Preventing Ransomware Damages using In-Operation Off-Site Backup to Achieve a 10^{-8} False-Negative Miss-Detection Rate

Hiroshi Fujinoki
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL, USA 62026-1656
hfujino@siue.edu

Alexander Towell
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL, USA 62026-1656
atowell@siue.edu

Vamshi Anirudh Thota
Department of Computer Science
Southern Illinois University Edwardsville
Edwardsville, IL, USA 62026-1656
vthota@siue.edu

***Abstract*—This paper proposes a backup-based defense mechanism that transparently and continuously secures production data against ransomware while ensuring controlled growth of backup copies. Ransomware attacks continue to intensify, often bypassing conventional static or dynamic detection-based security measures and inflicting irreparable harm on critical organizational data. Our approach focuses on a "just-in-time" backup strategy—termed In-Operation Off-Site Backups—that interposes continuous and verifiable file duplication at each update, combined with a detection mechanism that can halt malicious encryption attempts as soon as they are discovered. The solution leverages Bloom filters and carefully managed linked-list backups to maintain a low false-negative miss detection rate on the order of 10^{-8}, providing extremely high confidence that unauthorized data modifications will be detected before damage spreads. By employing fake fields, locality-aware thresholds, and fine-tuned probabilistic data structures, the system discriminates malicious activities from legitimate ones, preventing denial-of-service threats associated with frequent backups. Simulation results demonstrate that this approach can achieve highly reliable detection and sustainable storage utilization. The proposed method fills a crucial gap in ransomware protection by ensuring that backups remain both secure and manageable, thereby mitigating the risk of catastrophic data loss.**

Keywords*—*ransomware, offsite data backups, malware detection, Bloom filter, security risk management

I. Introduction

Despite significant effort by security administrators, ransomware attacks show no clear signs of slowing down, and attackers' demands are increasing for recent targets. Based on recent trends, we focus on computer systems owned by organizations rather than personal users. Although three types of ransomware—lockers, exposures, and encrypters—have been reported [1], this work focuses on encrypters, as they are the most common form today. Specifically, we focus on protecting production data (i.e., document files) from malicious encryption as well as other forms of malicious modifications, such as data obfuscation and data shuffling [2, 3, 4].

The primary reasons for the difficulties in preventing ransomware are twofold. First, the vast attack surface—including viruses, URL forgeries in web-based applications, system vulnerabilities, and threats from insiders (both intentional and unintentional)—makes it challenging to secure all potential entry points [5]. Second, it is difficult to prevent damages using existing proactive solutions [6], such as virus scanners, anomaly detection systems, and firewalls. Although these proactive solutions have positive effects, the challenge is that preventing damage from ransomware allows no time margin between when the ransomware becomes active and when it is detected. Although many existing methods for ransomware detections have been proposed [7-16], none of them satisfies all of the following requirements: ① a solution without depending on metrics that are under controls of ransom attackers (such as malware signatures, anomy detections, and entropy values after encryption), ② effectiveness to obfuscation ransomwares, ③ protections against zero-day ransomwares, and ④ a control over false negatives.

A straightforward approach would be to back up production data each time any update is made. Although this approach logically offers a safe solution, those backup-based solutions require detection of ransomware activities (otherwise they can be a cause of denial-of-service attacks [17]). We propose a backup-based protection using a detection metric out of attackers' control (i.e., reference counts by distinct users). We propose a method that avoids uncontrolled growth of backup copies by (i) unambiguously detecting malicious updates using fake fields and (ii) evaluating the trustworthiness of the record's contents using reference counts without depending on existing detection methods. While our work should be extended to files with any internal structure, this paper assumes only those files that have a block structure, where each block is implemented as a record consisting of fields (typical database files).

II. Existing Work

Many of the existing backup-based solutions consist of two functional components: ① detections of ransomware activities and ② recovering the last known good production data from backup copies after a detection of ransomware activities. Park proposed a solution using entropy value in production data for detection of ransomware activities [18]. However, as Chen argued, the entropy of production data will remain low if the production data is obfuscated [4]. Gomez-Hernandez proposed detections using honeypots [19]. Detections using honeypots have two weaknesses: delay before detections and high false negatives ($\cong 10^{-2}$). Mir [20] and Chen [4] applied machine-learning to ransomware detections, which share the same weaknesses for [19]. It is hard to achieve low false negatives ($< 10^{-6}$) especially to zero-day ransomwares. Continella proposed a detection method using anomaly detection by detecting deviations from the expected legitimate file I/O patterns [21]. The major risk is that ransom attackers will counter-detect the expected legitimate file I/O patterns to evade detections. The backup-based solutions without ransomware activities [22, 23] will not be a reliable solution, which may result in a huge volume of backups, as well as their incapability of identifying the last known good copy of production data. TABLE 1 summarizes the limitations of the existing solutions.

TABLE 1. LIMITATIONS OF EXISTING BACKUP-BASED SOLUTIONS

Proposed by	Primary Detection	Issue(s)
Chen et. al. [4]	sandbox	delay before detections, high false postives
Park et. al. [18]	entropy	not effective to obfuscation ransomwares
Gomez-Hernandez [19]	honeypots	delay before detections, high false postives
Mir [20], Chen [4]	machine learning	not effective to zero-day ransomwares, high false negatives
Continella et. al. [21]	anomaly detection	not effective to zero-day ransomwares, high false negatives
Jin [23], Vrable [24]	no detection	a large volume of backup-copies, hard to retrieve last good data

III. PROPOSED SOLUTIONS

A new solution is designed for coping with the four limitations in the existing solutions: ① without depending on metrics that are under controls of ransom attackers, ② effectiveness to obfuscation ransomwares, ③ protections against zero-day ransomwares, and ④ a control over false negatives. The proposed solution consists of both hardware and software components. Its hardware components include production host computers (denoted production hosts) and one or more backup servers. Production hosts are where users create production data, which is stored on the backup servers as files. Users issue commands to backup servers to read, update, add, or delete their data.

