of data on the disk. In NTFS, all data are stored in files. In addition to the data stored as regular files, the internal data for structuring the file system are also stored as files. These auxiliary files are called *metadata* files. Among the metadata files in NTFS, the most important is the Master File Table (MFT), which contains the description of all files on a disk volume. This information includes the file name, time stamps, *e.g.,* date created and date last accessed, security identifiers and file attributes, *e.g.,* hidden and read-only. Table 1 shows a list of selected attributes for files on NTFS volumes.

Table 1. Selected attributes for NTFS files. The complete list is available at [22].

Attribute	Attribute Type Name	Description
Standard Information	`$STANDARD_INFORMATION`	File attributes such as read-only, archive, and so on; time stamps, including when the file was created or last modified.
File Name	`$FILE_NAME`	The file's name in Unicode 1.0 characters.
Data	`$DATA`	The contents of the file.
Attribute List	`$ATTRIBUTE_LIST`	A list of the attributes that make up the file and the file record number of the MFT entry where each attribute is located.

Therefore, since the decoy files are not supposed to be accessed by users, its reasonable to expect that their access pattern shows difference from genuine files. Under NTFS, it is possible to obtain certain parts of information about access patterns, for example, `$STANDARD_INFORMATION` attribute stores last modification time. Furthermore, one can access file creation time along with other file attributes. This basic information, statistically analyzed, can be used to distinguish decoy and genuine files.

In fact, this observation was made by Rowe [21] in 2006. Rowe developed a tool which computes statistics on a file system and compares this data to that of a typical real system. If there exist a significant discrepancy in the two statistics, this suggests the existence of a deception-based defense. The proposed tool computed 36 metrics for each directory using features based on file attributes, including (i) file name; (ii) file size; (iii) file type; and (iv) last modification time. Furthermore, these 36 metrics are augmented with their standard deviations, resulting in a feature vector of 72 metrics. For a file system i, let μ_{ik} and σ_{ik} denote the mean and standard error of metric k. Rowe assumed that all metrics have equal weights and used the following formula to compute the difference

between systems i and j:

$$s_{ij} = \left(\frac{1}{72}\right) \sum_{k=0}^{35} \left[\frac{|\mu_{ik} - \mu_{jk}|}{\sqrt{\sigma_{ik}^2 + \sigma_{jk}^2}} + \frac{|\sigma_{ik} - \sigma_{jk}|}{2\sigma_k} \right] \tag{3}$$

According to results of Rowe's experiments, Eq. 3 excels in finding discrepancies based on time & date information.

The feasibility of this technique, among with its efficiency makes it a valuable tool for a decoy-aware ransomware. Unless the anti-ransomware system updates the decoy files in such a way to mimic the user behaviour, Rowe's results shows that statistical techniques increase the chances of attackers against decoy-based defenses. We elaborate on this issue more in §5.

4.3 Monitoring User to Reveal Non-decoy Files

An anti-ransomware system that uses decoy files is supposed to be designed in such a way to let legitimate users either be able to differentiate between genuine and decoy files, or not to be able access decoy files for instance by hiding them. Either way, the goal is to prevent the user from accessing a decoy file.

Relying on this consideration, a decoy-aware ransomware can obtain a list of genuine files by monitoring the user activity. In a metaphor, by following the user's steps, the ransomware can pass unharmed through the minefield of decoys. We imagine two of such decoy-aware ransomware strategies:

- (see Alg. 3): inject a spy module into `Explorer.exe` to monitor which files are accessed by user applications Ransomware can further compute the hash of the file at first access time and check it later for changes to detect modifications – this might be a sign of a "valuable" file (though, not always this property holds: e.g. pictures are rarely changed, and they are very valuable for ransomware).
- (see Alg. 4): Enumerate all processes and inject an interceptor module which hooks `WriteFile` API Replace the `WriteFile` API with the encryption routines (however, this strategy also requires the ransomware to keep information about which parts of the files have been overwritten to be able to properly decrypt it later).

