# デコイファイルの動的配置による ランサムウェア被害抑制機構

榎本 秀平[1]　葛野 弘樹[1]　山田 浩史[2]　白石 善明[1]

**概要**：サーバを標的としたランサムウェアによるファイル暗号化の被害が広く確認されている．近年開発されたランサムウェアは，マルチスレッドや暗号化専用命令の採用によるファイル暗号化の高速化を実現するための設計を備えており，マルチコア CPU を搭載するサーバにおいて，短時間で多くのファイルを暗号化する．既存のランサムウェア検出機構は，振る舞いの分析時間を必要とし，検出までに一定の時間を要するため，検出される前に数多くのファイル暗号化が完了する可能性がある．本研究では，ファイル暗号化の被害を最小としながら，既存のランサムウェア検出機構がランサムウェア検出を実現するための機構を提案する．提案手法では，ランサムウェアが暗号化対象のファイルパスを取得するため，ディレクトリ配下のファイル情報を格納したデータ構造であるディレクトリエントリを要する点に着目する．既存のランサムウェア検出機構による検出がなされるまでの間，ディレクトリエントリの取得に応じてデコイファイルを動的に配置し，デコイファイルに対するファイル暗号化を許容しながら，通常ファイルに対するファイル書き込みをブロックし，通常ファイルのファイル暗号化被害を抑制する．提案手法を Linux Kernel Module として実装し，評価実験を行った．評価結果より，実際のランサムウェアによる通常ファイルに対する暗号化を抑制可能であることを確認した．

**キーワード**：ランサムウェア，ファイル暗号化，オペレーティングシステム，ソフトウェアセキュリティ

# Ransomware Encryption Mitigation using Dynamic Placement for Decoy Files

Shuhei ENOMOTO[1]　Hiroki Kuzuno[1]　Hiroshi YAMADA[2]　Yoshiaki SHIRAISHI[1]

***Abstract:*** Ransomware attacks on server machines are widely reported in the world. Latest ransomware families can encrypt many files in a short time, as they employ software design for fast file encryption on server machines. Existing ransomware detections require analysis time to identify the malicious behaviors from user processes, and many files are encrypted before the detection. This study proposes a novel software mechanism to mitigate ransomware encryption damages in the analysis time of existing ransomware detections. The proposal mechanism is based on the insight that ransomware requires directory entries to search for target files before the file encryption. We designed and implemented the software mechanism to place decoy files in response to require directory entries and block file writes to original files during accepting file encryptions to decoy files. Our experimental results show that the proposal mechanism is available to mitigate file encryption damages during analysis time in real-world ransomware attacks.

## 1. はじめに

ランサムウェアによるファイル暗号化の被害が広く確認されている．ランサムウェアが標的とするコンピュータシステムは，PC，モバイル端末，ならびにサーバホストと

[1] 神戸大学 大学院工学研究科 電気電子工学専攻
Dept. of EE, Kobe University
[2] 東京農工大学
TUAT

多岐に渡る．近年では，サーバホストの脆弱性や設定不備を突いた攻撃からインストールされたランサムウェアによる，ファイル暗号化被害が広く報告されている [1]．例として，LockBit [2] は VMware ESXi の脆弱性 [3] を突いた攻撃によりサーバホストのファイルシステムに配置される仮想マシンのディスクファイルを不正に暗号化し，復旧の見返りとして金銭を要求する．

近年開発されたランサムウェアは，ファイル暗号化にかかる時間を短縮するための設計がなされている．

- **マルチスレッドによるファイル I/O の並列化**
  実行時に複数のスレッドを生成し，ファイル暗号化を各スレッドが分担することで，一定時間あたりに暗号化可能なファイル数を増やし，すべてのファイルの暗号化にかかる時間の短縮を実現する．
- **専用命令による暗号化演算の高速化**
  暗号化のアルゴリズムを専用の CPU 命令により実装することで，汎用の CPU 命令により実装される暗号化演算と比較し，高速な暗号化演算を実現する．

ランサムウェア対策として，既存研究ではユーザプロセスの振る舞い分析によるランサムウェア検出があり，未知の種類を含めた幅広いランサムウェアファミリの検出を可能とする．また，被害発生後に焦点を当てたランサムウェア対策として，ファイル暗号化後にファイルリカバリを可能とするストレージシステムが提案されている [4–9]．既存の対策手法においてもランサムウェア検出やファイルリカバリは可能であるが，以下の課題が存在する．

