Available online at www.sciencedirect.com

ScienceDirect

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

Computers & Security

TC 11 Briefing Papers

Rcryptect: Real-time detection of cryptographic function in the user-space filesystem

Seungkwang Lee[a,b,*], *Nam-su Jho*[b], *Doyoung Chung*[a,b], *Yousung Kang*[b], *Myungchul Kim*[a,*]

[a] *Department of School of Computing, KAIST, Daejeon, 34141, South Korea*
[b] *Cryptography Research Team, ETRI, Daejeon, 34129, South Korea*

ARTICLE INFO

Article history:
Received 12 March 2021
Revised 16 October 2021
Accepted 18 October 2021
Available online 22 October 2021

Keywords:
Device security
Ransomware
Cryptographic function detection
Entropy
FUSE

ABSTRACT

The existing methods of ransomware detection have limitations. To be specific, static analysis is not effective to obfuscated binaries, while dynamic analysis is usually restricted to a certain platform and often takes tens of minutes. In this paper, we propose a block-level monitoring system to detect potentially malicious cryptographic operations. We carry out statistical analysis to find heuristic rules to distinguish between normal and encrypted blocks. In order to apply the heuristic rule to the filesystem without kernel modification, we adopt Filesystem in Userspace (FUSE) and define our filesystem *Rcryptect* for real-time detection of cryptographic function. We demonstrate the protection of well-known ransomware and show that various cryptographic functions can be detected with about 13% overhead.

1. Introduction

Cryptographic primitives providing confidentiality, integrity, and availability can be used by ransomware for malicious purposes. Recently, ransomware has become an increasingly attractive business model for malicious attackers; the average amount paid by victims was reportedly $1,077 Growth in ransomware. The first report of ransomware, which had a great impact, was known as locker-ransomware that locks access to a victim's device. Since then, crypto-ransomware has begun to replace locker-ransomware. This encrypts the victim's entire or some important files and demands ransom for the decryption key. In general, in order to encrypt as many files as possible within a short period of time, encryption is performed by a symmetric key algorithm, and the secret key used is protected by the attacker's asymmetric key. Even though there is no guarantee that the files or devices will be restored, victims keep paying up; otherwise most people would panic.

Ransomware can infiltrate the victim's computer by posing as an Adobe Flash installer like in the case of Bad Rabbit, which targeted Russian media group Interfax and Fontanka as well as public transportation targets in Ukraine Bad rabbit ransomware. It is also possible to use freeware such as CCleaner. Of 130 million users, the attackers exploited CCleaner to af-

* Corresponding authors.
E-mail addresses: yiskwang@kaist.ac.kr, skwang@etri.re.kr (S. Lee), nscho@etri.re.kr (N.-s. Jho), thisisdoyoung@etri.re.kr (D. Chung), youskang@etri.re.kr (Y. Kang), mck@kaist.ac.kr (M. Kim).
https://doi.org/10.1016/j.cose.2021.102512

fect more than two million computers Ransomware and free software. More seriously, large-scale ransomware can penetrate through networks and use anonymous digital currencies such as bitcoin to demand ransom. WannaCry, for example, spread this way. It took advantage of security vulnerabilities in some Windows operating systems, infecting more than 200,000 computers across 150 countries, with total financial losses to billions of dollars WannaCry and financial loss. Microsoft released patches for the vulnerability, but unpatched computers are still at risk. Recently, Kia Motors America has allegedly suffered a ransomware attack by the DoppelPaymer gang, demanding 20 million dollars Kia motors and ransomware. If ransomware targets hospitals, more computers can be damaged, and even life can be at risk. For example, patient and doctor records at Hollywood Presbyterian Hospital in Los Angeles were infected by ransomware in February, 2016 Hospital and ransomware. In this case, the only solution may be to pay the ransom and get the decryption key to restore the systems and administrative functions. For this reason, ransomware attacks and defense mechanisms have been an attractive area of research in recent years.

Detection techniques for crypto-ransomware generally use local static, local dynamic, or network traffic information. Due to the main characteristic of crypto-ransomware, some static and dynamic methods often aim to detect cryptographic functions based on an in-depth understanding of the target algorithm. Static analysis finds instruction chains, constant values and signatures (Matenaar et al., 2012; Wang et al., 2009) as well as the data flow graph isomorphism (Lestringant et al., 2015). However, such analysis often becomes expensive in terms of time and shows little effect when analyzing obfuscated binaries (Linn and Debray, 2003; Moser et al., 2007; Popov et al., 2007).

To detect cryptographic functions in obfuscated binaries, CryptoHunt (Xu et al., 2017), a dynamic method of a bit-precise symbolic loop mapping, was proposed. Later, its computational cost was reduced by 34 times using ATOS (Sun et al., 2019). CryptoHunt detects the target cryptographic code inside a loop body. In particular, it compares the core transformations with the reference implementations. So, the loop mapping methods can be bypassed by implementing a cryptographic algorithm in a different way than known reference implementations. For instance, a table-based implementation in the presence of dummy operations (Banescu and Pretschner, 2018) may be effective to make their mappings costly. Specifically, a white-box cryptographic implementation consisting of key-instantiated lookup tables (Chow et al., 2002a,b) can make it more difficult to analyze the internal algorithm when combined with dummy tables and lookups. In a technical point of view, the loop mapping is not effective to detect unknown cryptographic functions. Also, the detection takes tens of minutes to several hours, making it difficult to detect cryptographic functions immediately, and the target binary has to be executed in a secure environment.

The virtual machine introspection (VMI) can provide an out-of-VM solution of ransomware detection with memory forensics on the live guest OS (Ajay Kumara and Jaidhar, 2017; Tang et al., 2020). However, the slow performance of the guest OS is one of the serious drawbacks. In addition, it imposes an engineering effort of setting virtual environments. Park and Park (2020) proposed a hardware-assisted tracing method, recording the change-of-flow instructions from the CPU at run-time, for detecting symmetric-key cryptographic routines. There are also several techniques based on the avalanche effect of input-output dependencies (Caballero et al., 2010; Li et al., 2014) and unique input-output relations (Calvet et al., 2012; Gröbert et al., 2011; Zhao et al., 2011). However, the input and output can be protected by simple techniques such as data encoding.

There are also practical defense systems monitoring abnormal activities. Hosfelt adopted (Intel Pin) and machine learning in order to extract features and analyze the behavior of the target binary at run-time (Hosfelt, 2015). The authors in Ahmed et al. (2020) also used machine learning techniques to propose a filter method, removing the noisy features and selecting the real behaviour of ransomware from Windows API calls. Almgren et al., 2015 observed filesystem activities and filtered out particular I/O requests by hooking their method into the system service descriptor table. In Kharraz et al. (2015), the authors suggested to monitor the changes in Master File Table in the NTFS filesystem and suspicious sequences of Windows API calls. Kharraz *et al.* introduced UNVEIL (Kharraz et al., 2016). It utilizes an entropy-based detection which compares the entropy of read and write requests at the same file offset to detect encryption using the Windows Filesystem Minifilter Driver framework FltMgr (Microsoft file driver). Scaife et al. (2016) selected three indicators that broadly cover the transformation caused by ransomware: changes in file formats, entropy of created files, and the number of deleted files and file formats accessed by a process. A drop-in driver ShieldFS for the Windows native filesystem also monitors the entropy of write operations and the frequency of read/write/list/rename operations as well as the memory pages of potentially malicious process (Continella et al., 2016). While ShieldFS is not resistant to unknown cryptographic functions, Redemption Kharraz and Kirda (2017) does not rely on cryptographic primitive identification but requires modification of the operating system to perform behavioral monitoring on the common characteristics of ransomware. Most of the above countermeasures are restricted to a certain type of OS or CPU. Such Windows-based analysis using API calls, drivers, and filesystem is expected to be redesigned for other platforms. Although 85% of ransomware attacks target Windows systems, now ransomware is adapting to compromise Linux servers that administer enterprise networks, massive databases, and web services. Their owners, such as businesses or governmental institutions, are juicy targets because they can afford to pay ransom with sizeable budgets Linux ransomware characteristics; Linux ransomware threats. Depending on the CPU in the device, Intel PIN also needs to be replaced with other DBI tools such as Valgrind Valgrind home page; it is time consuming. At the flash-based storage level, a new buffer management policy was proposed to detect the repetitive overwriting-following-reading access (Paik et al., 2018). However, it cannot detect ransomware that erases the original file, nor can it distinguish between repetitive benign encryption.