Algorithm 3 Monitor User.

```
1: function MONITOR
2:     Exp ← FindProcess(Explorer)
3:     InjectProcess(Exp, SpyModule)
4:     GenList ← ∅
5:     while true do
6:         f ← Listen(SpyModule)
7:         GenList ← GenList ∪ {f}
8:     return GenList
```

Algorithm 4 Replace `WriteFile`.

```
1: function REPLACE
2:     PList ← EnumAllProcesses()
3:     for all p ∈ pList do
4:         InjectProcess(p, InterceptMod)
5:         wf ← GetFuncAddr(WriteFile)
6:         if wf ≠ NULL then
7:             Replace(wf, encFile)
8:     return Success
```

Decoy-aware ransomware implementing Alg. 3 and Alg. 4 can run in user-space, *i.e.,* no kernel-mode component is required; so, the ransomware would typically have the sufficient privileges to run.

5 Discussion: the Endless Battle

History suggests that the malware mitigation is a multifaceted combat where the cyber-criminals constantly searches for a hole in the battlefronts. It is not a secret that, to achieve their nefarious aims, ransomware authors also acquire new techniques to exploit the limitations of defense systems. A good deception-based anti-ransomware strategy, say $\mathtt{g}$, we argued in Section 2, should be, at least, such that to maximize the probability for a ransomware A to encrypt first any decoy files (*i.e.,* $\Pr[\![X_A^{\mathtt{g}}(\neg D) > 0]\!]$); similarly, it should also minimize the probability it starts encrypting some genuine files first (*i.e.,* $\Pr[\![X_A^{\mathtt{g}}(D) > 0]\!]$). Such probabilities can be enriched to consider for how long (*i.e.,* for how many encrypted files) g is capable of keeping the ransomware in check.

In §6 we give an experimental estimation of those measures of quality for some of the deception-based anti-ransomware strategies—those we could get access to plus two we designed ourselves and that we describe in this section—against the ransomware strategies that we have imagined to exist and that we implemented and run. Here, we discuss how they can minimize the damage of a ransomware attack.

To begin with, as we argued in §4.1, the static decoy files can be practically discovered by a decoy-aware ransomware and therefore should be avoided. In order to prevent fingerprinting of the decoys employed, the defense system should include randomness in the decoy generation procedure. Note that this property should not be understood as filling the decoy file with Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) outputs as the ransomware can also detect the unusually high entropy in the file content. Though, we cannot reach an ultimate decision in such case; ransomware may interpret the file as a trap and skip or a valuable data, *e.g.,* encrypted key vault, and attack.

As we argued in §4.2, ransomware can obtain crucial information from the metadata in a file system and use statistical techniques which might enable to unveil decoy files. RWGuard updates the decoy files *periodically* to mitigate this potential attack, however, we believe any update pattern would result in a discrepancy. Reasonably, the best protection level looks like reflecting user behaviour on decoys. We left such a decoy system to be realized in a future work. Randomness is also vital when updating the decoy files (see later). That said, an under-studied aspect of decoy files is the header-extension relation. An inconsistency in header bytes and file extension might make a decoy-aware ransomware suspicious, therefore, these two should be coherent. Moreover, the decoy updater process should be careful if a new content is added to a file randomly, and target the body of the decoy file. This can be usually achieved by skipping the first few bytes of the decoy file.

Decoy-aware ransomware that observes user-behaviour, whose existence we speculate in §4.3, could be quite hard to beat. As a mitigation strategy, an anti-ransomware could add the following functionalities to current decoy systems:

(**F**1) **Add noise to user activity** by emulating user's opening a decoy file so that the spy module adds a decoy file to *GenList*. If the file accessed by the user is modified, update a decoy file to mimic the user.

(**F**2) **Verify the data written to decoy file** to check if the decoy updater is compromised.

To bypass these strategies, a ransomware may ignore a series of user activities happening in a short time-frame. Therefore, to obfuscate the functionality of (**F**1), a decoy updater may choose to delay the update process for a random time period or – ideally – according to user's access pattern. Predetermined update patterns may also be identified by attackers. Therefore, (**F**2) must use randomized data while updating decoy files. On the other hand, (**F**2) should also avoid writing to the file headers so that file's magic value does not conflict with its extension. Although it might be unfeasible to locate the process that initiated the attack, a protection system might suspend all file system activity when an inconsistency is reported by (**F**2). In §6.2 we build such a anti-ransomware based on these ideas, called *DecoyUpdater*, and estimate its quality.

