

東京大学
情報理工学系研究科 電子情報学専攻
修士論文

ファイル単位のバックアップを用いた
ランサムウェア攻撃に対する
リアクティブなデータ保護手法
Reactive Data Protection against Ransomware Attacks
Using File-Level Backup

48-236427

手塚 尚哉

Naoya Tezuka

指導教員 落合秀也 准教授

2025 年 1 月

概要

ランサムウェアの脅威に対応するためには、ランサムウェアの活動を迅速に検知して被害の拡大を防ぐだけでなく、仮に被害を受けた場合でもデータを復旧することができる手法が必要となる。定期的なスナップショットによる復旧は一般的な手法だが、バックアップの粒度の粗さから頻繁な取得は難しく、その結果データ損失が発生するおそれがある。本研究ではランサムウェアのファイル侵害の直前にデータを隔離領域へ退避させることでデータ復旧を実現するシステム Fuga を提案する。Fuga はファイル単位のバックアップをリアクティブに取得することでスナップショット方式の課題を克服し、誤検知時のコストも軽減する。本稿では Fuga の設計を行い、ランサムウェアのファイル侵害方法に応じた実装方針を示した。さらに eBPF を用いた実装を行い、様々なファイルサイズにおけるデータ保護性能とオーバーヘッドを評価した。これにより、ランサムウェアからデータを保護するシステムとして Fuga が実用的であることを示した。

Abstract

To address the threat of ransomware, it is essential not only to detect ransomware activities quickly to prevent further damage, but also to have methods for recovering data in case the attack succeeds. While recovery using periodic snapshots is a common approach, their coarse granularity makes frequent backups impractical, increasing the risk of data loss. In this study, we propose Fuga, a system that ensures data recovery by evacuating data to an isolated storage area just before ransomware compromises files. Fuga reactively performs file-level backups, addressing the limitations of snapshot-based approaches while reducing the cost of false positives. We presented the design of Fuga and its implementation strategies tailored to different methods of ransomware file compromise. Furthermore, we implemented Fuga using eBPF, and evaluated its data protection performance and overhead across various file sizes. We conducted evaluation experiments. The results demonstrate that Fuga is effective and practical as a data protection system against ransomware.

目次

第 1 章	序論	1
第 2 章	ランサムウェア	2
2.1	概要	2
2.2	ランサムウェアの分類	3
2.3	ランサムウェアの影響	6
2.4	主要なランサムウェアインシデント	8
第 3 章	ランサムウェア対策	11
3.1	感染リスクの緩和	11
3.2	ランサムウェア検知	11
3.3	ランサムウェア被害からの復旧	14
3.4	既存の復旧手法の課題	18
第 4 章	提案手法	20
4.1	既存手法の課題への対応	20
第 5 章	提案手法の設計	22
5.1	概要	22
5.2	各コンポーネントの設計	22
第 6 章	提案手法の実装	25
6.1	各コンポーネントの実装	25
第 7 章	評価	29
7.1	データ保護性能の評価	29
7.2	オーバーヘッドの評価	32
	発表文献と研究活動	35
	参考文献	36

vi 目次

付録 A	eBPF	43
A.1	歴史的背景	43
A.2	eBPF のアーキテクチャ	44

目次

2.1	Development of major ransomware families (1989–2021). The first know ransomware, AIDS Trojan, was introduced in 1989. [1].	3
2.2	Breakdown of encryption algorithms used by major ransomware families (1989–2021). [2]	5
2.3	The outcomes of ransomware attacks based on whether victims paid the ransom and the success rate of data recovery. [3]	7
2.4	Recovery costs from the most severe ransomware attacks in 2023 and 2024. The cost includes downtime, labor, device and network costs, and lost revenue. [4]	8
2.5	A post by BlackSuit detailing their alleged access to KADOKAWA’ s networks, encryption of the infrastructure, and theft of 1.5TB of data. [5]	10
3.1	Overview of each stage in MITRE ATT&CK framework. Some stages such as Reconnaissance are omitted for simplicity. [6]	12
3.2	Distribution of detection techniques and detection features used in previous ransomware detection studies for PC/workstation. [1]	13
3.3	Overview of PayBreak. [7]	16
3.4	On the right ShieldFS shadowing a file offended by ransomware malicious write (MW), in comparison to standard filesystems (on the left). [8]	18
5.1	The overview of the proposed system.	23
6.1	Implementation of the Evacuation Module with parallelization and pipelining. The indices shown in the diagram are used only for illustrative purposes to indicate the order of data blocks and to visualize the reordering process. In the actual implementation, such indices are not explicitly assigned.	28

7.1	Match rates by original file size, comparing results from the preliminary and main experiments. The horizontal axis in the left figure uses a common logarithmic scale.	30
7.2	Boxplot of processing times for the three operations in the Evacuation Module: Read, Decode, and Write. Entries with processing times exceeding 1000 μ s were excluded; only 2 out of 976 total entries were removed.	31
7.3	Variation of match rate by file size for different ring buffer sizes. The horizontal axis uses a logarithmic scale (base 2).	32
7.4	Variation of match rate by original file size for different degrees of parallelism p	33
7.5	The transition of CPU usage.	34
7.6	The transition of I/O wait.	34
A.1	Overview of eBPF system. [9]	44

表目次

3.1	Techniques for Initial Access and general countermeasures	12
6.1	Implementation environment for the proposed method.	26
7.1	Environment of the physical machine used for the evaluation.	30
7.2	Estimated performance metrics in MB/s.	34

第1章

序論

これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。

これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。

これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。

これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。

これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。これは、序論の文章である。

参考文献を引用してみる [?, ?]. もうひとつ引用する [?]. 日本語の文献 [?, ?] も引用する。創造情報学専攻のウェブページを引用する [?].

第 2 章

ランサムウェア

2.1 概要

ランサムウェアとはマルウェアの一種であり、攻撃者が要求した金額が支払われるまで、システムやデータへのアクセスを制限する。言い換えると、データや計算資源、サービスなどのリソースを人質に取って被害者を脅迫することで身代金を要求するマルウェアがランサムウェアである。

ランサムウェアはリソースへのアクセスを制限する方法に基づいて暗号化ランサムウェアとロッカーランサムウェアに分類される [10]。暗号化ランサムウェアは感染先ホストのファイルやデータを暗号化し、元のファイルを削除または上書きする。ロッカーランサムウェアは暗号化を行わず、デスクトップのスクリーンやブラウザをロックすることで被害者がシステムを利用できないようにする。本研究は暗号化ランサムウェアを対象としているため、本稿では暗号化ランサムウェアを単に「ランサムウェア」と呼ぶことにする。

図 2.1 に示すように、AIDS Trojan [11] は 1989 年に最初のランサムウェアとして登場した。AIDS Trojan は被害者に郵送されたフロッピーディスクを介して感染し、Windows システムを対象としていた。その後インターネットの普及に伴い、ランサムウェアによる被害が増加し始めた。2005 年に登場した GPCode [12] はフィッシングメールを介して感染し、独自の暗号化アルゴリズムによってファイルを暗号化した。

現代のランサムウェアはますます高度化している。ランサムウェアの進化における重要な要素を以下に列挙する。

- AES などの対称鍵暗号化アルゴリズムや RSA、楕円曲線暗号などの非対称鍵暗号化アルゴリズムを使用して暗号化を行うようになっている [1]。これにより、復号鍵を入手することができなければデータの復号はほぼ不可能となった。
- Windows だけでなく、Linux, macOS, Android などの他の OS を対象としたランサムウェアも登場するようになった。
- ビットコインに代表される仮想通貨が普及したことで身代金の支払いが匿名で行えるようになり、攻撃者の特定が難しくなった。

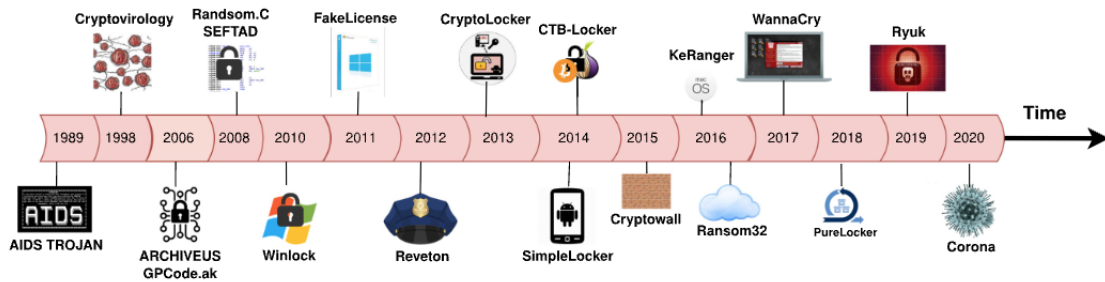


図 2.1: Development of major ransomware families (1989–2021). The first know ransomware, AIDS Trojan, was introduced in 1989. [1].

- ランサムウェアの開発と配布を有料で行うサービスである Ransomware as a Service (RaaS) が登場し、専門知識が無くとも容易に攻撃を実施することができるようになった。
- 無差別的な攻撃から、特定の高価値な組織 (政府機関や大企業など) を対象とした高度な攻撃に移行しつつある [13].

2.2 ランサムウェアの分類

2.2.1 悪意ある振る舞いに基づく分類

2.1 節で述べたように、ランサムウェアは被害者が身代金を支払うまでリソースへのアクセスを制限するが、アクセスを制限する方法には多様性が見られる。本稿では Oz らの分類 [1] を参照し、その方法として暗号化、データ破壊、データ窃取を扱う。

暗号化：ランサムウェアは暗号化鍵を用いてデータを暗号化し、元のデータを削除するか、暗号化後のデータで上書きする。この時使用する鍵はランサムウェアの実行ファイルに埋め込まれているか、感染先ホスト上で生成されるか、C2 サーバとの通信から取得されるかのいずれかである。ファイルを暗号化するランサムウェアの中には、暗号化の対象とするファイルを限定するものも存在する。例えば CTB-Locker [14] は、被害者にとってより高価値なファイルのみを暗号化するために、.pdf や.zip などの拡張子を持つファイルを暗号化対象としている。また、Jigsaw [15] は 10MB 以下のファイルのみを暗号化する。このように、ランサムウェアの一部は暗号化の対象とするファイルを限定することで、ランサムウェアの活動が検出されるリスクを緩和している [16] と考えられる。

データ破壊：破壊活動を目的としているがランサムウェアに擬態して攻撃者の意図を隠蔽しようとするマルウェアが確認されている。例えば、2017 年に発見された NotPetya [17] は、ハードディスク全体を暗号化した後、ビットコインの送金先として無効なアドレスを提示していた。このアドレスはランダムに生成されており、攻撃者が金銭を回収する意図がないことから、NotPetya は破壊活動を目的として作成されたと考えられる [17]。同様の攻撃として、暗

4 第2章 ランサムウェア

号化を行わず、ランダムなデータでファイルを上書きするマルウェアを作成し使用することも可能である。なお、このタイプのマルウェアの被害者は身代金を支払ってもリソースを復旧することができないが、本研究ではランサムウェアとして扱う。

データ窃取：ランサムウェアは機密文書や顧客の個人情報などの重要データを摂取する可能性がある。データの暗号化または破壊とデータの窃取を組み合わせる脅迫を行うランサムウェアを「二重脅迫ランサムウェア」と呼ぶ。二重脅迫ランサムウェアは、データの復旧のために一回、窃取したデータの公開を防ぐためにもう一回、被害者に身代金を要求する。二重脅迫ランサムウェアによる被害は近年増加しており、SOPHOS 社が発表したレポート [4] によると、2023 年に発生したランサムウェアインシデントのうち 32% においてデータの摂取も発生している。加えて、データの窃取のみによって脅迫を行う「ノーウェアランサム」[18] と呼ばれる手法も確認されている。

2.2.2 暗号化アルゴリズムに基づく分類

データを暗号化するランサムウェアは、ISO/IEC [19] などの標準化団体が採択した標準的なアルゴリズムを使用する場合と、攻撃者によって独自に設計された暗号化アルゴリズムを使用する場合がある。Begovic ら [2] が調査した、1991 年から 2021 年までに確認された著名なランサムウェア変種の暗号化アルゴリズムの使用状況を図 2.2 に示す。図 2.2 によると 8.2% のランサムウェアが独自の暗号化アルゴリズムを使用しているが、近年のランサムウェアは AES や RSA といった標準的な暗号化アルゴリズムを使用する傾向が強いことがいくつかの先行研究 [1, 20] にて指摘されており、この数値には初期のランサムウェアが多く含まれていると考えられる。より具体的には、攻撃者が独自に設計した暗号化アルゴリズムは強度が不十分で暗号解読者による解読が容易であることが多い [20] ため、2000 年代後半から 2010 年代前半にかけて、十分に評価され強度が高い暗号化アルゴリズムが採用されるようになっていった [1]。

ランサムウェアは**対称鍵暗号化**、**非対称鍵暗号化**、**ハイブリッド暗号化**のいずれかの暗号化技術を採用することができる。

対称鍵暗号化：対称鍵暗号化では、暗号化と復号のために 1 つの鍵のみが使用される。非対称鍵暗号化よりも高速に暗号化を行うことができるが、被害者が鍵を入手してファイルを復号することができる可能性がある。例えば、PayBreak [7] は、暗号化機能を提供する Windows API の関数をフックして暗号化に使用される共通鍵を取得することで復号を可能にしている。そのため攻撃者は、鍵が被害者からアクセスできないようにする必要がある。図 2.2 より、ランサムウェアが採用する対称鍵暗号化アルゴリズムとしては AES が最も人気であることがわかる。

非対称鍵暗号化：非対称鍵暗号化では、暗号化鍵（公開鍵）と復号鍵（秘密鍵）の 2 つの鍵が使用される。被害者が公開鍵を入手しても暗号化されたデータを復号することはできないため、対称鍵暗号化に比べて暗号化速度は劣るが、暗号化鍵の保護を行う必要がない。常に同一の鍵ペアを使用する場合、一度秘密鍵が漏洩（または、ある被害者が身代金を支払って秘密鍵を取

