Computers & Security 97 (2020) 101997

Contents lists available at ScienceDirect

Computers & Security

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

RansomSpector: An introspection-based approach to detect crypto ransomware

Fei Tang[a], Boyang Ma[a], Jinku Li[a,*], Fengwei Zhang[b], Jipeng Su[a], Jianfeng Ma[a]

[a] School of Cyber Engineering, Xidian University, Xi'an, China
[b] Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China

ARTICLE INFO

Article history:
Received 5 March 2020
Revised 7 July 2020
Accepted 3 August 2020
Available online 7 August 2020

Keywords:
Crypto ransomware
Malware detection
Virtual machine introspection
Filesystem activities monitoring
Network activities monitoring

ABSTRACT

Crypto ransomware encrypts user files and then extorts a ransom for decryption, thus it brings a big threat to users. To address this problem, we propose RansomSpector, an introspection-based approach to detect crypto ransomware. Compared to previous solutions, our approach makes progress in two aspects. First, RansomSpector is based on the virtual machine introspection technique, and it resides in the hypervisor layer under the operating system (OS) where ransomware runs. Thus it is capable of analyzing OS-level ransomware and difficult to be bypassed by privilege escalation attacks. Second, RansomSpector monitors both the filesystem and network activities for ransomware detection, thus it achieves a higher precision and earlier warning than the approaches that only leverage the filesystem activities as the detecting basis. To validate our approach, we have implemented a prototype of RansomSpector, and collected 2,117 recent ransomware samples to evaluate it. The evaluation results indicate that our system effectively detects ransomware with a low performance overhead ($< 5\%$ on average).

1. Introduction

A class of malware called ransomware has drawn the worldwide attention recently. The growing number of high-profile variants and increasing economic losses caused by ransomware have made it one of the most severe security threats to users. In May 2017, a variant of ransomware called WannaCry infected more than 230,000 computers in over 150 countries (CNN, 2017), which was the most striking ransomware attack that year. After draining an estimated $5 billion from the global economy last year, ransomware had retained its position at the top of the malware threat lists in 2018, according to Europol's fifth annual Internet Organized Crime Threat Assessment (IOCTA) (for Law Enforcement Cooperation, 2018). It is no doubt that ransomware has become a worldwide concern.

More recently, there are more ransomware attacks against enterprises. For instance, Malwarebytes has witnessed an almost constant increase in business detections of ransomware, rising a shocking 365% from Q2 2018 to Q2 2019 (Malwarebytes, 2019). According to a new survey of senior executives (MarketsInsider, 2020), 46% of small businesses have been the targets of ransomware attack. And of those companies that have been hit with ransomware attack, almost three-quarters (73%) have paid a ransom. 43% of small businesses paid between $10,000 and $50,000 to ransomware attackers, and 13% paid more than $100,000. Of those who paid, however, 17% recovered only some of the company's data. On the other hand, cloud computing is gradually becoming the preferred choice of businesses to streamline different business processes. As per industry reports around 68% of the businesses use cloud technology while 19% are planning to integrate cloud computing into their business operations (Customerthink, 2020). The volume of attacks on cloud services more than doubled in 2019, in line with the trend of organizations increasingly moving operations to the cloud, according to the 2020 Trustwave Global Security Report (Trustwave, 2020). Therefore, as cloud computing becomes widely used in enterprises, ransomware defense in cloud and virtualized environments is of great significance.

Ransomware, whose behaviors are quite different from other malware, mainly operates in two different ways. The first one is screen lockers that maliciously lock the screen of an infected computer, and the other is crypto ransomware that systematically encrypts the victim's files with cryptographic algorithms. Then, the victims will be asked for a ransom to unlock their computers or decrypt their files. The victims can mitigate the threat of screen lockers by reinstalling their operating systems (OSes). However, as for crypto ransomware, it is nearly impossible for the victims to decrypt their encrypted files without a key considering that attackers use strong encryption algorithms and long-enough keys, so this

* Corresponding author.
E-mail address: jkli@xidian.edu.cn (J. Li).

https://doi.org/10.1016/j.cose.2020.101997
0167-4048/

type of ransomware is extremely harmful to users. Therefore, we target crypto ransomware in this paper.

To address the threat of crypto ransomware, researchers have proposed a number of systems and most of them detect attacks by monitoring the low-level filesystem activities (Continella et al., 2016; Kharraz et al., 2016; Kharraz and Kirda, 2017; Scaife et al., 2016). For instance, UNVEIL (Kharraz et al., 2016) presents a dynamic analysis system that is able to detect crypto ransomware by modeling and monitoring its file-access behaviors. In particular, UNVEIL categorizes (crypto) ransomware into three classes of attack based on the file I/O access requests, and then monitors the low-level filesystem activities to identify ransomware attacks. Redemption (Kharraz and Kirda, 2017) monitors the file I/O request patterns for every process to observe possible ransomware activities, and then terminates the offending processes. Note that the file-access behaviours of ransomware are akin to some benign applications, e.g., the encryption and compression applications. Moreover, ransomware can directly invoke the encryption-related functions available in the victim's system to accomplish the corresponding tasks. Therefore, the limitation of such systems is that they possibly generate false positives as they only consider the characteristics of ransomware's filesystem activities as the detecting basis. Further, nearly all the existing ransomware detecting systems run in the same environment (i.e., the same OS) with ransomware applications, and thus they can be bypassed if ransomware process elevates its privilege to kernel level. Indeed, FlashGuard (Huang et al., 2017) and our initial study on 2,117 recent ransomware samples both indicated that many ransomware families and samples attempt to elevate their privileges (see Section 2.1).

In this paper, we present RansomSpector, an introspection-based approach to detect crypto ransomware. Compared to previous systems, RansomSpector makes progress in two aspects. First, RansomSpector is based on the virtual machine introspection (VMI) technique (Garfinkel and Rosenblum, 2003), and it resides in the hypervisor layer under the OS where ransomware runs (i.e., "out of the box"). It does not require to make any modifications to the OS and is transparent to ransomware. Thus, it is difficult, if not impossible, to be bypassed by ransomware which is "in the box". Second, RansomSpector monitors both the low-level filesystem and network activities of ransomware, and models its file I/O access and network activity patterns. After that, RansomSpector detects attacks by enforcing policy to match the modeled patterns; if the condition hits, then ransomware is identified or an alert is raised.

Our system is proposed based on two key observations. First, each file or network operation (e.g., file open/create/read/write/close/rename, or network connect/bind/send/receive/disconnect) corresponds to a specific system call in the OS kernel, which can be captured by the hypervisor and whose context information (e.g., the caller process, the parameters, and the return value of the system call) can be introspected in the hypervisor layer on modern processors (e.g., on x86 or x64). Second, we observe that besides filesystem activities, most crypto ransomware samples connect to the network and produce a large amount of network activities with certain patterns (the evaluation results in Section 4.2 indicate that 93.77% of identified ransomware samples match at least one network activity pattern observed by us). For instance, the destination IP address of ransomware continues to vary in a short period. By monitoring the ransomware process's interactions with the filesystem and network, we achieve a higher precision and earlier warning when detecting ransomware attacks.

To validate our approach, we have developed a prototype with the open-source KVM hypervisor (KVM, 2018), and collected 2,117 recent ransomware samples in Microsoft Windows to evaluate it. The evaluation results show that our system successfully identified 771 crypto ransomware samples from 31 families with zero false positives. Note that even if a sample is labelled as ransomware by anti-virus vendors, it does not mean that the sample is crypto ransomware which will encrypt user files. For example, the sample may be screen locker ransomware, and many other reasons cause the sample to perform no crypto ransomware behaviors (see Section 4.1). On average, only 2.67 user files were lost before the ransomware sample was detected by our system. To the best of our knowledge, RansomSpector is the first to combine the filesystem and network activities for ransomware detection. The performance evaluation results indicate that the overhead introduced by RansomSpector is small ($< 5\%$ on overage).

In summary, this paper makes the following contributions:

- We present an approach to detect crypto ransomware. Our technique is based on VMI and resides in the hypervisor layer under the OS where ransomware runs. Thus, it is transparent to ransomware and difficult to be bypassed. We believe that it is helpful in cloud and virtualized environments because service providers need to protect their customers while they cannot trust virtual machines.
- We implemented a prototype which monitors both the filesystem and network activities of ransomware, and then leverages these monitoring information to match certain file I/O access and network activity patterns to identify ransomware attacks.
- We performed evaluation to show that our approach can effectively detect crypto ransomware attacks with a small performance overhead. We successfully identified 771 ransomware samples from a dataset of 2,117 recent malware with zero false positives.

The rest of the paper is structured as follows. First we describe the threat model and assumptions, and present the overall design as well as the key techniques in Section 2. Then we show the implementation details and evaluation results in Sections 3 and 4, respectively. After that, we discuss possible limitations of our current prototype in Section 5 and describe related work in Section 6. Finally, we conclude the paper in Section 7.

2. System design

2.1. Threat model and assumptions

In this work, our goal is to detect crypto ransomware. Our threat model assumes that ransomware leverages all techniques that other classes of malware use. For instance, ransomware can leverage zero-day attacks to compromise a victim machine and then install malware in it. In addition, we assume that a successful ransomware attack performs one or more of the following activities.

Greedy Encryption and Deletion (or Overwriting) of User Files. To successfully extort a victim, ransomware tends to encrypt user files greedily. The function that an attacker uses to encrypt files may be customizable, or provided by the OS. The potential way to delete victim's files is to rename the file after the original file is covered by ciphertext, or delete the original file after producing the corresponding encrypted file. And some ransomware samples might simply overwrite the original file with ciphertext.