The software components consist of a set of secure system calls installed at each production host and a set of the processes executed at backup servers. These processes manage files and their backup copies as a linked list for each record in each backed-up file. We denote the set of processes at a backup server as Server-Side Implementation (SSI). SSI consists of processes at backup servers that monitor commands from users to detect malicious modifications by ransomware, halt their file accesses to prevent damage, and dynamically prune unnecessary backup copies. Fig. 1 depicts the overall structure of the proposed backup-based solution.

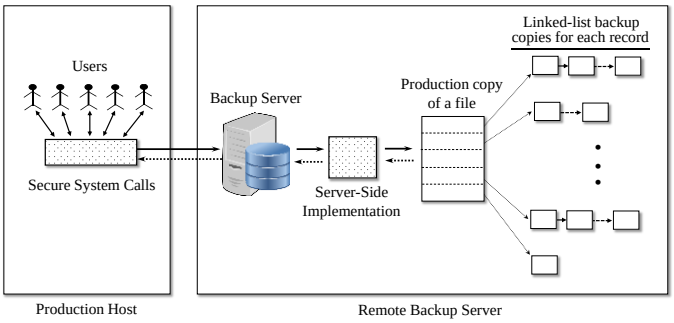

Fig 1. Overview of the system architecture of the backup-based solution.

A. Hardware Component

The two primary hardware components are:

- Production Hosts: Computers where users interact with applications that generate data and use (read, update, add, and delete) the data as files. A set of new system calls is installed on each production host as part of the operating system by a qualified security administrator, ensuring users cannot install a bogus set of system calls or tamper with them once installed.
- Backup Server: A remote file server that maintains the files created by users and their backup copies. It creates backup copies of production files and investigates file I/O commands from users to (1) detect malicious modifications by ransomware and prevent further damage, and (2) purge unnecessary backup copies to prevent uncontrolled growth of backups.

B. Software Components

- Secure System Calls: Each time a user at a production host accesses a record in a file stored at a backup server, the secure system calls transmit file I/O commands to a backup server.
- Server-Side Implementation (SSI): Upon receiving a file I/O command from a user, the SSI at a backup server investigates the command, consisting of three possible outcomes:

  (a) If SSI confirms the legitimacy of the contents of the record using reference counts, it purges all backup copies of the record.

  (b) If SSI confirms that the command is ransomware activity by using fake record fields, it freezes the entire file system to prevent further damage to any files stored on the backup server.

  (c) If SSI can neither confirm if the command is ransomware activity nor if the contents of the record is legitimate, it attaches the modified record to the end of the linked list of its backup copies.

C. Proposed Procedure

SSI employs two methods for detecting ransomware activity:

1. **Fake Record Fields**: Dummy fields inserted into each record by SSI to help identify ransomware activity. Legitimate users are not expected to interact with fake fields, while ransomware may attempt to read, modify, or encrypt them without their knowledge about the fake fields.
2. **Reference Counts**: SSI tracks four reference counts for evaluating the trustworthiness of the record's contents or if backed-up copies of a deleted record can be permanently purged.

   - $C_{\text{RECORD-READ}(i,j)}$: The distinct user count of users who read record *i* in file *j*. Initialized as 0 when a new file is created or a new record is inserted into a file.
   - $C_{\text{FILE-READ}(j)}$: The distinct user count of users who read any record in file *j*. Initialized at 0 when a new file is created.
   - $C_{\text{FILE-UPDATE}(j)}$: The distinct user count of users who update any record in file *j*. Initialized at 0.
   - $C_{\text{DELETED-RECORD}(i,j)}$: The distinct user count of users who would have read or updated record *i* in file *j after the record is deleted*. Initialized at 0 when a record or a file is deleted.

In addition to the four reference counts, SSI attaches $U_{\text{UPDATE}(i,j)}$ to record *i* in file *j*. It is initialized as an invalid user ID, such as "-1", when a new record *i* is added to file *j* or file *j* is created.

- $U_{UPDATE(i, j)}$: The ID of the user (an unsigned integer) who made the last update to record *i* in file *j*.

Detection by Fake Fields

Backup servers insert fake fields into each record when a new file is created. Each fake field is a dummy field whose content is automatically generated by the backup servers, which should be deceptive enough to prevent attackers from distinguishing them from production fields. Only the backup servers can distinguish fake fields from production fields. See Fig. 2 for visualizations of fake fields.

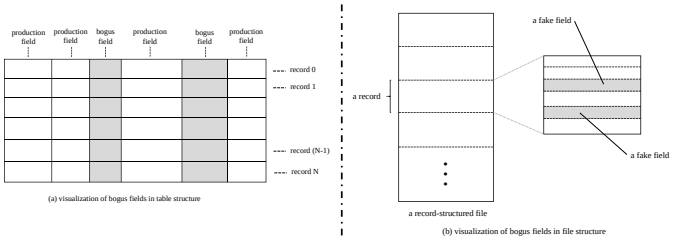

Fig 2. Visualizations of fake fields. Left (a) is the record structure and right (b) is the file structure.