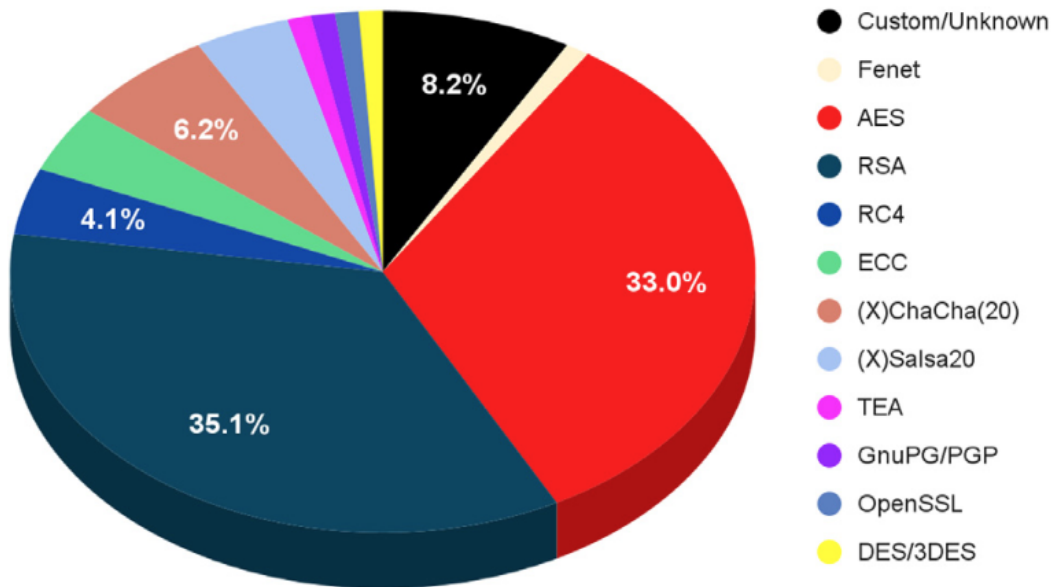


図 2.2: Breakdown of encryption algorithms used by major ransomware families (1989–2021). [2]

得)すると、その鍵ペアで暗号化されたデータは全て解読可能となる。そのため CryptoLocker [21] などの一部のランサムウェアは、被害者ごとに異なる鍵ペアを生成する戦略を採用している。RSA が最も頻繁に、楕円曲線暗号が次いで使用されていることが図 2.2 よりわかる。

ハイブリッド暗号化：ハイブリッド暗号化では、対称鍵暗号を用いてデータを暗号化した後、その暗号化鍵を非対称鍵暗号化アルゴリズムを用いて暗号化する。これにより大量のデータの暗号化を効率よく実行しつつ、対称鍵暗号化における問題点であった鍵の保護を解決することができる。近年の著名なランサムウェアはハイブリッド暗号化を採用しているものが非常に多い [2]。代表例としては WannaCry [22] や CTBLocker [14] が挙げられる。

2.2.3 暗号化の対象に基づく分類

ランサムウェアは OS 上でユーザが扱うデータファイル (e.g. .docx, .xlsx, .jpg) を暗号化することが一般的であるが、ファイル以外の単位で暗号化を行うランサムウェアも存在する。Mamba [23] はハードディスク全体を暗号化したのちマスターブートレコード (MBR) を書き換えて OS の正常な起動を阻害し、OS 起動時にランサムノートが表示されるようにする。また、Petya [23] は Windows システムのマスターファイルテーブル (MFT) ^{*1}を暗号化することでファイルのアクセスを不可能にする。DarkSide [24] ランサムウェアは VMWare ESXi [25] のホストマシン上で実行される。DarkSide は実行中の仮想マシン (VM) を強制終了さ

^{*1} MFT は Windows システム内に存在するファイルの物理的な位置、ファイル名、作成者などのメタデータを管理するデータ構造である。

6 第2章 ランサムウェア

せ、VM の仮想ディスクなどの関連ファイルを暗号化する。これらのファイルは、典型的には `/vmfs/volumes` ディレクトリ以下のファイルであるが、仮想マシンからは認識できない。

今日のクラウドサービスの隆盛に伴い、クラウド環境を対象としたランサムウェア攻撃が増加している。ALIBABA Cloud の仮想ストレージサービスは 2023 年の第三四半期のみで 1000 件以上の被害報告をユーザから受けており、2021 年と比較して 118% の増加率を示している [26]。さらに Zscaler 社は、クラウドサービスやクラウド上のワークフローに最適化されたランサムウェアが開発されることを予測 [27] しており、新しいタイプのランサムウェアの出現に備える必要がある。

2.3 ランサムウェアの影響

2.3.1 データ損失

PGPCode [12] に代表される原始的なランサムウェアの一部は、強度が不十分な独自の暗号化アルゴリズムを採用していた、鍵の保護が不十分だったなどの理由から、暗号解読者によって解読されるケースが存在した。しかし 2.2.2 節で述べたように、近年のランサムウェアは標準的な暗号化アルゴリズムを利用してデータを暗号化することが一般的であり、暗号化されたデータを独力で復号することは現実的ではない。したがって身代金を支払わない場合、ランサムウェア被害者は暗号化されたデータを失うことになる。

ランサムウェアによって利用不可能になったデータは、攻撃者に身代金を支払ったとしても復旧される保証はない。SpyCloud 社のレポート [3] (図 2.3) によると、2024 年にランサムウェア攻撃者に対して身代金を支払った組織のうち、暗号化されたデータを完全に復旧することができたのは約 50% である。

身代金を支払わない場合、あるいは攻撃者によって復旧の手段が提供されない場合でも、定期的なバックアップを取得していればデータ損失を緩和することが可能である。そこで身代金を支払うインセンティブを高めるために、近年の攻撃者はバックアップデータの侵害を試みることが多い [3]。2023 年にランサムウェア被害を報告した組織の 94% がバックアップへの侵害も試行され、さらにバックアップ侵害の試行の 57% が成功している [4]。例えば、CryptoWall や CTB-Locker といったランサムウェアは Windows OS のスナップショットを削除する機能を持っている [28] ことで知られる。またクラウドストレージによるバックアップは、暗号化されたローカルのデータがクラウドと同期された時点で利用不可能になる恐れがある。さらに、クラウドサービスにおける認証認可に使用するデバイスデータも暗号化することでクラウドストレージへのアクセスを不可能にするランサムウェアが増加していることが指摘されている [29]、

バックアップを安全に作成および運用できている場合でも、ランサムウェア被害によるデータ損失を完全に回避することは困難である。なぜなら、最新のバックアップからランサムウェア攻撃が発生するまでの間に更新されたデータは失われるからである。したがってバックアップの取得間隔はなるべく短いことが望ましいが、頻繁なバックアップ取得の課題としてスト



図 2.3: The outcomes of ransomware attacks based on whether victims paid the ransom and the success rate of data recovery. [3]

レージ容量の圧迫や運用コストの増加が挙げられる [26].

2.3.2 金銭的損失

攻撃者に支払う身代金がコストとして発生する。さらに、システムダウンやデータアクセスの制限により、業務が停止することで本来のサービスが提供できなくなるためサービスのダウンタイム中に機会損失が発生するほか、復旧作業の費用（データ復旧、セキュリティ対策や調査の person 費、デバイスやネットワークの修理費用など）も生じる。

ランサムウェア攻撃の対象は無差別な個人から高価値な組織へと移行しつつあり、それに伴って身代金額も増加している。SOPHOS 社のレポート [4] によると、要求される身代金の中央値が 200 万 USD、63% が 100 万 USD 以上であり、30% が 500 万 USD 以上であった。同レポートの調査対象において身代金の中央値と平均値が最も高かったのはアメリカ合衆国中央政府であった。

図 2.4 に示すように、身代金の支払いを除く復旧コストは 2024 年に大幅な増加が確認された。復旧コストの平均値は 2023 年の 182 万 USD から 273 万 USD に増加し [4, 3], 図 2.4 では年間の収益が中程度のグループにおいて復旧コストの増加が顕著である一方、収益が高いグループではコストは微増または減少している。

8 第2章 ランサムウェア



図 2.4: Recovery costs from the most severe ransomware attacks in 2023 and 2024. The cost includes downtime, labor, device and network costs, and lost revenue. [4]

2.3.3 信頼性の低下

ランサムウェア攻撃を受けた企業は、データの保護やセキュリティへの投資が不十分であるとの印象を顧客や取引先に与えることがある。これにより既存の顧客や取引先との取引関係が損なわれ、逸失利益が発生する可能性がある。図 2.4 に示される復旧コストには逸失利益も含まれていることに注意する。また非営利の組織であっても、メディア報道によってネガティブなイメージが拡散された結果組織の活動に支障が出るおそれもある。

2.3.4 長期的なリスク

CyberReason 社の調査 [30] によると、ランサムウェア攻撃を受けて身代金を支払った企業は、その 8 割が再度ランサムウェア攻撃を受ける。したがって身代金支払いによって一度の攻撃に対処したとしても、その後も継続的に攻撃の対象となるリスクがある。

2.4 主要なランサムウェアインシデント

2.4.1 選定基準

- 2015 年 (本稿の執筆時点から 10 年前) 以降に発生したインシデントであること。
- インシデントのタイムラインや被害状況などの情報が広く公開されていること。
- インシデントの規模が大きく、社会的影響が大きかったこと。

2.4.2 WannaCry の事例 (2017 年)

ランサムウェア WannaCry による攻撃は 2017 年 5 月に発生したランサムウェアインシデントである。WannaCry は Windows サーバの任意コード実行脆弱性 (CVE-2017-0143) を利

用して、Microsoft による脆弱性修正パッチが適用されていない Windows7 マシンを中心として感染を広げた。

スペインの大手通信事業者であるテレフォニカが最初の標的であり、その後ヨーロッパ諸国および世界中で感染が拡大して最終的には 100 カ国以上で被害が発生した [31]。英国における NHS (National Health Service, 国民保健サービス) の被害は特筆すべきものであり、NHS 全体で 19,000 件以上の来院予約のキャンセルと 9200 万ポンドの損失が生じ、手術や診察の中止により人名にかかわる被害も報告された [32]。

2.4.3 コロニアル・パイプライン社への攻撃 (2021 年)

RaaS を提供する犯罪者グループである DarkSide ^{*2} は、VPN システムの認証の不備についてコロニアル・パイプライン社のネットワークに侵入し、ランサムウェア攻撃を展開してデータの窃取と暗号化を行った [33]。コロニアル・パイプライン社はランサムウェア攻撃を受けて一時的にパイプラインの操業を停止した。同社はアメリカ合衆国の東海岸にて石油製品のパイプラインを運営しており、インシデント発生当時には東海岸にて 45% の燃料供給を担っていたため、操業停止の発表によってパニック的なガソリンの買い占めが発生し、一部の給油所で備蓄が枯渇した。このインシデントで影響を受けたのは同社の社内 IT システムのみであったため輸送パイプラインは 1 週間以内に稼働を再開することができたが、パイプライン制御システムが攻撃されていた場合操業停止が長引いて燃料のサプライチェーンに深刻な影響を及ぼす可能性があった。

2.4.4 名古屋港への攻撃 (2023 年)

2023 年 7 月 4 日に、名古屋港のコンテナターミナルシステムを対象としたランサムウェア攻撃が発生し、名古屋港の全ターミナルが約 3 日間作業停止となる障害が発生した [34]。名古屋港は総貨物取扱量や貿易額などの項目で日本一 [35] であり、復旧までに約 2 万本のコンテナの輸出入に影響が発生した。

本インシデントではコンテナターミナルシステムが稼働するすべての物理マシンおよび仮想マシン上のデータが暗号化され、システムバックアップによる復旧が行われた。ただしログはバックアップの対象外であり復旧作業中にログが消失したため、感染経路や感染の原因は特定されていない。

2.4.5 株式会社 KADOKAWA への攻撃 (2024 年)

株式会社 KADOKAWA は 2024 年 6 月にランサムウェア攻撃を受けたことを発表し、動画配信サービスをはじめとする主要サービスの一部が一時的に停止した [36]。この攻撃の対象は株式会社ドワンゴが使用するファイルサーバであり、ファイルの暗号化のみならずデータ窃取

