future internet

MDPI

Article

Ransomware-Resilient Self-Healing XML Documents

Mahmoud Al-Dwairi [1,†], Ahmed S. Shatnawi [2,*,†], Osama Al-Khaleel [1,†] and Basheer Al-Duwairi [3,†]

[1] Department of Computer Engineering, Jordan University of Science and Technology, P.O. Box 3030, Irbid 22110, Jordan; mndwairi14@cit.just.edu.jo (M.A.-D.); oda@just.edu.jo (O.A.-K.)
[2] Department of Software Engineering, Jordan University of Science and Technology, P.O. Box 3030, Irbid 22110, Jordan
[3] Depatment of Network Engineering & Security, Jordan University of Science and Technology, P.O. Box 3030, Irbid 22110, Jordan; basheer@just.edu.jo
* Correspondence: ahmedshatnawi@just.edu.jo; Tel.: +962-7910-803-57
† These authors contributed equally to this work.

Abstract: In recent years, various platforms have witnessed an unprecedented increase in the number of ransomware attacks targeting hospitals, governments, enterprises, and end-users. The purpose of this is to maliciously encrypt documents and files on infected machines, depriving victims of access to their data, whereupon attackers would seek some sort of a ransom in return for restoring access to the legitimate owners; hence the name. This cybersecurity threat would inherently cause substantial financial losses and time wastage for affected organizations and users. A great deal of research has taken place across academia and around the industry to combat this threat and mitigate its danger. These ongoing endeavors have resulted in several detection and prevention schemas. Nonetheless, these approaches do not cover all possible risks of losing data. In this paper, we address this facet and provide an efficient solution that would ensure an efficient recovery of XML documents from ransomware attacks. This paper proposes a self-healing version-aware ransomware recovery (SH-VARR) framework for XML documents. The proposed framework is based on the novel idea of using the link concept to maintain file versions in a distributed manner while applying access-control mechanisms to protect these versions from being encrypted or deleted. The proposed SH-VARR framework is experimentally evaluated in terms of storage overhead, time requirement, CPU utilization, and memory usage. Results show that the snapshot size increases proportionately with the original size; the time required is less than 120 ms for files that are less than 1 MB in size; and the highest CPU utilization occurs when using the bzip2. Moreover, when the zip and gzip are used, the memory usage is almost fixed (around 6.8 KBs). In contrast, it increases to around 28 KBs when the bzip2 is used.

Keywords: ransomware; XML documents; secure document engineering self-healing

Citation: Al-Dwairi, M.; Shatnawi, A.S.; Al-Khaleel, O.;Al-Duwairi, B. Ransomware-Resilient Self-Healing XML Documents. *Future Internet* **2022**, *14*, 115. https://doi.org/10.3390/fi14040115

Academic Editor: Leandros Maglaras

Received: 12 March 2022
Accepted: 5 April 2022
Published: 7 April 2022

1. Introduction

The progression of cybercrime and the development and adoption of new techniques to jeopardize sensitive information and impart damage across the Internet present an alarming threat to businesses, governments, and nations. Recent cybersecurity research (e.g., the works in [1–6]) confirms cybercriminals' determination to develop newer techniques for achieving their malicious objectives. Ransomware is just one of the methods that have been used recently by cybercriminals to achieve financial gains in return for releasing ransomware-encrypted files to their rightful owners. Ransomware attacks represent a real security threat to users' data files and various network resources that would contain backup files. Amongst others, a conservative estimate is that ransomware criminals received USD 412 million in payments in 2020 [7]. Ransomware attacks impact individuals and organizations in the public and private sectors, including, amongst many, the health sector, e-commerce, educational institutions, government agencies, and the business sectors, in a

manner that leads to economic and moral loss. In 2017, the WannaCry Ransomware [8], a recent massive Ransomware attack, impacted up to 300,000 users in 150 countries worldwide, preventing them from accessing their devices and demanding Bitcoin payments in exchange for unlocking the files involved.

With an ever-increasing rate of storing and sharing data, document security is becoming one of the biggest challenges that faces both individuals and organizations. Here, digital documents are represented in many formats, one of the most popular of which includes the Extensible Markup Language (XML). When Ransomware attacks victims' machines, it will seek to lock or encrypt users' crucial files and documents, including XML-based documents such as ".docx" and ".odt" file types.

Since 2010, the rate of infection by Ransomware has increased significantly. This growing threat has received significant attention from both academia and industry. Many research studies have intensely served to analyze Ransomware and develop new techniques to detect it, as long as it considers backup. However, a significant portion of all proposed detection techniques claims to have a high detection success rate. Nonetheless, most detection and protection systems in use have several limitations.

In this study, we address the problem of recovering XML documents once a ransomware attack has taken place. We propose a self-healing version-aware XML recovery framework to combat Ransomware to achieve this goal. The proposed framework takes advantage of the structure of XML documents and combines link-based version control with well-known access-control mechanisms.

The Version-Control System (VCS) manages all the changes made to documents, including tracking and storing versioning data. In this paper, VCS will be tapped into by presenting a novel approach directed at recovering ransomware-infected XML-based files and documents. Version-Aware XML-based documents are part of a distributed version-control system that does not rely on a central repository but refers to the document file itself in tracking each subsequent version of a document.

The work presented in this paper focuses mainly on protecting XML-based documents such as ".docx" and ".odt" files from being encrypted by Ransomware. The proposed framework integrates decentralized version control that utilizes file links with access-control mechanisms to prevent Ransomware from tampering with the protected file version. Therefore, It ensures complete recovery of protected XML-based documents from ransomware infection. To that end, the main contributions of this work are as follows:

- A self-healing version-aware ransomware recovery framework for XML-based documents is identified.
- The proposed framework is evaluated according to different performance metrics, including storage overhead, CPU utilization, and memory requirements for about 500 XML-based documents of various sizes, ranging from a few kilobytes to 30 Megabytes.

The rest of this paper is organized as follows: Section 2 provides background information on information security, Ransomware, and version-control systems. Section 3 reviews some pieces of related work. Section 4 presents the proposed system. The performance evaluation part is presented in Section 5. Finally, we conclude in Section 6.

2. Background

The field of Information Security is one of the most critical fields in the IT world. Ensuring the protection of information assets is a top priority for users and organizations because the data stored on a computer are certainly worth more than the computer itself. Cybersecurity's critical goal is to protect data transferred over the network and its connected resources against any security threat. There are three main objectives for information security that are deemed primary pillars of cybersecurity. These pillars are Confidentiality, Integrity, and Availability; otherwise referred to as the Security Requirements Triad [9] or the CIA triangle. These three objectives are highly recognized across the security-concerned communities. Confidentiality means that the information is accessed only by authorized parties with sufficient privileges. It guarantees privacy, meaning that the individuals control

what information is related to them, who can collect such information, and to whom a set of given data can be revealed. Integrity guarantees that the data stored on computers and other resources are correct and that either unauthorized people or malware do not manipulate pieces of data. It is more critical than availability and confidentiality. On the other hand, availability ensures connectivity for authorized users of network resources.

Two additional objectives are sometimes added to these pillars: Authenticity and Accountability. The extended model is known as the CIA+ model, as elaborated in [10]. Authenticity ensures that the message received is the same as the one sent without alteration or tampering; it ensures that it was sent from trusted sources; something that warrants truthfulness of origins. Accountability is related to the individual or organization's responsibility to trace the actions performed on their systems and perform preventive and defensive measures to counter these threats. This includes taking backup for essential data, instating fault isolation, ensuring proper intrusion detection and prevention, conducting after-action recovery, and taking legal action.

2.1. Ransomware

Ransomware is defined as a form of malware that prevents users from accessing their resources and files either by encryption or blockage until a ransom is rendered to restore access to infected files. It provides a means for money-based extortion that affects both individuals and organizations [11]. It is a piece of software designed and implemented by cybercriminals to gain access to legitimate users without their knowledge and to perform malicious activities such stealing sensitive data and asking for a ransom. Due to a lack of proper technical background with little knowledge of how to preserve their data, short of making necessary file backups, some users, especially naive ones, end up paying ransom to restore access to their files. This ultimately leads cybercriminals and attackers to gain more significant revenues and helps to make this an opportunity for thriving businesses [12].

In 1989, the first ransomware attack was reported when infected floppy disks with AIDS Trojan were distributed amongst biologists. The malware encrypted all the victims' system files with a ransom of USD 189 to undo the damage. The earliest variants of Ransomware were developed in 1980 [13]. Ransom was paid via postal mail. Today, ransomware authors order that payment is rendered via credit cards or cryptocurrency such as bitcoin [14].

In recent years there has been an increasing proliferation rate of different types of ransomware families that are spread like a worm, which involve advanced recovery-prevention schemes. This impacts home users, organizations, and the infrastructures of vital governmental establishments around the world [11].