As a proactive measure, SSI employs fake fields to flag suspicious activity. We consider two types of commands: *read* and *update*. We define an update command to be one that replaces the content in one or more fields in a record. When SSI receives a read or a update command, it executes the following procedure for identifying suspicious activities using fake fields in each record. If a read or update operating results in either case (a) or (c), the ransomware detections by reference counts start.

(a) If the read or the update command is applied only to non-fake fields, SSI proceeds to detection using the reference counts. For example, SELECT * FROM table WHERE field = <non-fake-field>.

(b) If the read or the update command is applied only to fake fields, SSI raises an attack alert and halts the system. For example, SELECT * FROM table WHERE field = <fake-field>.

(c) If the read or the update command is applied to both fake and non-fake fields (e.g., using SELECT * FROM table), SSI proceeds to detection using the reference counts.

Fig 3. READ/UPDATE command procedure.

Detection by Reference Counts

When SSI cannot determine the legitimacy of an update record or detect ransomware activities using fake fields, it proceeds to the second phase. To differentiate ransomware activity from legitimate user actions, we introduce reference counts along with threshold values that characterize *normal* behavior. When a new file is created, SSI creates a backup copy of each record and attaches the first three reference counts (i.e., $C_{RECORD\text{-}READ(i, j)}$, $C_{FILE\text{-}READ(j)}$, and $C_{FILE\text{-}UPDATE(j)}$) to a record or a file after they are initialized. When a record or a file is deleted, $C_{DELETED\text{-}RECORD\ (i, j)}$ is attached to each deleted record.

SSI uses the following Bloom filters to identify distinct users.

- $BF_{RECORD\text{-}READ(i, j)}$: A Bloom filter that approximates the set of user IDs who read record *i* in file *j*. Initialized at ∅ when a new file is created or a new record is inserted into a file.
- $BF_{FILE\text{-}READ(j)}$: A Bloom filter that approximates the set of user IDs who read at least one record in file *j*. Initialized at ∅ when a new file is created.
- $BF_{FILE\text{-}UPDATE(j)}$: A Bloom filter that approximates the set of user IDs who updated at least one record in file *j*. Initialized at ∅.
- $BF_{DELETED\text{-}RECORD\ (i, j)}$: A Bloom filter that approximates the set of user IDs who would have read or updated record *i* in file *j* *after* the record is deleted. Initialized at ∅ when a record is inserted into a file or a file is deleted.

To detect ransomware activity that deviates from expected data access patterns, SSI uses the following parameters:

k: The minimum number of distinct users who read the contents of a record to ensure it has not been maliciously updated, obfuscated, or shuffled by ransomware.

w: The number of timeslots (T_{SLOT}) considered in the sliding window for recording the IDs of users who have updated at least one record in a file.

q: The ratio the number of the files in which at least one record is updated by a user to the total number of the existing files, calculated over the most recent *w* timeslots ($[T_{SLOT}]_w$).

y: The threshold determining when a deleted record can be permanently purged. SSI retains deleted records even after they are "deleted" from a production file, storing them in a linked list of deleted records. A deleted record is purged when a certain number (*y*) of legitimate users *implicitly* agree to its removal.

T_{SLOT}: A fixed time interval during which the IDs of users who have updated at least one record in a file are recorded. To detect deviations from temporal locality, we use a sliding window approach with a window size of *w*. The *w* most recent timeslots is denoted by $[T_{SLOT}]_w$. Since *w* is a fixed parameter, we do not need to retain the entire history of timeslots, and so may truncate it to the most recent *w* timeslots for efficiency.

$N_{FILES(s)}$: An integer counting the number of files updated by user *s* during the most recent *w* timeslots $[T_{SLOT}]_w$.

U_{MAX_FILES}: An integer representing the largest number of files updated by a user during the most recent *w* timeslots ($[T_{SLOT}]_w$).

Procedures Executed by SSI

The following procedures are executed by SSI for detecting ransomware activity. When a new record *i* is added to file *j* by user ID *s*, SSI executes the procedure shown in Fig. 4. When SSI receives a read command from user *s* to record *i* in a file *j*, SSI executes the procedure shown in Fig. 5. When SSI receives a

command for updating record *i* in file *j* by user *s*, SSI performs the procedure shown in Fig. 6.

1. SSI inserts *s* to $BF_{RECORD\text{-}READ(i,\ j)}$ and $BF_{FILE\text{-}READ(j)}$.
2. SSI sets both $C_{RECORD\text{-}READ(i,\ j)}$ and $C_{FILE\text{-}READ(j)}$ to 0. Then, SSI proceeds to 3.
3. SSI adds record *i* to file *j*. SSI terminates the procedure.

Fig 4. ADD command procedure.

1. If ($s = U_{UPDATE(i,j)}$), SSI terminates the procedure. Otherwise, proceed to 2.
2. If ($s \notin BF_{RECORD\_READ(i,j)}$), SSI increments $C_{RECORD\_READ(i,j)}$ by one (+1) and inserts *s* to $BF_{RECORD\_READ(i,j)}$. SSI proceeds to 3.
3. If ($s \notin BF_{FILE\_READ(j)}$)), SSI increments $C_{FILE\_READ(j)}$ by one and inserts *s* to $BF_{FILE\_READ(j)}$. SSI proceeds to 4.
4. If ($C_{RECORD\_READ(i,j)} < k$), SSI terminates this procedure. Otherwise, proceed to 5.
5. SSI performs the following tasks:
   i. Delete all backup copies of record *i* in file *j*.
   ii. Reset $BF_{RECORD\text{-}READ(i,\ j)} = \varnothing$ and $C_{RECORD\text{-}READ(i,\ j)} = 0$.
   iii. Terminate the procedure.

Fig 5. READ command procedure.