Another under-studied aspect of anti-ransomware solutions employing decoy files is the *usability*. This topic deserves an independent research, but we would like point out two issues. The placement of decoys are studied fairly well in the past; however, the effects to user's daily workflow needs more research. For example, if the decoy files are generated with the hidden attribute, it would be safe for ransomware to attack only visible files. This may suggest to generate the decoys as visible files, and therefore at least an estimation of our measure of confoundedness, $\Pr\llbracket Y_U^{\mathbf{g}}(D) = \mathbf{1} \rrbracket$ is required.

The number of decoys has also another significance: to evade ordering strategies described in §3.4, decoy-aware ransomware might utilize a random ordering. In this case, obviously, the more decoy files, the faster detection speed. It should be noted that, the number of decoys may not be useful against selective attacks, though.

Lastly, deception-based defense systems are highly linked to the security of the host OS. If for example, ransomware can write to Master Boot Record (MBR) and reboots the target machine, it might be able to load a malicious kernel and encrypt the files, as NotPetya does. However, this is rather a generic issue about runtime protection systems and applies to the most of the other anti-ransomware solutions.

6 Experiments and Quality Measures

We demonstrate the feasibility of our speculated decoy-aware ransomware, and we measure against it, the quality of decoy of CryptoStopper, the only anti-ransomware we could have access among the ones we described in §3, and of

DecoyUpdater, a proof-of-concept of an anti-ransomware that implements the mitigation strategy that we described in §5.

6.1 Revealing Static Decoys

To demonstrate the feasibility of the avoiding static decoys, we have developed a prototype implementing Alg. 1. We conducted the experiments on a clean install of Windows 10 (version 1809) virtual machine (VM) running atop VMware Fusion. In the experiments, we populate the VM with 30 files, namely 5 from each of the following file types that are typically selected by ransomware: `.txt`, `.jpg`, `.png`, `.docx`, `.xlsx` and `.pdf`. These 30 files are placed in user's directories targeted by ransomware, including Desktop and Documents. Once the artificial environment is ready, we have tested it on the latest version CryptoStopper at the time of writing.

We implemented Alg. 1 in C# language and run it on the test system. As shown in Fig. 1, our prototype successfully identified all the 25 decoy files generated by CryptoStopper and skipped them while building the list of targeted files to encrypt.

AntiStaticDecoy Console

```
AntiStaticDecoy Prototype
Target directory: C:\Users\RWTest\Desktop

[GENUINE] C:\Users\RWTest\Desktop\accesscontrol.jpg
[GENUINE] C:\Users\RWTest\Desktop\AccountSummary.xlsx
...
[GENUINE] C:\Users\RWTest\Desktop\weakprng.png
[GENUINE] C:\Users\RWTest\Desktop\WeeklySchedule.txt

Total number of genuine files: 30

[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Add Resolve.docx
[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Backup Convert.docx
...
[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Unregister Test.avi
[DECOY] C:\Users\RWTest\Desktop\Parlay Permit\Use Initialize.pptx

Total number of decoy files: 25

Proceed with encryption? y/n
```

Fig. 1. Console output of the prototype of Alg. 1.

The fact that CryptoStopper (CS) relies on a static strategy means that the set of decoy files has features that do not change in time. Thus, we can

safely infer from our experimental result that an estimation for our measure of quality of decoy strategy is $\Pr[\![X^{CS}_{Alg_1}(\neg D) > 0]\!] = 0$ and $\Pr[\![X^{CS}_{Alg_1}(D) > 0]\!] = 1$. Actually, we estimate, $\Pr[\![X^{CS}_{Alg_1}(D) > n]\!] = 1$ for all $0 \leq n \leq |F|$. Thus our version of a hypothetical ransomware outsmarts CryptoStopper. In defence of CryptoStopper we have to say that we tested it against a potential but a non-existing ransomware variant. So the lesson we can learn from this experiment is not precisely on CryptoStopper, but rather on a strategy that generates decoy files with fixed and communal properties. Knowing it, a ransomware designer can easily implement a counter-measure that sieves those files from the rest.