Network Activity. Ransomware might conduct some network activities when encrypting the victim's files. For example, the key used during the encryption and taken to ask for a ransom, might be obtained from a remote server or sent to a server after having been generated on the victim machine. Although ransomware can encrypt user files using symmetric encryption algorithms and hardcode the encryption keys to ransomware itself, which avoids the conduction of network activities, this approach is very rare

now as the encryption keys are easy to be reversed by malware defenders (Sevtsov, 2017). Therefore, there might be both network and filesystem activities during the execution of a ransomware program. Moreover, doxware (Instinct, 2017), a variant of ransomware, steals and transmits user's private data through the network.

Privilege Escalation. A number of ransomware samples try to bypass the detecting system by privilege escalation, although the detecting system usually runs in the kernel level. FlashGuard (Huang et al., 2017) showed that eight ransomware families attempt to delete backup files, which is an indication for privilege escalation. Additionally, in Microsoft Windows, a process can elevate its privilege by calling *OpenProcessToken()* function to get the handler to its access token that contains system-level privileges, then it modifies this access token by invoking *AdjustTokenPrivileges()* function, one of whose parameters is Local Unique Identifier (LUID) that references target privilege. The invocation of these functions is also an indication of attempting for privilege escalation. Therefore, we performed static analysis for ransomware to make an initial study whether an ransomware sample attempts to elevate its privileges in this way. Specifically, we leveraged IDA PRO (IDAPro, 2019) to disassemble each sample and wrote a python script to automatically check if *OpenProcessToken()* and *AdjustTokenPrivileges()* had been called; if so, it was an indication of attempting for privilege escalation. In the result, we found that 572 (27%) of 2,117 ransomware samples attempted to call *OpenProcessToken()* and *AdjustTokenPrivileges()*. Our further study indicated that 61 samples requested specific privileges, e.g., *SeShutdownPrivilege, SeTcbPrivilege, SeDebugPrivilege*, etc. Note that a request of privilege(s) does not indicate it always succeeds. The decision will eventually be made by the operating system. We think an exploit might be required to achieve that, e.g., a system configuration error, or a vulnerability to break the access control when requesting the privileges. However, nearly all the existing solutions ignore this action and are likely to be bypassed by ransomware through privilege escalation attacks.

Finally, we assume that the guest OS is untrusted but the hypervisor is trusted. Namely, the hypervisor is free of malicious code and ransomware running in the guest OS with kernel privilege cannot attack the hypervisor.

2.2. Overall design

Crypto ransomware needs to encrypt user files to ask for a ransom. Thus, ransomware performs multiple file-related operations, e.g., file open, read, write, rename, delete, close, etc. It is a natural way to detect ransomware attacks by monitoring the low-level filesystem activities. However, it possibly generates false positives if we only consider the characteristics of ransomware's file activities as the detecting basis. This is because the file behaviours of ransomware are akin to some benign applications, e.g., the encryption and compression applications. To overcome this limitation, we observe that besides filesystem activities, a number of ransomware samples may also perform network activities. For instance, they may need to transmit the key for file encryption through the network, or steal and send out user's private data for further extortion. Therefore, by monitoring the ransomware program's interactions with both the filesystem and network, we can achieve a higher precision and earlier warning when detecting ransomware attacks.

Additionally, as mentioned earlier, nearly all the existing ransomware solutions run in the same OS with ransomware itself, and thus they can be bypassed if the ransomware program elevates its privilege to kernel level (this possibility has been shown in Section 2.1). To counter this threat, it is better to move the ransomware detecting system outside of the OS. Therefore, we can leverage the VMI technique to monitor the filesystem and network activities of ransomware from the hypervisor layer. This is practical as we observe that each file or network operation corresponds to a specific system call in the guest OS kernel, which can be captured by the hypervisor and whose context information can be restored in the hypervisor layer on modern processors. Compared to existing solutions, our approach does not make modifications to the OS in which ransomware runs, and it is transparent to ransomware. Thus, it is difficult to bypass our system.

Putting everything together, we propose RansomSpector, an introspection-based approach to detect crypto ransomware. The overall design of RansomSpector is shown in Fig. 1.

RansomSpector consists of two modules: *Monitor* and *Detector*. When a process in the guest OS issues a file or network I/O

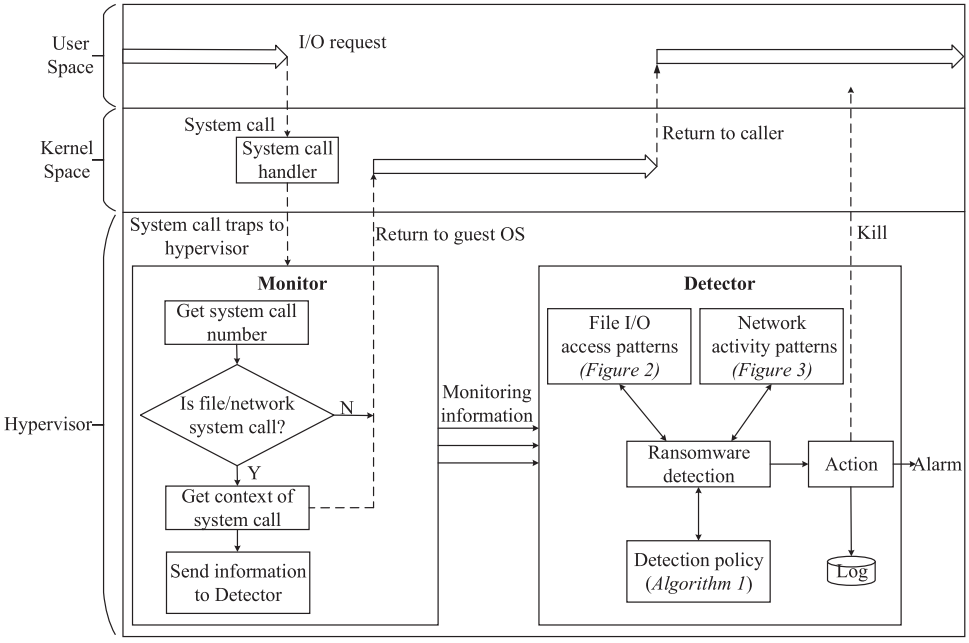

Fig. 1. The overall design of RansomSpector.

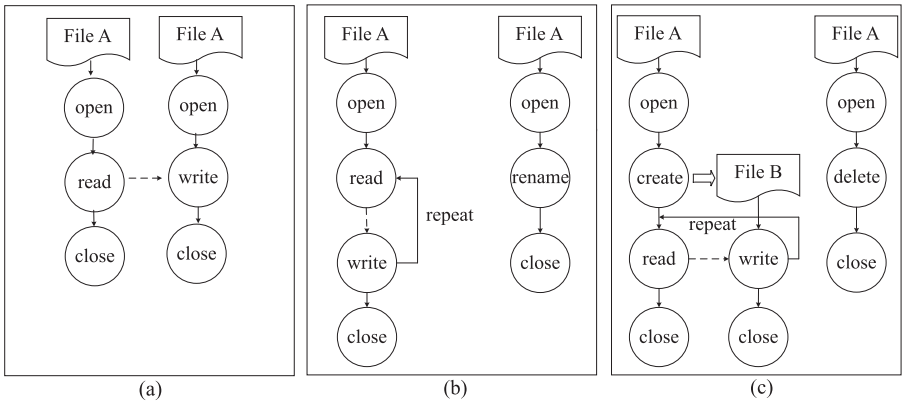

Fig. 2. File I/O access patterns of ransomware.

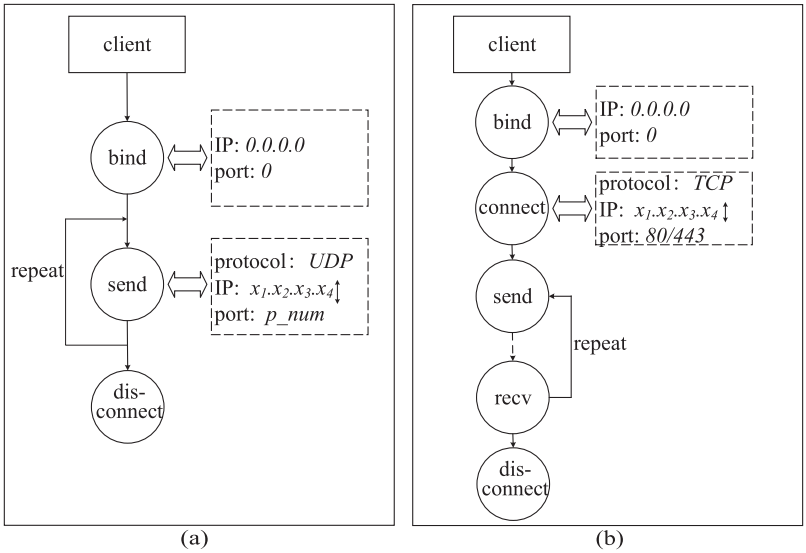

Fig. 3. Network activity patterns of ransomware.

request, the OS switches to kernel mode to execute the corresponding system call handler. At this moment, we make the system call trap to the hypervisor, and then the *Monitor* in the hypervisor captures it. The *Monitor* introspects the guest OS to obtain the context information of the captured system call, e.g., the caller process, the parameters, and the return value of the system call. After that, the execution returns to the guest OS (which will return to the caller in user space after the system call handler is finished). At the same time, the *Monitor* sends the information (i.e., monitoring information) to the *Detector*. When receiving the monitoring information, the *Detector* enforces detecting policy (as shown in Algorithm 1 in Section 2.4) to match certain file I/O access and network activity patterns (as shown in Fig. 2 and Fig. 3 in Section 2.3) to identify ransomware attacks. If the condition hits, one or more actions are performed, e.g., an alarm is raised, the log is recorded, or the ransomware process is killed.