- **振る舞い分析に要する時間が長い**
  ユーザプロセスの振る舞いに基づいたランサムウェア検出手法 [4–8] では，振る舞いの観測および分析を要するため，検出までの間のランサムウェア動作を許容する必要がある．例として，機械学習によるランサムウェアの動的検出手法 [8] では，ランサムウェアの起動から検出までに 30 秒 から 120 秒程度の時間を要する．近年のランサムウェアは高速なファイル暗号化が可能であるため，検出までの間に多くのファイルが暗号化される可能性がある．
- **ファイルバックアップによるオーバーヘッド**
  ファイルリカバリを可能とするストレージシステムでは，専用領域に対するバックアップを要し，実行性能およびストレージ消費量にて，オーバーヘッドが生じる．例として，ランサムウェア検出までの間に暗号化されたファイルのリカバリを実現する手法 [9] では，ファイル書き込みのたびに専用領域に対しファイルのコピーを行い，380 % 程度のランタイムオーバーヘッド，14.73 GB 程度のストレージ消費量増加が生じる．ユーザープロセスの実行性能を重視するサーバシステムにおいては，適用が困難である．

本研究では，既存のランサムウェア検出機構により検出されるまでの間，ランサムウェアによるファイル暗号化の被害を抑制するための機構である CryptoDelay を提案する．ランサムウェアはファイルシステムからファイルの探索を行うため，ディレクトリ配下のファイル情報を格納したデータ構造であるディレクトリエントリを要する．CryptoDelay はユーザプロセスから OS に対するディレクトリエントリの要求を監視，対象のディレクトリに対しデコイファイルを動的に生成および配置，デコイファイルへの暗号化を許容しながら，通常ファイルに対する変更について，一定時間のブロッキングを施行する．

CryptoDelay は既存のランサムウェア検出機構の検出精度を落とすことなく，正規のアプリケーションの実行性能およびストレージに対するオーバーヘッドを最小としながら，検出がなされるまでの間のファイル暗号化被害を抑制する．本研究の貢献を以下に示す．

(1) サーバホストにて，既存のランサムウェア検出機構が検出を行うまでの間に，ランサムウェアによるファイル暗号化被害を抑制するための新たな機構 CryptoDelay を提案した．CryptoDelay は幅広いランサムウェア検出機構との協調が可能であり，多様なランサムウェアファミリのファイル暗号化被害を抑制可能である．
(2) CryptoDelay を実現するためのソフトウェア設計を示した．CryptoDelay は OS カーネルやミドルウェアのソースコードの変更を必要とせず，サーバホストに対し容易な導入が可能である．
(3) CryptoDelay を Linux Kernel Module として実装し，セキュリティ，パーフォーマンスに関する評価を実施した．セキュリティ評価として 7 種類のランサムウェア実検体を実行し，ファイル暗号化を抑制可能であることを確かめた．加えて，UnixBench [18] を実行し，2.90 % 程度の性能劣化が生じることを確かめた．

## 2. 背景

### 2.1 ランサムウェアによる攻撃

ランサムウェアは，標的とするホストのリソースを使用不可能とし，復旧の見返りとして金銭を要求する．使用不可能とする手法は，スクリーンのロックやファイルの暗号化がある．使用不可能とすることに加え，ファイルの盗取を行い，ファイルコンテンツの公開を脅迫する種類についても確認されている．既存の調査 [10] では，ファイル暗号化を行う種類が，全ランサムウェア攻撃の 90 % 程度を占めることを報告している．

#### 2.1.1 ファイルの探索と暗号化

ファイル暗号化を行うランサムウェアは，実行中のオペレーティングシステム (OS) におけるファイルシステムから，標的とするファイルの探索，およびファイルの暗号化を行う．ファイルの探索には，あるディレクトリ配下におけるファイルリストが格納されたデータ構造であるディレ

クトリエントリを要する．OS に対し再帰的にディレクトリエントリを要求，取得したディレクトリエントリに含まれるファイルの一覧を確認する．標的とするファイルが存在する場合においては，ファイルを開き，コンテンツを読み込み，暗号化を行い，コンテンツの書き込みを行う．

例として，Linux を動作対象とした RansomEXX [11] では，getdents システムコールを用いてディレクトリエントリを取得する．取得後，ディレクトリエントリ内のファイルリストを先頭から検査し，16 バイト以上のサイズのファイルを発見した場合にて，ファイル暗号化を実施する．