Decoy-aware ransomware implementing Alg. 2 were supposed to be tested against the other anti-ransomware systems in §3; we requested the prototypes of [18] and of [12] to conduct experiments but not received any response from the authors yet. The method proposed in [17] is not published, so it is unavailable.

6.2 Revealing non-Decoy Files by Monitoring Users

To demonstrate the feasibility of our hypothetical ransomware monitors users, we implemented Spy and Replace. They realize Alg. 3 and Alg. 4, respectively.

Spy is written in C# and uses `FileSystemWatcher`. The target directory to watch for events is set to `%USERPROFILE%\Desktop`. Spy implements `OnChanged` and `OnRenamed` event handlers to receive file change notifications, and `OnCreate` for watching new files.

Replace is implemented as a Dynamic-Link Library (DLL) module using C++ language, and injected into the target process by calling `CreateRemoteThread`. Once the DLL is loaded into the target application's memory area, all hooking operations are performed using Detours library [13] from Microsoft Research. After loading the malicious DLL, Replace hooks `WriteFile` API and whenever `WriteFile` is called, it invokes `Fake_WriteFile` that encrypts the whole content of the file using `CryptEncrypt` with a hardcoded key[6].

The experiments were conducted on a similar setup environment as the previous one, namely a Windows 10 VM running atop VMware Fusion with 30 user files created and placed similarly as before. In addition to these user files, we have also added two decoy files from each file type.

During the experiments, we first run Spy, and then use various applications to open and read the content of all of the 30 user files, by writing at various time intervals to the `.txt`, `.doc` and `.xls` files only. As shown in Fig. 2, Spy is able to successfully observe all the user files that have been modified.

Thus, speculatively Spy (as well as Replace) is able to nullify existing decoy methods, be this one CryptoStopper or one of the solutions we described in §3.

The only significant comparison is against the anti-ransomware that employs (**F**1) and (**F**2) (see §5) in a decoy-file based defense system. This is the prototype we called DecoyUpdater[7] (DU). It should be noted that our aim is not to provide

[6] For the sake of proof-of-concept: a real ransomware would use a strong key-management strategy.

[7] Available under GPLv3 at https://github.com/ziyagenc/decoy-updater.

Spy Console

```
Target Directory: C:\Users\RWTest\Desktop

20:58:07.814 CHANGED C:\Users\RWTest\Desktop\MyPasswords.txt
20:58:13.814 CHANGED C:\Users\RWTest\Desktop\TermProject.doc
20:58:21.002 CHANGED C:\Users\RWTest\Desktop\Essay.doc
20:58:30.439 CHANGED C:\Users\RWTest\Desktop\MyNotes.txt
20:58:42.626 CHANGED C:\Users\RWTest\Desktop\AccountSummary.xls
20:58:46.955 CHANGED C:\Users\RWTest\Desktop\Costs.xls
```

Fig. 2. Console output of Spy on unprotected system.

a full-fledged deception system. Rather, we attempt to evaluate our technique and prove the validity and efficiency of the underlying idea (see §8).We have developed DecoyUpdater in C# language. For ease of implementation, we have used the `System.IO.FileSystemWatcher` class, as it is very useful for monitoring file system events, such as for opening/deleting/renaming files and directories or detecting changes in file contents.

We have started DecoyUpdater and have repeated the same previous actions on files (namely, reading and writing), seen in Fig. 3. This time, however, event logs in the Spy also show the file activities performed on the decoys, as in Fig 4. Since the logic behind Spy is bound to the OS and does not depend on other factors, from the only experiment we have run, we obtain $\Pr[\![X^{DU}_{Spy}(D) > 0]\!] < 1$ and $\Pr[\![X^{DU}_{Spy}(\neg D) > 0]\!] > 0$. A more precise estimate requires to equip *Spy* with a decisional strategy over the collected files, and we leave this a future work.

DecoyUpdater Console

```
Target Directory: C:\RWTest\RWTest\Desktop

07:15:31.608 CHANGED C:\RWTest\RWTest\Desktop\ToDoList.txt
07:15:33.616 UPDATED Decoy file: Addresses.txt
07:15:50.790 CHANGED C:\RWTest\RWTest\Desktop\MyPasswords.txt
07:15:51.792 UPDATED Decoy file: PhoneNumbers.txt
07:15:58.641 CHANGED C:\RWTest\RWTest\Desktop\MyNotes.txt
07:15:58.643 UPDATED Decoy file: Addresses.txt
```

Fig. 3. Console output of DecoyUpdater while Spy is active. The decoy files `Addresses.txt` and `PhoneNumbers.txt` are randomly picked and updated after a random delay of 5 seconds maximum.