2.3. The feature of ransomware behaviors

Our system detects ransomware based on its file I/O access and network activity patterns. So first we need to get ransomware's filesystem and network activities and model its patterns. To achieve that, we leverage RansomSpector with only its *Monitor* enabled to run 302 recent active ransomware samples collected from UNVEIL (Kharraz et al., 2016) and VirusTotal (VirusTotal, 2017), and record each sample's file and network activity information for modeling. Specifically, 116 samples are from UNVEIL, and the rest 186 ones are collected from VirusTotal. Note that for the samples from UNVEIL, they have been identified as crypto ransomware by researchers. While for the samples from VirusTotal, we only select those samples that are labelled as ransomware by most of anti-virus vendors on the VirusTotal site. Since it is rare for more than half of the anti-virus engines to label a sample as same family (Sebastián et al., 2016), we adopt a plurality vote rather than a majority vote to select samples. And we further manually run each sample to confirm that it is indeed ransomware. The list of ransomware families for modeling is shown in Table 1. We believe that these samples cover most of the main families of the state-of-the-art ransomware. In RansomSpector, the *Monitor* allows us to get complete information of file- and network-related operations for a process in the guest OS, including the operation type, process ID, parent process ID, process name, and timestamp. In addition, for file operations, the monitoring information also includes the name of accessed file, the new file name if renamed and the written data; and for network operations, the monitoring information also includes the type of network operation, IP address, and port number.

When we run a ransomware sample, a process with the same name as the sample is created in the guest OS, and this process may create one or more child processes during a sample runs. In the log, we can find the process ID by the name of the process that is created at the moment we run the sample, and then we are able

Table 1
The list of ransomware families for modeling.

Family	Samples
cerber	115(38.08%)
cryptolocker	12(3.97%)
cryptowall	11(3.64%)
filecoder	26(8.61%)
generic!bt	1(0.33%)
generickd	13(4.30%)
injector	6(1.99%)
kazy	4(1.32%)
kovter	4(1.32%)
kryptik	10(3.31%)
locky	8(2.65%)
razy	63(20.86%)
symmi	1(0.33%)
teslacrypt	25(8.28%)
yakes	1(0.33%)
zusy	2(0.66%)
Total Samples	302

to find all descendants of this process using the parent process ID. For each process, we sort the file I/O access requests based on file names and timestamps, and sort the network activities based only on timestamps. By doing so, we get ransomware's file I/O access and network activity patterns, which reveal how ransomware operates. We leverage a python script to automatically search the logs of these 302 ransomware samples and derive the patterns. After that, we find that a ransomware sample might have the same file I/O access pattern for different types of file, and the samples from different families might have a same file I/O access pattern. The same thing happens to the network activities. Finally, we identify three file I/O access patterns and two network activity patterns.

Fig. 2 shows the identified file I/O access patterns, i.e., file pattern *a, b*, and *c*. Note that in Fig. 2, *File A* stands for a user file that ransomware will encrypt. (1) For file pattern *a*, it first opens *File A*, reads some data (marked as *data R*) from the file, and closes it. Then it opens *File A* again, writes some data (marked as *data W*) to the file, and closes it. We believe that *data W* is the encrypted version of *data R*, which is done by ransomware. Note that for file pattern *a*, ransomware only reads and writes each file once regardless of its size. So for file pattern *a*, ransomware might only encrypt a part of the user file. (2) For file pattern *b*, it first opens *File A*, reads data (marked as *data R*) from and writes data (marked as *data W*) to the file repeatedly, and closes it. Then it opens *File A* again, renames the file, and closes it. Similarly, we believe that *data W* is the encrypted version of *data R*. So for file pattern *b*, ransomware repeatedly overwrites the original file with the encrypted data, and then renames the file. (3) For file pattern *c*, it first opens *File A*, creates a new file called *File B*, then reads data (marked as *data R*) from *File A* and writes data (marked as *data W*) to *File B* repeatedly, and closes the two files. Then it opens *File A* again, deletes the file, and closes it. Similarly, we believe that *data W* is the encrypted version of *data R*. So for file pattern *c*, ransomware creates a new file that is the encrypted version of the original file and then deletes the original file.

Fig. 3 shows the identified network activity patterns, i.e., network pattern *a*, and *b*. (1) For network pattern *a*, it first binds IP address *0.0.0.0* and port number *0*, then repeatedly sends UDP packets to a large number of different hosts (with the same port number, e.g., port 6892) in a short period, and finally it disconnects. Note that after infecting a computer, ransomware might communicate with its C&C (command and control) servers. For example, it might collect various information of the infected computer, such as the OS version, the installed service pack list, the user name, the computer name, and the type of CPU for launching a further attack; or it might steal and transmit user's private data to its servers. Note that ransomware may use domain generation algorithm (DGA) to hide the real address of its C&C server (Wikipedia, 2020). Therefore, the data is sent over UDP (which does not require server's response) to a wide range of IP addresses and thus minimizes the ability for security products to pinpoint the real location of C&C servers (Sevtsov, 2017). (2) For network pattern *b*, it first binds IP address *0.0.0.0* and port number *0*, then connects to multiple remote hosts' 80 or 443 port with TCP protocol in a short period; after that, it repeatedly sends and receives packets with the established connection, finally it disconnects. We think one possible reason for ransomware to connect to multiple remote hosts is to improve the success rate of attack. In our experiment, we observed that a large amount of samples have both network activity patterns.

2.4. Detection approach

Our detection approach is based on the file I/O access and network activity patterns that we observed in Section 2.3. When receiving monitoring information, the *Detector* enforces detection policy to determine if the process that generates the file or network activity is malicious. Algorithm 1 shows the detection policy enforced by the *Detector*.

Algorithm 1 Detection Policy.

```
1:  while True do
2:      receive monitoring information;
3:      if operation type is file-related then
4:          if accessed file is a user file then
5:              add received information into a file operation queue
    based on file name;
6:              if a file I/O access pattern is matched then
7:                  file_pattern_match ← True;
8:      else
9:          add received information into network operation queue;
10:         if a network activity pattern is matched then
11:             network_pattern_match ← True;
12:     if file_pattern_match and network_pattern_match then
13:         ransomware is identified;
14:     else if file_pattern_match then
15:         calculate average entropy of the written data received so
    far;
16:         if average_entropy > α then
17:             alert user;
```

For each received monitoring information (line 2), Algorithm 1 first determines if the operation type of the information is file-related (line 3); if yes, it then determines if the accessed file is a user file (line 4); if yes, it adds the received information into a file operation queue based on file name (line 5). After that, it determines if a file I/O access pattern is matched (line 6); if yes, it sets variable *file_pattern_match* to *True* (line 7). If the operation type of the information is not file-related, which means it is network-related (line 8) as the *Monitor* in Fig. 1 only sends file or network monitoring information to the *Detector*, then it adds the received information into the network operation queue (line 9); further, it determines if a network activity pattern is matched (line 10); if yes, it sets variable *network_pattern_match* to *True* (line 11). Next, the algorithm determines if both *file_pattern_match* and *network_pattern_match* are true (line 12); if yes, ransomware is identified (line 13); otherwise, it determines if *file_pattern_match* is true (line 14); if yes, it calculates the average entropy of the received written data so far (line 15); if the average entropy is bigger than α, which is a threshold set by us (line 16), it alerts the user that a potential ransomware is detected (line 17), which

needs the user to confirm whether an attack happened. This is a necessary step to avoid false positives, as in some special cases, the file behaviors of benign compression or encryption applications match one pattern in Fig. 2 while the data written by them also has a high value of entropy. Fortunately, our evaluation indicates that only a very small ratio (i.e., 2.27%, see Section 4.2) of ransomware need such confirmation by users, as the behaviors of most crypto ransomware match both the file I/O access and network activity patterns.

3. Implementation

To validate our approach, we have developed a proof-of-concept prototype of RansomSpector. Our prototype is implemented on top of the open-source KVM hypervisor (version 3.13.0) (KVM, 2018), which runs on a Ubuntu 16.04/x86-amd64 system, and the ransomware sample runs in the virtual machine (VM) on top of the KVM hypervisor. The operating system of the VM is 64-bit Microsoft Windows 7. To introspect the file and network activities in the VM, we need to extend the KVM hypervisor to capture the system calls invoked by ransomware and get the context information of the system calls. Later, the context information will be sent to the *Detector* of RansomSpector for ransomware detection. To achieve that, our prototype has added about 2,100 lines of *C* code to KVM.

3.1. Monitor

Capturing System Calls. In RansomSpector, the file and network operations are monitored by capturing system calls and getting the context information using VMI. A key technical problem to leverage VMI is the semantic gap (Jain et al., 2014), which means the difference between the high-level OS abstractions from internal guest OS and the hardware-level abstractions from the hypervisor. For instance, in the guest OS we can see semantic-level objects such as processes and files, while we only see memory pages and disk blocks from the hypervisor (which is outside the guest OS). One way to bridge the semantic gap is that the hypervisor incorporates knowledge of the hardware architecture and the guest OS to interpret the low-level state information about the guest OS, which can avoid attacks that result from changing the guest OS architecture (Pfoh et al., 2011). To capture the system calls by the hypervisor, it requires to make the system call trap to the hypervisor. Note that system call does not own such a capability but system interrupt (e.g., page faults, exceptions, etc.) is able to achieve that, with the support of hardware extensions on modern CPUs, e.g., Intel Virtualization Extensions (VT-x) (Corporation, 2018). Therefore, we force the OS to enter system interrupt when a system call occurs. On x64 platform, systems calls are implemented with *SYSCALL* and its counterpart *SYSRET* instructions to construct a mechanism that can fast call system service routines. And this mechanism can be turned off by clearing the *SCE* flag in the Extended Feature Enable Register (*EFER*). It will lead to an invalid opcode exception if the *SCE* flag is cleared, which would trap to the hypervisor when a system call occurs. By doing so, we can capture the system call in the hypervisor, and then determine whether current instruction is *SYSCALL* or *SYSRET*; if so, we get context information of current system call, emulate the instruction and return to the guest OS.