1. If $s = U_{UPDATE(i,\ j)}$, then the process terminates. Otherwise, proceed to 2.
2. SSI performs the following tasks in order:
   i. Append the current record (one before that update is applied) at the end of the backup linked list of record *i* in file *j*, and then apply the update(s) to the current record.
   ii. Set $s = U_{UPDATE(i,\ j)}$ to the current record.
   iii. Insert *s* to $BF_{FILE\text{-}UPDATE(j)}$. Then, proceed to 3.
3. Calculate the number of the files updated by user *s*, during the *w* most recent timeslots by scanning $BF_{FILE\text{-}UPDATE(k)}$ for each file in the system. Save the total number of the files updated by user *s* to $N_{FILE(s)}$. Proceed to 4.
4. Identify the IDs of the users who updated the largest number of files during the most recent timeslots as follows. Then proceed to 5.
$$U_{MAX\_FILES} = max\left(N_{FILE(s)}\right).$$
5. If $U_{MAX\_FILES} < \frac{n}{q}$, where n is the total number of the existing files in the system, then SSI terminates the procedure. Otherwise, proceed to 6.
6. SSI freezes the whole system by declaring an ongoing ransomware. Then, SSI terminates the procedure.

Fig 6. UPDATE command procedure.

1. SSI sets $s = U_{UPDATE(i,\ j)}$, resets $BF_{RECORD\text{-}READ(i,\ j)}$ to $\varnothing$, and resets $C_{RECORD\text{-}READ(i,\ j)} = 0$. Then, SSI proceeds to 2.
2. SSI appends record *i* at the end of the linked list of the deleted records. Then, it proceeds to 3.
3. SSI deletes record *i* from file *j*. SSI terminates the procedure.

Fig 7. On-command procedure for DELETE command

1. When another user *u* applies either read or update command to record *i* in file *j*, SSI scans the linked list of the deleted records to identify the records (i.e., record ID *i* and the file ID *j*) that match with the record specifiers (e.g. WHERE field for queries). If a matching record is not found in the linked list of the deleted records, SSI terminates post-command procedure. Otherwise, SSI proceeds to 2.
2. For record (*i*, *j*) found in the linked list of the deleted records, SSI checks if its $U_{UPDATE(i,\ j)} = u$. If it is, SSI terminates the post-command procedure. Otherwise SSI proceeds to 3.
3. SSI checks if *u* is in $BF_{RECORD\text{-}READ(i,\ j)}$. If it is, SSI terminates the post-command procedure. Otherwise, it increments $C_{RECORD\text{-}READ(i,\ j)}$ by one. SSI proceeds to 4.
4. SSI checks if the number of the distinct users who would have referenced a deleted record ($C_{RECORD\text{-}READ(i,\ j)}$) exceeds a threshold, *y*, by checking the condition
$$\textbf{\textit{if}}\ (C_{RECORD\text{-}READ(i,\ j)} \geq y)$$
If the count does not exceed the threshold value (*y*), SSI terminates the procedure. Otherwise, SSI proceeds to 5.
5. SSI permanently purges the *deleted record* (record (*i*, *j*)) from the linked list of the deleted records. Then, SSI terminates the post-command procedure.

Fig 8. Post-command procedure for DELETE

In Fig. 6, the last four steps in the **Update Command** (steps 3-6, Fig. 6)) are for detecting malicious modifications by ransomware based on the expected locality in data references. Essentially, if a user modifies a large number of records in a large number of files (deviations from the spatial locality) in a short amount of time (deviations from temporal localities), a set of such modifications should be flagged as suspicious. As the Update Command shows, parameter *q* determines the sensitivity in the spatial locality and parameter *w* determines the sensitivity in the temporal locality.

The procedure for deleting an existing record in a file consists of the **on-command procedure**, which SSI performs when it receives a command to delete existing record *i* in file *j* and the **post-command procedure**, which SSI performs after SSI *deletes* an existing record from the production copy of a file.

SSI executes the following procedure before SSI deletes record *i* in file *j* by user *s*. The on-command procedure is as

shown in Fig. 7. Fig. 8 shows the procedure SSI executes *after* record *i* is deleted from file *j*.

IV. PERFORMANCE EVALUATIONS

We evaluate the performance of the proposed offsite backup mechanism. We provide two analyses to assess its effectiveness.

Analysis 1: False negative miss-setection rate and the required reference counts by distinct users (λ)

In this analysis, we evaluate the effectiveness of the offsite backup by assessing the false negative miss detection rate, P_{miss} which quantifies the probability that all users fail to detect a malicious update in any file.

Frequency of Occurrence			Degree of Harm				
Class-5	Frequent	$10^{-3} < R$	C6	B3	A1	A2	A3
Class-4	Probable	$10^{-3} \le R < 10^{-4}$	C5	B2	B3	A1	A2
Class-3	Occasional	$10^{-4} \le R < 10^{-5}$	C4	B1	B2	B3	A1
Class-2	Remote	$10^{-5} \le R < 10^{-6}$	C3	C4	B1	B2	B3
Class-1	Improbable	$10^{-6} \le R < 10^{-7}$	C2	C3	C4	B1	B2
Class-0	Incredible	$R \le 10^{-8}$	C1	C2	C3	C4	C5
			None	Negligible	Marginal	Critical	Catastrophic
			Category-0	Category-I	Category-II	Category-III	Category-IV

Fig. 9 the R-map matrix for risk management.

We adopt the R-map matrix as the benchmark for safety levels in preventing ransomware damage [26] (Fig. 9). For example, a false negative miss detection rate $P_{miss} = 10^{-6}$ means the system fails to detect one out of 10^6 ransomware attempts.