WannaCry and Petaya [8] are examples of recent Ransomware which spreads through insecure compromised websites, exploiting weaknesses inherent in Microsoft Windows. On 12 May 2017, WannaCry was first observed as part of massive attacks over multiple countries [15]. These attacks affected many vital sectors, including government organizations and the healthcare and telecommunications sectors. WannaCry is an example of crypto Ransomware that is based on public-key cryptography; something that is rather challenging to mitigate or recover from, as the encryption keys are stored on a remote command and control server (C&C). In the following subsections, we explain the ransomware lifecycle and main ransomware categories:

2.1.1. Ransomware Lifecycle

The authors of [16] analyzed 25 ransomware families and found that they all possess similar dynamics. They differ somewhat, however, according to the ransomware versions in place, but exhibit a similar overall high-level pattern. In general, the ransomware lifecycle spans the following six steps [16]:

- Ransomware distribution: Like other malicious software programs, Ransomware uses social-engineering strategies to seduce victims to click links that lead to ridiculous content or download a malicious dropper or payload that causes infection.

- Infection: The malicious code is downloaded at this stage, and the execution of the code begins. At this stage, a victim's machine will have been compromised by Ransomware, with the underlying files still not yet encrypted. Encryption is a reversible process, involving highly intensive CPU calculations operations. Encryption does not readily happen in a typical ransomware attack as it requires time for data evaluation by the malware and the scope for data encryption. Once this stage becomes active, all the automatic detection systems will have stopped. The firewall, proxy, antivirus, and intrusion detection programs will have been compromised to allow all malicious communications to take place, ultimately putting the ransomware in total control.
- C2 Communications: The malicious code continues to maintain access to its command-and-control server (C2) at this stage. Here, an attacker manages a C2 server and begins to send commands to the compromised system. The primary C2 communications objective with Ransomware entails the acquisition of an encryption key. Once that is complete, different systems are changed, and persistence is determined.
- File search-scanning: This is when things start to slow down a bit. The malware searches the computer to find files to encrypt first. It also scans for cloud data that are synced through folders and shown as local data. Then it starts searching for file shares. This may take time, depending on how much activity there is across the network. The goal is to examine the available information and determine the victim's level of permissions (e.g., list, published, delete).
- Encryption: The encryption starts once all data have been inventoried. Local file encryption may take minutes, but it may take several hours to encrypt a network file; this is because data on network file shares are locally copied and encrypted in most ransomware attacks. Then this is followed by uploading the encrypted files and removing the original ones. This phase takes a bit of extra time.
- Ransom demand: At this stage, a victim will receive a ransom message instructing them to render ransom; the Ransomware message is issued immediately once encryption has taken place. The Ransomware shows a screen that instructs its victim to pay before criminals delete the key to decrypt the files. The last function usually performed by Ransomware is to end and uninstall itself from a victim's machine. At this point, the hackers are ready to receive the ransom to their Bitcoin wallet.

2.1.2. Ransomware Categories

Ransomware falls under three main categories ranging from severe to damaging: Scareware, Locker Ransomware, and crypto Ransomware. Table 1 summarizes these categories. Scareware is a form of malicious software that overwhelms users' screens with warnings and pop-ups claiming that issues are detected on the users' PC and it requires money to fix them. If the victim falls in for this trick and installs the malware on their machines, the cybercriminal/s would use this malware to access their files, send out fake emails in their names, and/or track their online activity. Locker Ransomware is malicious software that infects the operating system and prevents users from accessing their files and data. It hijacks one or more of the victim's system services, such as desktops, smartphones, and applications, depriving users of those tools from accessing them [11]. This attack usually takes the form of a locking computer interface asking the user to pay a ransom for re-access. Often, infected computers are left with limited capabilities to allow the user to communicate with ransomware and conduct-related activities to pay the requested ransom. For example, W32. Rasith is a worm that locks the victim's desktop, making the system unusable [17]. This type is not limited to PCs or servers alone, but it also affects mobile devices. Android.Lockdroid.H is an example of a trojan that locks the screen of mobile devices and displays a ransom message [17]. Since Locker ransomware is designed to prevent access to the device's interface, the underlying system and files are left untouched. It is possible to restore the computer to a state close to its original condition. Thus, Locker ransomware is less effective at eliciting ransom payments.

Although cryptography is regarded as a critical defense mechanism in computer and network applications [18], it can also be used to perform crypto crimes. The work in [19] is one of the earliest research studies on fraudulent cryptographic use. What distinguishes Ransomware from conventional malware is that it utilizes cryptography techniques, including symmetric and asymmetric key-based encryption, against victims, as discussed in [20]. This type is the most common type of Ransomware. It is the most harmful type and can cause a great deal of damage, thereby extorting vast amounts of money. This type of Ransomware is considered the most dangerous because once the attacker gets hold of the files, there is no way to restore them until a ransom is rendered for file restoration. Here, WannaCry [8] is one famous example.

Crypto ransomware encrypts victims' files, file contents, and file names without notification by utilizing different cryptographic methods and notifies victims that their data have been encrypted, forcing them to pay a ransom to decrypt files [12]. Since 2016, crypto Ransomware attacks have increased dramatically. According to a report by [21], 58.43% of ransomware attacks are conducted by a crypto Ransomware strain called TeslaCrypt. CTB-Locker was considered one of the primary ransomware attacks in 2016. CTB-Locker can attack multiple victims at the same time. Thus, during the same attack, it can extort several victims. This infects web servers by encrypting webroot, causing web servers, host applications, and websites to become paralyzed [21].

Table 1. Ransomware Categories.

Category	Symptoms	Example
Locker	prevents users from accessing their files and data	W32. Rasith Data
Crypto	Encrypts victims' files, file contents, and file names without notification by utilizing different cryptographic methods and notifies victims that their data have been encrypted, forcing them to pay a ransom to decrypt files.	WannaCry
Double extortion	Encrypts files and asks victims to pay a ransom. Attackers threaten to publicize stolen data if their demands are not met.	Maze
RaaS	Involves perpetrators leasing access to ransomware from the ransomware author, who delivers it as a paid service.	Locky

2.2. Version-Control System (VCS)

Version-control systems (VCS) are used to manage all changes made to documents, including tracking and storing version data. In this paper, VCS will be tapped into by presenting a novel approach to recovering XML documents affected when Ransomware attacks victims' machines, causing locking of file encryption. Version-Aware XML-based documents is a distributed version-control system that does not rely on a central repository but refers to the document file to utilize the changes between different versions of the same document. version-control is a system used for tracking all files or file set changes over time to allow for the subsequent release of a specific version of the file so that you can obtain a specific version of the file later. As VCS became popular, new techniques continued to evolve. It uses two main techniques to store versions of data. The first one is to keep a copy of each new version of the file, while the second one would keep only the deltas, which are the data differences between the two versions of the file. There are two major version-control types: centralized and distributed. A centralized version-control system is based on client–server architecture where a central repository is used to store the document versions. Centralized VCS must be used online as it requires the end-user (client) to be connected to the system (central repository) at all times. Using this approach makes it possible to elicit single points of failure [22].

A distributed version-control system, also known as Version-Aware XML document (used in our approach) was first introduced in [22]. In contrast to centralized VCS, version–aware VCS does not depend on a central repository to store versions data. It utilizes reverse deltas stored inside the document file itself, which are the data differences between the two versions of a file, rather than storing the whole document every time. By using Version-Aware XML document technology, users are not worried about the need to use a repository or network connection to remote servers. LibreOffice documents (ODT) are XML schemas that store files, styles, and settings. The authors of [23] created a Custom Microsoft Word plugin to support Version-Aware XML documents technology. Revisions of the document content are stored as a separate copy (snapshot) in a sub-directory inside the document. Shatnawi et al. [24,25] proposed a secure framework for XML documents that improves security for XML documents and their provenance and provides persistent integrity, detects tampering, and provides tools for performing forensics by utilizing version-aware XML document technology. Their approach provides an extensive document history with author signatures at each step, which also enhances the performance when applying security policies applied to documents.

3. Related Work

Cybersecurity researchers have extensively investigated malware attacks over the last few years. In particular, Ransomware has received significant attention among existing research works. Many researchers have studied Ransomware, analyzed its characteristics and properties, and explored how it affects impacted victims. Meanwhile, they have conducted their research work by proposing different approaches to detect and recover from ransomware attacks.

3.1. Ransomware Analysis

To recover from a ransomware attack and mitigate its impacts, we should understand how Ransomware is staged and, in the process, analyze what takes place. Analysis can be achieved by looking at the structure of Ransomware and what it does by invoking a reverse-engineering approach for multiple occurrences. The authors of [26] used reverse engineering to study ransomware samples based on code quality, functionality, and cryptographic primitives, if any. In their study, they concluded that the code is relatively basic for the most part, with high-level languages used in most instances. Both symmetric and asymmetric cryptography were employed. The analyzed samples were mainly purposed to masses, with no specific objects being targeted. While reverse engineering provides an in-depth look inside the structure of Ransomware, it is not considered a cost-effective alternative to performing reverse engineering for every ransomware sample to find a way to prevent attacks due to the complications and overheads involved.