Getting Context Information of System Calls. When we capture a system call in the hypervisor, we first need to determine if it is file- or network-related with the number of the system call. Note that on x64 platform, it is straightforward to get the number of system call as the number will be put into *RAX* register when the system call occurs, while the parameters may be an integer or a pointer to a data structure. Thus the key is to get the addresses of the system call's parameters. Before that, we need to understand the structure of the stack frame when a function (or a system call) is invoked in x64 Windows (Corporation, 2012). For x64 Windows, all the stack operations are performed by *RSP* register, which is the top pointer of the stack. When a system call is invoked, its first four parameters (if needed) are put into *RCX, RDX, R8*, and *R9* registers from left to right, respectively, and the remaining parameters (if needed) are pushed on the stack from left to right. Note that the *call* instruction which invokes the system call pushes an 8-byte return address before the system call is invoked, which subtracts *RSP* by 8. Therefore, when we capture a system call, its first four parameters are stored in *RCX, RDX, R8*, and *R9* registers, respectively. And the address of the fifth parameter is *RSP*+5*8, and so on. In this way, we can get the address of each parameter of the system call. As for the integer return value of the system call, it is stored in *RAX* register in x64 Windows.

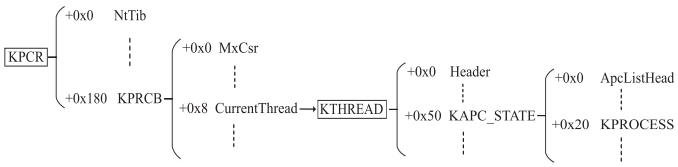

Fig. 4. The path to reach *EPROCESS* from *KPCR* in 64-bit Windows 7.

Further, we obtain information of the caller process that invokes the system call, e.g., the process name, process ID, etc. Note that in Windows, the attributes relating to a process are stored in the executive process (*EPROCESS*) structure, which is a Windows kernel-mode data structure that contains information about current process. In Windows, the address of the *EPROCESS* can be obtained in Kernel Processor Control Region (*KPCR*). In *KPCR*, there is a substructure called Kernel Processor Control Block (*KPRCB*), which contains a pointer to the thread object of current thread that contains the *EPROCESS* of current process. Following this path, we can get the information of caller process that invokes the system call. Fig. 4 shows the path of the data structures to reach *EPROCESS* from *KPCR* in 64-bit Windows 7. Then the key is to get the address of *KPCR*. Note that in x64 Windows, when a *SYSCALL* instruction is executed, in order to make kernel to use the *GS* prefix on normal memory references to access kernel data structures, *SWAPGS* instruction (Corporation, 2018) is used to swap current *GS* base register with the value stored in *MSR* register, which contains the pointer to *KPCR*. Thus we can get the address of *KPCR* in *MSR* register when a system call is captured.

Note that all the above addresses of the context information are in the address space of the VM instead of the hypervisor. To get the values in the hypervisor, we integrate LibVMI (community, 2017) into our prototype to map the addresses of VM into the address space of hypervisor, and then we can read the values in the hypervisor accordingly.

3.2. Detector

When receiving the monitoring information from the *Monitor*, the *Detector* enforces the detection policy (as shown in Algorithm 1) to identify ransomware attacks. At this moment, if both the file and network patterns are matched (for network pattern, we think it is matched if current process has connected more than 3 different IP addresses within three seconds, which is derived by us with the samples in Table 1), we deem the program as ransomware. However, if only the file I/O access pattern is matched, we further calculate the average entropy of written data so far and compare the result to the threshold (i.e., α) set by us to identify attacks. Note that entropy is used to measure data uncertainty, and the encrypted data has a high value of entropy. We

calculate the Shannon entropy (Lin, 1991) of an array of bytes contained in the monitoring information with Eq. 1.

$$H = \sum_{i} -p_i \log_2 p_i \quad (1)$$

In Eq. 1, p_i is the frequency of the value of the i-th byte in the array. The entropy of the data in an encrypted file is commonly higher than that of normal data, as each byte of the encrypted file should have a uniform probability of occurrence. *Detector* calculates the entropy of the data written to user files by current process every time, and then calculates the mean value of these entropy.

To get a reasonable threshold to distinguish ransomware from benign applications, we calculated the average entropy of the data written by each ransomware sample in Table 1. Meanwhile, we ran some benign applications (such as Microsoft Word, Notepad, Adobe Reader, etc.) to perform operations on user documents, and calculated the entropy of the data written by them. As a result, we found that the maximum/minimum values of average entropy for ransomware samples are 7.34/6.08 respectively. While for the benign applications, the average entropy is less than 4. Therefore, we set the threshold of entropy to 6. If the average entropy of the written data by the sample is greater than 6, then we display an alarm message at the top of the screen to alert the user. Compared to existing solutions that need to calculate the entropy of the written data for all the samples (e.g., UNVEIL (Kharraz et al., 2016), Redemption (Kharraz and Kirda, 2017)), our approach makes progress as it performs computation only for those samples with only file I/O access patterns matched, which is a small ratio in our evaluation (only 2.27%, see details in Section 4.2). Further, our system can automatically terminate the malicious process in the guest OS to prevent its malicious behaviors. This can be achieved by clearing the memory page where the target process code is stored with VMI.

4. Evaluation

In this section, we conduct a series of experiments to evaluate that RansomSpector is able to effectively detect ransomware with a low overhead.

4.1. Experimental setup

In our experiments, the OS of the VM where the samples run is 64-bit Microsoft Windows 7, as Windows is the main target for most ransomware attacks (Continella et al., 2016; Kharraz and Kirda, 2017). Since ransomware encrypts user files, we first built a user document directory consisting of a large number of different types of files. These files were from five real-world users' workstations in the lab. They included documents such as *.docx.pptx.xlsx.pdf.txt*, and media files such as *.jpg.png.bmp.gif.mkv.mp4.mp3.rmvb*, as well as archive files such as *.zip.rar*. In addition, there were also some program source files like *.cpp.py.java*. In particular, these files contained some images and regular documents, however, they did not contain user's sensitive data, e.g., personal account information, password, personal photos, etc. After filtering user's sensitive data, to maintain the structure of the directory in real-world users, we copied the directories and files from each source workstation to the experimental environment. To remove the factors that prevent ransomware from running successfully, we shut down the firewall, anti-virus software, and user access control in the VM. Furthermore, the VM had access to the Internet so that the samples could communicate with their C&C servers. We ran each sample with administrator privilege for 30 minutes to ensure that the sample could complete the attack on user files. We reverted the VM to the snapshot before the sample ran in order to ensure that the execution of the sample was not affected by previous samples.

In order to evaluate that RansomSpector can detect ransomware from malware samples in the real world, we obtained 2,117 recent samples from VirusTotal (2017) and VirusShare (2017) using ransomware-related search terms (e.g., ransomware, ransom, etc.) or known variant names, and then put them into the dataset for evaluation. Further, we leveraged AVClass (Sebastián et al., 2016) for family name labelling. The families of samples used in our experiment are shown in Table 2, and these samples cover most families of popular ransomware.

It is worth pointing out that the samples for evaluation in Table 2 do not overlap with those samples used to find the feature of ransomware behaviors in Table 1 in Section 2.3. Note that even if a sample is labelled as ransomware by one or more antivirus vendors, it does not mean that the sample will attack user files. For example, the sample may be a screen locker that only locks the user's system maliciously. And many other reasons also cause the sample to perform no ransomware behaviors. For instance, the sample may need to receive a key from a remote server before encrypting the user files, but the remote server may be shut down or unavailable anymore. Additionally, some advanced samples will stop attacking user files if they detect that they are running in a VM (i.e., evasive malware) (Kirat and Vigna, 2015; Zhang et al., 2015). Moreover, we collected a number of benign applications with ransomware-like behaviors (e.g., file encryption or compression), and analyzed how their network or file behaviors differ from ransomware to reduce false alarms.

4.2. Detection results

In the experiments, we found that RansomSpector successfully identified 771 crypto ransomware from 2,117 malware samples

Table 2
The list of ransomware families for evaluation.

Family	Number	Rate
cerber	312	14.74%
sodinokibi	226	10.68%
gandcrab	195	9.21%
urausy	135	6.38%
lockscreen	127	6.00%
teslacrypt	125	5.90%
yakes	118	5.57%
filecoder	97	4.58%
razy	91	4.30%
cryptowall	90	4.25%
kovter	79	3.73%
reveton	67	3.16%
cryptolocker	56	2.65%
bitman	46	2.17%
crilock	43	2.03%
locky	40	1.89%
symmi	37	1.75%
winlock	35	1.65%
injector	30	1.42%
genasom	18	0.85%
satan	17	0.80%
wannacry	16	0.76%
dmalocker	15	0.71%
cryptxxx	15	0.71%
ctblocker	13	0.61%
sagecrypt	13	0.61%
tobfy	12	0.57%
globeimposter	9	0.43%
tescrypt	8	0.38%
ryuk	5	0.24%
antix	5	0.24%
others (5 families)	22	1.03%
Total (36 families)	2117	-

Table 3
The list of detected ransomware families.

Family	Number	Rate
cerber	235	30.47%
sodinokibi	128	16.60%
filecoder	60	7.77%
razy	53	6.87%
teslacrypt	38	4.93%
gandcrab	34	4.40%
cryptowall	32	4.15%
yakes	29	3.76%
urausy	21	2.72%
crilock	16	2.08%
reveton	14	1.82%
cryptolocker	12	1.56%
kovter	11	1.43%
wannacry	11	1.43%
cryptxxx	10	1.30%
symmi	8	1.04%
ryuk	5	0.65%
expiro	5	0.65%
winlock	5	0.65%
satan	5	0.65%
locky	5	0.65%
tobfy	5	0.65%
sagecrypt	4	0.52%
globeimposter	4	0.52%
tescrypt	4	0.52%
bitman	4	0.52%
ctblocker	3	0.39%
others (4 families)	10	1.30%
Total (31 families)	771	-

Table 4
Statistics of ransomware behavior patterns.