We compute λ for different values of p and various target false negative miss detection rate (P_{miss}) with w fixed at 10 and N (N: the number of files in a system) fixed at 10 (Fig. 10). We see that as p increases, the required λ decreases for a given P_{miss}. At $p = 0.01$, achieving $P_{miss} = 10^{-8}$ requires 19 distinct users in each timeslot, while achieving $P_{miss} = 10^{-4}$ requires 16 distinct users. As p increases, the required λ decreases.

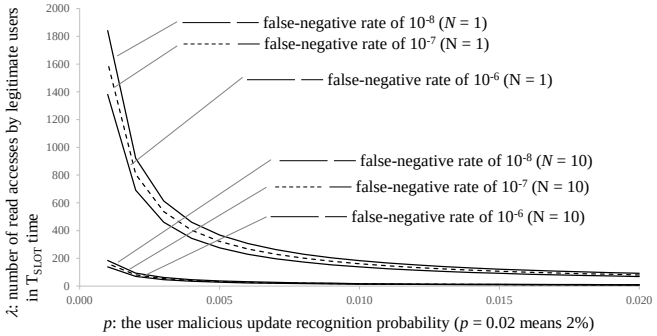

Fig. 10 Expected number of distinct users λ for achieving different target false negative detection rates P_{miss} or different user malicious update recognition probabilities p.

Fig. 11 shows the expected number of user references (λ) to achieve a particular miss rate (P_{miss}) when we reduce N to 1. We see that the number of files significantly impacts the expected λ, especially when p is small (e.g., $p = 0.001$). This observation suggests that the proposed backup mechanism is more effective in systems with a larger number of files.

With $p = 0.001$ and λ = 10, Fig. 12 shows how P_{miss} decreases as w increases. For $N = 1000$, P_{miss} drops below 10^{-8} at $w = 5$, while for smaller N, achieving the same P_{miss} requires larger w. These results suggest that the proposed backup mechanism is well-suited for organizational systems with a reasonable number of files and frequent read accesses.

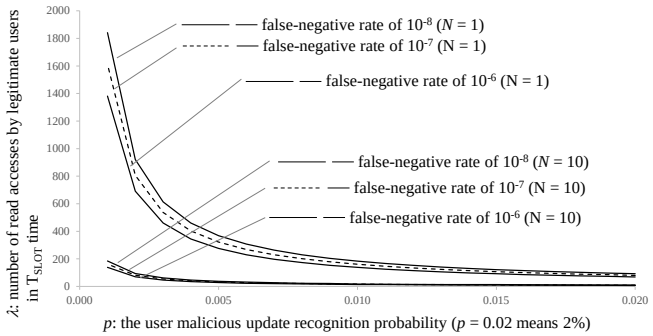

Fig. 11 Expected number of distinct users λ when N = 1 and all other parameters identical to Fig. 10. As the number of files increases, the expected λ also increases.

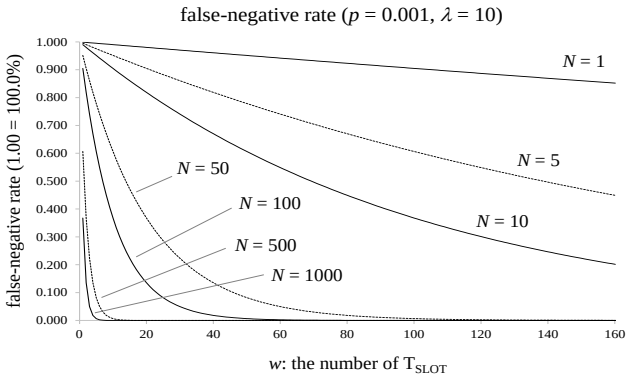

Fig. 12 False negative miss detection rate P_{miss} for $p = 0.001$ and λ =10.

We developed a formula to deduce the expected λ (the reference counts by λ distinct users) for target w, N, and P_{miss}. The following is the process of our deduction.

Theorem 1: Consider a system with N files subject to potential malicious updates by ransomwares, monitored by λ distinct users over w timeslots. Let p, $0 < p < 1$, be the probability that a single user detects a malicious update in a file once it has occurred.

We refer to P_{miss} as the *false negative detection rate*, a key metric for assessing the effectiveness of the offsite backup in preventing ransomware damage.

Proof: By assumption, a user detects a malicious update with probability p, so the probability that this user does not detect it is 1 - p. Assuming independence across all λ users, w timeslots, and N files, the probability that no user detects the malicious update in any file over all timeslots is the product of the individual non-detection probabilities. Thus,

$$P_{\text{miss}} = (1-p)^{\lambda \times w \times N} \quad (1)$$

Next, we derive the expected number of distinct users, λ required to achieve a target false negative detection rate P_{miss}.

Corollary 1: Given a target false negative detection rate P_{miss} and known parameters p, w, and N, the expected number of distinct users λ required to achieve this target rate is:

$$\lambda = \frac{log(P_{\text{miss}})}{log(1-p)\times w\times N} \quad (2)$$

Proof: Starting from Theorem 1, taking the logarithm of both sides yields:

$$ln(P_{\text{miss}}) = log\left((1-p)^{\lambda wN}\right) = \lambda wNlog(1-p) \quad (3)$$

Solving for λ, we obtain:

$$\lambda = \frac{log(P_{\text{miss}})}{wNlog(1-p)} \quad (4)$$

Analysis 2: Bloom Filter Configuration and Performance and its impact to the required reference counts by distinct users

A Bloom filter is a probabilistic data structure often used for membership queries, trading space efficiency for a controlled false-positive rate, denoted ε. For this study, Bloom filters are critical in managing backup data operations, ensuring minimal redundancy while safeguarding against malicious modifications. The key parameters for Bloom filters are:

- Size (m): The number of bits in the Bloom filter.
- Hash Function Count (k): The number of hash functions used to map elements to the Bloom filter.