^{*2} DarkSide という名前は犯罪者グループだけでなく、彼らが作成したランサムウェア [24] も指すが、本節では犯罪者グループを指す。

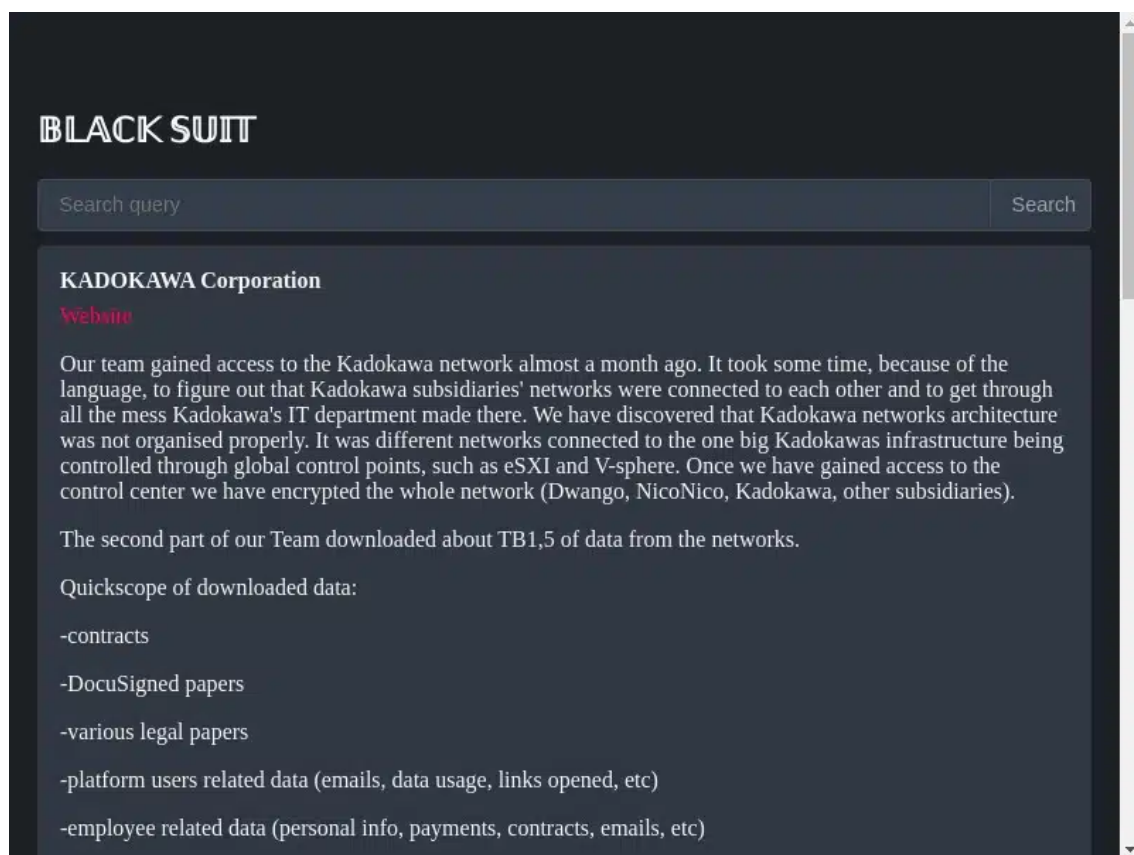


図 2.5: A post by BlackSuit detailing their alleged access to KADOKAWA's networks, encryption of the infrastructure, and theft of 1.5TB of data. [5]

も行われた。その結果、株式会社ドワンゴの従業員やユーザなど、合計で 25 万人以上の個人情報流出したとされる。この個人情報データは BlackSuit というランサムウェア犯罪グループによってダークネット上のリークサイトに公開された [5]。

第 3 章

ランサムウェア対策

3.1 感染リスクの緩和

現実世界の攻撃を戦術と使用技術の観点から分類したフレームワークである MITRE ATT&CK [37] によると、ランサムウェアのデータ侵害は、攻撃の最終段階である Impact ステージの Data Destruction, Data Encrypted for Impact, Data Manipulation のいずれかに分類される。つまり、ランサムウェアによるデータ侵害は Initial Access (初期アクセス) や Privilege Escalation (権限昇格) などのステージを完了した後に発生するといえる。したがって、Impact より前のステージにおけるセキュリティ強化もランサムウェア対策の重要な要素である。なお、本研究の提案手法はランサムウェアの Impact ステージの活動に対する対策であるため、本節の内容はスコープ外であることに注意する。

本節では、すべてのランサムウェアが通過するステージである Initial Access ステージに焦点を当て、感染リスクの緩和について述べる。SpyCloud 社 [3] によると、ランサムウェアの Initial Access に利用される手法として 2023 年に最も多く報告されたものはフィッシング、サードパーティアプリケーションの IAM 設定の不備、cookie 窃取によるセッションハイジャックであった。これらの手法に対する一般的な対策を表 ?? に示す。

3.2 ランサムウェア検知

3.2.1 検知に使用するデータ

あるアプリケーションがランサムウェアであるかどうかを判断するための入力データは、実行ファイルの解析から得られる静的データと、実行ファイルを実行した際にプロセスの振る舞いから得られる動的データに分類される。[38] では検知のための入力データを local static (バイナリファイルの構造的特徴)、local dynamic (プログラム実行時の振る舞いから得られるデータ)、network based (実行中のプログラムが送信または受信するネットワークトラフィック) の 3 種類に分類しているが、プログラム実行中に収集されるデータであることから、本稿ではネットワークトラフィックを動的データに含める。[1] が調査した、ランサムウェア検知の先行研究において採用された技術および特徴の統計を図 3.2 に示す。

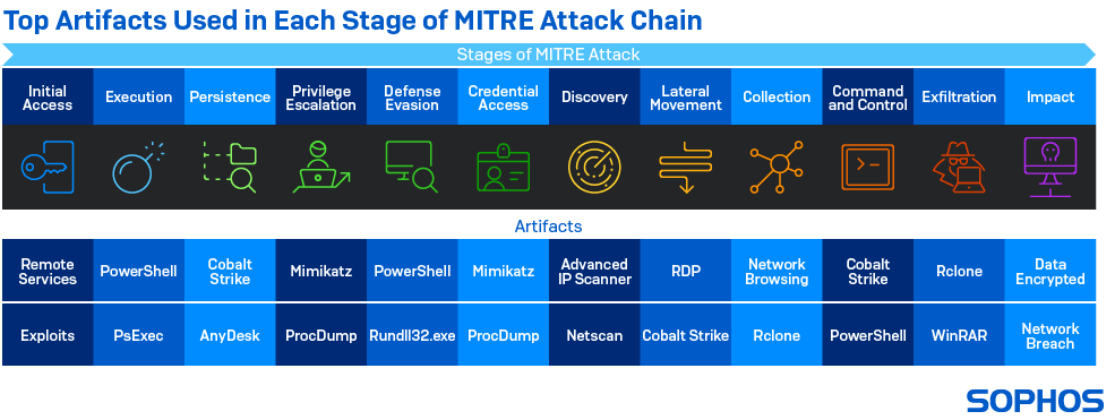


図 3.1: Overview of each stage in MITRE ATT&CK framework. Some stages such as Reconnaissance are omitted for simplicity. [6]

表 3.1: Techniques for Initial Access and general countermeasures

Initial Access の手法	対策
フィッシング	メールフィルタリングを強化する 組織構成員の教育を行う
IAM 設定の不備	多要素認証を導入する
Cookie 窃取によるセッションハイジャック	Secure 属性や HTTPOnly 属性を強制する 多要素認証を導入する

典型的な静的データはファイルのハッシュ値、文字列、ライブラリの API 呼び出しなどである。文字列としては”ransom”や”bitcoin”などのランサムノートに頻出する文字列、過去にランサムウェアが使用していたドメイン文字列や IP アドレスなどが確認される [38]。しかし、静的データは一般に難読化や圧縮などの手法に弱く、さらに新種のランサムウェアに対して有効ではないことが多いという問題が指摘されている [39]。API 呼び出しについては、暗号化やファイルアクセスといった操作の有無を、ランサムウェアと良性アプリケーションの区別に使用することができる [1]。API 呼び出しは実行ファイル中のシステムライブラリの動的リンクまたは静的リンクから取得される。また実行ファイルに含まれる命令オペコードも静的データとして利用されることがある [40]。

動的検知では実行中のプロセスの振る舞いを分析する。ランサムウェアは特定のディレクトリ以下のファイル一覧を取得して各ファイルを暗号化する挙動を繰り返し行うことから、ファイルシステム上の操作、暗号化ライブラリや API の呼び出しが特徴量として使用されることが多い [1]。またランサムウェアがファイルに書き込むデータは暗号化されており情報学的エントロピーが高いため、書き込まれるデータのエントロピーを利用する手法も複数存在する [41, 42]。ただし書き込みデータのエントロピーはファイル圧縮などの正常アプリケーションにおいても高くなるので、エントロピーはその他の特徴量と組み合わせて使われる傾向があ



図 3.2: Distribution of detection techniques and detection features used in previous ransomware detection studies for PC/workstation. [1]

る [38]. 一部のランサムウェアは暗号化が完了したファイルの拡張子を変更することがあるため、ファイル拡張子の変更に動的データとして利用されることがある [43]. ネットワークトラフィックに関しては、主にランサムウェアと C2 サーバとの通信を検知するために使用可能である。送信元や宛先の IP アドレス、ポート番号、プロトコル、ドメイン名が使用されることが多い [1].

3.2.2 代表的な検知手法

機械学習による検知

図 3.2 が示すように、ランサムウェア検知手法では機械学習が広く使用されている。[44] は 2017 年から 2022 年までに提案された機械学習ベースの検知手法を調査しており、本稿ではこの調査結果をもとに機械学習による検知手法を紹介する。

SSD-Insider [45] は SSD 上のデータのランサムウェアによる上書きのパターンを検知するために、SSD 上にランサムウェア検出器を導入した。筆者らはリクエストデータそのものではなく軽量の I/O リクエストヘッダから検知を行うことで検知器のオーバーヘッドを改善している。I/O リクエストヘッダには論理ブロックアドレス、リクエストタイプ (read or write)、データサイズのみが含まれており、これらの情報から 6 種類の特徴量を作成して決定木に入力する。SSD-Insider はランサムウェアサンプルによる実験でほぼ 0% の誤検知率および見逃し率を達成し、ランサムウェアに類似した高頻度の書き込みが行われるシナリオにおいても誤検知率は 5% 未満に抑えられた。

機械学習ベースの検知手法では静的データと動的データを組み合わせることで検知精度の向上が得られることが多い [44]。たとえば Wang ら [46] はプロセスのパケットデータをフローに要約した特徴と実行ファイル内の文字列やメタデータといった静的データを組み合わせ特徴量を作成し、決定木によって検知を行う。

ルールベースの検知

Medhat ら [47] は YARA ルールを作成するランサムウェア検出システムを提案した。提案手法では実行ファイルの暗号ライブラリおよびファイル操作の API 呼び出し、文字列を静的に解析することで対象のファイルに悪意スコアを割り当てる。Redemption [42] はプロセスの振る舞いベースの特徴量とアクセスするファイルコンテンツベースの特徴量を用いて MSC (Malice Score Calculation, 悪意スコア計算) 関数を設計した。あるユーザプロセスの悪意スコアを MSC 関数によって計算し、閾値を超えた場合にランサムウェアと判定する。

デコイファイルによる検知

デコイファイルとは、正常なアプリケーションがアクセスしないように作成されたダミーファイルのことである。あるプロセスがデコイファイルを上書きした場合、そのプロセスはランサムウェアである可能性が高いと考えられる。RWGuard [48] はプロセスおよびファイルの監視とデコイファイルを組み合わせた検知手法で、事前定義されたファイルのみ暗号化するランサムウェアか、デコイファイルの配置を認知している内部犯を想定してデコイファイルを使用する場合と使用しない場合の実験を行った。結果からデコイファイルはランサムウェアの検知を高速化するために有効であることがわかった。

3.2.3 検知レイテンシ

動的検知あるいはハイブリッド検知において、ランサムウェアが実行されてから検知されるまでの時間を本稿では検知レイテンシと呼ぶ。検知レイテンシが小さければランサムウェアの被害を抑えることができるため、検知手法の評価指標として重要である。例えば RATAFIA [49] はオートエンコーダを用いた検知手法で、WannaCry を含む変種を最大 5.3 秒で検知した。また、RWGuard [48] はデコイファイル、プロセス監視、ファイル監視を組み合わせるレイテンシの削減を目指した手法であり、評価実験ではランサムウェアのプロセスを 9 秒未満で特定した。Brownor ら [50] は提案した検知手法の検知レイテンシを CPU 使用率ごとに測定し、使用率が 90% の設定でも評価に用いたランサムウェアサンプルを 3 秒未満で検知することができた。

検知レイテンシはランサムウェア攻撃を迅速に終了させられるかどうかという観点において重要な指標であるが、検知レイテンシの評価を行っている研究が少ないことが指摘されている [49]。例えばいくつかの研究では、提案手法の検知精度を最大化するために、30 日間などの非常に長い期間での評価を行っているものがある [39]。

3.3 ランサムウェア被害からの復旧

ランサムウェア被害からの復旧とは、ランサムウェアによって侵害されたデータを身代金を支払うことなく攻撃前の状態に戻すことを指す。したがって復旧手法は、データの復旧を実施

するトリガとして、人間の明示的な入力または手法自体の自動的なランサムウェア検知を必要とする。

3.3.1 スナップショットによる復旧

スナップショットとは、特定の時点でのファイルシステムやデータの状態を記録する機能である。ランサムウェア攻撃が発生する前に取得したスナップショットを用いることで、攻撃前の状態にデータを復元することができる。そのため一定間隔でスナップショットを取得し、隔離されたストレージに保存しておくことはランサムウェア対策として広く採用されている [26]。