A network-based method for detecting command and control (C&C) severs in the black list was proposed (Zahra and Shah, 2017). However, fixing the address of the C&C server in its binary is not a common practice for ransomware. Usually, ransomware locates C&C servers using the resolution of a DNS name to avoid network filtering (Cabaj and Mazurczyk, 2016; Quinkert et al., 2018). In addition, ransomware can attempt to resolve dozens of pseudo-randomly generated domain names in case the specific domain names are blocked. In Cabaj et al. (2018), the sequences of the HTTP messages and their respective content sizes were analyzed based on software-defined networking. Another network-based intrusion detection system was implemented by employing packet- and flow-level classifiers (Almashhadani et al., 2019). Although many ransomware families try to connect to C&C servers, the network connection is not always available, and also ransomware does not always connect to the outside.

In this paper, we present a novel method, *Rcryptect* (Real time + CRYPto + deTECT) to detect potentially malicious cryptographic functions at run-time in the filesystem. This is an entropy-based technique to filter out malicious blocks encrypted in the user-space filesystem. The file I/O in the user-space filesystem consists of block I/O. Therefore, block-level granularity allows us to take action at the earliest point in time to determine whether a set of blocks is encrypted, regardless of the boundaries of the files; this will result in a minimal loss of files. Our key idea is to observe the entropy of incoming blocks that will be written because standard cryptographic algorithms compute ciphertexts with outstanding randomness of 0's and 1's. To compute the entropy, we adapt the NIST randomness test suite (NIST SP 800-22) (Rukhin and others. 2010), a set of statistical tests for the validation of random and pseudo-random number generators. Since the frequency test provides the most basic evidence for the existence of non-uniformity among the 15 different types of tests, we utilize the frequency test in our heuristic rules to distinguish normal and encrypted blocks. If the suspicious blocks of high entropy are detected, the permission of the writing or deleting process is checked in order to distinguish between benign and malicious operations. To implement these functionalities in the filesystem without kernel modification, we use Filesystem in Userspace (FUSE) Dokan FUSE; FUSE examples; Winfsp, which is supported on various operating systems such as Linux, Windows, and macOS. In short, the advantages of *Rcryptect* are as follows:

- Block-level detection of cryptographic functions
- No kernel modification
- Applicable to various operating systems
- Effective to obfuscated ransomware
- No need for the network connection

The rest of this paper is organized as follows: Section 2 explains the basic concepts of FUSE and NIST randomness test suite. Afterwards, Section 3 proposes *Rcryptect* with our statistical analysis of normal and encrypted blocks. It also provides details of implementation and evaluation. We discuss limitations and conclude this paper in Section 4 and 5, respectively

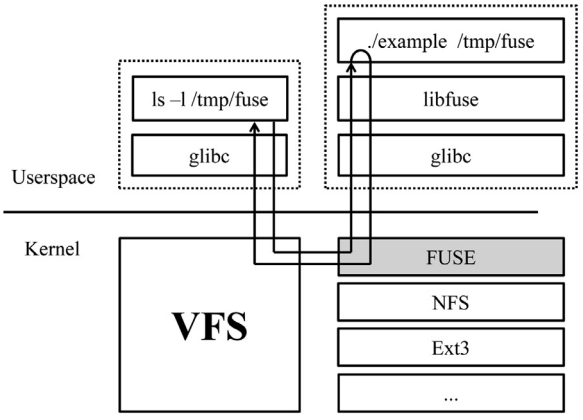


Fig. 1 – Structural diagram of FUSE.

2. Preliminaries

In this part, two open-source components coupled in the proposed method are introduced. First, FUSE will be used to define our filesystem in the user space without having to update kernel. Note that modifying kernel is not practical for industry practitioners because it requires a lot of labor and cost. Second, the NIST randomness test suite will evaluate the entropy of the writing blocks. By detecting cryptographic primitives at the block level, not at the file level, we can provide real-time monitoring on the filesystem. In the connection with the entropy test, the permission will also be checked by using FUSE to prevent repetitive malicious attempts.

2.1. *FUSE*

FUSE, a loadable kernel module, enables to develop a user-space filesystem that supports personalized features and extended capabilities without having to modify filesystem internals or kernel modules. FUSE is available for various platforms including Linux, FreeBSD, OpenBSD, OpenSolaris, Minix 3, Android, Windows, and macOS and is free software released under the GNU General Public License and the GNU Lesser General Public License. There are the reference implementations FUSE examples and a lot of open-source projects such as GamilFS, SSHFS, and LoggedFS.

To implement a new filesystem, a user has to first write a handler program that specifies how the filesystem responds to read/write/stat requests. This program linked to *libfuse* library is also used to mount the filesystem and is registered with the kernel when the filesystem is mounted. Upon receiving read-/write/stat requests for the mounted file system, the kernel forwards them to the handler and sends its response back to the user-space program that originally made the request. Fig. 1 shows an example of how FUSE works. An *ls* command to list files gets redirected by the kernel through VFS to FUSE which in turn executes the registered handler *example* on the request (ls -l /tmp/fuse). *example* then returns a response back to the user through FUSE. To achieve this, the callbacks connected to *struct fuse_operations* must to be written.

```
struct fuse_operations {

int (*getattr) (const char* path, struct stat* stbuf, struct fuse_file_info* fi);
int (*unlink) (const char* path);
int (*write) (const char* path, char* buf, size_t size, off_t offset, struct fuse_file_info* fi);
int (*read) (const char* path, char* buf, size_t size, off_t offset, struct fuse_file_info* fi);
... //omitted
};
```

In the case of *Rcryptect*, the writing and deleting functions are customized and pointed by *write* and *unlink*, respectively. The details will be explained later. After unmounting the FUSE filesystem, the files under the mount point can still be accessed by the underlying file operations.

2.2. NIST Randomness test suite

Each test in the NIST randomness test suite includes relevant randomness statistics that were formulated to test a *null hypothesis* (H0) that the sequence to be tested is random. The entropy of binary sequences is tested with respect to the following assumptions:

- Uniformity: The probability of occurrence of 0 or 1 at any point in the sequence to be tested is the same, and each occurrence probability is exactly one-half. Then we know that the expected number of zeros (or ones) is $\ell/2$, where ℓ is the length of the sequence.
- Scalability: Any test that can be applied to a sequence can be also applied to a randomly selected subsequence. If a sequence is random, its subsequence must be random (Rukhin and others, 2010).

A theoretical reference distribution of a statistic under an assumption of randomness is determined by mathematical methods, and a *critical value* is selected from this reference distribution. For each test, a test statistic value computed on the target binary sequence is compared to the critical value. If this value exceeds the critical value, H0 is rejected; otherwise it is accepted. The *level of significance*, denoted as α, is the probability that the test will conclude that the random sequence being tested is not random. Commonly, α in cryptography is about 0.01 (Rukhin and others, 2010). For example, an α of 0.01 indicates that a sequence in 100 random sequences would be rejected by the test. The test statistic is used to calculate *P-value*, a probability that a perfect random number generator produces a less-random sequence than the sequence being tested. If $P\text{-}value = 1$, this implies that the sequence appears to be perfect random. In this test package, if $P\text{-}value \geq \alpha$, then H0 is accepted; otherwise the sequence appears to be non-random. For example, $P\text{-}value \geq 0.01$ gives us that the sequence would be random with a probability of 99%.