#### 2.1.2 近年のランサムウェアのファイル暗号化設計

近年開発されたランサムウェアは，ファイル暗号化を高速化するための設計を備える．例として，RansomEXX2 [12] では CPU コア数分のスレッドを生成し，各スレッドが暗号化対象となるファイルの I/O を分担する．マルチコア CPU を搭載するホストにおいては，マルチスレッドを用いたファイル暗号化により，実行性能の向上が可能となる．AvosLocker [13] では Intel x86 における AES アルゴリズムの命令セット AES-NI [14] を用いたファイル暗号化を行う．AES-NI をはじめとする命令セットを持つ CPU を搭載するホストにおいては，専用の CPU 命令を用いた暗号化演算により，実行性能の向上が可能となる．

#### 2.1.3 近年のランサムウェアの動作環境

ランサムウェアが標的とするコンピュータシステムは，クライアント PC，モバイル端末，IoT 機器など多岐に渡るが，近年ではサーバホストにおける被害が広く報告されている．例として，Microsoft Active Directory の奪取によりファイルサーバをファイル暗号化の標的とする REvil [15]，Ryuk [16]，VMware ESXi の脆弱性 [3] を突いた攻撃により，仮想マシンのストレージファイルをファイル暗号化の標的とする LockBit [2]，RedAlert [17] がある．

### 2.2 既存の対策手法

振る舞いの監視による検出 [4–8] では，ユーザプロセス実行中の振る舞いを監視および分析し，悪性と識別された際には，ユーザプロセスの停止や管理者への警告を行う．監視内容として，ファイルシステムの操作や，ファイルエントロピーの変化といった項目が挙げられる．ファイルシステム操作やファイルエントロピー変化は，多くのランサムウェアファミリで類似の傾向にあり，未知の種類を含めた，幅広いランサムウェアファミリの検出を可能とする．

振る舞いの分析を行うためには，ユーザプロセスを一定時間動作させる必要があるほか，分析結果の算出にて一定時間を要する．例として，ユーザプロセスの振る舞いに基づいた機械学習によりランサムウェアを検出する手法 [8] においては，ランサムウェア起動から検出までに 30 秒から 120 秒程度の時間を要することが報告されている．

バックアップによるファイルリカバリ [9] では，ファイル暗号化被害発生前におけるファイルのバックアップを用いて，被害発生後に暗号化済みファイルを復旧させる．ストレージ内にバックアップ領域を構築し，ファイル書き込み時にバックアップ領域にファイルをコピー，被害発生判明後にバックアップ領域からファイルをロールバックする．バックアップ時に生じるパフォーマンスおよびストレージオーバーヘッドについては，許容する必要がある．

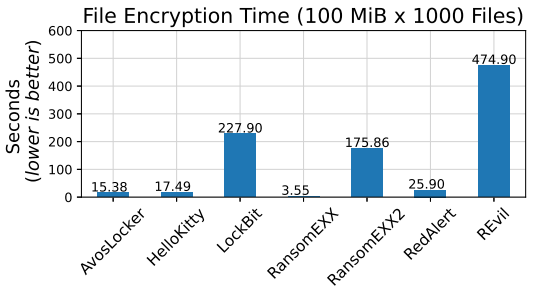


**図 1** ランサムウェア実検体におけるファイル暗号化時間: 100 MiB × 1000 ファイルの暗号化にかかる時間をランサムウェア実検体 7 種類にて計測した．Intel(R) Core(TM) i7-8700K (3.70GHz, x86_64) CPU, 48 GiB RAM における Ubuntu 24.04.2 LTS (Linux 5.4.0-153-generic) にて計測を行った．

### 2.3 ファイル暗号化時間の計測

近年のランサムウェアファミリにおけるファイル暗号化速度を示すため，ファイル暗号化にかかる時間の計測を行った．ファイルシステムに対し 100 MiB サイズのファイルを 1000 ファイル配置し，ランサムウェア実検体 7 種類にて，ユーザプロセスとして起動してからすべてのファイル暗号化が完了するまでの時間の計測を行った．

実験結果を図 1 に示す．1 秒間につき 2 ファイル (REvil) から 281 ファイル (RansomEXX) 程度の暗号化がなされていることが確認された．ユーザプロセスの振る舞いに基づいた既存の検出機構を用いて，これらのランサムウェアの検出を試みた場合，検出までの間に暗号化の被害を受けるファイルが多数生じる可能性を示している．