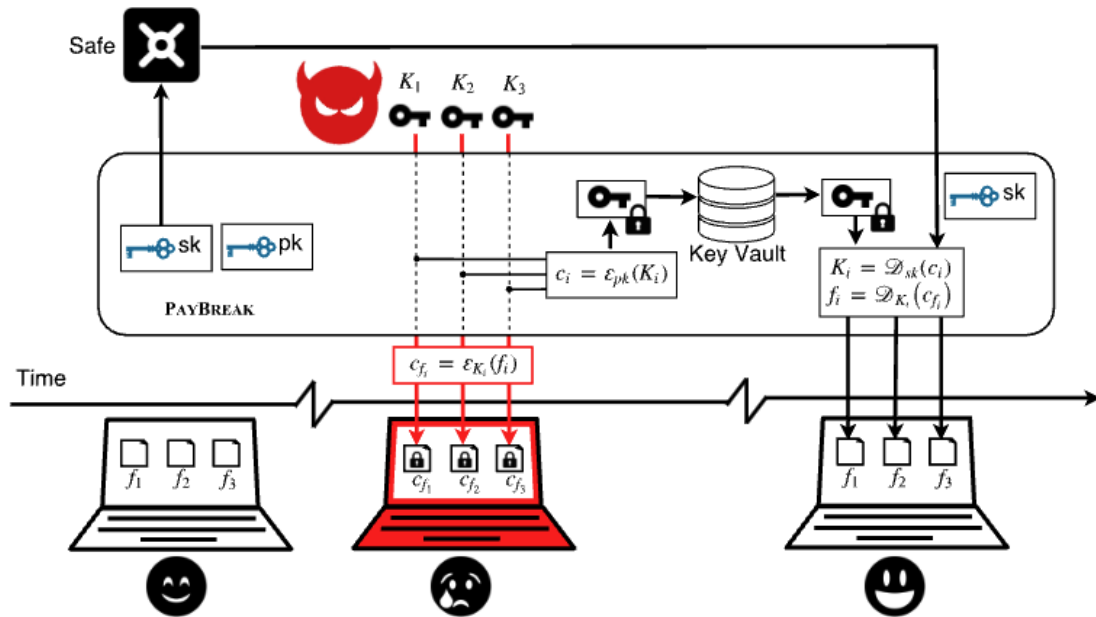
ZFS [51] や Btrfs [52] はファイルシステムとしてネイティブにスナップショット機能を提供している。ZFS は差分スナップショット方式を採用しており、スナップショット取得後に更新されたファイルのみを COW (Copy-On-Write) によってコピーする。これによりスナップショット作成を高速に行い、ストレージ容量を節約している。ZFS のスナップショットはファイルシステム単位で作成され、スナップショットを取得したファイルシステム上に保存される。Btrfs も ZFS と同様に差分スナップショット方式をサポートするが、スナップショットはサブボリューム単位で作成される。サブボリュームはファイルシステム上でディレクトリとして扱えるデータ領域で、ユーザが作成・削除することができるため ZFS よりも柔軟なスナップショット管理が可能である。

一部の商用 OS はスナップショット機能を提供している。Windows の VSS (Volume Shadow Copy Service) はファイルシステムのスナップショットを取得する機能であり、MacOS の Time Machine はファイルシステムに加えてアプリケーションや環境設定なども 1 時間ごとの差分バックアップにて保存する機能である。なお、Locky などのランサムウェアは VSS を削除することでスナップショットを無効化することがある [1]。

ハイパーバイザの機能を利用して仮想マシンまたは仮想ディスクのスナップショットを取得することもできる。例えば Xen Hypervisor [53] ではインターフェースなどの VM の設定情報を含むスナップショットを作成する機能を提供しており、作成したスナップショットをもとに VM を新規作成したり、スナップショットをマージすることでデータを復元したりすることができる。Azure [54] などの主要なクラウドサービスプロバイダもユーザ使用する仮想マシンの OS および仮想ディスクのスナップショット取得機能を提供している。近年のクラウドサービスを対象としたランサムウェア攻撃の増加 [27] を受け、ランサムウェア対策に特化したスナップショット機能を提供するクラウドストレージサービスも登場している [55]。

3.3.2 暗号化鍵の取得による復旧

2.2.2 節で述べたように、共通鍵暗号では暗号化と復号に同じ鍵を使用する。そのため共通鍵暗号を利用するランサムウェアに対しては、暗号化に使用される鍵を取得できればデータを復号することができる。PayBreak [7] はこの点に着目して開発された手法で、



☒ 3.3: Overview of PayBreak. [7]

Windows OS が提供する暗号化ライブラリをフックして暗号化鍵を取得する。取得された鍵は事前にユーザが登録した公開鍵によって暗号化されたのち、追記のみが可能なデータストアに保存される。PayBreak は暗号化ライブラリの動的リンクと静的リンクの両方に対応している。さらに代替となる暗号化ライブラリが登場した場合でも、マルウェア解析によって一度暗号化関数を特定することができれば、その関数をフック対象として容易に追加することができる。しかしながら、PayBreak は非対称鍵暗号を利用するランサムウェアには対応していない。

3.3.3 SSD の特性を利用した復旧

SSD の特性を活用してランサムウェアにより侵害されたデータを復旧する手法が提案されている。HDD と SSD はブロックデバイスとしての抽象化により論理的には同一の I/O 操作を受け付けるが、ハードウェアレベルではデータ上書き時の挙動が異なる。HDD では論理的な上書きに対して物理的な上書きが即時実行される (in-place 書き込みされる) 一方で、SSD における物理的な上書きは複数回の論理的な上書きに対してまとめて実行される (上書きされる古いデータが out-of-place 書き込みされ、GC: Garbage Collection によって一定時間の後削除される)。これは SSD 上で物理ページを消去するレイテンシが大きいためである。つまり、SSD は上書きされたファイルや削除されたファイルのコピーを、GC までの期間だけ保持することになる。

FlashGuard [16] は上記の SSD の特性を利用したデータ復旧システムであり，SSD ファームウェアとして実装された．FlashGuard はランサムウェアによって上書きまたは削除された可能性のある物理ページに「GC によって回収しない」というフラグを立てることで，SSD

上にそのデータを保持してデータ復旧に使用できるようにする。筆者らは 1477 個のランサムウェアサンプルを用いて FlashGuard の有効性を評価し、ランサムウェアによって暗号化されたファイルを効率的に復元可能であり SSD に与える性能劣化も最小限であることを示した。

FlashGuard は保守的にデータの保持を行うため、実際には保持しておく必要がない古いデータまで保持してしまうという問題がある。SSD-Insider [45] はブロック I/O リクエストのヘッダを参照してあるデータがランサムウェアによって侵害されたかどうかを判定することでこの問題を改善した。

SSD の特性を活用したこれらの手法は、ランサムウェア検知がデータ復旧のトリガとなるが、検知において誤検知や見逃しが頻繁に発生する点が先行研究によって指摘されている [56]。そのため本来必要のないリカバリが発生してデータが消失するリスクがある [57]。さらに、ハードウェアに依存した復旧手法は大規模な展開が難しいという課題も指摘されている [26]。特殊なハードウェアまたはプロトタイプ的なハードウェアに依存していること、ランサムウェアの進化に追隨してファームウェアを頻繁に更新することが非現実的であるからだ。

3.3.4 ファイルシステムの拡張による復旧

いくつかの先行研究では、ファイルシステムを拡張することでランサムウェアによるデータ侵害からの復旧を実現している。ShieldFS [8] は OS のネイティブなファイルシステムにカーネルモジュールを適用し、任意のプロセスのファイル書き込みまたは削除の I/O リクエストに対して COW を行うデータ保護システムである。概要を図 3.4 に示す。ShieldFS はすべてのプロセスの低レベルの I/O リクエスト、および不審なプロセスの暗号化処理を監視してプロセスごとの悪意スコアを計算するモジュールを持つ。あるプロセスの悪意スコアが閾値を超えた場合、COW のコピー元となっているファイルを用いて、そのプロセスによって更新されたファイルを復旧する。Redemption [42] は ShieldFS と類似した手法で、ファイルシステムへの書き込みおよび削除の I/O リクエストをインターセプトし、保護領域に作成されるコピーファイル（「リフレクションファイル」と呼ばれる）にリクエストをリダイレクトする。リフレクションファイルは変更履歴が保持されており、ランサムウェアによるファイルの暗号化や削除を検知した場合、リフレクションファイルを用いてファイルを復元する。

Matos らは RockFS [29] を提案し、悪意ある第三者によるファイルデータの侵害からの復旧を行うシステムの設計と実装を行なった。ランサムウェアなどがバックアップサービスのクライアントデバイス上に保存されているデータを侵害すると、データの更新がクラウドに同期されるため、バックアップが利用不可能になってしまう。RockFS は既存のクラウドバックアップサービスに存在するこの問題に注目し、クライアントデバイス上のファイルシステム操作のログをクラウドストレージ上に保存するというアプローチを取っている。ランサムウェア攻撃が発覚した場合に、ログ上の正常な操作のみをファイルの初期バージョンに適用することでデータを復元する。攻撃者がクラウドストレージへのアクセス情報を窃取してログを改ざんすることを防ぐために、RockFS はログを完全性を保証するための手法を提案している。さらに RockFS は操作ログを複数のクラウドストレージに分散して保存する仕組みを提供してお

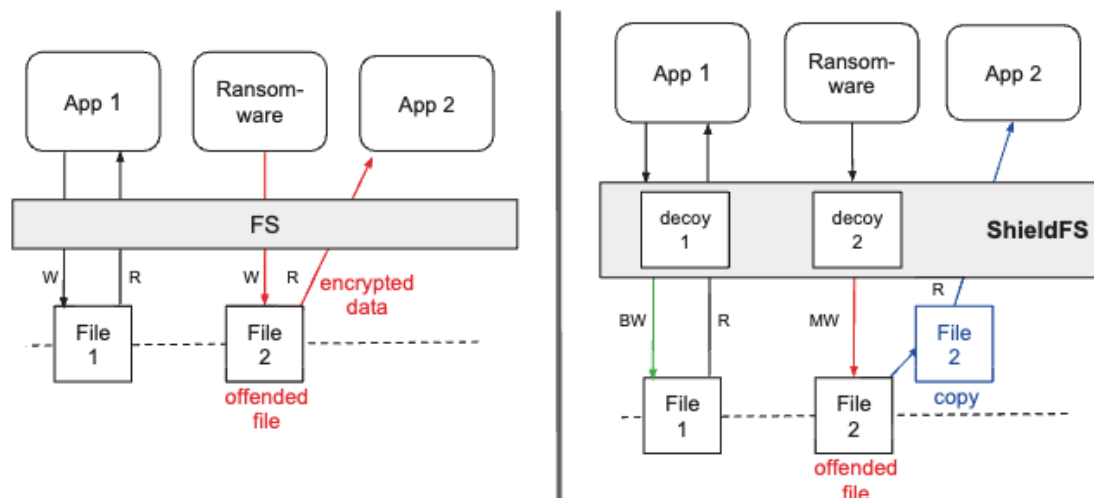


図 3.4: On the right ShieldFS shadowing a file offended by ransomware malicious write (MW), in comparison to standard filesystems (on the left). [8]

り、ログの改ざんを試みる攻撃者は複数のクラウドサービスに侵入する必要がある点が特徴的である。

3.4 既存の復旧手法の課題

3.4.1 スナップショットによる復旧の限界

取得しておいたスナップショットを利用してデータをランサムウェア攻撃前の状態に復旧するためにはスナップショットを一定以上の頻度で取得しておく必要があるが、高頻度の取得には課題が伴う。まず、金銭的なコストが大きくなる。Wang ら [26] の試算によると、ALIBABA Cloud における仮想ディスクのスナップショット取得サービスを 1 時間ごとに利用する場合、発生するコストは仮想ディスク使用料金の 2.5 倍に相当する。このコストはスナップショット取得による追加のストレージ容量および処理時間によるものであるため、クラウドサービスを利用せずオンプレミスでスナップショットを取得する場合でも同様のコストが発生する。

上述したコストを受容する場合でも、スナップショット方式による復旧ではデータ損失が発生する可能性が高いことが指摘されている [26]。第一に、最新のスナップショット取得時からランサムウェア攻撃発生までの間に更新されたデータは復旧されない。第二に、3.3.1 節で述べたように、スナップショットはファイルシステム、OS、ディスクなどの単位で取得される。したがってスナップショットでは、ランサムウェアによって侵害されたデータと正常なアプリケーションによって更新されたデータが区別されずにバックアップされる。この性質により、データ更新が頻繁に発生する環境においてスナップショットによるデータのコピーがデータ更新に追いつかず、データ損失が発生するおそれがある。

3.4.2 復旧のトリガとなる検知の課題

ランサムウェア検知手法は一般に誤検知と見逃しのトレードオフに悩まされる [1, 38, 39]. 攻撃を見逃すと侵害されたデータの復旧が不可能になるため見逃し率は可能な限り小さくする必要があるが、その場合トレードオフにより誤検知率が高くなり、正常なアプリケーションがランサムウェアとして判定され、システムの可用性が低下するおそれがある. また 3.3.3 で述べた SSD ベースの手法 [16, 45] にて誤検知が発生した場合、誤ったデータ復旧によってデータが消失する危険性が指摘されている [57].

実行時の振る舞いからランサムウェアを検知する手法では、必然的に一定時間のランサムウェアの活動を許容する. 被害を最小限に抑えるためには検知までの時間、検知レイテンシを短縮することが重要であるが、短時間での検知では誤検知率および見逃し率が高くなると考えられる [39]. あるプロセスがランサムウェアであるかどうかを判定するための情報が不足するほか、ランサムウェアが実行後すぐにデータ暗号化などの特徴的な挙動を行うとは限らないからだ.