As pointed out previously, the most basic evidence of non-randomness is supplied by the frequency test, often called the monobit test. For this reason, this is conducted before any other tests are carried out. This test assesses whether or not the numbers of zeros and ones in the tested sequence appear with approximately the same probability. Our FUSE-based filesystem will use it to distinguish encrypted and non-encrypted bit streams.

3. Real-time detection of cryptographic functions

We aim to design a real-time solution to detect cryptographic functions and variations in obfuscated binaries. Rather than applying complex policies based on typical behaviors of ransomware and providing a more fine-grained control over backup and restore operations, this proposes a first line of defense against potentially malicious cryptographic operations. To achieve it, we perform statistical analysis using the frequency test on various types of samples to distinguish the encrypted blocks from benign ones. Based on the entropy statistics, we define a heuristic rule for reducing false positives and adapt it to the filesystem. We call our proposed method *Rcryptect* and explain how to customize it below.

3.1. Threat model and assumptions

Many crypto-ransomware attacks, including samples used in the experiments below (Section 3.5), encrypt files after checking the whitelist or blacklist of file extensions to prevent the victim's system from crashing during encryption WannaCry report; Clop report; GandCrab report; JSWorm report; Ransom X report; PXJ report. Based on this fact, we first assume that ransomware performs selective encryption on specific file types, such as text, MP3, JPEG, and ZIP. Above all, it helps to finish the attack quickly. Even from a business perspective, brute-force encryption, which can destabilize the victim's system, is unnecessary. It is also assumed that the user stores the files of those extensions under the filesystem protected by *Rcryptect*. By mirroring the entire filesystem, *Rcryptect* may counteract ransomware attacks encrypting arbitrary files. However, not only does it increase the time required to attack, but it also reduces the performance of the entire file system. This study only considers attacks and defenses for limited file types and areas.

Second, we assume that the administrator always responds appropriately to vertical privilege escalation Privilege escalation toward a higher level of permission than the user already has. Therefore, ransomware is supposed to be executed with an account of non-root users. Based on this assumption, the root privileges are trusted while the users are not trusted.

3.2. Overview

Rcryptect protects the files under the mount point in the following ways with respect to writing and deleting. It sets the I/O size to 64KB, and thus each input/output block is not greater than 64KB. For every incoming block to be written, *Rcryptect* measures the entropy using the frequency test. If unencrypted, it is less likely that bit streams will appear to be random. Based on this fact, the entropy of multiple blocks will be analyzed to reduce false positives. By analyzing the statistics of sample blocks, we will distinguish encrypted blocks, which are called suspicious blocks. Based on the number of times suspicious blocks appear, *Rcryptect* will adjust the writing permission. As mentioned earlier, the root account is assumed to be secure. If *Rcryptect* aims to defend against attacks by privilege escalation, the high-entropy writing can be blocked to both root and non-root users after the first appearance of suspicious blocks. In this case, even benign encryption should be performed outside the *Rcryptect* filesystem. Our work does not take privilege escalation into account. Since the original files can be overwritten by encrypted files or can be deleted by ransomware, the file deletion is required to be customized. For this purpose, *Rcryptect* only allows the file deletion to authorized users once the writing of suspicious blocks takes place.

3.3. Statistical analysis of sample blocks

Among the target file extensions in the whitelist of crypto-ransomware, we chosen the four sample types: text files without figures, MP3, JPEG, and ZIP. The last three formats are likely to show relatively high entropy due to the use of statistic modeling techniques to reduce repetitive patterns. In particular, the MP3 and JPEG formats were chosen to test the entropy of lossy compression, and the ZIP format was chosen to test the lossless compression.

For each type of files, we prepared 1000 files for training and 200 files for testing the entropy. First, we used a python crawler (newspaper) to collect 1200 articles consisting of more than 1000 characters from BBC, BBC Sports, ABC, CNN, Slate, USA TODAY, ESPN, and Sky Sports. Second, the MP3 files were 1200 songs of KPOP and POP. Third, we also used a python crawler (google_images_download) collecting 1200 JPEG files of more than 10 Megapixels with the following keywords: BTS, Maroon 5, Justin Bieber, Drake, Luke, Michael Jackson, Bruno Mars, Taylor Swift, Billie Eilish, Beatles, Weeknd, and Arian Grande. Unlike encryption, compression does not dramatically increase the entropy of an original file. To test ZIP samples of relatively high entropy, 1200 ZIP files were created by compressing JPEG files collected in the same way as above using the following keywords: cloud, flower, forest, mountain, river, rain, sky, sea, and tree.

Table 1 shows the average size and standard deviation of the sample files. After checking the distribution of entropy for each type without encryption, we will show the increase in entropy after encryption. We analyzed the frequency test results on each block of the sample files. From the frequency test implementation, which is publicly available on Rukhin and others (2010), we slightly modified (flipped) the return value and denoted this version by $\mathcal{F}$ as shown in Algorithm 1 . $\mathcal{F}$ takes the input bit stream *buf* of *size* bytes and returns 0 if *P-value* $<$ α (0.01), which means the tested sequence is non-random; the return value 1 means that the sequence is random. Hereafter, high-entropy and low-entropy blocks indicate a set of blocks that returns 1 and 0 from $\mathcal{F}$, respectively.

Table 1 – Sample size (KB) of the training and testing sets.

	Training Average size (s.d)	Testing Average size (s.d)
Text	7.7 (9.9)	7.2 (9.6)
MP3	7922 (2946)	8557 (3011)
JPEG	2829 (2814)	2605 (2091)
ZIP	3695 (3600)	3652 (3323)

Algorithm 1: Frequency test $\mathcal{F}$.

Input : *buf*, *size*
Output: γ `// high and low entropy signal`

$\alpha \leftarrow 0.01$
for $i \leftarrow 0$ **to** $size - 1$ **do**
  $mask \leftarrow 0x80$
  for $j \leftarrow 0$ **to** 7 **do**
    if $(*(buf + i)$ & $mask)$ **then**
      $num\_1s \leftarrow num\_1s + 1$
    else
      $num\_0s \leftarrow num\_0s + 1$
    end if
    $mask \leftarrow mask \gg 1$
  end for
end for
$S_{obs} \leftarrow |num\_0s - num\_1s|/\sqrt{size \times 8}$
$P\text{-}value \leftarrow erfc(S_{obs}/\sqrt{2})$
if $(P\text{-}value < \alpha)$ **then**
  $\gamma \leftarrow 0$ `// low entropy`
else
  $\gamma \leftarrow 1$ `// high entropy`
end if
return γ

Table 2 summarizes the entropy of the samples. *P-value* is indicated up to the fourth decimal place. Thus, *P-value* $<$ 0.0001 is written as 0. The number of high-entropy blocks in text and MP3 files is less than 1 percent of the total blocks. The average *P-value* is less than α. In contrast, in the case of JPEG and ZIP files, the portion of high-entropy blocks is around 8 percent, but the entropy of some blocks is very large. So the average *P-value* is greater than α, and the standard deviation becomes large.

Table 3 shows the distribution of the length of consecutive blocks of high entropy for each type of training samples. Based on the entropy of the training set, we made a simple rule on the benign files: less than 90% blocks of all 40 consecutive blocks in the every type were high entropy. To check its correctness, we observed the testing set of each type. Table 4 shows the distribution of block entropy in the testing set, and

Table 2 – Frequency test results on blocks of the sample files.

	Training		Testing	
	P-value avg. (s.d)	High entropy (%)	P-value avg. (s.d)	High entropy (%)
Text	0 (0)	0	0 (0)	0
MP3	0.0002 (0.007)	0.1	0.0001 (0.01)	0.03
JPEG	0.022 (0.11)	7.4	0.025 (0.11)	7.9
ZIP	0.027 (0.12)	8.8	0.021 (0.1)	7

Table 3 – Distribution (%) and the maximum value of the length of consecutive high-entropy blocks in the training set (cutting off after the second decimal place).