Impact of Bloom Filter Size (m) and Hash Function count (k)

We limit this analysis to a fixed number of distinct users N = 100 and we estimate the sensitivity to the false positive rate (FPR) with respect to the Bloom filter size m and number of hash functions k. TABLE 2 shows the expected FPR for filter size (m = 256 to 8192 bits) with k = 1, 2, 3, 4, and 5 (Fig. 13).

For a given number of distinct users (λ = 100), the size of the Bloom filter directly impacts the probability of false positives. A Bloom filter size of m = 256 bits results in a large false positive rates:

- 29.4, 32.9, 39.0, and 46.5% for hash function counts k = 2, 3, 4, and 5, respectively.

Doubling the Bloom filter size to m = 512 bits reduces the false positive rate to around 10% (ε ≈ 0.1), while Bloom filter sizes of m = 1024 bits or larger reduce this probability to under 3.2%.

In addition, it is important to know the number of distinct users interacting with the system, denoted λ as before. The probability that the Bloom filter reports a false positive ε for a non-interacting user, is given by:

$$\varepsilon \approx \left(1-e^{-k\lambda/m}\right)^k \quad (5)$$

The optimal k for a given m and λ is given by:

$$k^* = \frac{m}{\lambda} ln2 \quad (6)$$

In practice, λ varies over time or is not known apriori, necessitating a range of k (and m) values to accommodate different scenarios. A larger value of k also increases the computational cost of hashing, potentially impacting system performance. Balancing ε and computational overhead is thus critical.

TABLE 2. EXPECTED FPR FOR FILTER SIZE AND NUMBER OF HASH FUNCTIONS

***m* (bits)**	***k* =2**	***k* =3**	***k* =4**	***k* =5**
256	29.39%	32.88%	39.03%	46.54%
512	10.46%	8.72%	8.64%	9.42%
1,024	3.15%	1.64%	1.09%	0.86%
2,048	0.87%	0.25%	0.10%	0.05%
4,096	0.23%	0.04%	0.01%	0.00%
8,192	0.06%	0.00%	0.00%	0.00%

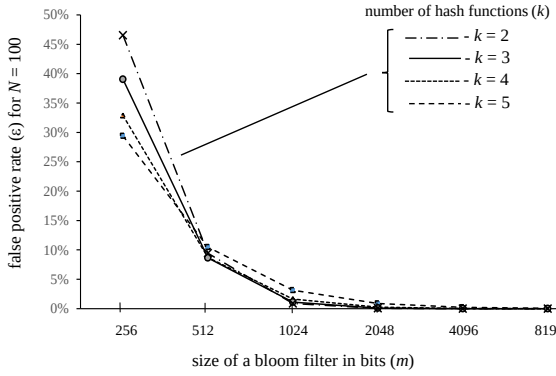

Fig. 13 False positive rate as a function of Bloom filter size m and hash function count k.

Theorem 1: If we report the number of distinct users that test positive in the Bloom filter as λ_{app}, the *expected* number of users that test positive in the Bloom filter is given by:

$$E(\lambda_{app}) = \lambda + \varepsilon \times n \quad (7)$$

a standard deviation:

$$\sigma(\lambda_{app}) = \sqrt{\varepsilon(1-\varepsilon)\times n} \quad (8)$$

a bias is given:

$$bias(\lambda_{app}) = \varepsilon \times n \quad (9)$$

a 95% confidence interval (using the CLT) given by

$$\lambda + n\varepsilon \pm 1.96\sqrt{n\varepsilon(1-\varepsilon)} \quad (10)$$

where n are the number of users not inserted into the Bloom filter, ε is the false positive rate of the Bloom filter, and λ is the number of users inserted into the Bloom filter.

Proof: We have λ positives and n negatives. The Bloom filter has a true positive rate of 1, so all λ positives will test positive. However, the false positive rate ε will cause some of the n negatives to test positive.

To compute the expectation and the variance, we model each of the n negatives as a Bernoulli random variable with probability ε of testing positive. The sum of these n Bernoulli random variables has an expectation of εn and a variance of ε(1-

$\varepsilon)n$. Thus, the expectation of the number of users that test positive in the Bloom filter is given by the number of positives λ plus the expectation of the number negatives n that test positive, which in sum is $\lambda + \varepsilon n$.

The standard deviation is just the square root of the variance, the bias is just the expectation of the number of false positives, and the confidence interval is a direct application of the Central Limit Theorem (CLT).

The accuracy and precision of the $\hat{\lambda}$ is important, as many of our parameters depend on the number of distinct users λ. We see that λ_{app} is a positively biased estimate of λ due to the false positives. As ε increases, the bias in the estimate also increases. We may adjust the estimate of λ by subtracting the bias to obtain a less biased estimate of λ. This is given by:

$$\hat{\lambda} = \lambda_{app} - \varepsilon n \quad (11)$$

which is an unbiased estimator.

Reasonable Configurations

A 64-byte (m = 512 bits) Bloom filter represents a reasonable compromise between space and performance, achieving a false-positive rate of approximately 10%.