The work in [27] performed a long-term ransomware attack analysis and reports the results of examining over 1300 samples collected between 2006 and 2014 belonging to 15 separate Ransomware families. They show that monitoring the activities in the file system would help with Ransomware detection. They concluded that families of Ransomware share very similar features in their core part, though their implementation differs. The author of [28] conducted their study on malware samples, which is readily valid for Ransomware. They proposed TTAnalyze, which can analyze the behavior of malware that comes as a Windows-executable file process on a virtual processor under an isolated environment. Other researchers were involved in studying the behavior of ransomware families on the network rather than on the local machine. The authors of [29] have, in particular, sought to analyze the network behavior of the CryptoWall Ransomware family. Here, they used HoneyPot technology, which is based on dynamic analysis concepts and an automatic run-time malware analytical system. They completed their study with the conclusion that they could identify infected machines in a dedicated environment and understand ransomware samples' network behavior. Malicious parties commonly associate Ransomware with a particular type of server called Command and Control (C&C)

servers. These are used to automatically control Ransomware and anonymously instruct it on what to do to infect other machines on the network. An approach is presented in [30] to detect communication activities between infected hosts and Command and Control servers by finding communication aggregates from multiple internal hosts that share common characteristics. The authors concluded that three aggregation functions could detect communication based on the hosts' destination, payload, and platform.

Another research effort was conducted in [31] to study how Command and Control servers operate. Instead of detecting communication activities to these servers, the authors proposed a way to make automata that can reveal the hidden specification of closed-type protocols. The solution they created does not require any information upfront, such as source code or specifications about the implementation, and was found to be able to successfully develop automata for FTP traces. The same principle could be applied to C&C servers, which are closed-type protocol automata that send replies to ransomware requests. The work in [32] presents the analysis of 14 strains of ransomware families that infect Windows platforms. This study compares the baseline of standard operating-system behavior operations, and Windows Application Programming Interface (API) calls made through Ransomware processes. This study reports notable features of Ransomware, as indicated by the frequency of API calls, without identifying code signatures within the ransomware code in order to provide a better understanding of what a particular Ransomware does to the system in API calls. The work in [33] applies data-mining techniques to connect components of multi-level code to find unique association rules to classify ransomware families through implementing static or dynamic reverse-engineering processes. The authors carried out this study using 450 ransomware samples in which they were able to identify the strong connection between the different code components that emerged from the experiments.

In [34], the authors examined ransomware attacks in a healthcare setting, duties, and the costs related to such infections as they would affect the healthcare business in general. They also discussed risk-impacts mitigation. They suggested that healthcare facilities should have a disaster plan with appropriate data backups and recovery plans and increase employees' awareness.

3.2. Ransomware Detection

In this section, we discuss the main research efforts for ransomware detection, mitigation and prevention. Detection methods rely on ransomware attack behaviors that affect computer systems such as files or network systems. They give an alarming signal to the end-users to prompt responses towards their files and important data. A SDN-based system that can improve protection against Ransomware by observing the ransomware attack is presented in [29]. By analyzing the behavior of two popular Ransomware, Cryp-toWall and Locky, they could be leveraged to detect Ransomware based on HTTPS messaging sequences and content size based on network-communication observations.

The authors of [35] proposed a Paybreak recovery solution to recover corrupted files on a victim's machine by extracting the encryption keys used to decrypt infected files following a Ransomware attack. PayBreak effectively implements a key escrow mechanism to store session keys in a key vault that can be encoded with a public user key; thus, the user may decrypt the key vault with his private key following ransomware attack. In another research work, Continella proposed ShieldFS in [36]. In this approach, the proposed scheme acts upon the operating and file system levels and serves as a shield to detect and correct any suspicious activities.

Kharraz in [27] carried out a long-term study of ransomware attacks and presents results leveraging analysis of more than 1300 samples collected between 2006 and 2014 that belong to 15 different Ransomware families. Further, the study showed that monitoring activities in the file system would ultimately help with Ransomware detection. R-locker, a general technique intended to prevent crypto Ransomware action, was first introduced by [37]. The researchers used the honeyfile technique to prevent a ransom once it accessed

a trap file. Therefore, the honeyfile technique helps to preserve the data on the system. Moreover, while the ransom is blocked, a countermeasure to eliminate the issue would be beneficial to eradicate the environment's problem.

The study presented in [29] came with the ultimate objective of detecting the underlying Ransomware and mitigating its impact on the systems. The work in [38] provides a signature-based detection approach by observing the original semantics of the dataset of malware. Here, semantics are required to be as effective as malware. However, the authors conclude that malware could be detected commensurate with these signatures at higher error rates with broad classes such as Trojans. In [39], the authors introduced CryptoDrop; an early warning system for ransomware attacks to notify users during any unusual file operation. Based on popular ransomware behavior criteria, the proposed solution tracks victim data and identified Ransomware in the process. Their study conducted experiments on 492 real-world samples of Ransomware, representing 14 families, and was able to achieve high detection rates with low false positives. Ransomware designers continually keep improving their techniques to spread their attacks, especially for Ransomware types that are not easily detected. They use encryption algorithms to hide malicious code within benign code to be executed later.

Shafiqq, Khayam, and Farooq [40] proposed a detection scheme to detect embedded malware, malicious code that is hidden within benign files, using statistical abnormal detection. Yfuksel, den Hartog, and Etalle [41] described a protocol-aware anomaly detection framework that aims to monitor a network from embedded malware access by scanning a network for SBM and Microsoft Remote Procedure Call (RPC) messages. The work presented in [42] studies the whole life cycle of Ransomware creation, design, and implementation using Dynamic Data Exchange (DDE) in Python scripting language and REST APIs in PHP, with the back-end being a MySQL database. Their study aimed to prove that even though many security measures and several top-quality antivirus programs are currently in use, ransomware authors continue to develop and write dangerous malicious codes that can be distributed easily through connected devices. Meanwhile, various research endeavors have widely explored analysis and detection of Ransomware based on its characteristics, leveraging machine learning techniques. In [43], Lim and Ramli applied machine learning techniques to classify extracted static and behavioral analysis, and they developed an efficient malware analysis framework based on the mentioned analysis features addressed thus far.

An approach to efficiently detect Ransomware was presented in [44]. The authors incorporated feature-generation engines and machine learning in a reverse-engineering framework. The purpose of malware code segments is to achieve better examination and interpretation in the proposed framework by performing multilevel analyses such as raw binaries, libraries, function calls, and assembly language. Binaries are decoded to assembly level instructions and DLL libraries using the object-code dump tool (Linux) and portable executable (PE) parser. The experiments were conducted using supervised ML techniques on both Ransomware and normal binaries. Seven of the eight ML classifiers that were tested had a detection rate of at least 90%.

In [45], G. Cusack, O. Michel, and E. Keller proposed a solution using programmable data-transmission from the network-traffic-monitoring engines between the infected computer and command and control server. They derived high-level flow features from this traffic and used this dataset to detect Ransomware. A detection rate of around 0.86 was achieved in this classification model.

While Ransomware is commonly found to infect personal computers rather more frequently, the rapid spread and increased usage of mobile devices and smartphones have often led Ransomware writers and hackers to pay particular attention to this evolving market. Although mobile applications are subject to specific standards by stores before they are made available to end-users, users can still find and download infected applications from these stores. Andronio, Zanero, and Maggi [46] developed a detection scheme based on training ransomware samples called HelDriod. Their approach detects whether a

particular application will attempt to lock or encrypt a mobile device without the user's approval. It can also detect ransom requests from within the text of the application itself.

Stokkel, M. [47] proposed a code using an open-source intrusion detection system called Bro to detect many samples. Alfredo Cuzzocrea, Fabio Martinelli, and Francesco Mercaldo [48] presented a fuzzy logic classification method to identify whether a mobile application exhibits Ransomware behavior; they performed their evaluation based on a dataset containing 10,052 legitimate and illegitimate android mobile applications.

The work presented in [49] proposed a detection method leveraging a Support Vector Machine (SVM). This, inherently, is considered one of a group of supervised algorithms for machine learning. By using this approach, they can identify the API calls logs of Ransomware samples based on their features. These authors evaluated this scheme using 276 real Ransomware samples and they concluded that their technique indeed increases the predictive accuracy and the correct Ransomware detection rate. Ref. [50] conducted a survey on Ransomware Detection Using the Dynamic Analysis and Machine Learning from 2019 to 2021.

3.3. Recovery from Ransomware

This section provides an overview of the literature for recovery from ransomware attacks, the proposed schemes to counter them, and the efficiencies involved. Zimba A, Wang Z, and Simukonda in [51] examined samples from crypto Ransomware through reverse engineering and dynamic analysis to evaluate a Ransomware's underlying attack structures and deletion techniques. They conclude that no matter how disruptive a crypto Ransomware attack is, the key to data recovery is the underlying attack structure and the deletion technique applied. They show that data recovery based on the structure of the attack is possible. The work presented in [52] studies the recovery of lost files due to ransomware attacks in a network-shared volume scenario. It presents a software tool that monitors the traffic and records all user actions on the file. The authors demonstrate that their proposed tool can recover the file from previous and subsequent operations without taking the encrypted content as valid data. This tool, which could recover files successfully, is evaluated based on test-traffic records of 18 different families. The work presented in [53] presents a tool to perform evaluations for Ransomware backup systems during security-risk assessment; this study would make auditors analyze backup systems effectively and improve organizational abilities to detect and recover from Ransomware attacks.