	Length							
	1–5	6–10	11–15	16–20	21–25	26–30	30 - 35	max
Text	100	-	-	-	-	-	-	0
MP3	100	-	-	-	-	-	-	1
JPEG	95.55	3.83	0.33	0.16	-	0.05	0.05	31
ZIP	94.47	3.74	1.17	0.34	0.11	0.07	0.07	34

Table 4 – Distribution (%) and the maximum value of the length of consecutive high-entropy blocks in the testing set (cutting off after the second decimal place).

	Length							
	1–5	6–10	11–15	16–20	21–25	26–30	30 - 35	max
Text	100	-	-	-	-	-	-	0
MP3	100	-	-	-	-	-	-	2
JPEG	98.14	1.58	0.26	-	-	-	-	14
ZIP	94.67	4.11	0.96	0.24		0.25	-	19

Table 5 – Tested cryptographic building blocks.

Type	Primitive
Block cipher	AES-128
Stream cipher	RC4
Asymmetric cipher	RSA-2048
Hash function	SHA-256, MD5

there was no 40 consecutive blocks containing more than 36 high-entropy blocks.

Let's see how entropy changes when the cryptographic algorithms listed in Table 5 are applied. AES is the most well-known block cipher widely used by ransomware to encrypt victim's files. One of the software benchmarking results shows that AES-128 has the lowest cycle-per-byte count, followed by SPECK, TWINE, PRESENT, LED, and SIMON that are lightweight block ciphers (Diehl et al., 2017). This means that AES is a fast and secure block cipher. RC4 is the stream cipher candidate used by standard network protocols such as wired equivalent privacy (WEP). This algorithm is optionally used by secure shell (SSH), remote desktop protocol (RDP), and Kerberos. RSA, as an asymmetric cipher, is widely used for digital signatures, key exchanges, and encryption. In the case of ransomware, the secret key used in the block cipher is often protected by the attacker's public key. At last, SHA-256 and MD5 are hashing algorithms commonly used for the integrity checking in various applications. For these cryptographic algorithms we adapted OpenSSL, a general-purpose cryptography library, because this provides highly standardized and correct implementations and is also used by many programmers in practice.

We executed each cryptographic primitive with the training set of JPEG files and analyzed the entropy of the output blocks. As summarized in Table 6, the listed cryptographic primitives produced blocks with *P-value* of approximately 0.5, and about 99% of the output blocks were evaluated to be high entropy by $\mathcal{F}$. The maximum number of consecutive low-entropy blocks was just 2, but hardly found. The encrypted (or signed, hashed) blocks of the corresponding testing set also contained about 99% high-entropy blocks. Here we note that these cryptographic primitives produce high-entropy outputs regardless of the entropy of the plaintext (or message) and the secret key. Therefore, this entropy level should be considered not to be significantly affected by the sample.

By combining the statistics of the original sample blocks, we made our heuristic rule: if 90% or more of the n consecutive blocks are to be high entropy, the blocks are probably the outcome of a cryptographic function. Let $m = \lceil n \times 0.9 \rceil$. Considering the samples that contain high-entropy blocks, the candidate pair of (n, m) includes

$$(n, m) \in \{(40, 36), (42, 38), \ldots, (50, 45), \ldots\}.$$

Table 6 – Frequency test results on encrypted blocks of the sample files.

	Training		Testing	
	P-value avg. (s.d)	High entropy (%)	P-value avg. (s.d)	High entropy (%)
AES-128	0.49 (0.28)	98.9	0.49 (0.28)	99
RSA-2048	0.48 (0.29)	98.7	0.48 (0.28)	98.76
RC4	0.49 (0.28)	98.9	0.50 (0.28)	99.15
SHA-256	0.49 (0.28)	99	0.49 (0.28)	98.88
MD5	0.50 (0.28)	98.9	0.50 (0.29)	98.84

Here, the larger n, the lower false positives. Let's call n consecutive blocks a window. In our samples, setting the window size n to 40 could eliminate false positives/negatives for the every type of files. If it is an IoT-like environment, where only text or sound data is collected, setting $n \in [10, 20]$ may not result in false positives.

3.4. *Rcryptect implementation*

The FUSE library provides the communication with the socket and forwards the requests to user-defined functions. This is accomplished by using the callbacks, customized functions for each file operation, and by filling *fuse_operations*, a set of pointers to the callbacks. A callback function is called by FUSE when the designated event occurs in the filesystem. For example, when a user writes a chunk of data, the callback function pointed by *write* will be called.

Rcryptect customizes *write* and *unlink*. The corresponding callback functions are aptly named *Rwrite* and *Runlink*, respectively. When *Rcryptect* is mounted with the *allow_root* option, the user who mounts the filesystem and the root are allowed to access the filesystem. This will be used as a means of access control in connection with the writing of suspicious blocks or the deleting operations.

Implementing *Rwrite*. The *write* function defined in the *fuse_operations* structure takes several parameters to write *size* bytes from the buffer *buf*. This data will be written to the *offset* bytes of the file located in *path*. This returns the number of the bytes written successfully. *Rwrite* adds our features to these basic operations.

First of all, high-entropy blocks created by cryptographic operations will be detected by applying the parameters obtained from the statistical analysis. For $n = 40$ and $m = 36$, *Rcryptect* counts the number of high-entropy blocks that appear within 40 incoming blocks. Unless the current block is not the 36th high-entropy block, *Rcryptect* writes it. If it is the first time that a window contains the 36th high-entropy block, *Rcryptect* enters the stage of suspecting cryptographic operations, in which non-root users cannot complete the writing of the second suspicious blocks. At this stage, only the user with the root privilege will be allowed to write the suspicious blocks. If non-root users try to build the second suspicious blocks, *Rcryptect* considers it a malicious attack and forces the process to terminate. As a relaxed countermeasure, this policy can be modified to allow the process to write only low-entropy blocks instead of forcing it to terminate. To achieve this strategy, the root can be confirm by checking $uid = 0$ and $gid = 0$, and FUSE gets the caller *uid* and *gid* by `fuse_get_context()->uid` and `fuse_get_context()->gid`, respectively. After reaching the end of a window or detecting suspicious blocks, *Rcryptect* initializes to count the blocks of the next window and the number of high-entropy blocks. Through this block-level detection of cryptographic functions, *Rcryptect* prevents encryption by crypto-ransomware while allowing encryption by authorized users.

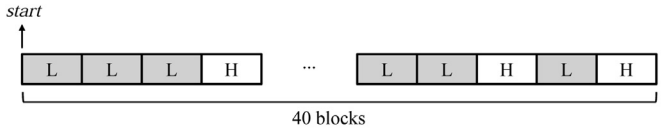
Fig. 2 shows three examples of incoming blocks to *Rwrite*. Suppose the writer is a non-root user. In the first window, more than 10% of the block elements are low entropy, and hence *Rcryptect* initializes the counters for the next window and high-entropy blocks. In the second window, *Rcryptect* encounters the 36th high-entropy block at the 38th block. This is the first detection of suspicious blocks. Afterwords, the writing of suspicious blocks is only allowed to the root user, and *Rcryptect* initializes the counters. In the third window, this user tries to write the 36th high-entropy block again. This process is now considered to be malicious.

Implementing *Runlink*. In the FUSE functions, *unlink* removes (deletes) the given file, symbolic link, hard link, or special node. Before entering the stage of suspecting cryptographic operations, the deletion is allowed to the mount owner and the root users. After entering the stage, the deletion is only allowed for the root users. By doing so, *Rcryptect* protects data from ransomware, attempting encryption and deletion. In addition, if encryption and deletion operations are suspended, it will be impossible to establish a business model for ransomware.