第 4 章

提案手法

本章では、ランサムウェアによるファイル侵害からファイルデータを保護するための新しい手法を提案する。ここでいう「ファイルの侵害」とは暗号化・上書き・削除などの操作によってファイルを利用不可能にすることを指す。提案手法はランサムウェアのオンライン検知手法を使用し、ランサムウェアの疑いがあるユーザプロセスを特定する。検知手法は閾値を調整するなどしてランサムウェアの見逃しを削減するように設定する。そして特定されたプロセスが実行する、事前に定義したファイル侵害の処理をフックする。フックされた処理が実行されると、その処理のエントリーポイントにおいて侵害対象のファイルはランサムウェアから隔離された領域にコピーされる。ランサムウェアによって侵害される直前の状態でファイルのデータが保護されるため、ランサムウェア被害からの復旧が可能となる。

4.1 既存手法の課題への対応

3.4 節で述べた課題を提案手法が緩和または解決する方法を示す。

4.1.1 スナップショットによる復旧の課題への対応

提案手法は、理想的にはランサムウェアによって侵害されるデータのみをバックアップし、正常なアプリケーションによって更新されたデータはバックアップしない。これによりバックアップデータによるストレージ容量の圧迫を軽減する。さらに、スナップショット方式ではどの時点のスナップショットを利用して復旧を行うかを選択する必要があるが、提案手法ではランサムウェアによる侵害の直前のデータが保護される。つまり提案手法を用いると復旧時に使用すべきバックアップデータは単一かつ最新に保たれるため、データ復旧の手順の簡略化と作業コストの削減が期待できる。

提案手法はスナップショット方式において発生するデータ損失を回避することができる。第一に、リアクティブなバックアップ取得によって、最新のバックアップ以降に発生したデータ更新は保護される。第二に、提案手法はバックアップが必要なデータのみを保護するため、正常なアプリケーションによる高頻度のデータ更新が発生する環境におけるデータ損失は発生し

ない。

実際には誤検知と見逃しの両方を完全に排除することは現実的ではなく、提案手法が採用するランサムウェアのオンライン検知手法が誤検知を行った場合に余計なデータバックアップが作成される。しかし依然として、提案手法はバックアップによるストレージ容量の消費を軽減し、上述したデータ損失を防ぐことができる。

4.1.2 検知の課題への対応

見逃し率を小さくしてデータ損失を防ぐ検知手法を、「保守的な検知」と呼ぶことにする。提案手法は保守的な検知手法を利用し、各ユーザプロセスがランサムウェアの疑いがあるかどうかを判別する。この調整によって、実際にはランサムウェアであるプロセスを検出できずデータが侵害される可能性を減らし、検知レイテンシを短縮する。検知レイテンシが短縮される理由は、ユーザプロセスの「ランサムウェアらしさ」が十分確からしくなっていない、例えば Redemption [42] において計算される悪意スコアが高くない、時点で「ランサムウェアである」と判断することができるからである。

見逃しを減らす検知ポリシーを設定するとトレードオフとして誤検知が増加する。しかし提案手法では、誤検知が発生したとしても一部のファイルが余計にバックアップされるだけであり、ストレージ容量に発生する影響は小さい。提案手法を用いてデータバックアップを取得している間、一定時間の経過後にフック対象のプロセスの「ランサムウェアらしさ」は保守的な検知においても十分に高くなるか、または十分に低くなることが期待される。前者では該当プロセスを停止すればよく、後者ではバックアップ取得の対象となるプロセス群から該当プロセスを除外すればよい。

第 5 章

提案手法の設計

5.1 概要

提案手法の設計を図 5.1 に示す。提案手法はコンポーネントとして **Detector**, **Process Monitor**, **Evacuation Module**, **Data Shelter** を含む。

ランサムウェアの疑いがあるプロセスを Detector が検出して Process Monitor に通知する。Process Monitor は指定されたプロセスが侵害しようとするファイルのデータを取得して Evacuation Module に送信する。Evacuation Module は Process Monitor から受け取ったデータを処理し、Data Shelter 内に侵害対象のファイルを退避させる。

5.2 各コンポーネントの設計

5.2.1 Detector

Detector はユーザ空間で実行されるプロセスで、自身以外のユーザプロセスを監視している。ランサムウェアの可能性が閾値よりも高いプロセスを発見した際に、そのプロセスの識別子を Process Monitor に通知する。なお、3.2 節で述べたように、ランサムウェアの検知手法は先行研究が多数存在する。したがって本研究において Detector の機能は利用可能であるという前提を置き、その内部実装はスコープ外とする。

5.2.2 Process Monitor

Process Monitor はカーネル空間で実行されるプログラムであり、Detector が指定したユーザプロセスを監視している。暗号化前の平文データを取得する **Capturer** モジュールと、ファイルのメタデータを収集する **Collector** モジュールで構成される。Capturer が取得したデータと Collector が収集したメタデータは、一つのデータ構造に格納されて Evacuation Module に送信される。

Capturer は監視対象のプロセスが実行する暗号化関数をフックし、関数の引数に渡される平文データのポインタをキャプチャする。その後ポインタが指すバッファのデータをコピーす

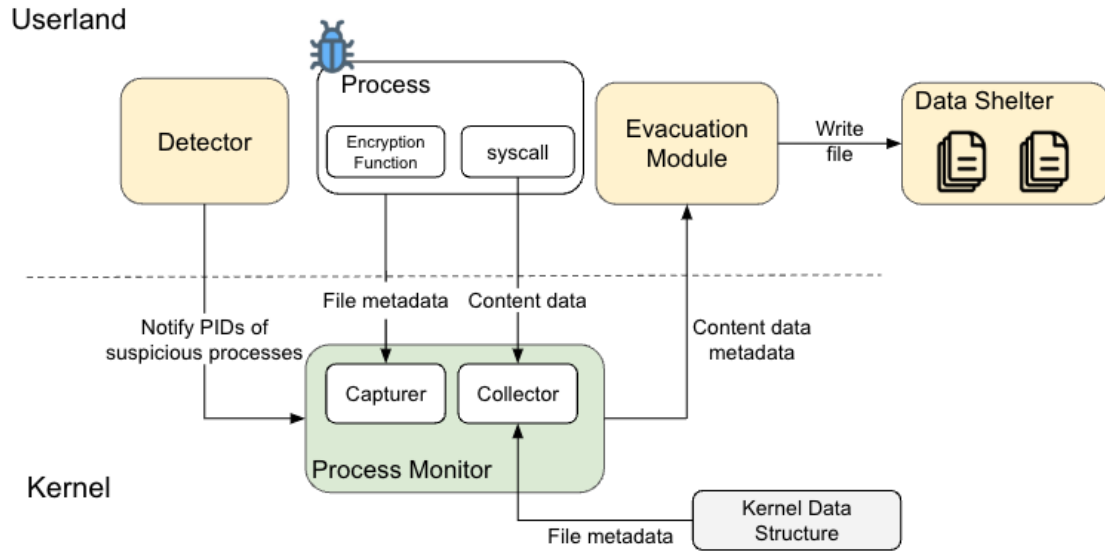


図 5.1: The overview of the proposed system.

ることによって平文データを取得する。

Collector は Capturer と連携し、ファイルの復旧に必要なメタデータ、具体的にはファイルパスとコンテンツデータのオフセットを以下のように収集する。

ファイルパスの取得：ファイルを開くシステムコールをフックし、ファイルの相対パスをシステムコールの引数から取得する。さらに、カーネル内構造体にアクセスして対象プロセスの現在の作業ディレクトリ（Current Working Directory: CWD）を取得する。

オフセットの取得：あるファイルが監視対象プロセスによって開かれたら、そのファイルのオフセットを管理する変数を初期化する。ファイルのコンテンツデータをメモリ上に読み込むシステムコール、およびファイルの現在のオフセットを操作するシステムコールをフックし、各ファイルのオフセットの変化量を取得する。この値を用いてファイルのオフセットを動的に更新する。

5.2.3 Evacuation Module

Evacuation Module はユーザ空間で実行されるコンポーネントである。Process Monitor から受け取ったファイル関連のデータを基に、退避する対象のファイルを構成し、Data Shelter に書き込む。

5.2.4 Data Shelter

ランサムウェアによる侵害から隔離されたデータ領域を、本稿では Data Shelter と呼ぶ。ランサムウェアを使用した攻撃が発覚した後、本システムの利用者は Data Shelter に退避されたファイルを取得し侵害されたファイルを復旧することができる。

24 第 5 章 提案手法の設計

Data Shelter として使用するデータストアにはいくつか選択肢が考えられる．アクセス制限を適用したローカルのディレクトリが最もシンプルであるが，NAS などを利用してホスト外部に Data Shelter を配置してもよい．

第 6 章

提案手法の実装

5 章で述べた提案手法の設計をもとに提案手法を実装した。本稿の実装は、TMP 節で述べたケースの実例として、OpenSSL を利用して暗号化を行うランサムウェアを対象としている。カーネル空間のプログラムは eBPF [9] を利用して、ユーザ空間のプログラムは Go 言語を用いて実装した。本稿の執筆時点において eBPF は実用上 Linux 環境のみで利用可能であるため、Linux 環境にて実装を行った。実装に使用したシステムの環境を表 6.1 に示す。

6.1 各コンポーネントの実装

本節では提案手法の実装特有の詳細を説明する。Detector の実装は本研究のスコップ外であることに注意する。

6.1.1 Process Monitor

Capturer

Capturer は、OpenSSL 内の暗号化関数を uprobe でフックする eBPF プログラムとして実装した。暗号化関数には引数として平文データが格納されているバッファへのポインタが渡されるため、eBPF プログラム内でこのポインタをキャプチャしユーザ空間からカーネル空間へ平文データをコピーする。コピーされた平文データは、関連するメタデータとともに暗号化イベントを表す構造体に格納され、eBPF map のリングバッファを介して Evacuation Module に送信される。なお、本稿で示す実装においては OpenSSL の暗号化関数のみを対象としているが、OpenSSL 以外の暗号化ライブラリにも容易に同じ手法を適用することができる。

Collector

Collector は Capturer が取得した平文データに対して、元ファイルの絶対パスと元ファイル上のオフセットを取得する機能を持つ。上述したように Collector がアクセスできるのは平文データのポインタのみであり、ポインタから絶対パスやオフセットを直接取得することはできない。そこで本研究では、FD (File Descriptor) を仲介して必要なデータを取得する方法

表 6.1: Implementation environment for the proposed method.

Linux カーネル	6.3.0-060300-generic
CPU アーキテクチャ	amd64
clang	14.0.0-1ubuntu1.1
Go コンパイラ	go1.22.5 linux/amd64

を採用した. `read(2)` システムコールは FD とバッファのポインタを引数として受け取るため, `read(2)` をフックすることでポインタと FD の対応関係を取得することができる. この対応関係を参照し, `open(2)` システムコール (FD を返り値として戻す) のフックから相対または絶対パスを, `lseek(2)` および `read(2)` システムコール (FD を引数に取り, FD に紐づくファイルのオフセットを更新する) のフックからオフセットを取得可能である. `open(2)`, `read(2)`, `lseek(2)` の関数シグネチャをコード 6.1 に示す. より詳細には, 以下の手順で FD の値から元ファイルの絶対パスとオフセットを取得する.

元ファイルの絶対パス: `open(2)` のフックにより FD とファイルパスの関係が得られるので, FD を key とし, ファイルパスを value とするハッシュマップにこの関係を保存する. `open(2)` に渡された引数が相対パスであった場合, プロセスの CWD (Current Working Directory) と結合して絶対パスを取得する. CWD は Linux カーネルにおいて各ユーザプロセスの状態を管理するカーネル内変数である `task_struct` を介して取得する.

元ファイル上のオフセット: `lseek(2)` および `read(2)` システムコールをフックする. これらのシステムコールは FD を引数に取り, `lseek(2)` はシステムコール完了時のオフセットを, `read(2)` はシステムコールによって読み込まれたバイト数を返す. この性質を利用して, FD を key とし, オフセットを value とするハッシュマップにオフセットを保存する. `open(2)` のフックによりファイルが開かれた際にハッシュマップに `key = 0`, `value = FD` のエントリを追加しておき, `lseek(2)` および `read(2)` が実行された際に value を更新しておく.

Listing 6.1: Signatures of the related system calls. The comment lines indicate the return value on success / error.

```

1  // a file descriptor / -1
2  int open(const char *pathname, int flags, ...
3          /* mode_t mode */ );
4
5  // the number of bytes read / -1
6  ssize_t read(int fd, void buf[.count], size_t count);
7
8  // the resulting offset location in bytes / -1
9  off_t lseek(int fd, off_t offset, int whence);

```

6.1.2 Evacuation Module

Evacuation Module は、Capturer によってリングバッファにデータが書き込まれるたびに呼び出しを行い、平文データ、元ファイルの相対パスと CWD、オフセットを取得する。その後絶対パスを構成し、絶対パスで指定されるファイルが Data Shelter 内に存在しなければ設計で示したようにファイルを作成する。そしてオフセットで指定される位置に平文データを書き込む。

eBPF プログラムである Capturer がリングバッファにデータを書き込むと、本稿の実装において Evacuation Module は以下の 3 つの処理を順に行う。

- Read : data availability notification をカーネル空間から受信し、リングバッファからバイナリデータを読み込む。
- Decode : リングバッファから取得したバイナリデータを、平文データ、元ファイルの相対パスと CWD、オフセットにデコードする。
- Write : 平文データを元ファイルの絶対パスとオフセットを用いて Data Shelter に書き込む。

先行研究 [58] では上記の処理を逐次的に実行していたが、実験結果からランサムウェアのデータ暗号化の速度に対して Evacuation Module のデータ退避スループットが不十分であることがわかった。さらに予備実験から、上記 3 つの処理の中で Decode 処理がボトルネックとなっていることが判明した。これらの実験結果は 7.1.2 節で報告している。

本研究では、処理のパイプライン化と並列化によって Evacuation Module の性能を向上させることを目指した。図 6.1 に高速化を施した Evacuation Module の実装を示す。まず Read, Decode, Write の各処理を個別のスレッドで実行しスレッド間で処理すべきデータを授受するように実装して、パイプライン処理を実現した。なお、ここでいうスレッドは OS のスレッドではなく、Evacuation Module の実装に使用した Go 言語の機能である goroutine [59] のスレッドを指す。スレッド間の通信には Go 言語の機能である channel を用いた。パイプライン処理によって、Evacuation Module のスループットはボトルネックである Decode 処理に律速される。予備実験では Decode 処理が Read 処理や Write 処理と比較して 4 倍程度の時間を要していたため、Decode 処理を並列化することでボトルネックの解消を行った。並列化によって Decode 1 回あたりの実行的な処理時間を短縮させた。

図 6.1 にて示したように、Decode 処理の並列化によってデータはリングバッファから読み出された順番で Write 処理に渡されることが保証されない。つまりデータのリオダが発生するため、Write 処理のスレッドが Decode 処理のスレッドから受け取ったデータをそのままの順番で Data Shelter に書き込むとファイルデータが正しく復元されない。本研究の実装では Write 処理において平文データの元ファイル上のオフセットが利用可能であるため、オフセットで指定される位置にファイルポインタを移動してからデータを書き込むことでこの問題に対処した。