Pattern	File I/O Access			Network Activity			
	a	*b*	*c*	*a*	*b*	*a&b*	*none*
number	24	709	38	41	199	483	48
rate	3.11%	91.96%	4.93%	5.32%	25.81%	62.64%	6.23%

listed in Table 2. Further, the 771 detected samples belonged to 31 different families (as shown in Table 3), among which 10 families overlapped with those of the training set, i.e., cerber, filecoder, razy, teslacrypt, cryptowall, yakes, cryptolocker, kovter, symmi, and locky. Thus we detected 21 new families other than the training set, which indicated that the identified file I/O access and network activity patterns generalized across other families. Then we analyzed the file I/O access and network activity patterns of these 771 ransomware samples. The results are shown in Table 4.

As shown in Table 4, for 91.96% (709/771) of the ransomware samples, their file behaviors matched file I/O access pattern *b* in Fig. 2. When these samples attacked user files, they repeatedly overwrote the original file with encrypted data, and then renamed the original file. Only 8.04% (62/771) of the ransomware samples produced other file I/O access patterns, but RansomSpector successfully identified them, which indicates the effectiveness of our prototype.

As depicted in Table 4, only 6.23% (48/771) of the detected ransomware samples do not match any network activity patterns, which verifies that the network behaviors can be used as an important basis for detecting ransomware. 62.64% (483/771) of the ransomware samples produced both network activity patterns in Fig. 3; 5.32% (41/771) of the ransomware samples only produced network activity pattern *a*, and 25.81% (199/771) of the ransomware samples only produced network activity pattern *b*, which indicates that these two network activity patterns are ubiquitous in our tested samples.

Table 5
The confusion matrix of experimental results.

		Actual Results	
		Crypto ransomware	Non-crypto ransomware
Detected Results	Crypto ransomware	771 (TP)	0 (FP)
	Non-crypto ransomware	0 (FN)	1346 (TN)

Combining with file I/O access and network activity patterns, RansomSpector successfully identified 723 ransomware samples automatically. For the samples with only file I/O access patterns matched, we then calculate the average entropy of the data that the sample writes to user files. If the average entropy is larger than the threshold set by us, in order to avoid false positives, we alert the user that a potential ransomware is found, which needs the user to confirm whether the sample is malicious. Note that the ratio needed to be confirmed by users is very small, which is only 2.27% (48/2,117) in our experiments.

Next, we will discuss the detection accuracy, early warning, and behaviors of the ransomware-like benign applications. The experiments are conducted by setting the entropy threshold α to 6. We calculate the false positive and false negative rates based on the overall number of ransomware samples shown in Table 2.

False Positives. To evaluate the false positive rate of RansomSpector, we manually ran each sample that was detected as ransomware by RansomSpector in a clean VM, and checked whether the user files were encrypted. As a result, we found that at least one user file was encrypted and generated high entropy data in the process of encryption. So it has no false positives in our results.

False Negatives. Due to the large number of samples detected as non-crypto ransomware, to evaluate the false negative rate of RansomSpector, we wrote a python script to search the logs for file-related operations of the rest 1,346 samples, as it is necessary for an crypto ransomware sample to operate on files (e.g., file open, read, write, etc.) for file encryption. As a result, we found no file-related operations to the user files provided by us in Section 4.1, so the *approximate* false negative rate of RansomSpector is zero. The term "*approximate*" indicates that there may be ransomware samples in our dataset labelled as non-crypto ransomware by RansomSpector, but they did not perform any ransomware behaviors in this experiment for some reasons. For example, some samples may need to run for more than 30 minutes to start attacking user files, which results in the fact that they were not detected by our system. However, RansomSpector is a system that can dynamically detect ransomware, and is able to identify it as long as the sample performs crypto ransomware behaviors.

Precision and Recall Rates. We leveraged a confusion matrix to further present the experimental results in terms of precision and recall, as shown in Table 5. Of the 771 detected crypto ransomware samples, as we manually confirmed that all of them showed crypto ransomware behaviors, so the value of true positive (TP) was 771 and the value of false positive (FP) was zero. And of the remaining 1,346 samples, we found no file-related operations to the user files, which indicated that the value of true negative (TN) was 1,346 and the value of false negative (FN) was zero. We further calculated the precision and recall rates with Eq. 2 and 3. In the result, the precision and recall rates were both 100%.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Early Warning. We have shown that RansomSpector accurately detects ransomware samples. Furthermore, if RansomSpector can

detect ransomware and prevent it from attacking user files as early as possible, it will reduce the loss of user files. In our evaluation, 86.51% (667/771) of ransomware samples were detected when they were encrypting the second user file, and most of these ransomware samples were detected by combining file I/O access and network activity patterns, which indicates the effectiveness of our approach that combines the filesystem and network activities for ransomware detection. On average, 2.67 user files were lost before the ransomware sample was detected by our system. Compared to CryptoDrop (Scaife et al., 2016) with a median loss of 10 files, our system makes a further step.

Ransomware-like Benign Applications. To further demonstrate the impact of RansomSpector on ransomware-like benign applications, we collected 100 popular applications from Softonic (Softonic, 2019) for evaluation. In particular, we selected five related categories on the website, i.e., compression, encryption, productivity, network, and data deletion, and downloaded the top 20 applications of each. We first obtained the File I/O access and network activity patterns of each benign application using RansomSpector (with only its *Monitor* enabled). Then, we compared the behavioral features of ransomware and these benign applications. As shown in Table 4, most crypto ransomware samples detected by us perform both file and network activities, while we found that no network activity patterns were matched for the benign applications at runtime. So the key is to compare the file I/O access patterns to distinguish them.

For the compression applications such as WinRAR (2017) and 7zip, they first open and read the files that the user wants to compress, then create a temporary file and continuously write the compressed data to it, finally rename it with an extension name *.rar* or *.zip*. Note that usually the original files are not overwritten, deleted, or renamed, which is different from ransomware. However, if the user chooses the option to delete the original files after the compression is finished, the file I/O access pattern *c* (in Fig. 2) is matched, and then the *Detector* of RansomSpector calculates the average entropy of the written data to judge if the entropy is bigger than 6. Note that the compression applications such as WinRAR and 7-Zip produce high entropy output, which is bigger than 6, so the *Detector* alerts the user that a potential attack is found and requests the user to make a decision. This is a necessary step to avoid false positives, and such cases that need to be confirmed by users is very rare, which is only 2.27% (48/2,117) in our experiments. The same thing happens to the encryption applications such as FlashCrypt (Labs, 2018).

4.3. Case study

The Role of Network Activity in Ransomware Execution. We used Reflector (Redgate, 2017) to decompime ransomware samples, and generated the source code of these samples. With that, we could have a deeper understanding on how ransomware attacks user files and what data is transmitted by ransomware over the network.

Next, we conduct a case study with a specific ransomware sample to illustrate that. The MD5 value of the sample is 0*xccae*2400*a*50*a*3*f*2435*d*7*ef*1*b*11*e*7497*c*, and it belongs to *CryptoLocker* family.

Before attacking the user files, the sample sends the user's host hardware information (including its CPU ID, BIOS ID, disk ID), host name, user name, etc. to a remote server through the HTTP protocol, and the sample receives a response with a key for encryption from the server. We believe that the attacker can leverage the user's information to generate the key, or record user's information to decrypt the victim's files later. Then the ransomware sample checks if a file named *filelist.txt* (which contains the file names that the attacker wants to encrypt) exists. If *filelist.txt* does not exist, the sample creates it, and then it searches for files with specific extensions and add the paths of these files to *filelist.txt*. Next, if the sample can not successfully receive the key from the server, it would directly generate a key on the victim's machine. Finally, the sample encrypts all files recorded in *filelist.txt*, clears the key, and deletes all the original files.

Therefore, it shows that while attacking user files, ransomware usually transmits user information and keys used for encryption over the network. Additionally, we found that some other ransomware samples generate encryption keys locally and then send the keys to their C&C servers.

Doxware Detection. In our experiments, RansomSpector successfully detected doxware (Instinct, 2017), which not only encrypts user files, but also steals user's sensitive information and releases the information unless the victim pays for a ransom. By analyzing the behaviors of doxware, we found that doxware performs both file and network activities. The file activities of doxware match the file I/O access pattern *c* in Fig. 2. Namely, doxware creates a new file, repeatedly reads data from the user file and writes encrypted data to the new file, finally deletes the original user file. In addition, doxware performs a large number of network activities that match network activity pattern *b* in Fig. 3. We believe that doxware sends sensitive data collected from victims to attackers over the network. Based on the characteristics of ransomware's file and network behaviors, RansomSpector successfully detected it from our dataset.

4.4. Performance overhead

To evaluate the performance overhead introduced by our approach, first we used IOzone (IOzone, 2018) to evaluate the file I/O performance of RansomSpector, including write, rewrite, read, and reread operations. Then we used NetPerf (Enterprise, 2018) to evaluate the network performance of RansomSpector, including *tcp_s* (TCP bulk transfer), *udp_s* (UDP bulk transfer), *tcp_rr* (TCP request and response), and *udp_rr* (UDP request and response). We ran each test 10 times in a standard 64-bit Windows 7 system on top of an original KVM (Original) and RansomSpector (*RS*) respectively, and then calculated the average. As shown in Table 6 and Table 7, the results indicate that the average file I/O performance overhead introduced by RansomSpector is 4.96%, and the average network performance overhead introduced by RansomSpector is 2.49%.

Finally, we evaluated the impact of RansomSpector on some popular user applications, including AESCrypt, Chrome, IE, MS Word, NotePad, WinRAR, and Media Player. We wrote scripts with AutoIt (Team, 2018) to make each application execute the same tasks with and without RansomSpector deployed, and obtained

Table 6
File I/O performance overhead.

Mode	Original	RS	Overhead
write	132.7MB/s	129.8MB/s	2.19%
rewrite	145.3MB/s	138.9MB/s	4.40%
read	701.1MB/s	654.9MB/s	6.59%
reread	726.9MB/s	678.4MB/s	6.67%
average	-	-	4.96%

Table 7
Network performance overhead.