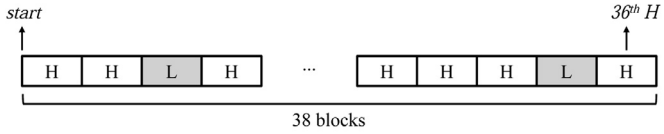
Killing the suspicious process. A non-privileged process can be terminated by using `kill(pid, SIGKILL)` if it continues to encrypt up to the second suspicious blocks, where the process ID (`pid`) can be obtained by `fuse_get_context()->pid`. The following experiments demonstrate how it works against well-known ransomware samples. Since *Rcryptect* detects malicious cryptographic functions in the filesystem, the ransomware samples are chosen based on encryption methods of ransomware.

3.5. *Experimental results*

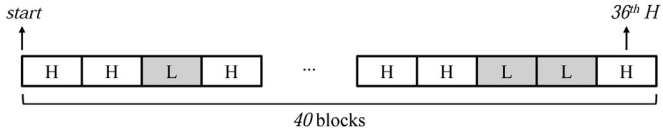
First, we tested WannaCry WannaCry report, JSWorm JSWorm report, PXJ PXJ report, and RansomEXX (also known as Ransom X) RansomExx report, which use a combination of (standard or customized) AES and RSA encryption. Many modern ransomware use the same hybrid encryption. The combination of key sizes varies slightly from ransomware to ransomware, but we have previously shown that AES-128 (smallest key size)



(a) A window of low-entropy blocks.



(b) First window of suspicious blocks.



(c) Second window of suspicious blocks.

Fig. 2 – Examples of incoming blocks in *Rwrite*. H: high-entropy block; L: low-entropy block.

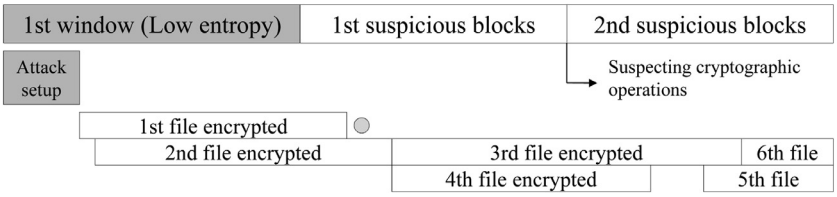


Fig. 3 – Brief description of the encryption by RansomEXX. Gray circle: incomplete writing of !TXDOT_READ_ME!.

guarantees high entropy. In particular, RansomEXX is the well-known ransomware used by criminals to attack both Windows and Linux environments explained in Section 1. Next, we executed Clop Clop report on our filesystem. It appeared in February 2019 and targeted companies with Active Directory servers, not individuals. This ransomware encrypts with RC4 and protects keys with RSA-1024. Finally, we selected Gandcrab v5 GandCrab report; GandCrab victims, which appeared in January 2018, in order to test ransomware using cryptographic algorithms not covered by Table 5. This is Ransomware as a Service (RaaS), which has infected over 500,000 victims world-wide and uses a customized Salsa20 algorithm to encrypt files and RSA-2048 to conceal the key.

A new set of sample files (five for each type) has been placed under the *Downloads* directory for easy detection by ransomware during file scanning. The average size of the files was 2.4 MB. If ransomware overwrites the original files, the expected loss of files is then approximately three, considering that some encryption can start from the center of the window. In the case of partial encryption, the number of losses can increase. The ransomware binaries were executed on Ubuntu 18.04 by using Wine WineHQ home page, an open-source compatibility layer that enables Unix-like operating systems to run Windows applications. Indeed, this is one of the ways the Windows-based ransomware invades Unix-based systems.

While scanning, WannaCry skips over files and folders related to binaries, libraries, and their attack because it aims to encrypt user data. After finishing the configuration, WannaCry began to encrypt a text file by which *Rcryptect* has entered the stage of suspecting cryptographic operations. Next, it encrypted an image file and was finally killed by *Rcryptect*. It is noteworthy that the original files were not deleted because WannaCry was terminated before the attack was completed. Of course, even if WannaCry tried, the file would not have been deleted because it did not have permission to delete. JSWorm first encrypted two files and then forced to terminate while encrypting the next file. PXJ was also killed while encrypting the third target file.

When testing RansomEXX, four files were encrypted, and two files were corrupted before its termination. As illustrated in Fig. 3, approximately one-third of the first window was devoted to writing low-entropy blocks for attack setup; the first window becomes low entropy. RansomEXX conducted two file encryption in parallel and encrypted only the first 1MB of each file. For optimization, ransomware often encrypts only the first few K/M bytes of large files. The first suspicious blocks appeared while encrypting the third and fourth files. The fifth and sixth files were not encrypted completely, but the beginning parts were corrupted by RansomEXX, which overwrites the original files.

Table 7 – Increase in the time for writing on *Rcryptect*.

	Elapsed time increase (%)
Text	12.3
MP3	15.1
JPEG	15.3
ZIP	12.2

Clop, which started encrypting files at 3/4 of a window, created the first suspicious blocks in the next window. At this point, encryption of the first file was not complete, and unauthorized users were unable to delete files. In the next window, the first file was encrypted, and ransomware was forcibly terminated in the process of encrypting the second file. Since it created a copy to be encrypted, the original files remained intact. In the case of Gandcrab v5, if an Internet connection is not provided, it is deleted by itself. Unlike RansomEXX conducting partial encryption for all types, Gandcrab v5 fully encrypted the first text file (approx. 2.4MB). While encrypting the first file, it entered the stage of suspecting cryptographic operations. The following window finished this encryption which overwrites the original file. The deletion operation was not involved. The second file was an MP3 file, and only the first 1MB part was encrypted. The ransomware was then forcibly terminated, encrypting the third file.

To put it simply, approximately 3 - 6 files were fully or partially corrupted when overwriting the original file with encrypted data, while no file loss occurred when creating an encrypted copy and erasing the original file. Considering the size of the encrypted blocks, it lost approximately less than 6MB.

3.6. *Overhead*

So far, we have proposed *Rcryptect* to mitigate ransomware attacks in the user-space filesystem. According to Vangoor et al. (2017), the performance of a FUSE-based filesystem is dependent on storage devices and FUSE configurations. In the case of writing, the performance is known to be degraded by less than 5%. The performance of write workloads on SSDs may improve in some cases.

When writing each type of samples on the proposed filesystem ($n = 40$) using 3.4GHz Core and 4GB RAM, the delays are shown in Table 7. The writing time was increased by about 10–15%, compared to a naive FUSE filesystem. This overhead was similar on other platforms including macOS and Windows 10. There was no noticeable proportional or inversely proportional relationship when changing n in the range of 30 - 50. The entropy level of each type does not also seem to have an impact on the overhead.

Table 8 shows the comparison results with existing work based on behavioral monitor introduced in Section 1. Like *Rcryptect*, ShieldFS and Redemption are end-point solutions whereas Ransomspector is one of VMI-based inspectors. The main advantages of *Rcryptect* are that it does not require kernel modification and can be easily applied to a variety of operating systems. In comparison, the other methods require engineering efforts (redesign or development) to apply them to different platforms. Note that the median (average) loss of files from *Rcryptect* includes the partially corrupted files. ShieldFS and Redemption provide minimal data loss using data backups. The overhead indicates how slow the performance of the filesystem has been due to the use of each solution. Our method imposes overhead only on write operations on the FUSE file system. When it comes to performance metrics in VMI environments (Tang et al., 2020), it should be considered that the guest OS is bound to have inherently slow performance. According to Zhang et al. (2010), the throughput of sequential block writes on the kernel-based virtual machine (KVM) turns out to be only about a half compared to the native machine. On the other hand, our work is not dependent on a virtual environment. Importantly, there is little or no performance degradation by FUSE in SSD-enabled environments. The downsides of *Rcryptect* are that it does not provide full data recovery and can be vulnerable when the root privilege is stolen.