RDS3 is a novel Ransomware Defense Strategy in which it stealthily backs up data in the spare space of a computing device so that the data encrypted by ransomware can be restored [54,55]. Kim et al. [56] proposed a method to decrypt Hive ransomware and recover infected data. Continella et al. [36] described a self-healing, ransomware-aware file system by monitoring low-level filesystem activity. If a process violates a previously trained model, its operations are deemed malicious, and the side-effects on the filesystem are transparently rolled back. The work carried out by Ye et al. [57] suggests monitoring and analyzing operating systems events to ensure that a back up is created whenever a suspicious event is detected. In case the misgiving comes true, it can be rolled back.

4. Proposed Version-Aware Ransomware Recovery Framework

In this section, we describe the proposed framework for Self-Healing Version-Aware Ransomware Recovery (SH-VARR). The main goal of the proposed framework is to serve as a version-control system and assist in recovery against ransomware attacks targeting XML-based documents. To achieve this goal, we implemented a distributed version-control system by adding the absolute URL path of the original file to keep track of file versions. Further, we employed access-control techniques to protect file versions from modification or deletion. These techniques ensure protection from ransomware attacks while allowing users to keep track of older versions of their files. Here, we point out that the novelty of our proposed framework relies on the way we combine well-known techniques from

access-control theory and version-control mechanisms to achieve the desired Self-Healing Version-Aware Ransomware Recovery of XM-based documents.

Figure 1 depicts the overall framework architecture. In this framework, all XML-based documents in a predefined directory go through the version-control module at the time of file closing to maintain the latest version of each document. The access-control module is activated by invoking the root daemon service to perform write protection for the snapshot version, which would be already pointing to the original file.

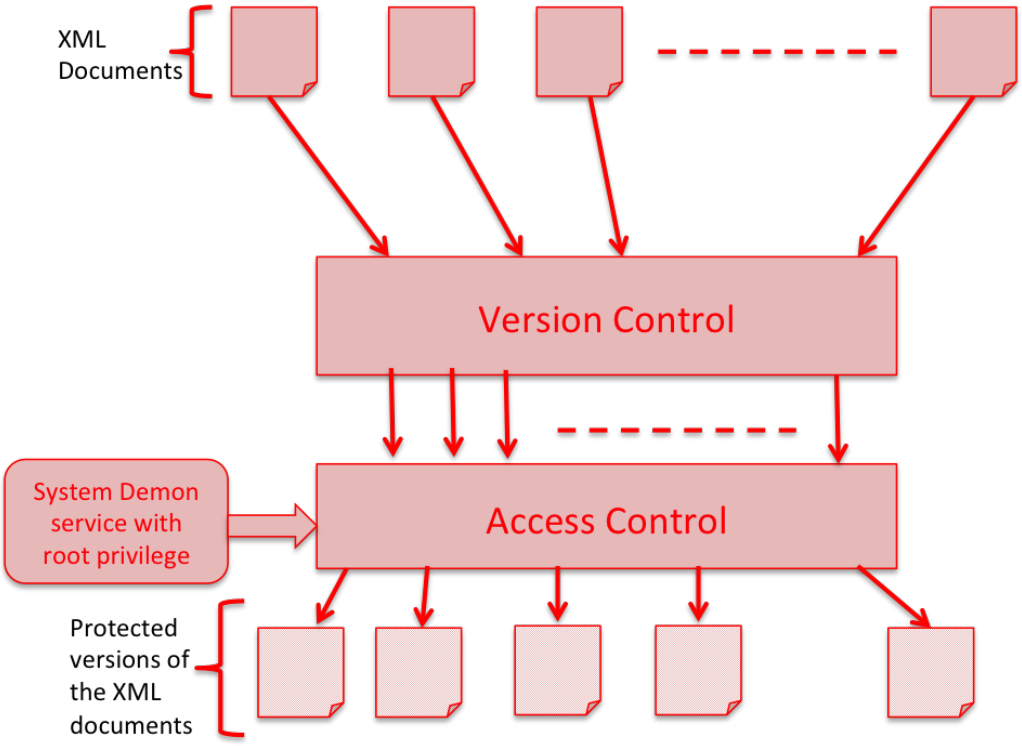

Figure 1. The overall architecture of SH-VARR framework.

4.1. Details of the Proposed SH-VARR Framework

We first describe the version-control module, illustrating the importance of using absolute URL links to keep track of old versions of a file. This is followed by a detailed description of the access-control module.

4.1.1. Version-Control Module

The version-control module is designed to maintain a copy of the XML-based file at the time of file closure so that the latest version can be retrieved in case of any corruption or system failure. We use the term *snapshot* to refer to the resulting file version. This can be achieved by adding a special plugin for Microsoft Word or LibreOffice. As part of this work, we have implemented a custom plugin for Microsoft Word 2013.

Our framework is specifically designed to recover XML-based documents in a predefined folder/directory in case of a ransomware attack. Microsoft documents and LibreOffice documents are XML-based documents that are originally compressed using the `zip` compression algorithm. To create a snapshot of a `.odt` or `.docx` file, the plugin performs the following steps:

- Step 1: Changing the `.odt/.docx` extension of the file to `.zip`.
- Step 2: Extracting the document archive. By unzipping the resulting `.zip` file, we obtain the document structure containing XML-based files and directories generated originally by Microsoft Word or LibreOffice. This includes configurations, meta information, content, settings, etc.

- Step 3: Adding a new XML file (`link.XML`) to the file archive that contains an absolute URL (i.e., a link) of the file version to be created in step 5.
- Step 4: Compressing the resulting ZIP archive, including the `link.XML` file.
- Step 5: Copying the resulting .zip file to a predefined directory that stores the protected versions. Access control permissions are added by the access control module as discussed in Section 4.1.2.
- Step 6: Changing the `.zip` extension of the file to `.odt`.

As an illustrative example, consider Figure 2, which shows the main steps performed by our distributed version-control module to obtain a new version for an XML-based file `abc.odt`. In this example, we assume that the file is in the user directory `/home/user/documents`. The version (i.e., a file snapshot) is created by renaming the file to `abc.zip` and then unzipping the resulting file to obtain the XML file archive. The main reason for performing this step is to add an absolute path (i.e., a link) to the location of the newly introduced version. Assuming that the file version will be stored in: `/home/user/versions` with the name`abc-version1.zip`, then the absolute path `/home/user/versions/abc-version1.zip` will be saved in the `link.XML` file that is added to the document archive in step 3. In step 4, the XML-based document archive is compressed back to obtain `abc.zip`. At this point, the file is copied to the predefined protected versions directory `/home/user/versions`. Finally, the file extension is changed to `.odt`.

Here, note that the version-control module is invoked at the time of closing the document. This ensures that a new snapshot of the XML-based document is saved each time the user closes the file. Here, we emphasize that keeping track of document history (i.e., versions) is achieved by following the absolute path stored in the link.XML file stored in each version. Figure 3 shows the approach used to retrieve older versions. Staring with the newest version (V_N), it is possible to retrieve the preceding version by following the link found in the link.XML file stored in the version itself. Older versions can be retrieved similarly. For recovery from a ransomware attack, it would be sufficient to keep the latest version only. However, suppose the objective was to retrieve older file versions while providing ransomware recovery capability. In that case, the system can be configured to store protected versions in precisely the same way as described in this section.

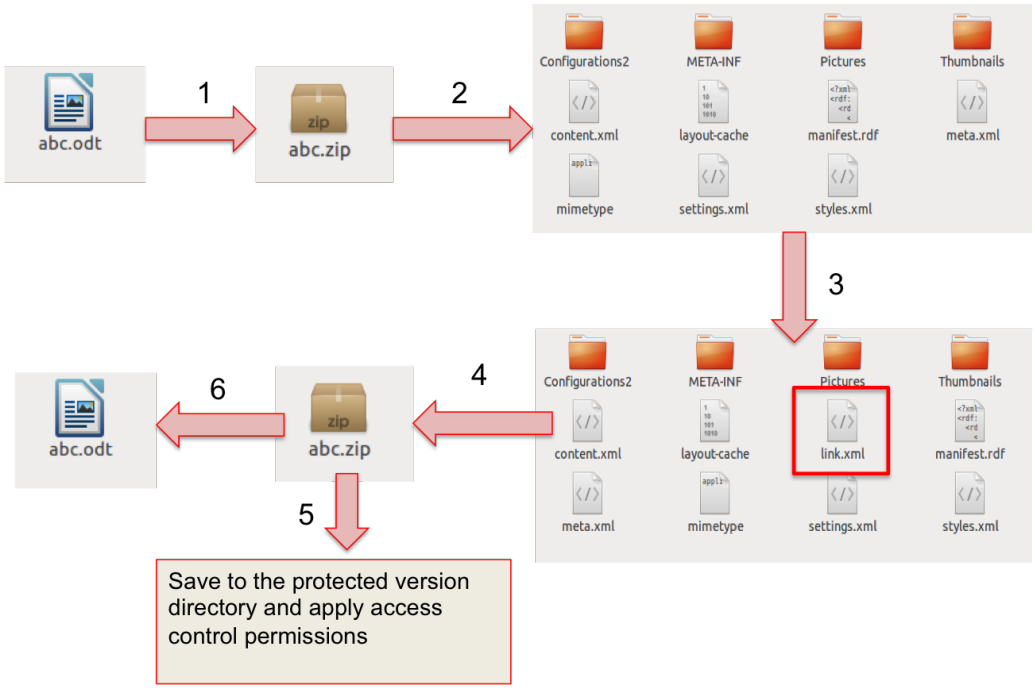

Figure 2. An illustrative example of the main steps of the version-control module.