Mode	Original	RS	Overhead
tcp_s	123.0MB/s	120.5MB/s	2.03%
udp_s	125.3MB/s	122.9MB/s	1.92%
tcp_rr	1,872.2/s	1,811.7/s	3.23%
udp_rr	9,748.7/s	9,477.5/s	2.78%
average	-	-	2.49%

Table 8
Runtime overhead on user applications.

Application	Original	RS	Overhead
AESCrypt	131.73s	137.25s	4.19%
Chrome	106.87s	109.31s	2.28%
IE	95.88s	98.36s	2.59%
MS Word	142.06s	146.92s	3.42%
NotePad	123.84s	128.33s	3.63%
WinRAR	83.61s	85.46s	2.21%
Media Player	123.18s	123.56s	0.31%
Average	-	-	2.66%

their running times. Similarly, we ran each application 10 times and calculated the average. As shown in Table 8, the average overhead on user applications introduced by RansomSpector is only 2.66%.

Note that our performance evaluation assumed that RansomSpector ran in a KVM hypervisor, and it did not count in the overhead introduced by KVM. This is reasonable as RansomSpector is an introspection-based approach, and it can be readily deployed in cloud and virtualized environments, which our approach mainly targets. In summary, the evaluation results indicate that RansomSpector introduces only a low performance overhead ($<$ 5% on average).

4.5. Comparison with research systems

We compared RansomSpector with four state-of-the-art research systems, i.e., ShieldFS (Continella et al., 2016), Redemption (Kharraz and Kirda, 2017), PayBreak (Kolodenker et al., 2017), and CryptoDrop (Scaife et al., 2016). The comparison results were shown in Table 9. Note that as the state-of-the-art research systems were not public, we could not evaluate them in a real experimental environment. Therefore, we collected public evaluation data of the research systems from their publications and then made comparison.

Dataset Size. As shown in the 2*nd* line of Table 9, RansomSpector held the maximum number of evaluation ransomware samples in the five systems, i.e., 2,117, while PayBreak held the minimum number of samples, which was only 107. Note that although Redemption collected 1,174 active ransomware samples in total, only 677 ones were used for evaluation and others were for training.

Detection Capabilities. The $3rd-6th$ lines of Table 9 showed the detection capabilities of the five systems, including the number of detected ransomware samples (*Detected ransomware*), the false positive rate (*False Positives*), the false negative rate (*False Negatives*), and the number of the lost files before ransomware was determined (*Lost files*). Specifically, RansomSpector detected the maximum number of ransomware samples in the five systems, i.e., 771. In terms of false positive and false negative rates, the values of RansomSpector were both zero. However, ShiledFS, Redemption, and CrytoDrop introduced a non-zero false positive or false negative rate. In addition, RansomSpector lost only 2.67 files on average before ransomware was detected, which was less than CryptoDrop (10 lost files). It is worth pointing out that as the detection results for comparison in Table 9 were obtained from different datasets, thus the comparison of these results is not rigorous and only serves as an indicator to show the detection capabilities of these research systems.

Other Features. We further compared RansomSpector with the research systems in other aspects. As shown in the 7*th* line of Table 9, compared to other four systems, RansomSpector did not require to modify the operating system, which was transparent to users. Further, since RansomSpector resided in the hypervisor layer while other four systems ran inside the OS, thus it had kernel-level detection capability and was difficult to be bypassed by ransomware (as shown in the $8th-9th$ lines of Table 9). The 10*th* line of Table 9 indicated that RansomSpector introduced a new indication (i.e., network activity) as the detecting basis, and thus it achieved a higher precision and earlier warning than other systems. As shown in the 11*th* line of Table 9, compared to ShieldFS and Redemption, RansomSpector and CryptoDrop had no file recovery capability, while PayBreak could recovery partial files.

Performance Overhead. As shown in the 12*th* line of Table 9, besides CryptoDrop which provided the performance overhead with latencies, the overhead introduced by RansomSpector was less than that of other systems, which was only 4.96% for I/O operations.

Overall, the comparison results indicate that RansomSpector has advantages in some aspects. First, RansomSpector resides in the hypervisor layer and does not require modification of the operating system, thus it has kernel-level detection capability and is difficult to be bypassed by ransomware. Second, RansomSpector introduces a new indication (i.e., network activity) as the detecting basis, thus it achieves a higher precision and earlier warning than other systems. Third, the performance overhead introduced by RansomSpector is small, which is less than 5% on average. Meanwhile, RansomSpector also has disadvantages compared to other systems. For instance, RansomSpector does not hold file recovery capability. We believe that compared to existing research systems, RansomSpector is a different approach that targets different environments, i.e, cloud and virtualized environments, and it could complement existing systems.

4.6. Comparison with commercial tools

To further demonstrate the effectiveness of RansomSpector, we compared it with four popular commercial ransomware detection tools, i.e., RansomDefender (Clonix, 2019), RansomStopper (Cybersight, 2019), Malwarebytes Anti-Ransomware

Table 9
Comparison results of RansomSpector with research systems.

-	ShieldFS (Continella et al., 2016)	Redemption (Kharraz and Kirda, 2017)	PayBreak (Kolodenker et al., 2017)	CryptoDrop (Scaife et al., 2016)	RansomSpector
Dataset size	305	677	107	492	2117
Detected ransomware	298	677	85	492	771
False Positives	0.0%	0.8%	N/A	3%	0.0%
False Negatives	2%	0.0%	N/A	0.0%	0.0%
Lost files	0	0	N/A	10	2.67
OS modification	Yes	Yes	Yes	Yes	No
Kernel-level detection	No	No	No	No	Yes
Bypass possibility	Easy	Easy	Easy	Easy	Difficult
Detection indicators	File activities	File activities	File activities	File activities	File&network activities
Recovery capability	Yes	Yes	Partial	No	No
Performance overhead	30%-380% (I/O)	5.6% (I/O)	150% (I/O)	$<$ 16ms (latencies)	4.96% (I/O)

Table 10
Comparison results of RansomSpector with popular ransomware detection tools.

-	RansomDefender (Clonix, 2019)	RansomStopper (Cybersight, 2019)	Malwarebytes (Malwarebytes, 2019)	RansomBuster (Trendmicro, 2019)	RansomSpector
Detected ransomware	614	132	758	473	771
False Positives	0%	0%	0%	0%	0%
False Negatives	20.36%(157/771)	82.88%(639/771)	1.69%(13/771)	38.65%(298/771)	0%(0/771)
Lost files	12.20	39.00	27.91	15.61	2.67
Kernel-level detection	No	No	No	No	Yes
Bypass possibility	Easy	Easy	Easy	Easy	Difficult
Detection indicators	File activities	File activities	File activities	File activities	File&network activities

(Malwarebytes, 2019), and RansomBuster (Trendmicro, 2019). The comparison results were shown in Table 10.

First, we tested each of the commercial ransomware detection tools with the same ransomware samples as RansomSpector listed in Table 2. To do that, we leveraged a similar experimental environment of RansomSpector in Section 4.1 for the commercial tools. Note that the difference was that the commercial tools were installed inside the OS image of the VM, while RansomSpector ran outside of the VM, i.e., in the hypervisor layer. For each commercial ransomware detection tool, we tested the 2,117 samples listed in Table 2 one by one, and each sample ran for 30 minutes. We reverted the VM to the snapshot before the sample ran in order to ensure that the execution of the sample was not affected by previous samples. We did the test for each commercial tool with 8 VMs simultaneously, and each VM ran a subset of the ransomware samples. In total, it took almost 26 days to complete the whole test for the four commercial tools.

After the test, we calculated the number of the detected ransomware samples (*Detected ransomware*), the false positive rate (*False positives*), the false negative rate (*False negatives*), and the number of the lost files before ransomware was determined (*Lost files*) for each commercial ransomware detection tool, as shown in the $2nd - 5th$ lines of Table 10.

Number of Detected Ransomware Samples. As shown in the $2nd$ line of Table 10, our system detected the maximum number of ransomware samples, i.e., 771, while RansomStopper detected the minimum number of ransomware samples, i.e., 132. Further, we found that all ransomware detected by four commercial tools were within the set detected by RansomSpector. In other words, these four commercial tools only detected a subset of ransomware that were detected by RansomSpector.

False Positives and False Negatives. Note that we manually confirmed that the 771 samples detected as ransomware by RansomSpector were active crypto ransomware in Section 4.2, and all the samples detected as ransomware by four commercial ransomware detection tools are within the 771 sample dataset, so the false positive rates for the commercial tools are all zero, as shown in the $3rd$ line of Table 10. While for the false negative rates, it is different for our system and the commercial tools. Similar to RansomSpector, we computed the *approximate* false negative rate of each commercial tool. As a result, the minimum false negative rate is zero from RansomSpector, and the maximum false negative rate is 82.88% from RansomStopper, as shown in the $4th$ line of Table 10.

Number of Lost Files. As mentioned before, if a ransomware detection tool can detect ransomware and prevent it from attacking user files as early as possible, it will reduce the loss of user files. So the number of lost files is an important vector for the ransomware detection systems. In order to get the number of lost files for each commercial tool, we traversed the log files and counted how many user files had been altered by the ransomware samples one by one. After that, we calculated the average number of lost files when the detection tool successfully blocked ransomware. As shown in the $5th$ line of Table 10, RansomSpector has the minimum number of lost files before ransomware was determined, i.e., 2.67 on average, while RansomStopper has the maximum number of lost files, i.e., 39.00 on average.

Additionally, we compared RansomSpector with the commercial ransomware detection tools from three other aspects, i.e., the ability of kernel-level ransomware detection (*Kernel-level detection*), the possibility to be bypassed by attacks (*Bypass possibility*), and the indicators for ransomware detection (*Detection indicators*). The results are shown in the $6th - 8th$ lines of Table 10.