4. Discussion

Attack variant. Ransomware attackers can make a variant in order to mitigate the protection of *Rcryptect*. For example, low-entropy cryptographic primitives may be chosen to bypass entropy-based detection. It should be noted that low-entropy primitives do not imply lightweight ones (Isa and Z'aba, 2012; Isa and Z'aba, 2014; Risqi and Windarta, 2017). Because reducing the entropy of the ciphertext also means that encrypted files are more likely to be recovered over time, an attacker will not be able to significantly lower the entropy of ciphertexts. Another possibility is to encrypt only certain parts of the file or pad low-entropy blocks at certain intervals. This variant can be detected with high probability by checking if the high-entropy block appears t times at a certain offset of the win-

Table 8 – Comparison of *Rcryptect* with existing research work. The numbers shown for Continella et al. (2016); Kharraz and Kirda (2017); Tang et al. (2020) are extracted from the respective papers.

	Rcryptect	ShieldFS Continella et al. (2016)	Redemption Kharraz and Kirda (2017)	Ransomspector Tang et al. (2020)
Kernel updates	No	Yes	Yes	No
OS dependency	No	Yes	Yes	Yes
Target activity	Filesystem	Filesystem	Filesystem	Filesystem & Network
Loss of data/files	3	0	0	2.67
Bypass possibility	Yes	Yes	Yes	Difficult
Overhead	13%	30 - 380%	3 - 9%	4.96% + KVM overhead

dows. However, it takes longer to detect. In the experiments above, such partial encryption overwrites the front part of the original files. As a result, only encrypted blocks were written consecutively in windows, making them "suspicious blocks".

False positive. There may be a false positive problem that benign write workloads of applications can be considered malicious. Since the proposed method takes parameters obtained from heuristic analysis of training and testing samples, it may indicate false positives for benign blocks with high entropy that were not found in our samples. For example, if a portion of a compressed file includes two or more windows consisting of 36 or more high-entropy blocks, it results in false positives. One of the simple solutions is to increase n, e.g., $n = 50$ or 60. Based on the ratio and density of high entropy in the benign blocks shown in Table 2 - 4, the parameters $(n, m) = (60, 54)$ will reduce the probability of false positives compared to (40, 36). Additionally, based on our statistical analysis, the heuristic rule is still valid even if we increase the size of m a little more than $\lceil n \times 0.9 \rceil$. This is because the standard cryptographic algorithms guarantee nearly 99% of high-entropy blocks. For example, the parameter set (60, 56) may detect encrypted blocks while reducing false positives. Of course, the drawback is an increased time required for detection.

Loss of the first few files. As shown in the experiments, some files can be damaged before *Rcryptect* terminates ransomware. There are two main reasons. First, *Rcryptect* sets the I/O block size to 64KB in order to reduce the number of file accesses. Based on scalability of the frequency test described in Section 2, a block size of 4KB probably results in the similar level of entropy on the output blocks of cryptographic algorithms. Thus, the use of 4KB blocks and small n can contribute to an early detection. Since it requires to use the root account for benign writing operations, this does not seem to be practical. Next, the second appearance of suspicious blocks determines whether the process is ransomware. The first appearance is currently an intermediate stage to make a careful decision. To reduce the loss of files, *Rcryptect* can make an early decision on the first suspicious blocks.

IoT. Our method can be effective to low-cost devices in the consideration of ransomware threats in Internet of Things (IoT) context (Humayun et al., 2020; Zahra and Shah, 2017; Zahra and Ahsan Chishti, 2019). Since low-cost IoT devices may have security flaws, ransomware will have catastrophic impacts that not only result in financial losses but also privacy violations and life risks. Because IoT data is periodically transmitted to the cloud storage there is not much incentive to pay to recover data that has already been backed up. However, ransomware is still notorious due to the characteristics of the IoT. For limited resources and computing power, real-time data is transmitted to the cloud at pre-defined time intervals from minutes to days Amano et al., 2020; IoT transmission interval, 2020. Without full-fledged protection against various threats, the interval can a security hole.

Fortunately, the IoT device has only a few missions, so it is relatively easy to manage the files by observing a few areas of the filesystem. Currently, it is not common to encrypt all data partly because they are implemented cheaply and also partly due to the lack of computing power. For this reason, most IoT devices collect sensing data in the filesystem without encryption and periodically send to the cloud over secure channels. From this real-world perspective, the detection of unauthorized cryptographic functions in the filesystem of IoT devices leads, with high probability, to the protection of ransomware.

5. Conclusion and future work

There is a need for a first line of defense that can be easily applied before the files are lost due to indiscriminate attacks by ransomware. In summary, we propose *Rcryptect*, a FUSE-based filesystem that examines the entropy of blocks to detect cryptographic operations in ransomware. With the frequency test defined in the NIST randomness test suite, we test whether or not a set of blocks has problematic entropy larger than the threshold. Because some compressed type of files may show relatively high entropy, multiple blocks are analyzed to make a right decision. Based on our experimental results, *Rcryptect* can detect cryptographic functions by using suspicious blocks such that 90% or more of n consecutive blocks appear to be high entropy. After the first detection of suspicious blocks, non-root users are not allowed to write the next suspicious blocks, and only the root user can execute the file deletion. We demonstrate that *Rcryptect* can detect and prevent the well-known ransomware. This can be applied to various platform without installing kernel update, and the additional cost is approximately a 13% increase of the writing time in the filesystem.

In addition to detection, an in-depth analysis sometimes requires identification of cryptographic functions. However, our experimental result indicates that the modern cryptographic functions produce the similar level of entropy on the output. For this reason, it cannot identify the cryptographic algorithm inside ransomware. Our future work is possibly to combine a set of indicators of well-known ransomware behaviors in order to distinguish benign cryptographic functions from malicious ransomware.

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

Seungkwang Lee: Conceptualization, Methodology, Software, Visualization, Writing – original draft. **Nam-su Jho:** Formal analysis, Funding acquisition, Investigation. **Doyoung Chung:** Software, Writing – original draft. **Yousung Kang:** Funding acquisition. **Myungchul Kim:** Resources, Supervision, Validation.

Acknowledgment

This work was supported in part by Electronics and Telecommunications Research Institute (ETRI) grant funded by the Korean Government (20ZR1300, Core Technology Research on

Trust Data Connectome), and in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government, Ministry of Science and ICT (MSIT) (No. 2021R1A2C1004993), and in part by Unmanned Vehicles Advanced Core Technology Research and Development Program through the National Research Foundation of Korea (NRF), Unmanned Vehicle Advanced Research Center (UVARC) funded by the Ministry of Science and ICT, the Republic of Korea (NRF-2017M1B3A2A01056680).

REFERENCES

Ahmed YA, Koçer B, Huda S, Saleh Al-rimy BA, Hassan MM. A system call refinement-based enhanced minimum redundancy maximum relevance method for ransomware early detection. Journal of Network and Computer Applications 2020;167:102753. doi:10.1016/j.jnca.2020.102753. https://www.sciencedirect.com/science/article/pii/S1084804520302277

Ajay Kumara M, Jaidhar C. Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. Digital Invest. 2017;23:99–123. doi:10.1016/j.diin.2017.10.004. https://www.sciencedirect.com/science/article/pii/S1742287617303328

Almashhadani AO, Kaiiali M, Sezer S, O'Kane P. A multi-classifier network-based crypto ransomware detection system: a case study of locky ransomware. IEEE Access 2019;7:47053–67. doi:10.1109/ACCESS.2019.2907485.

Analysis of WannaCry, https://gist.github.com/rain-1/989428fa5504f378b993ee6efbc0b168 Accessed: 2020-09-15.

Anatomy of a Linux Ransomware Attack, https://linuxsecurity.com/features/features/anatomy-of-a-linux-ransomware-attack, Accessed: 2021-03-05.