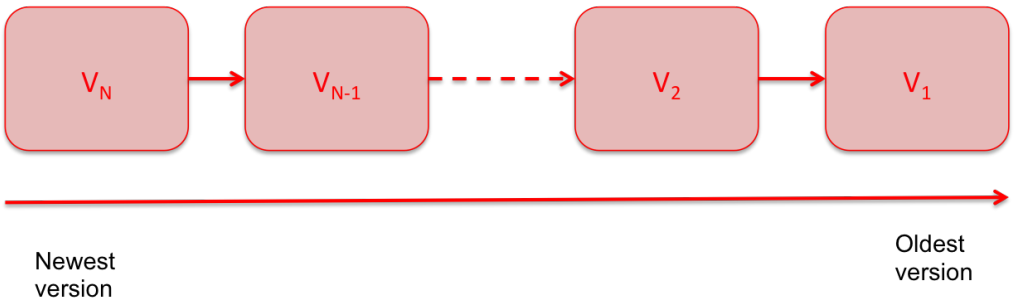

Figure 3. Keeping track of file version history based on link concept.

4.1.2. Access-Control Module

The access-control module is implemented as a root daemon that performs write/delete protection for the files produced by the version-control module each time a file version is created. This is achieved by running the `chattr` command (Change Attribute) with root privileges. `chattr` is a command line in Linux that is used to set/unset specific attributes to a file in a Linux environment to secure accidental deletion or modification of important files and folders, even by root users. Through this process, file snapshots are protected from corruption or deletion by using the change file attribute permissions with the immutable flag (i) under the Linux environment, preventing any user, including the root, from accidentally modifying and/or deleting files. An example using this command is shown in Figure 4.

```
justcb@justcb-virtual-machine:~/Desktop$ sudo chattr +i version1.zip
justcb@justcb-virtual-machine:~/Desktop$ lsattr
------------e-- ./testnew
------------e-- ./mate-terminal.desktop
------------e-- ./DigOutv2.txt
------------e-- ./file.zip
------------e-- ./testnew.zip
------------e-- ./in.txt
------------e-- ./P.class
----i-------e-- ./version1.zip
```

Figure 4. An example using `chattr` command to perform file write/delete protection.

It is important to note that the default setting for standard users is assumed to be non-admins, with the access-control module configured as a system daemon with root access privileges executing the `chattr` command; this would inherently ensure the protection of newly created versions in the version-control directory. Any attempt to modify or delete a protected file will not be permitted, as shown in the example in Figure 5. This is considered a valid setting for two reasons: (i) users usually do not log into their systems as admins. In fact, one of the best practices of computer usage emphasizes that users never log in as admins. (ii) A recent report showed that 90% of ransomware instances in the wild could infect systems and encrypt files without administrative privileges [58]. This indicates that while users log in as non-admins, there is still a high possibility that Ransomware may encrypt their files. In our proposed solution, ensuring a specific access control process with administrative privileges will protect files created/edited by non-admin users.

```
justcb@justcb-virtual-machine:~/Desktop$ rm version1.zip
rm: remove write-protected regular file 'version1.zip'? y
rm: cannot remove 'version1.zip': Operation not permitted
justcb@justcb-virtual-machine:~/Desktop$
```

Figure 5. The file is immutable when trying to write or delete.

4.2. Recovery from Ransomware Attack

The focus of our framework for ransomware recovery is all about maintaining control of the latest possible versions of the files. As the proposed framework preserves protected

versions of the files, we can gain access to the files in case of a ransomware attack. The result of the attack will corrupt the original file or even delete it. However, self-healing is achieved using the proposed SH-VARR framework by retrieving the protected version for each file stored in the version-control directory. In case the original file is deleted or encrypted by Ransomware, our SH-VARR framework allows immediate recovery of the last protected version of the file(s) involved, fulfilling the self-healing property. Based on the proposed framework, the protected snapshots will not be affected and can be recovered under root privileges assumed to be protected. The recovery process is performed by removing the sticky bit attribute to ensure that the file extension is `.odt`. Recovering a file from the protected versions directory is performed as follows:

- Removing the immutable flag (i) attribute. This is achieved by performing the command with root privileges only:
  `$chattr -i file.dot`.
- Changing the file name extension from `.zip` to `.odt` for Linux or `.docx` for a Windows environment.

4.3. Implementation Challenges and Limitations

Throughout this work, we conducted several experiments to ascertain that our goal of keeping a protected version of our XML-based files was achieved. Having set out to build a distributed version-aware control system for XML-based documents that ensures portability that would not depend on a centralized repository, the implemented approach was indeed found to warrant portability as it keeps a link to the original file as described above. During the implementation phase, the system was found to experience certain limitations, which can be summarized as follows:

- The proposed approach assumes a daemon is running with root privileges to keep versions protected.
- Under the Windows environment, and to ensure that our framework was well in place, we implemented a Microsoft office plugin working as a version-control system by keeping a complete snapshot of the active Word document inside the document itself upon document closure. A background process goes through iterations to span all files inside a directory or folder by calling this function. The main challenge here deals primarily with applying the permissions to the created version of each file; this is so because, under a Windows operating system, the read–write operation does not fall under permissions, but file attributes, which will be readily lost after compressing the file archive.

5. Performance Evaluation

In this section, we evaluate the proposed approach in terms of several performance metrics. To conduct our experiments, we use a repository of 500 `.odt` files collected from different sources, with different sizes ranging from 10 KB to 30 MB. All experiments were conducted on a Ubuntu 18.0 machine with a Core i5-1.8 GHz Intel processor and 4 GB RAM. Creating a protected version of each file was achieved by running a shell script that included all the steps outlined in the proposed framework discussed in Section 4. We performed multiple experiments to measure the performance of the proposed SH-VARR framework. SH-VARR uses `zip/unzip` for file compression/decompression as it is the default compression/decompression algorithm used in connection with XML documents. Meanwhile, SH-VARR still has the flexibility of operating with any other compression algorithm. Therefore, different compression algorithms were investigated investigated (`zip`, `gzip`, and `bzip2`) under our experimental set up. In this effort, we evaluate our proposed SH-VARR framework opposite storage overhead, time requirement, CPU utilization, and memory usage.

Creating a protected version of a file (i.e., a snapshot) represents a major step in our framework which results in extra storage requirements. Hence, our objective is to quantify the amount of the resulting storage. This overhead depends mainly on the compression

algorithm used to create the snapshot. Figure 6a–c show how the storage overhead increases with the original file size for the cases when using the `zip`, `gzip`, and `bzip2` compression algorithms. Figure 6d illustrates all cases together for the purpose of comparison. Generally, by increasing the file size, the size of the resulting snapshot increases proportionately. With that said, the size of the resulting file remains smaller than that of the original file. It is quite evident from the comparison that the bzip2-based SH-VARR slightly outperforms the other two versions. However, it consumes more time, as we will discuss next. This would also imply that there is a trade-off between time and storage overhead. Meanwhile, given the lower storage costs involved in today's technologies, the time required to create a protected snapshot may play out as a more pronounced factor.

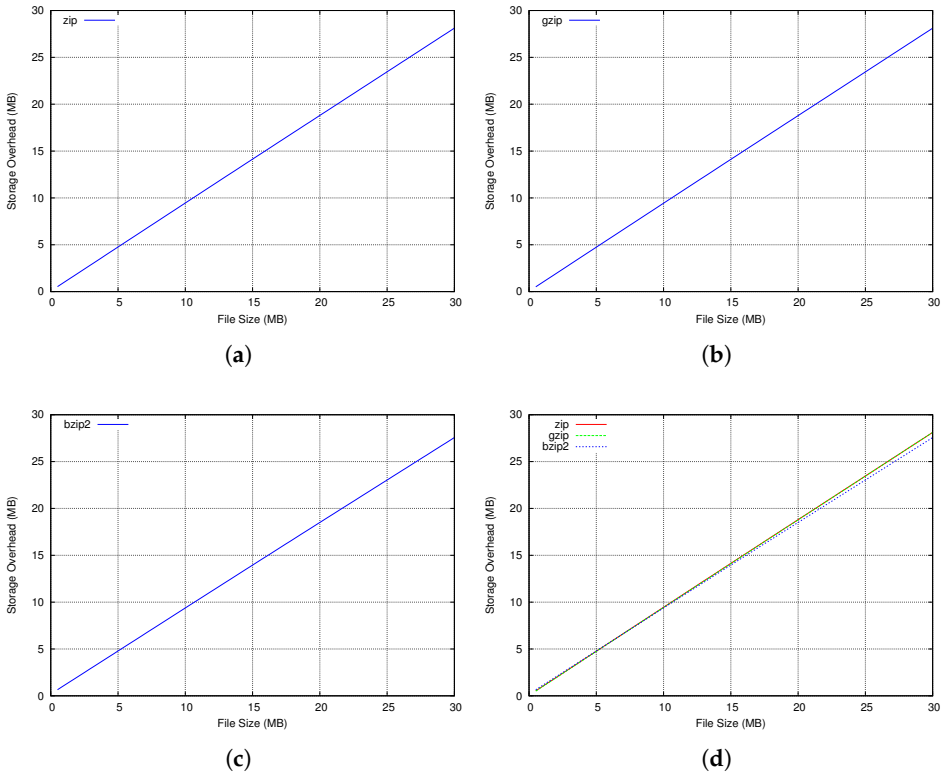

Figure 6. Storage overhead by SH-VARR snapshot based on three compression algorithms. (**a**) Using zip algorithm; (**b**) Using gzip algorithm; (**c**) Using bzip2 algorithm; (**d**) All algorithms.