As the final set of experiments, first we have executed a helper program to inject the Replace module into a target application, namely `Notepad.exe`. Using

Spy Console

```
Target Directory: C:\Users\RWTest\Desktop

07:15:31.608 CHANGED C:\Users\RWTest\Desktop\ToDoList.txt
07:15:33.616 CHANGED C:\Users\RWTest\Desktop\Addresses.txt
07:15:50.790 CHANGED C:\Users\RWTest\Desktop\MyPasswords.txt
07:15:51.792 CHANGED C:\Users\RWTest\Desktop\PhoneNumbers.txt
07:15:58.641 CHANGED C:\Users\RWTest\Desktop\MyNotes.txt
07:15:58.643 CHANGED C:\Users\RWTest\Desktop\Addresses.txt
```

Fig. 4. Console output of Spy while DecoyUpdater is active. Note that the list contains the decoy files `Addresses.txt` and `PhoneNumbers.txt`.

this target application, we have opened all the `.txt` files and added some random text into all of them, and saved them. After this operation, the `.txt` files were encrypted by Replace crypto-module. Second, we have activated DecoyUpdater and repeated the same steps in the previous experiment. In this scenario, at each try, DecoyUpdater's operations were intercepted by the Replace module. However, Replace's activities has been successfully reported[8] by DecoyUpdater in the logs, which is shown in Fig 5. Again, since the strategy's logic depends only on the OS, we can, from only one experiment, estimate that $\Pr[\![X^{DU}_{Replace}(D) > 0]\!] < 1$ and $\Pr[\![X^{DU}_{Replace}(\neg D) > 0]\!] > 0$. DecoyUpdater strategy has the potential to become robust deception-based anti-ransomware system, but demonstrating this claim is left for the future.

DecoyUpdater Console

```
Decoy Updater v1.0

Status: ACTIVE
Target Directory: C:\Users\RWTest\Desktop

22:00:27.553 CHANGED C:\Users\RWTest\Desktop\MyPasswords.txt
22:00:30.585 UPDATED Decoy file: PhoneNumbers.txt
22:00:30.585 WARNING Unexpected Write: PhoneNumbers.txt
```

Fig. 5. Console output of DecoyUpdater while Replace is injected into Notepad.exe and active. The logs shows that malicious activity on the decoy file `PhoneNumbers.txt` is detected.

[8] Due to the limited capability of `System.IO.FileSystemWatcher` class, we could observe the malicious activity, yet we were not able to identify the process ID of Replace and terminate it. That would be possible with developing a file system mini-filter, which is an implementation effort.

7 Related Work

In the previous sections, we have investigated some related work involved with the findings of this research. In this section, we summarize other works related to the use of decoys in ransomware mitigation.

One of the first honeypot systems against ransomware is proposed by Moore in [19], which tracks the number of files accessed on specified directories. The system implements a hierarchical multi-tier response model. Depending on the number of files accessed, the level of severity is determined and the corresponding countermeasure is applied.

Moussaileb *et al.* in [20] developed a post-mortem analysis system that detects ransomware activity using machine learning techniques. In their analysis, authors investigated the directory traverse patterns of processes and classified ransomware based on traversal traces in decoy directories.

Feng *et al.* in [9] intercepts `FindFirstFile` and `FindNextFile` APIs to manipulate file system traverse of processes so that whenever a process looks for a file, it is first served with a decoy file. Once the process finishes its task on the decoy file, the monitoring module of the system perform checks employed in behavioral analysis systems. After the checks, a process that shows malicious traits is terminated.

8 Ethical Considerations

Working on the ransomware threat by pointing out the potential limitations in current anti-ransomware defences raises an ethical question: could these insights be misused?