Amano M, Ohta M, Taromaru M. Transmission Timing Adjustment Algorithm in IoT System Equipped with Low-cost Oscillator with Low Frequency Stability. In: 2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC); 2020. p. 060–3.

Banescu S, Pretschner A. Chapter Five - a Tutorial on Software Obfuscation. In: Advances in Computers, Vol. 108. Elsevier; 2018. p. 283–353. doi:10.1016/bs.adcom.2017.09.004. https://www.sciencedirect.com/science/article/pii/S0065245817300499

Cabaj K, Gregorczyk M, Mazurczyk W. Software-defined networking-based crypto ransomware detection using http traffic characteristics. Computers & Electrical Engineering 2018;66:353–68. doi:10.1016/j.compeleceng.2017.10.012. https://www.sciencedirect.com/science/article/pii/S0045790617333542

Cabaj K, Mazurczyk W. Using software-defined networking for ransomware mitigation: the case of cryptowall. IEEE Netw 2016;30(6):14–20. doi:10.1109/MNET.2016.1600110NM.

Caballero J, Poosankam P, McCamant S, Babic D, Song D. Input generation via decomposition and re-stitching: finding bugs in malware. In: ACM Conference on Computer and Communications Security. ACM; 2010. p. 413–25.

Calvet J, Fernandez JM, Marion J-Y. Aligot: Cryptographic function identification in obfuscated binary programs. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. New York, NY, USA: ACM; 2012. p. 169–82.

Chow S, Eisen PA, Johnson H, van Oorschot PC. A White-Box DES Implementation for DRM Applications. In: Security and Privacy in Digital Rights Management, ACM CCS-9 Workshop, DRM 2002, Washington, DC, USA, November 18, 2002, Revised Papers; 2002. p. 1–15. doi:10.1007/978-3-540-44993-5_1.

Chow S, Eisen PA, Johnson H, van Oorschot PC. White-Box Cryptography and an AES Implementation. In: Proceedings of the Ninth Workshop on Selected Areas in Cryptography (SAC 2002). Springer-Verlag; 2002. p. 250–70.

Clop Ransomware, https://www.mcafee.com/blogs/other-blogs/mcafee-labs/clop-ransomware/, Accessed: 2021-05-15.

Continella A, Guagnelli A, Zingaro G, De Pasquale G, Barenghi A, Zanero S, Maggi F. Shieldfs: A self-healing, ransomware-aware filesystem. New York, NY, USA: Association for Computing Machinery; 2016. p. 336–47.

Diehl W, Farahmand F, Yalla P, Kaps J, Gaj K. Comparison of Hardware and Software Implementations of Selected Lightweight Block Ciphers. In: 2017 27th International Conference on Field Programmable Logic and Applications (FPL); 2017. p. 1–4.

Different ways to cook a crab: GandCrab ransomware-as-a-service (RaaS) analysed in depth, https://www.virusbulletin.com/virusbulletin/2019/11/vb2019-paper-different-ways-cook-crab-gandcrab-ransomware-service-raas-analysed-indepth/, Accessed: 2021-05-15.

Dokan - User mode file system for Windows with FUSE Wrapper, https://dokan-dev.github.io/, Accessed: 2021-01-15.

Efento NB-IoT/LTE-M wireless sensors, https://getefento.com/technology/efento-nb-iot-lte-m-wireless-sensors/, Accessed: 2020-09-15.

Filter Manager Concepts, https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/filter-manager-concepts, Accessed: 2020-09-15.

GandCrab Ransomware Master Decryption Keys Released, https://www.packetlabs.net/gandcrab-ransomware/, Accessed: 2021-07-26.

Gröbert F, Willems C, Holz T. Automated Identification of Cryptographic Primitives in Binary Programs. In: RAID. Springer; 2011. p. 41–60.

Hackers infiltrate free PC cleaning software, https://money.cnn.com/2017/09/18/technology/business/windows-ccleaner-hack/index.html, Accessed: 2020-09-15.

Hosfelt DD. Automated detection and classification of cryptographic algorithms in binary programs through machine learning. CoRR 2015 arXiv:1503.01186.

Hospital pays bitcoin ransom after malware attack, https://money.cnn.com/2016/02/17/technology/hospital-bitcoin-ransom/index.html, Accessed: 2020-09-15.

Humayun M, Jhanjhi N, Alsayat A, Ponnusamy V. Internet of things and ransomware: evolution, mitigation and prevention. Egyptian Informatics Journal 2020. doi:10.1016/j.eij.2020.05.003.

Isa H, Z'aba MR. Randomness analysis on LED block ciphers. In: Gaur MS, Elçi A, Makarevich OB, Orgun MA, Singh V, editors. In: 5th International Conference of Security of Information and Networks, SIN '12, Jaipur, India, October 22, - 26, 2012. ACM; 2012. p. 60–6. doi:10.1145/2388576.2388584.

Isa H, Z'aba MR. Randomness of the PRINCE Block Cipher. In: International Conference on Frontiers of Communications, Networks and Applications (ICFCNA 2014 - Malaysia); 2014. p. 1–6.

JSWorm: The 4th Version of the Infamous Ransomware, https://securityaffairs.co/wordpress/90811/malware/jsworm-4-ransomware-analysis.html, Accessed: 2021-05-15.

Kharraz A, Arshad S, Mulliner C, Robertson WK, Kirda E. UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware. In: Holz T, Savage S, editors. In: 25th USENIX