As the commercial ransomware detection tools run in the same environment (i.e., the same OS) with the ransomware applications, thus they are not able to detect kernel-level ransomware and easy to be bypassed by privilege escalation attacks. In comparison, RansomSpector runs outside of the OS, i.e., in the hypervisor layer, thus it is capable of analyzing and detecting kernel-level ransomware and difficult to be bypassed by privilege escalation attacks. Furthermore, RansomSpector leverages both the filesystem and network activities as ransomware detection indicators, thus it achieves a higher precision and earlier warning than the commercial tools that only leverage the filesystem activities as ransomware detection indicator.

In summary, compared to four popular commercial ransomware detection tools, RansomSpector makes a further step in almost all the items listed in Table 10.

5. Discussion

In this section, we discuss possible limitations of RansomSpector and suggest potential improvements.

First, it is worth pointing out that although the guest OS in our prototype of RansomSpector is 64-bit Windows 7 on x64 platform, we believe that the techniques presented in this paper are generic and can be applied to other commodity guest OSes as well. In fact, we have successfully introspected the file- and network-related system calls for Ubuntu 16.04/x86-amd64 guest OS with KVM hypervisor (KVM, 2018). It is promising to extend our system to other guest OSes with moderate engineering efforts, e.g., FreeBSD, Solaris, etc.

Second, RansomSpector is based on the VMI technique, and our current prototype requires to manually reconstruct the context information of file- and network-related system calls once captured by the hypervisor. In other words, it requires to develop a certain piece of reconstruction code for each type of guest OS, as the data structures to construct is not same in different OSes. However, all the construction code is developed in the hypervisor layer instead of the guest OS, and there is no need to modify (and recompile) the guest OS. Therefore, in a cloud or virtualized environment that our approach mainly targets, it is the responsibility of service providers to extend the hypervisor for detecting ransomware, which is transparent to average users. Moreover, one key idea of our approach is to leverage the VMI technique for detecting ransomware in the hypervisor layer, and we are not targeting to advance the VMI technique itself. However, any progress in the field of VMI could be borrowed by our system, as the ransomware

detection policy (i.e., Algorithm 1 in Section 2.4) enforced in our system is orthogonal to the VMI technique. Indeed, some promising solutions (Dolan-Gavitt et al., 2011; Fu and Lin, 2013) in the area of VMI have been proposed to automatically obtain the reconstruction code. Thus, our work can readily benefit from the advance in this direction.

Third, although the main task of RansomSpector is to detect ransomware attacks, and our evaluation results indicate that only 2.67 user files were lost before ransomware was detected by our system on average (see details in Section 4.2), it is better if RansomSpector could successfully recover the original files once encrypted. Note that it is almost impossible for victims to decrypt the encrypted files without a key, due to the high-intensity encryption algorithms and long-enough keys used by attackers. To address this challenge, our initial thought is to intercept the data in file read and write operations of ransomware with VMI, and record the data as well as its offset in the file. Once the user file is encrypted, we can leverage these data for file recovery. We leave it as our future work.

Fourth, although we believe that the samples leveraged to summarize the network and file patterns of ransomware (in Section 2.3) cover most families of state-of-the-art ransomware, and our evaluation results further indicate its effectiveness, it is possible to adopt a new network or file pattern for a new variant of crypto ransomware. Once such case is found, it is easy for us to add the new pattern into our system.

6. Related work

Ransomware Detection and Prevention. A number of systems have been proposed by researchers to detect or prevent ransomware for Windows (Continella et al., 2016; Huang et al., 2018; 2017; Kharraz et al., 2016; Kharraz and Kirda, 2017; Kharraz et al., 2015; Kolodenker et al., 2017; Mehnaz et al., 2018; Scaife et al., 2016) and Android (Andronio et al., 2015; Chen et al., 2018). We focus our discussion on the systems for Windows. For example, Kharraz et al. made a long-term study on the behaviors and evolution of ransomware and proposed an approach to detect ransomware by monitoring its filesystem activities (Kharraz et al., 2015), but they did not evaluate it. UNVEIL (Kharraz et al., 2016) can analyze and detect both the screen lockers and crypto ransomware. In particular, for crypto ransomware, UNVEIL monitors the low-level filesystem activities to match certain file I/O access patterns for ransomware identification. PAYBREAK (Kolodenker et al., 2017) restores encrypted files by hooking calls to cryptographic functions and gets the symmetric keys used to encrypt files. CryptoDrop (Scaife et al., 2016) detects ransomware by monitoring certain changes of user data, including the file type, file similarity, Shannon entropy, etc. ShieldFS (Continella et al., 2016) leverages a huge number of I/O requests generated by benign applications to train some models, and then uses the models to determine whether a process is malicious. Redemption (Kharraz and Kirda, 2017) redirects each write access request of user files to a protected area and evaluates the malicious degree of a process based on the changes of the target files and the behavioral characteristics of the process. RWGuard (Mehnaz et al., 2018) employs three techniques (i.e., decoy monitoring, process monitoring, and file change monitoring) to monitor file-related operations or changes for ransomware detection. As mentioned in Section 1, the above systems have two limitations, i.e., they work in the same OS with ransomware and they detect attacks only based on the filesystem activities. In contrast, RansomSpector moves the detection system out of the guest OS, and monitors both the file and network activities using VMI for attack identification. Thus, our system is immune to privilege escalation attacks and achieves a higher precision and earlier warning when detecting ransomware attacks.

Additionally, some approaches (Baek et al., 2018; Huang et al., 2017; Park et al., 2019) were proposed to provide firmware-level ransomware detection and file recovery by taking advantage of the inherent features of SSD, so they cannot be bypassed by ransomware with kernel privilege. But they only target for SSD-based file systems. In contrast, our system targets for the filesystem on common hard disks. Huang et al. established a measurement framework to perform a large-scale end-to-end analysis of ransomware revenue, affiliate schemes, and infrastructure (Huang et al., 2018), which has a different goal compared to our system.

Virtual Machine Introspection. As a new technique for intrusion detection, VMI combines high visibility of host-based intrusion detection system and strong isolation of network-based intrusion detection system. A variety of VMI systems have been proposed since its introduction in 2003 (Garfinkel and Rosenblum, 2003). For instance, Strider GhostBuster (Wang et al., 2005) and VMwatcher (Jiang et al., 2007) detect attacks by comparing the high-level view from the internal VM and the low-level view from the hypervisor with VMI. XenAccess (Payne et al., 2007) presents a monitoring library for VMs running on Xen (Barham et al., 2003). Process out-grafting (Srinivasan et al., 2011) reconstructs semantics for the key processes in the VM, and then monitors their behaviors to detect attacks. Compared to the systems that need to manually reconstruct semantics, recent systems mainly focus on how to automatically obtain the reconstruction code (Dolan-Gavitt et al., 2011; Fu and Lin, 2013; Saberi et al., 2014), e.g., Virtuoso (Dolan-Gavitt et al., 2011) and VMST (Fu and Lin, 2013). We believe that the research progress in the field of VMI can be borrowed by our system in the future.

7. Conclusion

In this paper, we propose RansomSpector, an introspection-based approach to analyze and detect crypto ransomware. In particular, RansomSpector resides in the hypervisor layer, and it introspects both the file and network activities of ransomware which runs in the VM to identify attacks. Thus, it is transparent to ransomware and difficult to be bypassed. Additionally, it achieves a higher precision and earlier warning compared to those systems that only use file behaviors as the detecting basis. We have developed a prototype of RansomSpector with KVM, and collected 2,117 ransomware samples to evaluate it. The evaluation results indicate that RansomSpector can effectively detect ransomware attacks with a small performance overhead.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Fei Tang: Methodology, Software, Validation. **Boyang Ma:** Formal analysis, Data curation, Writing - original draft. **Jinku Li:** Conceptualization, Project administration, Writing - review & editing. **Fengwei Zhang:** Conceptualization, Writing - review & editing. **Jipeng Su:** Investigation, Software. **Jianfeng Ma:** Supervision.

Acknowledgment

We would like to thank the authors of UNVEIL (Kharraz et al., 2016), as this work would not have been possible without the ransomware samples they provided. This work was supported in part by the Key R&D Program of Shaanxi Province of China under

Grant 2019ZDLGY12-06, and in part by Xi'an Science and Technology Planning Project of China under Grant 201809168CX9JC10.

References

Andronio, N., Zanero, S., Maggi, F., 2015. Heldroid: Dissecting and detecting mobile ransomware. In: Proceedings of the 2015 International Workshop on Recent Advances in Intrusion Detection, pp. 382–404.

Baek, S., Jung, Y., Mohaisen, A., Lee, S., Nyang, D., 2018. Ssd-insider: internal defense of solid-state drive against ransomware with perfect data recovery. In: 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), pp. 875–884.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., 2003. Xen and the art of virtualization. Proceedings of the ACM Symposium on Operating Systems Principles 164–177.

Chen, J., Wang, C., Zhao, Z., Chen, K., Du, R., Ahn, G.-J., 2018. Uncovering the face of android ransomware: characterization and real-time detection. IEEE Trans. Inf. Forensics Secur. 13 (5), 1286–1300.

Clonix, The preventive defense against ransomware with the multi-layered defense system (2019). http://www.ransomdefender.com/en/product.

CNN, World's biggest cyberattack sends countries into 'disaster recovery mode' (2017). http://money.cnn.com/2017/05/14/technology/ransomware-attack-threat-escalating/index.html.

Labs, F., Flashcrypt, a useful program that was created in order to help users protect their sensitive data stored in personal folders(2018). http://www.softpedia.com/get/Security/Encrypting/FlashCrypt.shtml.

Community, L., Libvmi-virtual machine introspection (2017). http://libvmi.com/.

Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S., Maggi, F., 2016. Shieldfs: a self-healing, ransomware-aware filesystem. In: Proceedings of the 32nd Annual Conference on Computer Security Applications, pp. 336–347.