Estimating the Number of Distinct Users

Since the integrity of the system depends on accurate estimates of the number of distinct users λ inserted into the Bloom filter, we compute the biased estimate λ_{app} and the unbiased estimate $\hat{\lambda}$. For this configuration, the apparent number of users is overestimated by 10%:

$$\lambda_{app} \approx 90 + 0.1 \times 100 = 110 \quad (12)$$

We apply the bias-correction to obtain the unbiased estimate:

$$\hat{\lambda} = 110 - 0.1 \times 100 = 100 \quad (13)$$

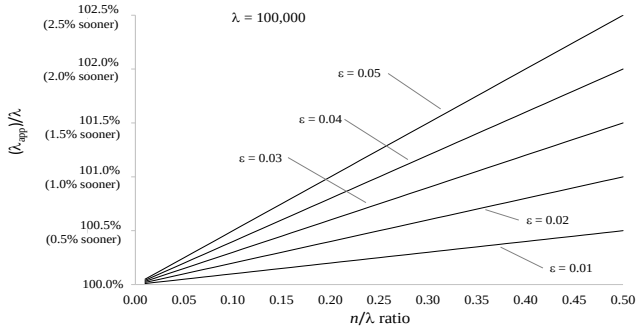

Fig. 14. the predictable impact of the false positives (how much sooner SSI will incorrectly halt a system due to the expected false positives in Bloom filter

Fig.14. visualizes $(\lambda_{app})/\lambda$ ratio for various FPRs (ε) when λ = 100,000. For n/λ ratio = 0.01 (n = 1 and λ = 100), the $(\lambda_{app})/\lambda$ ratio is estimated to be 0.01% (SSI will halt a system 0.01% sooner than it should in terms of reference counts) for ε = 0.01, 0.02% for ε = 0.02, 0.03% for ε = 0.03, 0.04% for ε = 0.04, and 0.05% for ε = 0.05. For n/λ ratio = 0.5 (n = 50 and λ = 100), the $(\lambda_{app})/\lambda$ ratio is estimated to be 0.5% (SSI halts a system 0.5% sooner in terms of reference counts) for ε = 0.01, 1.0% for ε = 0.02, 1.5% for ε = 0.03, 2.0% for ε = 0.04, and 2.5% for ε = 0.05.

These results (as shown by equation (7)) conclude that the impact of the false positives (i.e., the number of distinct users who did not do a read operation for a time slot, but tested positive) can be estimated using a liner regression model ($\varepsilon \times n$). For example, if $((\lambda_{app})/\lambda)$ ratio = 1.01 (i.e., 101.0%) because of the 1% FPR, the SSI will halt a system 1% sooner in terms of user reference count.

Justification for 64-byte Bloom filters

Given the space constraints and operational needs of backup servers, the results demonstrate that:

1. Increasing the Bloom filter size beyond 1024 bits yields diminishing returns, with false-positive probabilities already reduced to under 3.2.
2. A 512-bit Bloom filter offers a balance of accuracy and resource efficiency, with a manageable false-positive rate of around 10%.

Our simulations confirm that this configuration is well-suited for environments where a tradeoff between space overhead and accuracy is acceptable. For instance, maintaining a 10% false-positive rate does not significantly compromise the system's ability to protect production payloads, such as file records.

V. CONCLUSIONS AND FUTURE WORK

This paper introduced a novel backup-based solution, referred to as In-Operation Off-Site Backups, designed to protect critical organizational data against the escalating threats posed by ransomware. Unlike conventional detection-based strategies that rely solely on known signatures or behavior profiles, our method blends continuous, fine-grained versioning of production data with a probabilistic and locality-aware detection mechanism. By leveraging Bloom filters, fake record fields, and dynamically managed linked-list backups, the proposed system achieves a highly reliable false-negative detection rate - on the order of 10^{-8} - while simultaneously limiting backup explosion and preserving system performance.

A key innovation of the approach is its capacity to distinguish genuinely benign user updates from malicious activities attempting to encrypt or obfuscate large amounts of data. The combination of on-command backup creation, reference count monitoring, and strategically placed fake fields ensures that ransomware operations cannot silently propagate. Once anomalous behavior is detected, the system halts further modifications, effectively containing damage at the earliest possible stage. Our results indicate that In-Operation Off-Site Backups can be tuned to maintain extremely low false-negative probabilities without incurring unsustainable storage or computational overhead. For our future work, we currently recognize the following five tasks:

1. **Wider Application Scenarios**:

   While the current design assumes files with a record-based structure (e.g., typical database files), the approach should be extended and adapted to files of varying formats and internal structures. Enhancing the solution to handle diverse

file systems, big data storage formats, unstructured documents, and multimedia files will broaden its applicability.

2. **Adaptive Threshold Tuning**:

   The current system relies on fixed threshold values (e.g., k, w, q, and y) for triggering detection and pruning actions. Future research could explore adaptive, context-sensitive thresholding techniques that automatically adjust parameters based on observed access patterns, seasonal workloads, or emerging ransomware trends.

3. **Integration with Existing Security Frameworks**:

   Although our focus was on preventing ransomware damage at the backup layer, the solution could be integrated with other cybersecurity tools—such as anomaly detectors, intrusion prevention systems, or zero-trust frameworks—to form a more holistic, multi-layered defense.

4. **Performance Enhancements and Real-World Validation**:

   While simulation results demonstrate the feasibility of the proposed mechanism, future work should include prototyping and field trials. Deploying In-Operation Off-Site Backups in live production environments, measuring real-time latency effects, evaluating user experience, and assessing long-term storage impacts will provide valuable insights and refinement opportunities.

5. **Scalability and Distributed Architectures**:

   As organizational data environments grow, so do throughput requirements. Future research could investigate the scalability of the system, focusing on distributed backup servers, load-balancing mechanisms, and fault-tolerant architectures designed to handle larger user populations and global data centers.