## 3. 脅威モデル

攻撃者はサーバホストを標的とし，ネットワークサービスの脆弱性や，盗取された一般ユーザの認証情報を用いて，標的のサーバホストに対しアクセス可能であるものとする．攻撃者は実行ファイルをファイルシステムに配置，一般ユーザ権限にて実行可能であるものとする．

本研究にて対象とするランサムウェアは，ファイル暗号化によりユーザーに対し金銭を要求する種類とし，スクリーンのロックやファイルコンテンツ公開脅迫を行うランサムウェアは，本研究における対象外の種類とする．

OS におけるカーネル空間やハードウェアは安全であるものとし，カーネルやハードウェアの脆弱性を突いた攻撃は，本研究における対象外の攻撃とする．

## 4. 提案手法

本研究では，既存のランサムウェア検出機構によるランサムウェア検出がなされるまでの間，ファイル暗号化の被害を抑制するための機構 CryptoDelay を提案する．提案手法が実現する要件を以下に示す．

**要件 1**：既存のランサムウェア検出機構の検出時間および精度に悪影響を与えることなく，検出までの間のファイル暗号化を抑制する．

**要件 2**：ストレージ内に専用のバックアップ領域を必要とせず，ストレージに対するオーバーヘッドを抑える．

**要件 3**：OS やアプリケーションの実行性能に対する影響を最小とし，軽量に動作可能とする．

要件 1 を満たすため，CryptoDelay は既存のランサムウェア検出機構がファイル暗号化の振る舞いを観測可能としながら，ファイル暗号化の被害については抑制がなされるような仕組みを設計する．

要件 2 を満たすため，CryptoDelay はファイルのバックアップを保持することなく，ファイル暗号化の被害が生じる前に，既存のランサムウェア検出機構による検出が可能となるような仕組みを設計する．

要件 3 を満たすため，ファイルの操作を行う正規アプリケーションや，サーバーアプリケーションが稼働した際に，CryptoDelay がアプリケーションに対して与える影響が最小かつ限定的となるように設計する．

### 4.1 アプローチ

ランサムウェアはファイルシステムから標的とするファイルの探索を行うため，OS に対しディレクトリエントリを再帰的に要求する．CryptoDelay はユーザプロセスから OS に対するディレクトリエントリの要求に応じて，ディレクトリエントリが示すディレクトリに対し，複数のデコイファイルを動的に配置する．デコイファイルに対するアクセスを監視し，上書きや削除といったファイル変更操作を試行したユーザプロセスについて，通常ファイルに対するファイル変更操作を一定時間ブロッキングする．

CryptoDelay の概観を図 2 に示す．CryptoDelay は既存のランサムウェア検出機構との並列動作を想定とする．ランサムウェアは通常ファイルとデコイファイルに対する暗号化を試みる．CryptoDelay はデコイファイルに対する変更を許容しながら，通常ファイルに対する変更について，一定時間のブロッキングにより保留する．既存のランサムウェア検出機構は，デコイファイルに対するファイル暗号化の振る舞いを観測し，CryptoDelay によるブロッキングが実施される時間内にて，ランサムウェア検出を実現する．既存手法と異なり，ファイルのバックアップを必要としないため，デコイファイルへの変更なく通常ファイルへ書き込んだ際におけるオーバーヘッドは生じない．