Corporation, I., Intel® 64 and ia-32 architectures software developer's manual (2018). https://software.intel.com/sites/default/files/managed/a4/60/325384-sdm-vol-3abcd.pdf.

Corporation, I., Intel® virtualization technology (intel® vt) (2018). https://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html.

Corporation, I., c. Introduction to x64 assembly (2012). http://software.intel.com/en-us/articles/introduction-to-x64-assembly.

Customerthink, Cloud computing security risks and how to protect cloud customers from ransomware (2020). https://customerthink.com/cloud-computing-security-risks-and-how-to-protect-cloud-customers-from-ransomware/.

Cybersight, Ransomstopper protects from new and existing ransomware (2019). https://cybersight.com/products/free-home-personal/.

Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., Lee, W., 2011. Virtuoso: Narrowing the semantic gap in virtual machine introspection. In: Proceedings of the 2011 IEEE Symposium on Security and Privacy, pp. 297–312.

Enterprise, H.P., Netperf - a benchmark that can be used to measure the performance of many different types of networking (2018). https://hewlettpackard.github.io/netperf/.

Fu, Y., Lin, Z., 2013. Space traveling across vm:automatically bridging the semantic gap in virtual machine introspection via online kernel data redirection. ACM Trans. Inf. Syst. Secur. 16 (2), 586–600.

Garfinkel, T., Rosenblum, M., 2003. A virtual machine introspection based architecture for intrusion detection. In: Proceedings of the 10th Network and Distributed System Security Symposium, pp. 191–206.

Huang, D.Y., McCoy, D., Aliapoulios, M.M., Li, V.G., Invernizzi, L., Bursztein, E., McRoberts, K., Levin, J., Levchenko, K., Snoeren, A.C., 2018. Tracking ransomware end-to-end. In: Proceedings of the 2018 IEEE Symposium on Security and Privacy.

Huang, J., Xu, J., Xing, X., Liu, P., Qureshi, M.K., 2017. Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2231–2244.

IDAPro, Ida pro, an interactive, programmable, extensible, multi-processor disassembler (2019). https://www.hex-rays.com/products/ida/.

Instinct, D., The evolution of ransomware: From 'locker' to 'doxware' (2017). https://www.deepinstinct.com/blog/2017/12/08/the-evolution-of-ransomware-from-locker-to-doxware/.

IOzone, Iozone is a filesystem benchmark tool that generates and measures a variety of file operations (2018). http://www.iozone.org/.

Jain, B., Baig, M.B., Zhang, D., Porter, D.E., Sion, R., 2014. Sok: Introspections on trust and the semantic gap. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy, pp. 605–620.

Jiang, X., Wang, X., Xu, D., 2007. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 128–138.

Kharraz, A., Arshad, S., Mulliner, C., Robertson, W.K., Kirda, E., 2016. Unveil: A large-scale, automated approach to detecting ransomware. In: Proceedings of the 25th USENIX Security Symposium, pp. 757–772.

Kharraz, A., Kirda, E., 2017. Redemption: Real-time protection against ransomware at end-hosts. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 98–119.

Kharraz, A., Robertson, W., Balzarotti, D., Bilge, L., Kirda, E., 2015. Cutting the gordian knot: A look under the hood of ransomware attacks. In: Proceedings of the 2015 International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 3–24.

Kirat, D., Vigna, G., 2015. MalGene: Automatic extraction of malware analysis evasion signature. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS'15).

Kolodenker, E., Koch, W., Stringhini, G., Egele, M., 2017. Paybreak: defense against cryptographic ransomware. In: Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security, pp. 599–611.

KVM, Kernel-based virtual machine (2018). https://www.linux-kvm.org/page/Main_Page.

E.U.A. for Law Enforcement Cooperation, Internet organised crime threat assessment (iocta) 2018. https://www.europol.europa.eu/sites/default/files/documents/iocta2018.pdf.

Lin, J., 1991. Divergence measures based on the shannon entropy. IEEE Trans. Inf. Theory 37 (1), 145–151.

Malwarebytes, Labs quarterly report finds ransomware's gone rampant against businesses (2019). https://blog.malwarebytes.com/reports/2019/08/labs-quarterly-report-finds-ransomwares-gone-rampant-against-businesses/.

Malwarebytes, malwarebytes for windows (2019). https://www.malwarebytes.com/premium/.

MarketsInsider, Infrascale survey reveals close to half of smbs have been ransomware attack targets (2020). https://markets.businessinsider.com/news/stocks/infrascale-survey-reveals-close-to-half-of-smbs-have-been-ransomware-attack-targets-1029112584/.

Mehnaz, S., Mudgerikar, A., Bertino, E., 2018. Rwguard: A real-time detection system against cryptographic ransomware. In: Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 114–136.

Park, J., Jung, Y., Won, J., Kang, M., Lee, S., Kim, J., 2019. Ransomblocker: A low-overhead ransomware-proof ssd. In: 2019 56th ACM/IEEE Design Automation Conference (DAC), pp. 1–6.

Payne, B.D., Martim, D.d.A., Lee, W., 2007. Secure and flexible monitoring of virtual machines. In: Proceedings of the 23rd Annual Computer Security Applications Conference, pp. 385–397.

Pfoh, J., Schneider, C., Eckert, C., 2011. Nitro: Hardware-based system call tracing for virtual machines. In: Proceedings of the 2011 International Conference on Advances in Information and Computer Security, pp. 96–112.

Redgate, Reflector - decompile, understand, and fix any.net code, even if you don't have the source (2017). https://www.red-gate.com/products/dotnet-development/reflector/.

Saberi, A., Fu, Y., Lin, Z., 2014. Hybrid-bridge: Efficiently bridging the semantic gap in virtual machine introspection via decoupled execution and training memoization. In: Proceedings of the 21st Annual Network and Distributed System Security Symposium, pp. 1–15.

Scaife, N., Carter, H., Traynor, P., Butler, K.R., 2016. Cryptolock (and drop it): stopping ransomware attacks on user data. In: Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems, pp. 303–312.

Sebastián, M., Rivera, R., Kotzias, P., Caballero, J., 2016. Avclass: A tool for massive malware labeling. In: International Symposium on Research in Attacks, Intrusions, and Defenses, pp. 230–253.

Sevtsov, A., Ransomware network communication [part 3] (2017). https://www.lastline.com/labsblog/ransomware-network-communication/.

Softonic, Softonic, a platform that offers clean and safe app downloads (2019). https://en.softonic.com/.

Srinivasan, D., Wang, Z., Jiang, X., Xu, D., 2011. Process out-grafting: an efficient out-of-vm approach for fine-grained process execution monitoring. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 363–374.

Team, A., Autoit v3 is a freeware basic-like scripting language designed for automating the windows gui and general scripting (2018). https://www.autoitscript.com/site/autoit/.

Trendmicro, Ransom buster offers your valuable files an extra layer of protection (2019). https://ransombuster.trendmicro.com/.

Trustwave, 2020 trustwave global security report (2020). https://www.trustwave.com/en-us/resources/library/documents/2020-trustwave-global-security-report/.

VirusShare, Virusshare.com - because sharing is caring (2017). https://virusshare.com/.

VirusTotal, Analyze suspicious files and urls to detect types of malware including viruses, worms, and trojans (2017). https://www.virustotal.com.

Wang, Y.-M., Beck, D., Vo, B., Roussev, R., Verbowski, C., 2005. Detecting stealth software with strider ghostbuster. In: Proceedings of the 2005 International Conference on Dependable Systems and Networks, pp. 368–377.

Wikipedia, Domain generation algorithm (2020). https://en.wikipedia.org/wiki/Domain_generation_algorithm/.

WinRAR, Winrar archiver, a powerful tool to process rar and zip files (2017). https://www.rarlab.com/.

Zhang, F., Leach, K., Stavrou, A., Wang, H., Sun, K., 2015. Using Hardware Features for Increased Debugging Transparency. In: Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P'15).

7zip, 7-zip, a file archiver with a high compression ratio(2018). https://www.7-zip.org/.

Fei Tang received the B.S. degree in information security from Xidian University, Xi'an, China, in 2017, where he is currently pursuing the M.S. degree with the School of Cyber Engineering. His research focuses on malware defense.

Boyang Ma received the B.S. and M.S. degrees in communications engineering from North China Electric Power University, Baoding, China, in 2011 and 2015. He is currently pursuing the Ph.D. degree in the School of Cyber Engineering at Xidian University, Xi'an, China. His research interests include operating system security and malware defense.

Jinku Li received the B.S., M.S., and Ph.D. degrees in computer science from Xi'an Jiaotong University, Xi'an, China, in 1998, 2001, and 2005, respectively. From 2009 to 2011, he was a Research Associate with the Department of Computer Science, North Carolina State University, Raleigh, NC, USA. He is currently a Professor with the School of Cyber Engineering, Xidian University, Xi'an, China. His research focuses on system and mobile security.

Fengwei Zhang received the Ph.D. degree in computer science from George Mason University in 2015. He is currently an Associate Professor with the Department of Computer Science and Engineering, SUSTech, Shenzhen, China. Before that, he was an Assistant Professor at Wayne State University, Detroit, MI, USA. His research interests are in the areas of systems security, with a focus on trustworthy execution, transparent malware debugging, transportation security, and plausible deniability encryption.

Jipeng Su received the B.S. degree in computer science from North China Electric Power University, Baoding, China, in 2015. He is currently pursuing the M.S. degree in the School of Cyber Engineering at Xidian University, Xi'an, China. His research interests include operating system security and malware defense.

Jianfeng Ma received the Ph.D. degree in computer software and communications engineering from Xidian University, Xi'an, China, in 1995. From 1999 to 2001, he was with the Nanyang Technological University of Singapore as a Research Fellow. He is currently a Professor with the School of Cyber Engineering, Xidian University. His current research interests focus on information and network security.