The proposed SH-VARR framework involves several steps to create a protected snapshot for each file version. Therefore, it is important to measure the amount of time required to perform such an operation. Figure 7a–c show how the time requirement increases with the original file size for creating the snapshot in the proposed SH-VARR approach when leveraging the `zip`, `gzip`, and `bzip2` compression algorithms, respectively. Figure 7d illustrates all cases together for comparison purposes. Creating a protected version for small files (e.g., less than 1 MB) takes a negligible amount of time that would, on average, not exceed 120 ms. However, for larger file sizes exceeding 10 MBs, more time is required to create the protected version. It can be observed that the amount of time varies as file compression depends on the amount of redundancy in each file and the type of content (e.g., text, images, etc.) contained in each file. It is evident from the outcomes of using both the `zip` and the `gzip` algorithms that the results are fairly comparable and they are seen to offer much better results than when using the `bzip2` algorithm. In fact, the `bzip2` is observed to consume considerable amounts of time to create the protected version, especially when the file sizes involved are quite large.

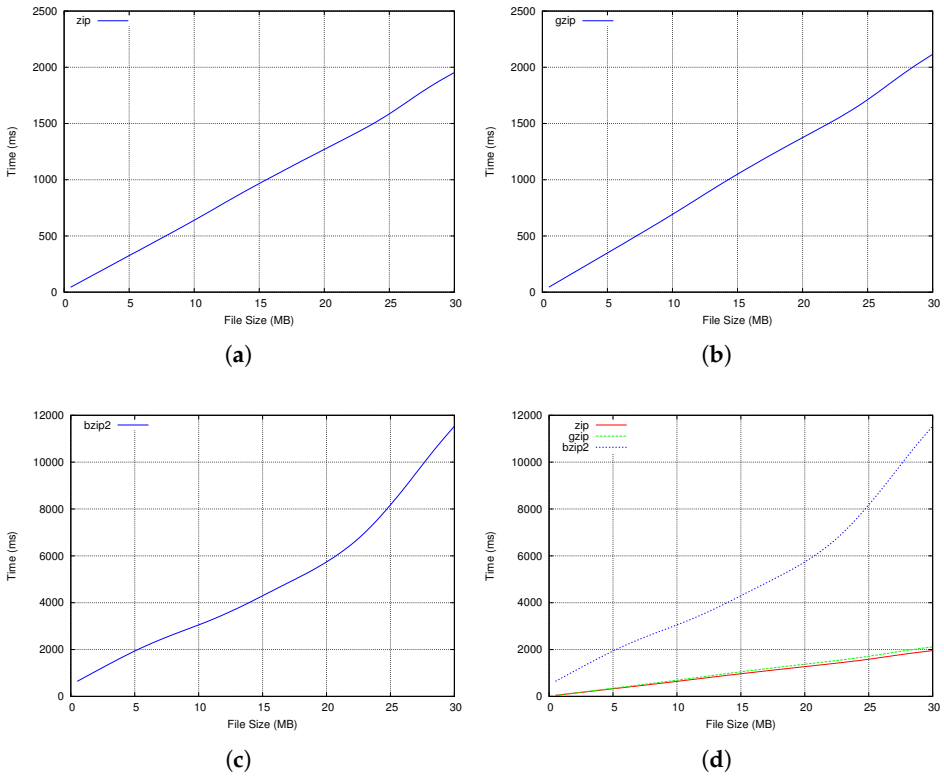

Figure 7. Time requirement for SH-VARR snapshot based on three compression algorithms. (**a**) Using zip algorithm; (**b**) Using gzip algorithm; (**c**) Using bzip2 algorithm; (**d**) All algorithms.

Figure 8a–c show how the CPU utilization varies against the original file size for creating the snapshot in the proposed SH-VARR schema when leveraging the `zip`, `gzip`, and `bzip2` compression algorithms, respectively. Figure 8d illustrates all cases together for the purpose of comparison. Here, CPU utilization is the amount of work handled by the CPU while creating a protected version for each file. Generally, for small files, CPU utilization increases with increasing file size. However, for larger file sizes, it levels off to some decent value. By monitoring the CPU utilization for each job executed when creating a protected version, we observed that when the `bzip2` compression algorithm was used the CPU utilization was evidently the highest.

Figure 9a–c show how the memory usage changes against the original file size to create the snapshot in the proposed SH-VARR schema when leveraging the `zip`, `gzip`, and `bzip2` compression algorithms. Figure 9d illustrates all cases together for comparison purposes. It is readily seen that the memory usage, for the cases when the `zip` and `gzip` compression algorithms are used, is almost fixed (around 6.8 KBs) where it does not show any dependence on file size. Meanwhile, memory usage for the case involving the `bzip2` compression algorithm is seen to increase with increasing file size, then it remains constant (around 28 KBs) for files with large sizes. This is because all the compression algorithms (`zip`, `gzip`, and `bzip2`) involved in our assessment of the proposed framework do not capture the entire file into the memory. Instead, they acquire it as a stream requiring a specific amount of memory each time (i.e., takes a chunk of data of a specific size each time), and the amount needed depends on the compression method used and the file size involved.

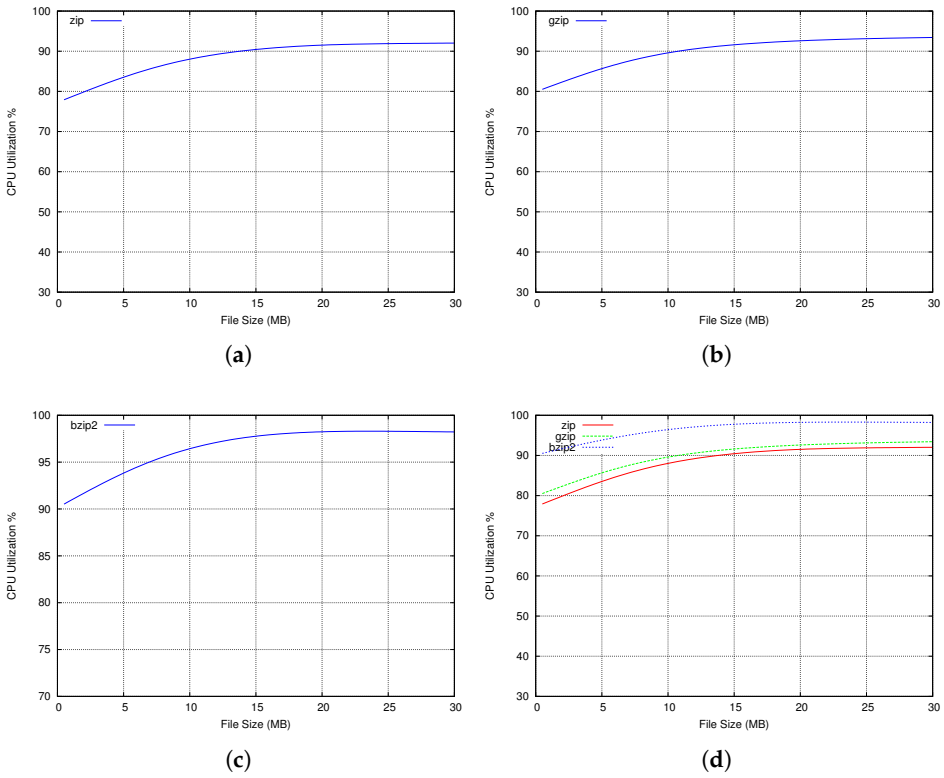

Figure 8. CPU utilization by SH-VARR snapshot based on three compression algorithms. (**a**) Using zip algorithm; (**b**) Using gzip algorithm; (**c**) Using bzip2 algorithm; (**d**) All algorithms.