Security Symposium, USENIX Security 16, Austin, TX, USA, August 10–12, 2016. USENIX Association; 2016. p. 757–72.
Kharraz A, Kirda E. Redemption: Real-Time Protection Against Ransomware at End-Hosts. In: Dacier M, Bailey M, Polychronakis M, Antonakakis M, editors. In: Research in Attacks, Intrusions, and Defenses - 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings. Springer; 2017. p. 98–119. doi:10.1007/978-3-319-66332-6_5.
Kharraz A, Robertson W, Balzarotti D, Bilge L, Kirda E. Cutting the gordian knot: A look under the hood of ransomware attacks. In: Proceedings of the 12th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9148. Berlin, Heidelberg: Springer-Verlag; 2015. p. 3–24. doi:10.1007/978-3-319-20550-2_1.
Kia Motors allegedly suffers ransomware attack, https://www.securitymagazine.com/articles/94633-kia-motors-allegedly-suffers-ransomware-attack-cybercriminals-demand-20-million-to-recover-sensitive-data, Accessed: 2021-02-27.
Lestringant P, Guihéry F, Fouque P. Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism. In: AsiaCCS. ACM; 2015. p. 203–14.
Li X, Wang X, Chang W. Cipherxray: exposing cryptographic operations and transient secrets from monitored binary execution. IEEE Trans. Dependable Sec. Comput. 2014;11(2):101–14.
Linn C, Debray SK. Obfuscation of Executable Code to Improve Resistance to Static Disassembly. In: ACM Conference on Computer and Communications Security. ACM; 2003. p. 290–9.
Matenaar F, Wichmann A, Leder F, Gerhards-Padilla E. CIS: The Crypto Intelligence System for Automatic Detection and Localization of Cryptographic Functions in Current Malware. In: MALWARE. IEEE Computer Society; 2012. p. 46–53.
Moser A, Kruegel C, Kirda E. Limits of Static Analysis for Malware Detection. In: ACSAC. IEEE Computer Society; 2007. p. 421–30.
New Ransom X Ransomware used in Texas TxDOT Cyberattack, https://www.bleepingcomputer.com/news/security/new-ransom-x-ransomware-used-in-texas-txdot-cyberattack/, Accessed: 2021-07-26.
New ransomware attack hits Russia and spreads around globe, https://money.cnn.com/2017/10/24/technology/bad-rabbit-ransomware-attack/index.html, Accessed: 2020-09-15.
New Ransomware Threat Jumps From Windows To Linux-What You Need To Know, https://www.forbes.com/sites/daveywinder/2020/11/08/new-ransomware-threat-jumps-from-windows-to-linux-what-you-need-to-know/?sh=1ae954da3265, Accessed: 2021-03-05.
Paik J-Y, Choi J-H, Jin R, Wang J, Cho E-S. A storage-level detection mechanism against crypto-ransomware. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. New York, NY, USA: ACM; 2018. p. 2258–60. doi:10.1145/3243734.3278491.
Park J, Park Y. Symmetric-Key cryptographic routine detection in anti-Reverse engineered binaries using hardware tracing. Electronics (Basel) 2020;9(6). doi:10.3390/electronics9060957.
Pin - A Dynamic Binary Instrumentation Tool, https://software.intel.com/content/www/us/en/develop/articles/pin-a-dynamic-binary-instrumentation-tool.html, Accessed: 2020-09-15.
Popov IV, Debray SK, Andrews GR. In: USENIX Security Symposium. Binary Obfuscation Using Signals. USENIX Association; 2007.
Privilege Escalation Techniques, https://attack.mitre.org/tactics/TA0004/, Accessed: 2021-05-15.
Pxj ransomware removal and recovery guide, https://sensorstechforum.com/pxj-virus-removal/, Accessed: 2021-05-15.
Quinkert F, Holz T, Hossain KSMT, Ferrara E, Lerman K. RAPTOR: Ransomware attack predictor. CoRR 2018 arXiv:1803.01598.
RansomExx: The malware that attacks Linux OS, https://resources.infosecinstitute.com/topic/ransomexx-the-malware-that-attacks-linux-os/, Accessed: 2021-02-15.
Ransomware attacks around the world grow by 50%, https://www.bbc.com/news/technology-39730407, Accessed: 2020-09-15.
Risqi YSS, Windarta S. Statistical Test on Lightweight Block Cipher-Based PRNG. In: 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA); 2017. p. 1–4.
Rukhin, A., others, 2010NIST SP 800-22 rev1a (dated April 2010), A Statistical Test Suite for the Validation of Random Number Generators and Pseudo Random Number Generators for Cryptographic Applications.https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software.
Scaife N, Carter H, Traynor P, Butler KRB. CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data. In: ICDCS. IEEE Computer Society; 2016. p. 303–12.
Sun H, Zhang C, Li H, Wu Z, Wu L, Li Y. ATOS: Adaptive program tracing with online control flow graph support. IEEE Access 2019;7:127495–510.
Tang F, Ma B, Li J, Zhang F, Su J, Ma J. Ransomspector: an introspection-based approach to detect crypto ransomware. Computers & Security 2020;97:101997. doi:10.1016/j.cose.2020.101997. https://www.sciencedirect.com/science/article/pii/S0167404820302704
The reference implementation of the Linux FUSE (Filesystem in Userspace) interface, https://github.com/libfuse/libfuse, Accessed: 2020-09-15.
Valgrind, https://valgrind.org/, Accessed: 2021-05-15.
Vangoor BKR, Tarasov V, Zadok E. To FUSE or Not to FUSE: Performance of User-Space File Systems. In: 15th USENIX Conference on File and Storage Technologies (FAST 17). Santa Clara, CA: USENIX Association; 2017. p. 59–72. https://www.usenix.org/conference/fast17/technical-sessions/presentation/vangoor
Wang Z, Jiang X, Cui W, Wang X, Grace M. ReFormat: Automatic Reverse Engineering of Encrypted Messages. In: ESORICS. Springer; 2009. p. 200–15.
WannaCry: Lessons Learned 1 Year Later, https://symantec-enterprise-blogs.security.com/blogs/feature-stories/wannacry-lessons-learned-1-year-later, Accessed: 2020-09-15.
Windows File System Proxy, https://github.com/billziss-gh/winfsp, Accessed: 2021-01-15.
WineHQ wiki, https://wiki.winehq.org/Main_Page, Accessed: 2020-10-15.
Xu D, Ming J, Wu D. Cryptographic Function Detection in Obfuscated Binaries via Bit-Precise Symbolic Loop Mapping. In: IEEE Symposium on Security and Privacy. IEEE Computer Society; 2017. p. 921–37.
Zahra SR, Ahsan Chishti M. RansomWare and Internet of Things: A New Security Nightmare. In: 2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence); 2019. p. 551–5.
Zahra A, Shah MA. IoT Based Ransomware Growth Rate Evaluation and Detection using Command and Control Blacklisting. In: 2017 23rd International Conference on Automation and Computing (ICAC); 2017. p. 1–6.
Zhang B, Wang X, Lai R, Yang L, Wang Z, Luo Y, Li X. Evaluating and optimizing i/o virtualization in kernel-based virtual machine (kvm). In: Ding C, Shao Z, Zheng R, editors. In:

Network and Parallel Computing. Berlin, Heidelberg: Springer Berlin Heidelberg; 2010. p. 220–31.
Zhao R, Gu D, Li J, Yu R. Detection and Analysis of Cryptographic Data Inside Software. In: ISC. Springer; 2011. p. 182–96.
Almgren M, Gulisano V, Maggi F (Eds.), 2015. Detection of Intrusions and Malware, and Vulnerability Assessment - 12th International Conference, DIMVA 2015, Milan, Italy, July 9–10, 2015, Proceedings. Lecture Notes in Computer Science, Vol. 9148, Springer; 2015.

Seungkwang Lee received his B.S. in computer science and electronic engineering from Handong University in 2009, M.S. in computer science from Pohang University of Science and Technology (POSTECH) in 2011, and Ph.D. in school of computing from Korea Advanced Institute of Science and Technology (KAIST) in 2021. Since 2011, he has been with the Electronics and Telecommunications Research Institute (ETRI). His research interests include side-channel analysis and white-box cryptography.

Nam-Su Jho received the B.S. degree in mathematics from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1999 and the Ph.D. degree in mathematics from Seoul National University, Seoul, Korea, in 2007. Since 2007, he has been with the Electronics and Telecommunications Research Institute (ETRI) as a senior researcher. His research interests include cryptography and information theory.

Doyoung Chung received the B.S. and Master's degrees in School of computing from Korea Advanced Institute of Science and Technology (KAIST). Currently, he is working as a senior researcher in the Electronics and Telecommunications Research Institute (ETRI) and is a doctoral candidate in School of computing, KAIST. His main research interests include quantum cryptanalysis and deep learning for cyber security.

Yousung Kang joined Electronics and Telecommunications Research Institute (ETRI) in 1999, and he is now a principal researcher. Since 2004, he has been the IT international standard expert of Telecommunications Technology Association (TTA). He was a visiting researcher at Queen's University Belfast, UK, from 2011 to 2012. His research interests include the areas of cryptographic protocol, side-channel analysis, key-hiding technology, and drone security. He obtained his Bachelor and Master of Engineering degrees in Electronics Engineering from Chonnam National University, Gwangju, Korea in 1997 and 1999, respectively. He received his Doctor of Philosophy in Electrical and Electronic Engineering from KAIST, Daejeon, Korea in 2015.

Myungchul Kim received his B.A. in Electronics Engineering from Ajou University in 1982, M.S. in Computer Science from the Korea Advanced Institute of Science and Technology (KAIST) in 1984, and Ph.D. in Computer Science from the University of British Columbia, Vancouver, Canada, in 1993. Currently, he is with the faculty of the KAIST as Professor in the School of Computing. Before joining the university, he was a managing director in Korea Telecom Research and Development Group during 1984–1997 where he was in charge of research and development of protocol and QoS testing on ATM/B-ISDN, IN, PCS and Internet. He has also served as a member of Program Committees for numerous numbers of conferences and served as chair of the IWTCS'97 and the FORTE'01. He has published over 150 conference proceedings, book chapters, and journal articles in the areas of computer networks, wireless mobile networks, protocol engineering and network security. His research interests include Internet, protocol engineering, mobile computing, and information security.