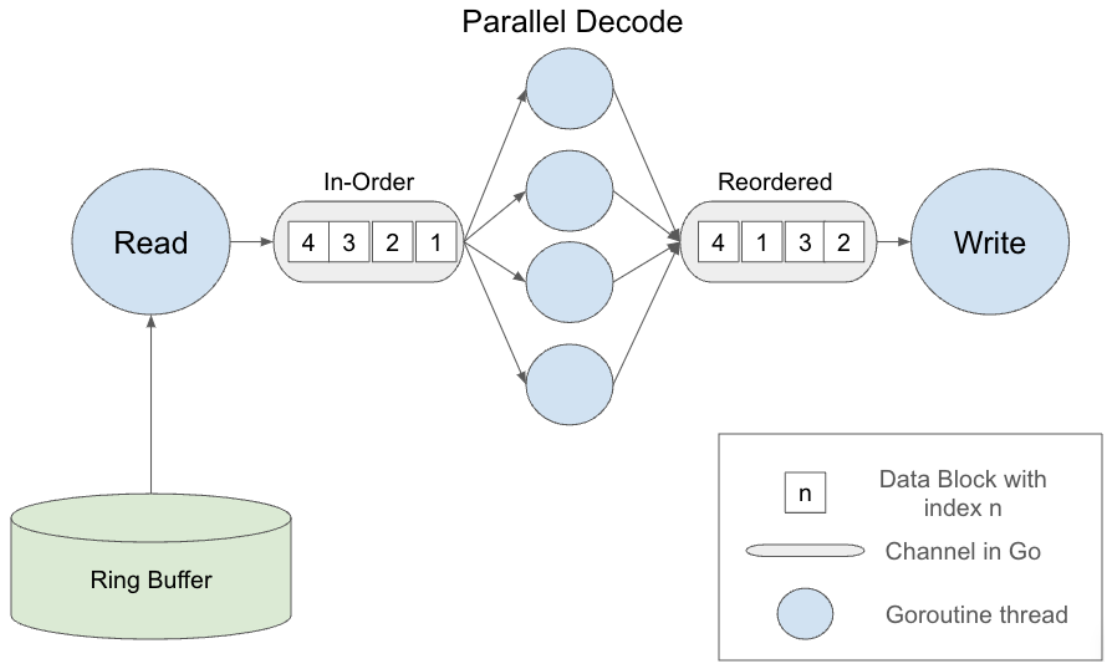


図 6.1: Implementation of the Evacuation Module with parallelization and pipelining. The indices shown in the diagram are used only for illustrative purposes to indicate the order of data blocks and to visualize the reordering process. In the actual implementation, such indices are not explicitly assigned.

6.1.3 Data Shelter

本研究の実装では、root 権限をもつユーザのみ読み書きできるディレクトリを作成し、これを Data Shelter として扱った。ユーザ権限で実行されるランサムウェアはこのディレクトリのファイルにアクセスできないため、ランサムウェアから一定レベルの隔離が実現されている。

第 7 章

評価

ランサムウェアによる暗号化からデータを保護する性能と、Fuga 稼働時のオーバーヘッドを評価した。評価はすべて表 7.1 に示す物理マシン上で行った。

7.1 データ保護性能の評価

7.1.1 実験シナリオ

指定されたパスのファイルを read し、OpenSSL 内の関数で暗号化したのち新規ファイルに write するプログラムを C 言語で実装した。暗号化方式は AES256 を使用し、鍵長と IV 長はそれぞれ 32B と 16B とした。read などの処理は 4096B のバッファに対して行うようにした。このプログラムをこれ以降「暗号化プログラム」と称する。

OpenSSL が提供するコマンドを使用して、指定したバイト長の擬似乱数を含むファイルを生成する。そして生成したファイルに対して以下の手続きを行う。

1. 実装した提案手法を実行する。
2. 暗号化プログラムでファイルを暗号化する。
3. 元のファイルと、Data Shelter 内に作成されたファイルを用いて評価を行う。

暗号化の対象となるファイルを「元ファイル」、Data Shelter 内に作成されるファイルを「退避ファイル」と呼ぶ。

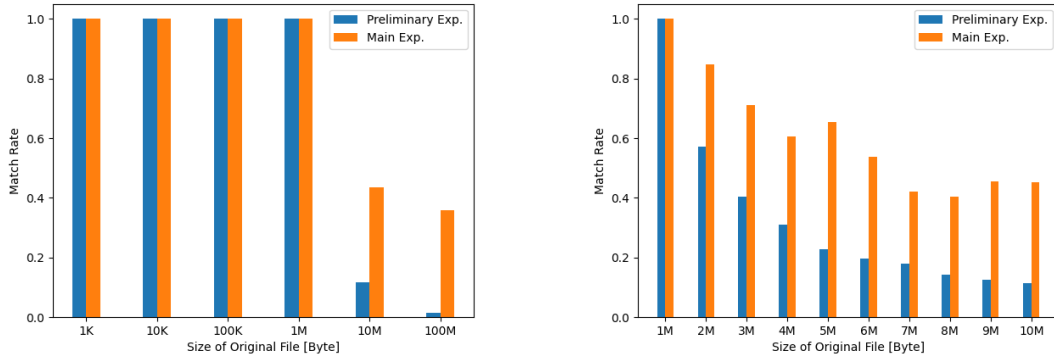
この実験におけるデータ保護性能の評価指標として、一致率を定義した。一致率は、退避ファイルと元ファイルがどの程度一致しているかを表す指標である。元ファイルと退避ファイルのデータをバイト値の配列とみなし、先頭から配列の値を調べて一致している個数を数え、その個数を元ファイルのバイトサイズで割ると一致率が求められる。元ファイルが [1, 2, 3, 4, 5]、退避ファイルが [1, 2, 4] の場合、先頭の 1 と 2 が一致しているのでこの時の一致率は $\frac{2}{5} = 0.4$ である。

表 7.1: Environment of the physical machine used for the evaluation.

OS	Ubuntu 22.04.4 LTS
ファイルシステム	ext4
カーネル	6.3.0-060300-generic
CPU	Intel Xeon Silver 4314 @ 2.400GHZ × 64
RAM	512GiB

7.1.2 予備実験の結果

先行研究 [58] で示した実装をによる提案手法の一致率を予備実験で評価した．結果を図 7.1a および図 7.1b に示す．Evacuation Module が実行する 3 つの処理 Read, Decode, Write のそれぞれの処理時間の分布を図 7.2 に示す．



(a) Match rates by original file size of 1KB, 10KB, ..., 100MB. (b) Match rates by original file size of 1MB, 2MB, ..., 10MB.

図 7.1: Match rates by original file size, comparing results from the preliminary and main experiments. The horizontal axis in the left figure uses a common logarithmic scale.

7.1.3 元ファイルのサイズに対する一致率の変化

本稿では先行研究 [58] と同様に, 1KB, 10KB, ..., 100MB のサイズ of 元ファイルに対して実験を行い, 一致率を計算した. Decode 処理の並列度は 4 に, リングバッファのサイズは 1MiB に設定した. その結果を図 7.1a に示す. また, 1MB から 10MB まで 1MB 刻みのサイズのファイルを生成し実験を行った. 並列度とリングバッファの設定は同一である. 結果を図 7.1b に示す. 先行研究 [58] で示された一致率からの向上が確認され, 高速化, すなわち Evacuation Module のパイプライン化とデコード処理の並列化, が保護性能の改善に寄与し

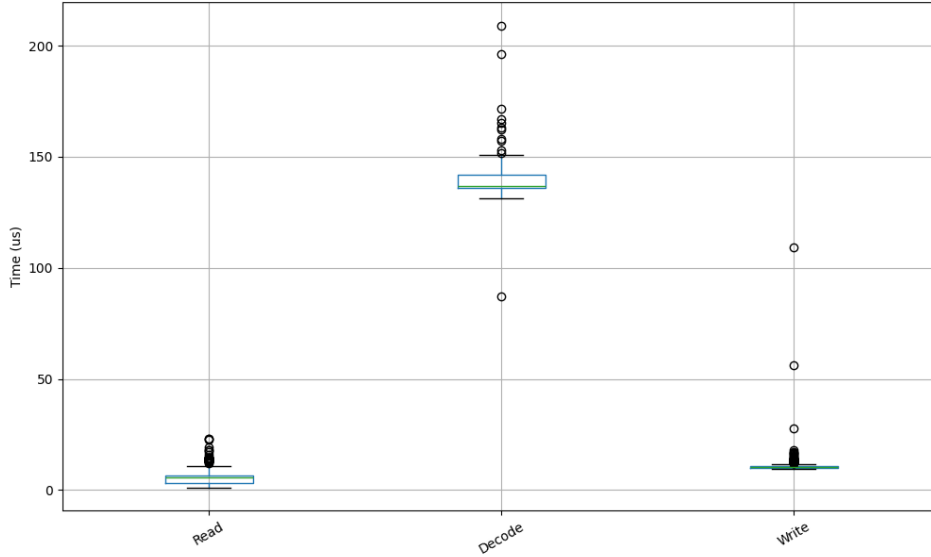


図 7.2: Boxplot of processing times for the three operations in the Evacuation Module: Read, Decode, and Write. Entries with processing times exceeding 1000 μ s were excluded; only 2 out of 976 total entries were removed.

たことがわかった。性能改善の程度は元ファイルのサイズが大きいほど顕著であり、100MB の元ファイルに対しては 20 倍以上の差が見られた。

7.1.4 パラメータの変更に対する一致率の変化

本稿で示した実装は、デコード処理の並列度 ($= p$) とリングバッファのサイズ ($= |RB|$) をパラメータとして持つ。2つのパラメータのうち一方を固定しもう一方を変化させて実験を行い、各パラメータが一致率に与える影響を調査した。

p を固定し $|RB|$ を 2MiB から 2 倍ずつ大きくした時の一致率の変化を図 7.3 に示す。 $|RB|$ より小さいファイルは完全に保護することができるが、 $|RB|$ より大きいファイルに対しては各 $|RB|$ 間で類似した傾向で保護性能が低下する。

$|RB|$ を固定し p を様々な値に設定した時の一致率の変化を図 7.4 に示す。 p を増加させるほど、完全に保護可能なファイルのサイズは増加する傾向が見られる。元ファイルのサイズの増加に対して一致率は単調に減少すると期待されるがそうっておらず、並列処理スレッドのスケジューリングの状況に依存して性能が変化する可能性がある。

パラメータ p を大きくすると多数のスレッドが生成されるため高いコア数が求められ、CPU 使用率が増加する。一方、 $|RB|$ を増やすことはカーネル空間のメモリ消費を伴うが、32MiB 程度の設定は現実的であり、計算リソースに対する保護性能の向上効果が大きいと我々は考え

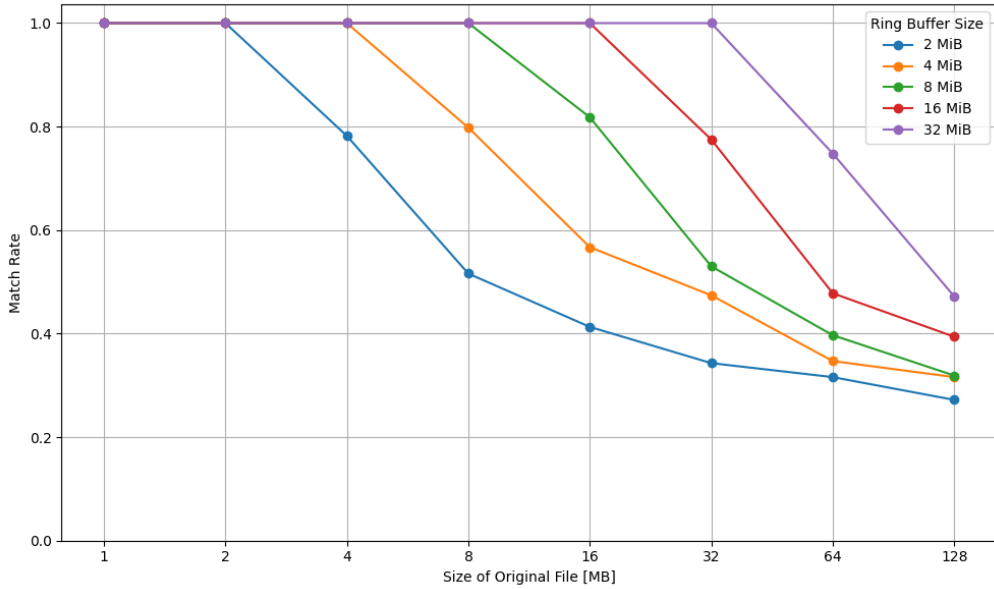


図 7.3: Variation of match rate by file size for different ring buffer sizes. The horizontal axis uses a logarithmic scale (base 2).

る. もちろん p と $|RB|$ の両方を同時に大きくすることも可能である.

しかしながら, パラメータを調整しても 100MB~数 GB オーダーの大きさのファイルを完全に保護することは現実的ではない. 頻繁に更新されないファイルならば定期的なスナップショットを利用してバックアップを確保することが望ましく, 保護したいファイルの大きさに応じて適切な戦略が異なるといえる.

7.2 オーバーヘッドの評価

7.2.1 CPU 使用率と I/O wait

本稿で示す実装では, Data Shelter として使用されているディレクトリに大量にデータの書き込みが発生する. また, ランサムウェアが高速にファイルを暗号化する場合, Process Monitor および Evacuation Module は高い頻度でデータを処理する必要がある. したがって提案手法は CPU 使用率およびディスク I/O にオーバーヘッドを発生させることが予想されるため, これらのメトリクスを計測した.