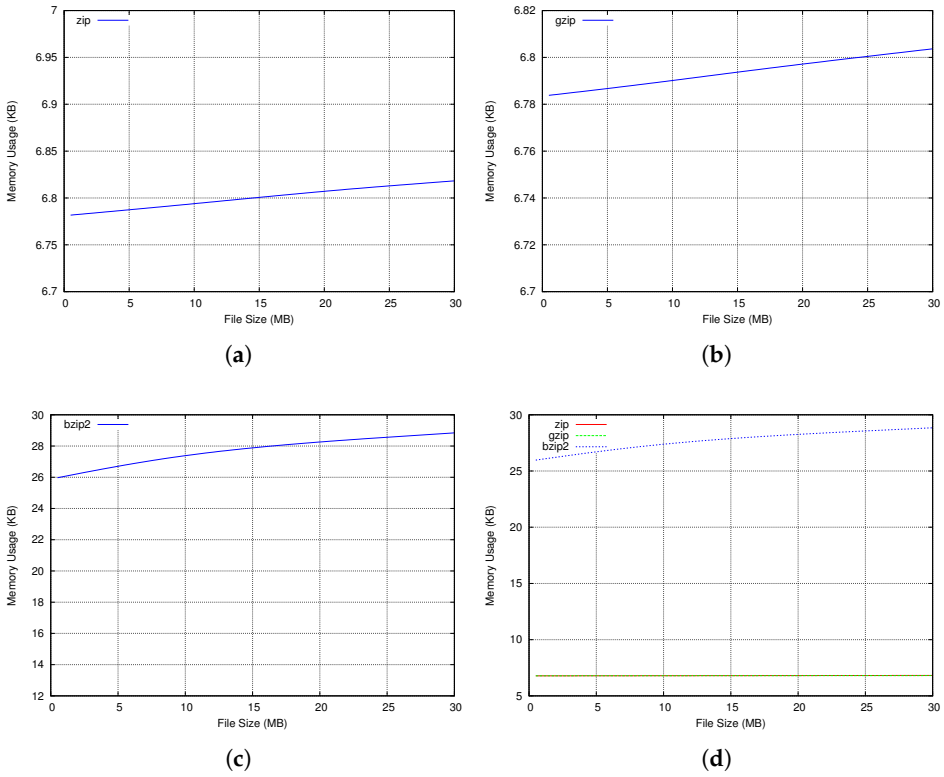

Figure 9. Memory usage by SH-VARR snapshot based on three compression algorithms. (**a**) Using zip algorithm; (**b**) Using gzip algorithm; (**c**) Using bzip2 algorithm; (**d**) All algorithms.

Finally, we compare the proposed mechanism with the work presented in [54,55]. In [55], the authors presented a Ransomware protection framework that depends on a network connection to backup files on a local or a remote server. However, they did not provide any performance evaluation of their framework in terms of time and storage requirements. In [54], the authors proposed backing up critical data in a fully isolated spare space that is not reachable by Ransomware, regardless of what privilege it can obtain. The authors assumed that the computing device has a particular portion of extra space, which can be utilized to create the backup volume to store encoded files with reverse deltas. This is different than the proposed work, where we can hold both reverse deltas and complete snapshots of files. We also used compression techniques to utilize the storage better. Moreover, our proposed work is portable because it can be shipped as a plugin that can be attached to documents; a feature that is not supported by [55] or [54].

6. Conclusions

In this paper, we introduced a Self-Healing Version-Aware Ransomware Recovery Approach (SH-VARR) of XML-based documents. This proposed system consists mainly of two modules. The first is a decentralized version-aware control system that periodically takes a backup version for each file and keeps the latest one. The second is the access-control module that executes special commands to protect the resulting versions from corruption or deletion caused by ransomware attacks; something that is carried out under administrator privileges.

The conducted set of experiments to assess the system focused on measuring the system performance in terms of the performance metrics: time, storage overhead, memory usage, and CPU utilization. Since compression is one of the main steps in the version-control system module, we evaluated these metrics by considering two commonly used compression algorithms: `bzip2` and `gzip`. Our technique (SH-VARR), introduced in this paper, uses the default zip algorithm. Comparisons show that the `zip` algorithm has the minimum time, size, utilization, and memory usage requirements. We conclude that this solution would protect XML-based files such as `.docx` and `.odt` files from ransomware attacks. The user can recover from such attacks even when the original files are deleted or encrypted. This is based on the assumption that these file types are compressed structures. In addition, we used a distributed version-aware control system to acquire a backup and keep track of each version. We observed access-control rules on these versions to achieve the core pillars of information security: Confidentiality, Integrity, and Availability.

Author Contributions: Conceptualization, A.S.S., O.A.-K. and B.A.-D.; methodology, M.A.-D.; software, M.A.-D. and A.S.S.; validation, M.A.-D.; writing—original draft preparation, M.A.-D.; writing—review and editing, M.A.-D., A.S.S., O.A.-K. and B.A.-D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Mashtalyar, N.; Ntaganzwa, U.N.; Santos, T.; Hakak, S.; Ray, S. *Social Engineering Attacks: Recent Advances and Challenges, HCI for Cybersecurity, Privacy and Trust*; Springer: New York, NY, USA, 2021; pp. 417–431
2. Mukhopadhyay, I. Cyber Threats Landscape Overview Under the New Normal. In *ICT Analysis and Applications*; Fong, S., Dey, N., Joshi, A., Eds.; Lecture Notes in Networks and Systems; Springer: Singapore, 2022; Volume 314. [CrossRef]
3. Djenna, A.; Harous, S.; Saidouni, D.E. Internet of Things Meet Internet of Threats: New Concern Cyber Security Issues of Critical Cyber Infrastructure. *Appl. Sci.* **2021**, *11*, 4580. [CrossRef]
4. Jang-Jaccard, J.; Nepal, S. A survey of emerging threats in cybersecurity. *J. Comput. Syst. Sci.* **2014**, *80*, 973–993. [CrossRef]
5. Zong, S.; Ritter, A.; Mueller, G.; Wright, E. Analyzing the Perceived Severity of Cybersecurity Threats Reported on Social Media. *arXiv* **2019**, arXiv:1902.10680.
6. Rudd, E.; Rozsa, A.; Günther, M.; Boult, T. A Survey of Stealth Malware Attacks, Mitigation Measures, and Steps Toward Autonomous Open World Solutions. *IEEE Commun. Surv. Tutor*. **2017**, *19*, 1145–1172. [CrossRef]