This ethical issue can be related to dual-use of research, which is mentioned in Article 2 of Council Regulation (EC) No 428/2009 [6] and its amendment Council Regulation (EU) No 388/2012. It defines dual-use items as "items, including software and technology, which can be used for both civil and military purposes [...]". Recently the EC has released a guidance note [8], where it comments on "Misuse of research", which has to be understood as "research that can generate knowledge, materials, methods or technologies that could also be used for unethical purposes" and in this phrasing, we recognize the ethical matter of our research. In adherence to the guidance, we comment on the risks of our research and we state ourselves that we behave to reduce the risk of misuse. By pointing out the potential and theoretical weaknesses in current anti-ransomware strategies, we may give suggestions on how to improve current variants, but we also warn cyber-security analysts and help them proactively to improve current anti-ransomware. It must be said that we work not by discovering bugs in applications—disclosing them will immediately have negative consequences. We rather discuss what we think are limitations in specific approaches against ransomware. Thus, using our arguments to build a fully-fledged malware requires to fill a non trivial knowledge and technological gap.

Whenever, in support to our research, we implement some piece of software to test a specific anti-ransomware application, we do not disclose any code. This

removes the risk that it may be re-used inappropriately. At the same time, we dutifully inform of our findings the authors of the application that we have put to a test. We invite them to challenge our arguments and evidence and we warn them that, were our speculations true, there could be a way to circumvent what they propose as a defence. We hope in this way to contribute to improve it too.

9 Conclusion and Future Work

Decoy-based strategies have been successfully used in providing evidence of an intrusion into a computer system. They have been called in different ways, the most common ones using the prefix 'honey-' as in honey pot, honey words, honey files, and honey token. Their use against malware, such as ransomware, is however still in its infancy, and there is little evidence that mitigating strategies that have worked against human intruders might work against ransomware. From one side, some applications may lack certain specific features that are usually exploitable to lure a human adversary into committing false steps – this makes malware immune to certain decision bias and vulnerabilities; from the other side, as the ransomware is running in the host system it might have access to additional capabilities, e.g. that of spying file activities, that are not available to a system intruder.

In this work we have looked into what limits decoy strategies may encounter when applied against ransomware. We first address the issue from a theoretical point of view, and then we have described a practical proof-of-concept that shows how some existing decoy-based solutions can be easily defeated. The results of our experiments show that we need to re-design the generation of honey documents so that their use against future ransomware will be as effective as their use against human intruders is. Our findings also provide the opportunity of investigation for two future directions. In the first one, an hypothetical strong adversary may be recognized and stopped by using other complementary strategies than those based on decoy files, and the research question is how to effectively combine different anti-ransomware strategies. In the second one, any anti-ransomware that relies on decoy files has to consider its usability, a quality that we have proposed be measured in terms of confoundedness, that is, how probably is that decoy files confuse a honest user into accessing them. We argue that finding the right balance for a decoy between being effective without confusing the user (who might then decide to switch off the defence, or change it for another) is a research challenge by itself that has to be addressed.

Acknowledgements

This work was partially funded by European Union's Horizon 2020 research and innovation programme under grant agreement No 779391 (FutureTPM) and by Luxembourg National Research Fund (FNR) under the project PoC18/13234766-NoCry PoC.