メトリクスの取得には `iostat` を利用した. `iostat` はユーザ空間およびカーネル空間の CPU 使用率の平均値, I/O wait 率の平均値, ブロックデバイスごとの read/write の量などを出力するコマンドである, 本研究では 1 秒ごとにデータを記録する設定を用いた. 汎用的なマクロベンチマークのセットである `sysbench` [60] の `fileio` ベンチマークを使って 16MiB のファイルを 128 個生成し, 暗号化プログラムによって各ファイルを暗号化することでワークロードを発生させた. 暗号化プログラムによる暗号化を開始した時点をも $t = 0$ とし, 30 秒の

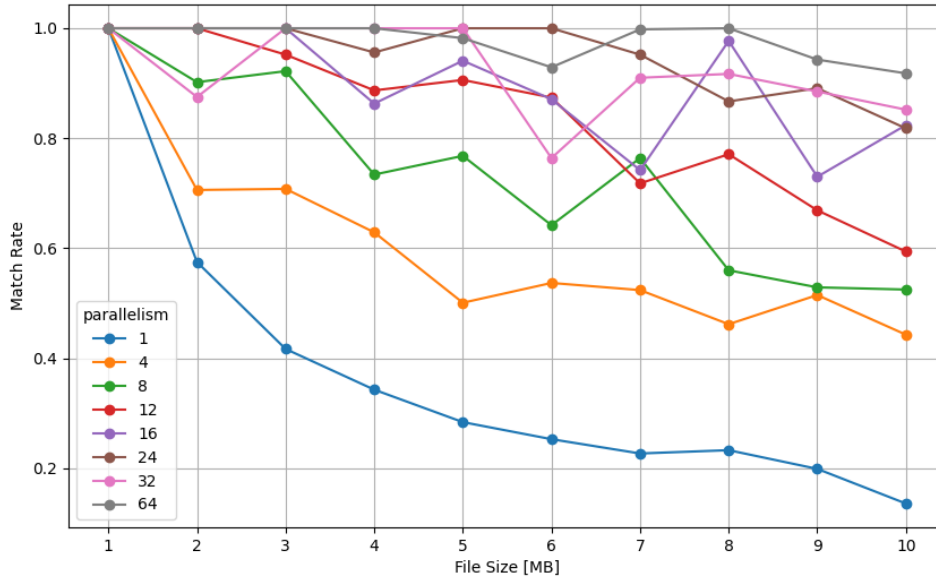


図 7.4: Variation of match rate by original file size for different degrees of parallelism p .

間 `iostat` の出力を続けた。

システム全体のベースライン (A), Fuga を使用せずに暗号化を行なった場合 (B), Fuga を使用して暗号化を行なった場合 (C) の 3 パターンで計測を行った。計測間のばらつきを抑えるために、計測は各パターンで 5 回ずつ行い平均値を計算した。

CPU 使用率の推移を図 7.5 に、I/O wait 率の推移を図 7.6 に示す。CPU 使用率は 1.5% 増加しており、Fuga のデータ退避処理が CPU リソースを多く消費することがわかる。一方 I/O wait 率については、一時的な増加は見られるものの増加幅は最大でも 0.05% 程度であり、Fuga がディスク I/O に与えるオーバーヘッドは非常に小さい。全体として、Fuga では CPU が主なボトルネックであると考えられる。

7.2.2 データ退避スループットの推定

前節で述べたパターン (C) では Fuga による書き込みが発生するが、パターン (B) では発生しない。したがってパターン (B) および (C) での書き込みスループットの差を取ることで、Fuga がファイルを退避させるスループットを推定することができる。30 秒ごとの `iostat` の出力を 5 回取得し、統計値を計算した。

表 7.2 に示す結果から、本稿で示した Fuga の実装は 19MB/s 程度のスループットでデータを退避させることができると示唆される。Huang ら [16] が行ったランサムウェア対策の開発の事前調査を参照すると、ランサムウェアによるファイル暗号化のスループットは 1MB/s から 4MB/s 程度であると計算される。したがって Fuga はランサムウェアによる暗号化よりも高速にデータを退避することができ、ランサムウェアのプロセスが検知技術によって特定お

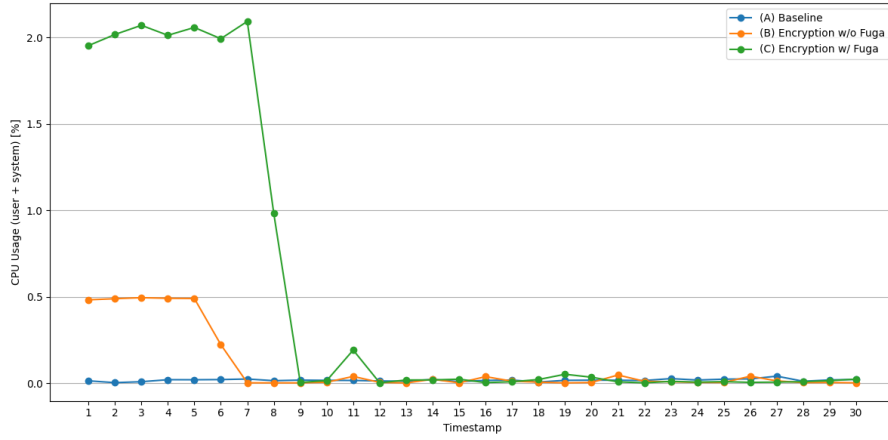


図 7.5: The transition of CPU usage.

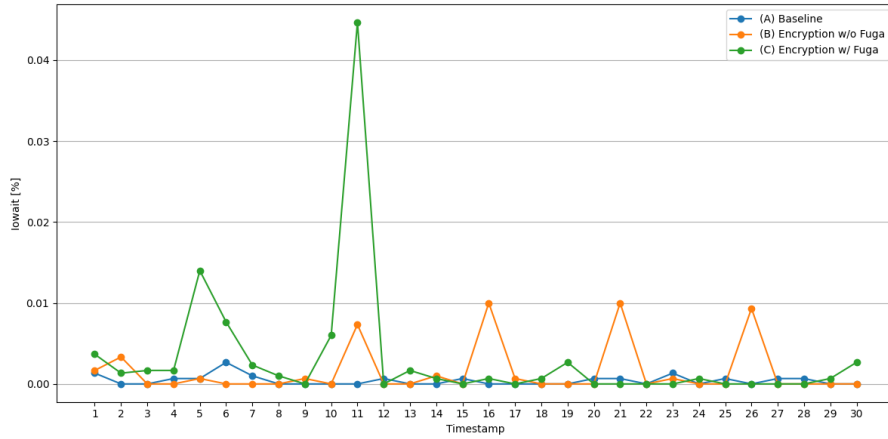


図 7.6: The transition of I/O wait.

表 7.2: Estimated performance metrics in MB/s.

Metric	Value [MB/s]
Mean	18.65
Median	117.53
Std Dev	825.78

よび停止されるまでの間ファイルデータを保護する性能を持つ。

表 7.2 に示す結果より，本稿で示した Fuga の実装は約 19MB/s のスループットでデータを退避できることが示された．一方，Huang ら [16] の調査に基づく計算によれば，ランサムウェアによるファイル暗号化のスループットは 1MB/s ～ 4MB/s 程度である．これにより，Fuga はランサムウェアが起動してから検知されるまでの間に発生する暗号化に対して，ファイルデータを退避によって保護できる性能を備えているといえる。

発表文献と研究活動

- (1) 手塚尚哉, 宮本大輔, 明石邦夫, 落合 秀也, ”ファイルの侵害をフックすることによるランサムウェアからのデータ保護システム”. CSS2024, <https://conferenceservice.jp/registration/css2024/mypage/proceedings/IPSJCSS-2024150.pdf>, 2024.10.24.

参考文献

- [1] Gourav Nagar. The evolution of ransomware: Tactics, techniques, and mitigation strategies. *Valley International Journal Digital Library*, pp. 1282–1298, 2024.
- [2] Kenan Begovic, Abdulaziz Al-Ali, and Qutaibah Malluhi. Cryptographic ransomware encryption detection: Survey. *Computers & Security*, Vol. 132, p. 103349, 2023.
- [3] SpyCloud. 2024 malware & ransomware defense report — spycloud. <https://spycloud.com/resource/2024-malware-ransomware-defense-report/>. (Accessed on 12/09/2024).
- [4] SOPHOS. 2024 Ransomware Report: Sophos State of Ransomware. <https://www.sophos.com/en-us/content/state-of-ransomware>. (Accessed on 11/21/2024).
- [5] comparitech. Ransomware group blacksuit adds kadokawa corporation to its data leak site - 1.5tb of data allegedly stolen - comparitech, 7 2024. [Online; accessed 2024-12-10].
- [6] Sophos Ltd. Mitre att&ck - mdr documentation, 1 2023. [Online; accessed 2024-12-17].
- [7] Eugene Kolodenker, William Koch, Gianluca Stringhini, and Manuel Egele. Pay-break: Defense against cryptographic ransomware. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 599–611, 2017.
- [8] Continella Andrea, et al. ShieldFS: A Self-healing, Ransomware-aware Filesystem. 2016.
- [9] eBPF.io. What is eBPF? An Introduction and Deep Dive into the eBPF Technology. <https://ebpf.io/what-is-ebpf/>. (Accessed on 11/19/2024).
- [10] Harun Oz, Ahmet Aris, Albert Levi, and A Selcuk Uluagac. A survey on ransomware: Evolution, taxonomy, and defense solutions. *ACM Computing Surveys (CSUR)*, Vol. 54, No. 11s, pp. 1–37, 2022.
- [11] Virus Bulletin. Trojan horse: Aids information introductory diskette version 2.0. <https://www.virusbulletin.com/uploads/pdf/magazine/1990/199001.pdf>. (Accessed on 12/08/2024).
- [12] Wikipedia. Pgp coder - wikipedia. <https://en.wikipedia.org/wiki/PGPCoder>. (Accessed on 12/08/2024).

- [13] Mingcan Cen, Frank Jiang, Xingsheng Qin, Qinghong Jiang, and Robin Doss. Ransomware early detection: A survey. *Computer Networks*, Vol. 239, p. 110138, 2024.
- [14] SOPHOS. The current state of ransomware: Ctb-locker — sophos news. <https://news.sophos.com/en-us/2015/12/31/the-current-state-of-ransomware-ctb-locker/>, 2015. (Accessed on 12/08/2024).
- [15] Dermot Byrne and Christina Thorpe. Jigsaw: an investigation and countermeasure for ransomware attacks. In *European Conference on Cyber Warfare and Security*, pp. 656–665. Academic Conferences International Limited, 2017.
- [16] Jian Huang, Jun Xu, Xinyu Xing, Peng Liu, and Moinuddin K Qureshi. Flashguard: Leveraging intrinsic flash properties to defend against encryption ransomware. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 2231–2244, 2017.
- [17] Cloudflare. What are Petya and NotPetya? — Ransomware attacks — Cloudflare. <https://www.cloudflare.com/learning/security/ransomware/petya-notpetya-ransomware/>. (Accessed on 11/22/2024).
- [18] 警視庁. 令和 5 年におけるサイバー空間をめぐる脅威の情勢等について. https://www.npa.go.jp/publications/statistics/cybersecurity/data/R5/R05_cyber_jousei.pdf. (Accessed on 11/23/2024).
- [19] ISO. Iso/iec 27001:2022 - information security management systems. <https://www.iso.org/standard/27001>, 2022. (Accessed on 12/09/2024).
- [20] Pranshu Bajpai, Aditya K Sood, and Richard Enbody. A key-management-based taxonomy for ransomware. In *2018 APWG Symposium on Electronic Crime Research (eCrime)*, pp. 1–12. IEEE, 2018.
- [21] Kevin Liao, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Behind closed doors: measurement and analysis of CryptoLocker ransoms in Bitcoin. In *2016 APWG symposium on electronic crime research (eCrime)*, pp. 1–13. IEEE, 2016.
- [22] Maxat Akbanov, Vassilios G Vassilakis, Ioannis D Moscholios, and Michael D Logothetis. Static and dynamic analysis of WannaCry ransomware. In *Proc. IEICE Inform. and Commun. Technol. Forum ICTF 2018*, 2018.
- [23] Qublai K Ali Mirza, Martin Brown, Oliver Halling, Louie Shand, and Abu Alam. Ransomware analysis using cyber kill chain. In *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 58–65. IEEE, 2021.
- [24] TrendMicro. Darkside on linux: Virtual machines targeted — trend micro (us). https://www.trendmicro.com/en_us/research/21/e/darkside-linux-vms-targeted.html, May 2021. (Accessed on 12/09/2024).
- [25] vmware. Vmware vsphere — virtualization platform. <https://www.vmware.com/products/cloud-infrastructure/vsphere>. (Accessed on 12/09/2024).