In conclusion, In-Operation Off-Site Backups addresses a critical gap in ransomware protection by coupling near real-time, record-level data versioning with robust probabilistic detection methods. By halting malicious encryption attempts before data becomes irreversibly compromised, our solution offers a promising new safeguard for organizations seeking to fortify their defenses against an ever-evolving threat landscape.

REFERENCE

[1] P. O'Kane, S. Sezer, and D. Carlin, "Evolution of ransomware," *Privacy, Data Assurance, Security Solutions for Internet of Things*, vol. 7, no. 5, pp. 321–327, Sep. 2018.

[2] A. Hessler, T. Kakumaru, H. Perrey, and D. Westhoff, "Data obfuscation with network coding," *Computer Communications*, vol. 35, no. 1, pp. 48–61, Jan. 2012.

[3] V. C. Craciun, A. Mogage, and E. Simion, "Trends in design of ransomware viruses," in *Proceedings of the international conference on security for information technology and communications*, Nov. 2018, pp. 259–272.

[4] Zhi-Guo Chen, Ho-Seok Kang, Shang-Nan Yin, Sung-Ryul Kim, "Automatic Ransomware Detection and Analysis Based on Dynamic API Calls Flow Graph," in *Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, pp. 196-201, September 2017.

[5] H. Oz, A. Aris, A. Levi, and A. S. Uluagac, "A survey on ransomware: Evolution, taxonomy, and defense solutions," *ACM Computing Surveys (CSUR)*, Feb. 2021.

[6] A. Ferreira, "Why ransomware needs a human touch," in *Proceedings of international carnahan conference on security technology*, 2018, pp. 1–5.

[7] A. Karnik, S. Goswami, and R. Guha, "Detecting obfuscated viruses using cosine similarity analysis," in *Proceedings of asia international conference on modelling & simulation*, Mar. 2007, pp. 1–6.

[8] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, "SplitScreen: Enabling efficient, distributed malware detection," *Journal of Communication and Networks*, vol. 13, no. 2, pp. 187–200, Apr. 2011.

[9] A. Malhotra and K. Bajaj, "A hybrid pattern based text mining approach for malware detection using DBScan," *CSI Transactions on ICT*, vol. 4, no. 2–4, pp. 141–149, Dec. 2016.

[10] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," *arXiv preprint arXiv:1609.03020*, Sep. 2016.

[11] A. Arabo, R. Dijoux, T. Poulain, and G. Chevalier, "Detecting ransomware using process behavior analysis," *Procedia Computer Science*, vol. 168, pp. 289–296, 2020.

[12] J. Stiborek, T. Pevný, and M. Rehák, "Probabilistic analysis of dynamic malware traces," *Computers & Security*, vol. 74, pp. 221–239, May 2018.

[13] E. P. T. P. and S. G. Yoo, "Detecting and neutralizing encrypting ransomware attacks by using machine-learning techniques: A literature review," *International Journal of Applied Engineering Research*, vol. 12, no. 18, pp. 7902–7911, 2017.

[14] Y. e, L. Chen, S. Hou, W. Hardy, and X. Li, "DeepAM: A heterogeneous deep learning framework for intelligent malware detection," *Knowledge Information Systems*, vol. 54, no. 2, pp. 265–285, Feb. 2018.

[15] M. hode, P. Burnap, and K. Jones, "Early-stage malware prediction using recurrent neural networks," *Computers & Security*, vol. 77, pp. 578–594, Aug. 2018.

[16] S. Kok, A. Abdullah, M. Supramaniam, T. R. Pillai, and I. A. T. Hashem, "A comparison of various machine learning algorithms in a distributed denial of service intrusion," *International Journal of Engineering Research for Techniques*, vol. 12, no. 1, pp. 1–7, 2019.

[17] H. Fujinoki and L. Manukonda, "Proactive damage prevention from zero-day ransomwares," in *International conference on computer communication and the internet*, Aug. 2023.

[18] J. Park, Y. Jung, J. Won, M. Kang, S. Lee, and J. Kim, "RansomBlocker: A low-overhead ransomware-proof SSD," in *Proceedings of ACM/IEEE design automation conference*, 2019, pp. 1–6.

[19] José Antonio Gómez‐Hernández, Raúl Sánchez‐Fernández, and Pedro García-Teodoro, "Inhibiting Crypto-Ransomware on Windows Platform," *IET Information Security*, vol. 16, no. 1, pp 64-74, 2022.

[20] Waqar Hassan Mir, Neeraj Goel, and Venkata Kalyan Tavva, "CARDR: DRAM Cache Assisted Ransomware Detection and Recovery in SSDs," in *Proceedings of the International Symposium on Memory Systems*, pp. 104-115, December 2024.

[21] Andrea Continella, Alessandro Guagnelli, Giovanni Zingaro, Giulio De Pasquale, Alessandro Barenghi, Stefano Zanero, and Federico Maggi, "ShieldFS: a Self-healing, Ransomware-aware File System," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pp. 336 – 347, December 2016.

[22] Yong Jin; Masahiko Tomoishi; Satoshi Matsuura; and Yoshiaki Kitaguchi, "A Secure Container-based Backup Mechanism to Survive Destructive Ransomware Attacks," in *Proceedings of IEEE International Conference on Computing, Networking and Communications*, pp. 1-6, March 2018.

[23] Michael Vrable, S. Savage, and G. Voelker, "BlueSky: a Cloud-Backed File System for the Enterprise, in *Proceedings of USENIX Conference on File and Storage Technologies*, pp. 1-14, February 2012.