References

1. Balfanz, D., Durfee, G., Smetters, D.K., Grinter, R.E.: In search of usable security: five lessons from the field. IEEE Security Privacy **2**(5), 19–24 (Sep 2004)
2. Bowen, B.M., Hershkop, S., Keromytis, A.D., Stolfo, S.J.: Baiting inside attackers using decoy documents. In: International Conference on Security and Privacy in Communication Systems. pp. 51–70. Springer (2009)
3. Bulazel, A., Yener, B.: A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In: Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium. pp. 2:1–2:21. ACM, New York (2017)
4. Cabaj, K., Mazurczyk, W.: Using software-defined networking for ransomware mitigation: the case of cryptowall. Ieee Network **30**(6), 14–20 (2016)
5. Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., Maggi, F.: Shieldfs: A self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. pp. 336–347. ACSAC '16, ACM, New York, NY, USA (2016)
6. Council of European Union: Council regulation (EU) no 428/2009 (2009), https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex:32009R0428, Last accessed on February 22, 2019
7. CyberEdge: 2018 Cyberthreat Defense Report. Tech. rep., CyberEdge Group, LLC (Mar 2018), https://cyber-edge.com/wp-content/uploads/2018/03/CyberEdge-2018-CDR.pdf
8. European Commission: Guidance note – Research involving dual-use items, http://ec.europa.eu/research/participants/data/ref/h2020/other/hi/guide_research-dual-use_en.pdf, Last accessed on February 22, 2019
9. Feng, Y., Liu, C., Liu, B.: Poster: A new approach to detecting ransomware with deception. In: 38th IEEE Symposium on Security and Privacy Workshops (2017)
10. Genç, Z.A., Lenzini, G., Ryan, P.Y.: No random, no ransom: a key to stop cryptographic ransomware. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 234–255. Springer (2018)
11. Greenberg, A.: The Untold Story of NotPetya, the Most Devastating Cyberattack in History (Aug 2018), https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/, Last accessed on February 22, 2019
12. Gómez-Hernández, J., Álvarez González, L., García-Teodoro, P.: R-locker: Thwarting ransomware action through a honeyfile-based approach. Computers & Security **73**, 389 – 398 (2018)
13. Hunt, G., Brubacher, D.: Detours: Binary interception of win32 functions. In: Proceedings of the 3rd Conference on USENIX Windows NT Symposium - Volume 3. pp. 14–14. WINSYM'99, USENIX Association, Berkeley, CA, USA (1999)
14. Juels, A., Rivest, R.L.: Honeywords: Making password-cracking detectable. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. pp. 145–160. CCS '13, ACM, New York, NY, USA (2013)
15. Kharraz, A., Kirda, E.: Redemption: Real-time protection against ransomware at end-hosts. In: Dacier, M., Bailey, M., Polychronakis, M., Antonakakis, M. (eds.) Research in Attacks, Intrusions, and Defenses. pp. 98–119 (2017)
16. Kolodenker, E., Koch, W., Stringhini, G., Egele, M.: Paybreak: defense against cryptographic ransomware. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 599–611. ACM (2017)
17. Lee, J., Lee, J., Hong, J.: How to Make Efficient Decoy Files for Ransomware Detection? In: Proc. of Int. Conf. on Research in Adaptive and Convergent Systems. pp. 208–212. RACS '17, ACM, New York, NY, USA (2017)

18. Mehnaz, S., Mudgerikar, A., Bertino, E.: Rwguard: A real-time detection system against cryptographic ransomware. In: Research in Attacks, Intrusions, and Defenses. pp. 114–136. Springer International Publishing, Cham (2018)
19. Moore, C.: Detecting ransomware with honeypot techniques. In: 2016 Cybersecurity and Cyberforensics Conference (CCC). pp. 77–81 (Aug 2016)
20. Moussaileb, R., Bouget, B., Palisse, A., Le Bouder, H., Cuppens, N., Lanet, J.L.: Ransomware's early mitigation mechanisms. In: Proceedings of the 13th International Conference on Availability, Reliability and Security. pp. 2:1–2:10. ARES 2018, ACM (2018)
21. Rowe, N.C.: Measuring the effectiveness of honeypot counter-counterdeception. In: Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06). vol. 6, pp. 129c–129c (Jan 2006)
22. Russinovich, M.E., Solomon, D.A., Ionescu, A.: Windows internals. Pearson Education (2012)
23. Scaife, N., Carter, H., Traynor, P., Butler, K.R.B.: Cryptolock (and drop it): Stopping ransomware attacks on user data. In: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS). pp. 303–312 (June 2016)
24. Sgandurra, D., Muñoz-González, L., Mohsen, R., Lupu, E.C.: Automated dynamic analysis of ransomware: Benefits, limitations and use for detection. CoRR **abs/1609.03020** (2016), http://arxiv.org/abs/1609.03020
25. WatchPoint Data: Cryptostopper (2018), https://www.watchpointdata.com/cryptostopper
26. Webroot: 2018 Webroot Threat Report Mid-Year Update. Tech. rep., Webroot Inc. (Sep 2018), https://www.webroot.com/download_file/2780
27. Yuill, J., Zappe, M., Denning, D., Feer, F.: Honeyfiles: Deceptive Files for Intrusion Detection. In: Proc. of the IEEE Work. on Information Assurance, United States Military Academy, West Point, NY (2004)