- [26] Zhongyu Wang, Yaheng Song, Erci Xu, Haonan Wu, Guangxun Tong, Shizhuo Sun, Haoran Li, Jincheng Liu, Lijun Ding, Rong Liu, et al. Ransom access memories: Achieving practical ransomware protection in cloud with {DeftPunk}. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 687–702, 2024.
- [27] Zscaler. 2023 threatlabz state of ransomware report — zscaler. <https://info.zscaler.com/resources-industry-reports-2023-threatlabz-ransomware-report>, 2023. (Accessed on 12/09/2024).
- [28] Mattias Weckstén, Jan Frick, Andreas Sjöström, and Eric Järpe. A novel method for recovery from crypto ransomware infections. In *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, pp. 1354–1358. IEEE, 2016.
- [29] David R Matos, Miguel L Pardal, Georg Carle, and Miguel Correia. RockFS: Cloud-backed file system resilience to client-side attacks. In *Proceedings of the 19th International Middleware Conference*, pp. 107–119, 2018.
- [30] CyberReason. Ransomware: True cost to business 2024. <https://www.cybereason.com/blog/ransomware-true-cost-to-business-2024>. (Accessed on 12/09/2024).
- [31] The Guardian. Massive ransomware cyber-attack hits nearly 100 countries around the world — cybercrime — the guardian, 2017. [Online; accessed 2024-12-10].
- [32] Kaspersky. Ransomware WannaCry: All you need to know. [Online; accessed 2024-12-10].
- [33] CloudGate. コロニアル・パイプライン社へのランサムウェア攻撃、その背景とは | セキュリティニュース | cloudgate クラウドゲート. [Online; accessed 2024-12-12].
- [34] 国土交通省. 名古屋港コンテナターミナルのサイバー攻撃におけるインシデント対応について, 7 2023. [Online; accessed 2024-12-10].
- [35] 名古屋港管理組合. 日本一の名古屋港 | 名古屋港管理組合公式ウェブサイト. [Online; accessed 2024-12-10].
- [36] KADOKAWA Corporation. Notification and Apology Concerning Information Leakage caused by the Ransomware Attack, 7 2024. [Online; accessed 2024-12-10].
- [37] MITRE. Mitre att&ck®. [Online; accessed 2024-12-17].
- [38] Eduardo Berrueta, Daniel Morato, Eduardo Magaña, and Mikel Izal. A survey on detection techniques for cryptographic ransomware. *IEEE Access*, Vol. 7, pp. 144925–144944, 2019.
- [39] Timothy McIntosh, ASM Kayes, Yi-Ping Phoebe Chen, Alex Ng, and Paul Watters. Ransomware mitigation in the modern era: A comprehensive review, research challenges, and future directions. *ACM Computing Surveys (CSUR)*, Vol. 54, No. 9, pp. 1–36, 2021.
- [40] James Baldwin and Ali Dehghantanha. Leveraging support vector machine for opcode

- density based detection of crypto-ransomware. *Cyber threat intelligence*, pp. 107–136, 2018.
- [41] Amin Kharaz, Sajjad Arshad, Collin Mulliner, William Robertson, and Engin Kirda. {UNVEIL}: A {Large-Scale}, automated approach to detecting ransomware. In *25th USENIX security symposium (USENIX Security 16)*, pp. 757–772, 2016.
 - [42] Amin Kharraz and Engin Kirda. Redemption: Real-time protection against ransomware at end-hosts. In *Research in Attacks, Intrusions, and Defenses: 20th International Symposium, RAID 2017, Atlanta, GA, USA, September 18–20, 2017, Proceedings*, pp. 98–119. Springer, 2017.
 - [43] May Medhat, Samir Gaber, and Nashwa Abdelbaki. A new static-based framework for ransomware detection. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pp. 710–715. IEEE, 2018.
 - [44] Amjad Alraizza and Abdulmohsen Algarni. Ransomware detection using machine learning: A survey. *Big Data and Cognitive Computing*, Vol. 7, No. 3, p. 143, 2023.
 - [45] SungHa Baek, Youngdon Jung, Aziz Mohaisen, Sungjin Lee, and DaeHun Nyang. Ssd-insider: Internal defense of solid-state drive against ransomware with perfect data recovery. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 875–884. IEEE, 2018.
 - [46] Yu-Lun Wan, Jen-Chun Chang, Rong-Jaye Chen, and Shiuh-Jeng Wang. Feature-selection-based ransomware detection with machine learning of data analysis. In *2018 3rd international conference on computer and communication systems (ICCCS)*, pp. 85–88. IEEE, 2018.
 - [47] May Medhat, Samir Gaber, and Nashwa Abdelbaki. A new static-based framework for ransomware detection. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*, pp. 710–715. IEEE, 2018.
 - [48] Shagufta Mehnaz, Anand Mudgerikar, and Elisa Bertino. Rwguard: A real-time detection system against cryptographic ransomware. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 114–136. Springer, 2018.
 - [49] Manaar Alam, Sarani Bhattacharya, Swastika Dutta, Sayan Sinha, Debdeep Mukhopadhyay, and Anupam Chattopadhyay. Ratafia: Ransomware analysis using time and frequency informed autoencoders. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 218–227. IEEE, 2019.
 - [50] Calvin Brownor, Patrick Andersen, Zachary Fischer, and Gabriel Osterberg. Ransomware detection using dynamic anomaly matrix for accurate and real-time threat

- identification. 2024.
- [51] OpenZFS. openzfs/zfs: OpenZFS on Linux and FreeBSD. <https://github.com/openzfs/zfs>. [Online; accessed 2024-12-22].
 - [52] The Linux Kernel. Btrfs — the linux kernel documentation. <https://docs.kernel.org/filesystems/btrfs.html>. [Online; accessed 2024-12-22].
 - [53] ZenServer Product Documentation. Vm snapshots — citrix hypervisor 8.2. <https://docs.xenserver.com/en-us/citrix-hypervisor/dr/snapshots.html>. [Online; accessed 2024-12-22].
 - [54] Microsoft. Create an Azure snapshot of a virtual hard disk - Azure Virtual Machines — Microsoft Learn. <https://learn.microsoft.com/en-us/azure/virtual-machines/snapshot-copy-managed-disk?tabs=portal>, 12 2024. [Online; accessed 2024-12-22].
 - [55] HUAWEI. Ransomware Protection Storage Solution — Huawei Enterprise. <https://e.huawei.com/en/solutions/storage/oceanprotect/ransomware>,. [Online; accessed 2024-12-23].
 - [56] Jaehyun Han, Zhiqiang Lin, and Donald E Porter. On the effectiveness of behavior-based ransomware detection. In *Security and Privacy in Communication Networks: 16th EAI International Conference, SecureComm 2020, Washington, DC, USA, October 21-23, 2020, Proceedings, Part II 16*, pp. 120–140. Springer, 2020.
 - [57] 榎本秀平ほか. ランサムウェアに対する破壊的書き込みの監視による仮想ディスク保護機構. コンピュータセキュリティシンポジウム 2024 論文集, pp. 1109–1116, 2024.
 - [58] 手塚尚哉ほか. ファイルの侵害をフックすることによる ランサムウェアからのデータ保護システム. コンピュータセキュリティシンポジウム 2024 論文集, pp. 1117–1125, 2024.
 - [59] Go. The Go Memory Model - The Go Programming Language. <https://go.dev/ref/mem>,, 6 2022. [Online; accessed 2024-12-27].
 - [60] akopytov. akopytov/sysbench: Scriptable database and system performance benchmark. <https://github.com/akopytov/sysbench>. (Accessed on 11/19/2024).
 - [61] Steven McCanne and Van Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX winter*, Vol. 46, pp. 259–270. Citeseer, 1993.
 - [62] Isovalent. Learning eBPF. [Online; accessed 2024-12-12].

謝辞

ありがとうございました！

付録 A

eBPF

A.1 歴史的背景

A.1.1 Berkeley Packet Filter

Steven と Van [61] は 1993 年に、Unix 系の OS 上でパケットキャプチャを効率的に行うためのアーキテクチャである Berkeley Packet Filter を提案した。これ以降、Berkeley Packet Filter を「BPF」と表記する。BPF が登場する以前のパケットキャプチャでは、カーネル空間で取得したパケットをすべてユーザー空間にコピーしてからフィルタリングしていて、これが無駄なオーバーヘッドの原因となっていた。[61] では特殊な 32 ビットの命令セットを解釈してパケットフィルタリングを行う擬似マシン (BPF pseudo-machine) を提案し、この擬似マシンをカーネル空間で動作させることでオーバーヘッドを軽減することを目指した。

BPF は Linux カーネルの v2.1.75 にて Linux Socket Filter として導入され、`tcpdump` コマンドの高速化に利用された。既存のシステムとの比較では、BPF は最大で 20 倍程度高速にパケットキャプチャを行うことができた。

A.1.2 eBPF: extended BPF

BPF は Linux カーネルの v3.18 にて大幅に拡張が行われ、extended BPF すなわち eBPF と呼ばれるようになった。eBPF 以前の BPF を明確に区別するために cBPF (classical BPF) と呼ぶことがある。拡張された項目は多岐にわたるが、代表的な項目を以下に示す。

- BPF 命令セットが 32 ビットから 64 ビットに書き直され、実行効率が向上した。
- eBPF map というデータ構造が導入され、ユーザ空間とカーネル空間の間でのデータ共有や、eBPF プログラム内のデータ保存またはプログラム間でのデータ共有が可能になった。
- eBPF プログラムが安全に実行できることを検証する eBPF verifier が追加された。

上述した拡張に加えて、eBPF はその適用範囲を大きく拡大してきた。ネットワーク分野においては、Linux ネットワークスタック内の様々なレイヤー、例えばトランスポート層 (e.g.

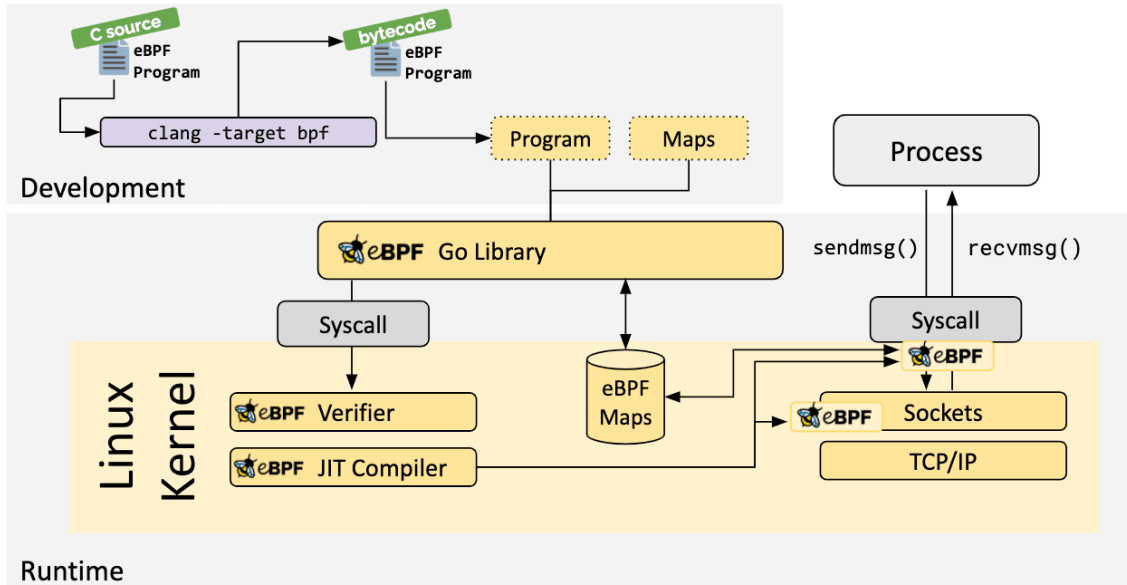


図 A.1: Overview of eBPF system. [9]

TCP/UDP) やデータリンク層 (e.g. ネットワークデバイスドライバ) などを対象に処理を記述することが可能となった. さらに eBPF はパフォーマンスレーシングやセキュリティ機能の向上にも活用されるようになっていく. これに伴い, 「BPF」という用語は元々の「Berkeley Packet Filter」という意味を超えて, 独立したフレームワークを指す言葉として認識されるようになった.

A.2 eBPF のアーキテクチャ

eBPF システムの概要を図 A.1 に示す. 特殊な C 言語で記述された eBPF プログラムは BPF バイトコードにコンパイルされ, bpf システムコールを通じてカーネルにロードされる. ロードされた eBPF プログラムは eBPF verifier によって検証され, 安全性が確認された後に JIT コンパイルされ, カーネル内で実行される. 本研究における Fuga の実装に関連性の高い要素について詳述する.

A.2.1 イベント駆動型アーキテクチャ

eBPF はイベント駆動型のアーキテクチャを採用している. すなわち, 開発者はカーネル空間またはユーザ空間で発生する特定のイベントを eBPF プログラムでフックし, イベント発生時にプログラムが実行されるようにカーネルの動作を拡張することができる. eBPF プログラムでイベントをフックすることを「eBPF プログラムをアタッチする」と表現するが, プログラムのロードおよびアンロード, アタッチおよびデタッチは動的に行えるように実装されている.

eBPF にはフックポイントという概念があり、eBPF プログラムが特定のイベントに応じて動作するために接続されるカーネルやユーザスペースの機能的なエントリポイントを指す。本研究における Fuga の実装では以下のフックポイントを利用している。

- **uprobe**：ユーザ空間プログラムが実行する関数のエントリ
- **fentry/fexit**：カーネル関数のエントリ/終了

これらのフックポイントではフックする先の関数を eBPF プログラム内に記述する。指定した関数が呼び出されると、その関数のエントリや終了時に eBPF プログラムが実行される。

A.2.2 eBPF Verifier

eBPF verifier はバイトコードに変換された eBPF プログラムを入力として受け付けて、そのプログラムがカーネル上で安全に実行できることを検証するプログラムである。verifier の検査項目としては、プログラムのメモリアクセス違反、正常終了性（無限ループする可能性があるかどうか）、スタックサイズや命令数の上限の超過などが挙げられる^{*1}。eBPF プログラムは verifier による安全性の検証を通過しない限り実行されないという点において、eBPF は一定レベルの安全性が保障されている。

A.2.3 eBPF Map

eBPF map（以下、map）は eBPF プログラムおよびユーザ空間の両方からアクセス可能なデータ構造である。map は複数の eBPF プログラム間でデータを共有したり、eBPF プログラムとユーザ空間プログラム間でデータを授受したりするために利用される。典型的なユースケースは以下の通り [62]。

- ユーザ空間にて設定情報を書き込み、eBPF プログラムがそれを取得する。
- ある eBPF プログラムがステートを保存し、後に同一プログラムまたは別のプログラムが参照する。
- eBPF プログラムがメトリクスを map に書き込み、ユーザ空間プログラムが map から取得して可視化を行う。

配列やキューなど複数のデータ構造が map として提供されているが、本研究においてはハッシュテーブルとリングバッファを利用している。eBPF におけるリングバッファは、eBPF プログラムからユーザ空間への高頻度なデータ転送に適している。

^{*1} 命令の上限数は、cBPF において 4096、eBPF において 100 万である。