7. Nakashima, E.U.S. Aims to Thwart Ransomware Attacks by Cracking Down on Crypto Payments. *The Washington Post*. 2021. Available online: https://www.washingtonpost.com/business/2021/09/17/biden-sanctions-ransomware-crypto (accessed on 19 October 2021).
8. Kumar, M.; Ben-Othman, J.; Srinivasagan, K. An Investigation on Wannacry Ransomware and its Detection. In Proceedings of the 2018 IEEE Symposium on Computers and Communications (ISCC), Natal, Brazil, 25–28 June 2018; pp. 1–6.
9. Stallings, W. *Network Security Essentials: Applications and Standards*; Pearson: London, UK, 2016.
10. Peter, A.; Peter, S.; Van Ekert, L. An ontology for network security attacks. In *Proceedings of the 2nd Asian Applied Computing Conference (AACC'04), LNCS 3285*; Springer: Berlin/Heidelberg, Germany, 2004.
11. Richardson, R.; North, M. Ransomware: Evolution, mitigation and prevention. *Int. Manag. Rev.* **2017**, *13*, 10.
12. Everett, C. Ransomware: To pay or not to pay? *Comput. Fraud Secur*. **2016**, *2016*, 8–12. [CrossRef]
13. Yaqoob, I.; Ahmed, E.; Rehman, M.; Ahmed, A.; Al-garadi, M.; Imran, M.; Guizani, M. The rise of ransomware and emerging security challenges in the Internet of Things. *Comput. Netw*. **2017**, *129*, 444–458. [CrossRef]
14. Shashank, M.; Agrawal, A.K. Multi Pronged Approach for Ransomware Analysis. Available online: https://deliverypdf.ssrn.com/delivery.php?ID=529106093087077008125066087007008126061069029053059024023024048119007044109100058011016111014009004006028061086001098107006013106127099006095000116044119113035023073115003083030043113078009059098044124031019004068007115065011000084085080125073117006075066113004076094086068087090001095082&EXT=pdf&INDEX=TRUE (accessed on 10 March 2022).
15. What You Need to Know about the WannaCry Ransomware. Available online: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/wannacry-ransomware-attack (accessed on 10 March 2022).
16. Leong, R.; Beek, C.; Cochin, C.; Cowie, N.; Schmugar, C. Understanding Ransomware and Strategies to Defeat It. 2016. Available online: https://www.mcafee.com/enterprise/en-us/assets/white-papers/wp-understanding-ransomware-strategies-defeat.pdf (accessed on 10 March 2022).
17. Al-rimy, B.; Maarof, M.; Shaid, S. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Comput. Secur.* **2018**, *74*, 144–166. [CrossRef]
18. Young, A.; Yung, M. Cryptovirology: The birth, neglect, and explosion of ransomware. *Commun. ACM* **2017**, *60*, 24–26. [CrossRef]
19. Young, A.; Yung, M. Cryptovirology: Extortion-based security threats and countermeasures. In Proceedings of the 1996 IEEE Symposium on Security and Privacy, Oakland, CA, USA, 6–8 May 1996; pp. 129–140.
20. Luo, X.; Liao, Q. Awareness education as the key to ransomware prevention. *Inf. Syst. Secur.* **2007**, *16*, 195–202. [CrossRef]
21. Gostev, A.; Unuchek, R.; Garnaeva, M.; Makrushin, D.; Ivanov, A. IT Threat Evolution in Q1 2016. Kapersky 2015 Report, Kapersky L. 2016. Available online: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2018/03/07192617/Q1_2016_MW_report_FINAL_eng.pdf (accessed on 10 March 2022).
22. Thao, C.; Munson, E. Version-aware XML documents. In Proceedings of the 11th ACM Symposium on Document Engineering, Mountain View, CA, USA, 19–22 September 2011; pp. 97–100.
23. Coakley, S.; Mischka, J.; Thao, C. Version-Aware Word Documents. In Proceedings of the 2nd International Workshop on (Document) Changes: Modeling, Detection, Storage and Visualization, Fort Collins, CO, USA, 16 September 2014; p. 2.
24. Shatnawi, A.; Ethan, V.M.; Cheng, T. Maintaining integrity and non-repudiation in secure offline documents. In Proceedings of the 2017 ACM Symposium on Document Engineering, Valletta, Malta, 4–7 September 2017; pp. 59–62.
25. Shatnawi, A.S.; Ethan, V.M. Enhanced Automated Policy Enforcement eXchange framework (eAPEX). In Proceedings of the ACM Symposium on Document Engineering 2019, Berlin, Germany, 23–26 September 2019; pp. 1–4.
26. Gazet, A. Comparative analysis of various ransomware virii. *J. Comput. Virol*. **2010**, *6*, 77–90. [CrossRef]
27. Kharraz, A.; Kirda, E. Redemption: Real-time protection against ransomware at end-hosts. In *International Symposium on Research in Attacks, Intrusions, and Defenses*; Springer: Cham, Switzerland, 2017; pp. 98–119.
28. Bayer, U.; Kruegel, C.; Kirda, E. TTAnalyze: A Tool for Analyzing Malware. 2006. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.7584&rep=rep1&type=pdf (accessed on 10 March 2022).
29. Cabaj, K.; Mazurczyk, W. Using software-defined networking for ransomware mitigation: The case of cryptowall. *IEEE Netw*. **2016**, *30*, 14–20. [CrossRef]
30. Yen, T.; Heorhiadi, V.; Oprea, A.; Reiter, M.; Juels, A. An epidemiological study of malware encounters in a large enterprise. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, 3–7 November 2014; pp. 1117–1130.
31. Zhang, T.; Antunes, H.; Aggarwal, S. Defending connected vehicles against malware: Challenges and a solution framework. *IEEE Internet Things J*. **2014**, *1*, 10–21. [CrossRef]
32. Hampton, N.; Baig, Z.; Zeadally, S. Ransomware behavioural analysis on windows platforms. *J. Inf. Secur. Appl.* **2018**, *40*, 44–51. [CrossRef]
33. Subedi, K.; Budhathoki, D.; Dasgupta, D. Forensic analysis of ransomware families using static and dynamic analysis. In Proceedings of the 2018 IEEE Security And Privacy Workshops (SPW), San Francisco, CA, USA, 24 May 2018; pp. 180–185.
34. Mansfield-Devine, S. Leaks and ransoms–the key threats to healthcare organisations. *Netw. Secur*. **2017**, *2017*, 14–19. [CrossRef]
35. Kolodenker, E.; Koch, W.; Stringhini, G.; Egele, M. PayBreak: Defense against cryptographic ransomware. In Proceedings of the 2017 ACM on Asia Conference on Computer And Communications Security, Abu Dhabi, United Arab Emirates, 2–6 April 2017; pp. 599–611.

36. Continella, A.; Guagnelli, A.; Zingaro, G.; De Pasquale, G.; Barenghi, A.; Zanero, S.; Maggi, F. ShieldFS: A self-healing, ransomware-aware filesystem. In Proceedings of the 32nd Annual Conference on Computer Security Applications, Los Angeles, CA, USA, 5–8 December 2016; pp. 336–347.
37. Gomez-Hernandez, J.; Gonzalez, L.; Garcia-Teodoro, P. R-Locker: Thwarting ransomware action through a honeyfile-based approach. *Comput. Secur.* **2018**, *73*, 389–398. [CrossRef]
38. Sathyanarayan, V.; Kohli, P.; Bruhadeshwar, B. Signature generation and detection of malware families. In *Australasian Conference on Information Security And Privacy*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 336–349.
39. Scaife, N.; Carter, H.; Traynor, P.; Butler, K. Cryptolock (and drop it): Stopping ransomware attacks on user data. In Proceedings of the 2016 IEEE 36th International Conference On Distributed Computing Systems (ICDCS), Nara, Japan, 27–30 June 2016; pp. 303–312.
40. Shafiq, M.; Khayam, S.; Farooq, M. Improving accuracy of immune-inspired malware detectors by using intelligent features. In Proceedings of the 10th Annual Conference On Genetic And Evolutionary Computation, Atlanta, GA, USA, 12–16 July 2008; pp. 119–126.
41. Yüksel, Ö.; Hartog, J.; Etalle, S. Towards useful anomaly detection for back office networks. In *International Conference on Information Systems Security*; Springer: Cham, Switzerland, 2016; pp. 509–520.
42. Hurtuk, J.; Chovanec, M.; Kičina, M.; Billik, R. Case Study of Ransomware Malware Hiding Using Obfuscation Methods. In Proceedings of the 2018 16th International Conference on Emerging ELearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 15–16 November 2018; pp. 215–220.
43. Lim, C.; Ramli, K. Mal-ONE: A unified framework for fast and efficient malware detection. In Proceedings of the 2014 2nd International Conference on Technology, Informatics, Management, Engineering & Environment, Bandung, Indonesia, 19–21 August 2014; pp. 1–6.
44. Poudyal, S.; Subedi, K.; Dasgupta, D. A Framework for Analyzing Ransomware using Machine Learning. In Proceedings of the 2018 IEEE Symposium Series on Computational Intelligence (SSCI), Bangalore, India, 18–21 November 2018; pp. 1692–1699.
45. Cusack, G.; Michel, O.; Keller, E. Machine learning-based detection of ransomware using sdn. In Proceedings of the 2018 ACM International Workshop on Security In Software Defined Networks & Network Function Virtualization, Tempe, AZ, USA, 21 March 2018; pp. 1–6.
46. Andronio, N.; Zanero, S.; Maggi, F. Heldroid: Dissecting and detecting mobile ransomware. In *International Symposium On Recent Advances in Intrusion Detection*; Springer: Cham, Switzerland, 2015; pp. 382–404.
47. Stokkel, M. Ransomware Detection with bro. Talk at BroCon '16. Available online: https://old.zeek.org/brocon2016/brocon2016_abstracts.html#toc-top (accessed on 20 January 2020).
48. Cuzzocrea, A.; Martinelli, F.; Mercaldo, F. A Novel Structural-Entropy-based Classification Technique for Supporting Android Ransomware Detection and Analysis. In Proceedings of the 2018 IEEE International Conference On Fuzzy Systems (FUZZ-IEEE), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–7.
49. Takeuchi, Y.; Sakai, K.; Fukumoto, S. Detecting ransomware using support vector machines. In Proceedings of the 47th International Conference on Parallel Processing Companion, Eugene, OR, USA, 13–16 August 2018; p. 1.
50. Urooj, U.; Al-rimy, B.A.S.; Zainal, A.; Ghaleb, F.A.; Rassam, M.A. Ransomware Detection Using the Dynamic Analysis and Machine Learning: A Survey and Research Directions. *Appl. Sci.* **2022**, *12*, 172. [CrossRef]
51. Zimba, A.; Wang, Z.; Simukonda, L. Towards data resilience: The analytical case of crypto ransomware data recovery techniques. *Int. J. Inf. Technol. Comput. Sci.* **2018**, *10*, 40–51. [CrossRef]
52. Berrueta Irigoyen, E.; Morató Osés, D.; Magaña Lizarrondo, E.; Izal Azcárate, M. Ransomware encrypted your files but you restored them from network traffic. In Proceedings of the 2018 2nd Cyber Security in Networking Conference, CSnet 2018, Paris, France, 24–26 October 2018.
53. Thomas, J.; Galligher, G. Improving backup system evaluations in information security risk assessments to combat ransomware. *Comput. Inf. Sci.* **2018**, *11*. [CrossRef]
54. Subedi, K.P.; Budhathoki, D.R.; Chen, B.; Dasgupta, D. RDS3: Ransomware defense strategy by using stealthily spare space. In Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI), Honolulu, HI, USA, 27 November–1 December 2017; pp. 1–8.
55. Martínez-García, H.A. Facing ransomware: An approach with private cloud and sentinel software. *Comput. Fraud. Secur.* **2020**, *2020*, 16–19. [CrossRef]
56. Kim, G.; Kim, S.; Kang, S.; Kim, J. A Method for Decrypting Data Infected with Hive Ransomware. *arXiv* **2022**, arXiv:2202.08477.
57. Ye, H.; Dai, W.; Huang, X. File Backup to Combat Ransomware. U.S. Patent 9,317,686, 19 April 2016.
58. 90 Percent of Ransomware Can Execute without Administrator Rights-Business Reporter. Available online: https://engageemployee.com/90-per-cent-ransomware-can-execute-without-administrator-rights/ (accessed on 30 December 2019).