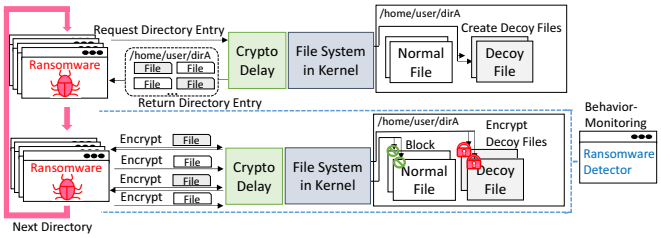


**図 2** ランサムウェアによるファイル暗号化に対する，CryptoDelay によるファイル暗号化被害抑制の概観

### 4.2 デザインチャレンジ

提案手法の実現にあたり，要求されるデザインチャレンジを以下に示す．

- **オーバーヘッドの抑制**: デコイファイルの生成は，ランタイムオーバーヘッドやストレージ消費量の増加を招く．正規のアプリケーションの実行性能やストレージへの影響を最小とする設計が求められる．
- **通常ファイルに対する変更の見逃し**: ディレクトリエントリ内のファイルの順番はファイルシステムの実装に依存する．2.1.1 節にて示した通り，ランサムウェアはディレクトリエントリ内のファイルの先頭から順番に暗号化を行う傾向にある．デコイファイルよりも前に通常ファイルが位置することで，ブロッキングが有効化される前に通常ファイルが暗号化される可能性がある．通常ファイルより先にデコイファイルが暗号化されやすくなる仕組みを設計する必要がある．
- **ランサムウェアによる回避**: CryptoDelay の設計をランサムウェア開発者が把握している状況においても，ランサムウェアから CryptoDelay に対する回避が依然として困難な設計が求められる．

## 5. 設計

CryptoDelay が導入された OS の概観を図 3 に示す．CryptoDelay はカーネル空間にて動作し，ユーザプロセスからのシステムコールに応じてファイルシステムおよびタスクスケジューラに対し操作を行う．

CryptoDelay のソフトウェアコンポーネントは，デコイファイルの配置および I/O を担う Decoy FS，ディレクトリエントリの制御を担う Directory Entry Shuffler，通常ファイルへの変更に対するブロッキングを担う Write Blocker の 3 種類より構成される．

### 5.1 デコイファイルの生成

ユーザプロセスからのディレクトリエントリ取得に応じて，Decoy FS は対象のディレクトリ配下に対してデコイファイルを生成する．

デコイファイルの生成は，対象のディレクトリ配下に存在する通常ファイルから，複数のデコイファイルとして

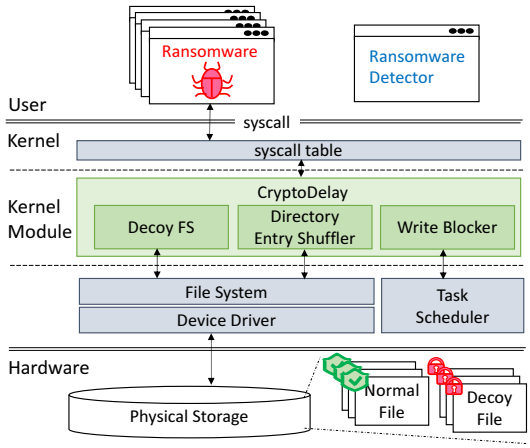


**図 3** CryptoDelay の概観

フォークするように配置される．デコイファイル生成時のオーバーヘッドを抑制するため，Decoy FS はファイルサイズが 0 のファイルとして，デコイファイルのフォークを行う．ユーザプロセスからのファイルサイズ取得に応じ，Decoy FS はデコイファイルのファイルサイズをフォーク元の通常ファイルの値に模倣を行う．

例として，1 KiB の通常ファイルが 1 ファイル配置されているディレクトリ配下に対し，ディレクトリエントリが要求された際には，Decoy FS により 0 バイトのデコイファイルが複数ファイル生成される．生成されたデコイファイルに対するファイルサイズ取得がユーザプロセスから要求された際には，Decoy FS により 1 KiB としてユーザプロセスに返される．

### 5.2 ディレクトリエントリのシャッフル

デコイファイルの生成後，ファイルシステムにより，対象のディレクトリ配下におけるディレクトリエントリの取得がなされる．

ディレクトリエントリ内のファイルの順番はファイルシステムの実装に依存し，2.1.1 節にて示した通り，ランサムウェアは先頭からファイル暗号化を行う傾向にある．デコイファイルを優先的に暗号化させるため，Directory Entry Shuffler により，ファイルの先頭がデコイファイルとなるように改変を行う．加えて，ファイルの順番からデコイファイルと通常ファイルの推測を行うことが困難となるようにするため，二番目以降のファイルについて，順番のシャッフルを行う．

### 5.3 デコイファイルの I/O

ユーザプロセスからデコイファイルに対する読み込み，および書き込みが要求された際には，Decoy FS によってファイル I/O の整合性が維持される．

読み込みは，フォーク元の通常ファイルに対し読み込み操作を行い，書き込みは，フォーク元の通常ファイルのコンテンツおよび変更内容となるコンテンツをデコイファイルに対し書き込み操作を行う．書き込み後については，デコイファイルに対しファイル I/O の発行がなされる．

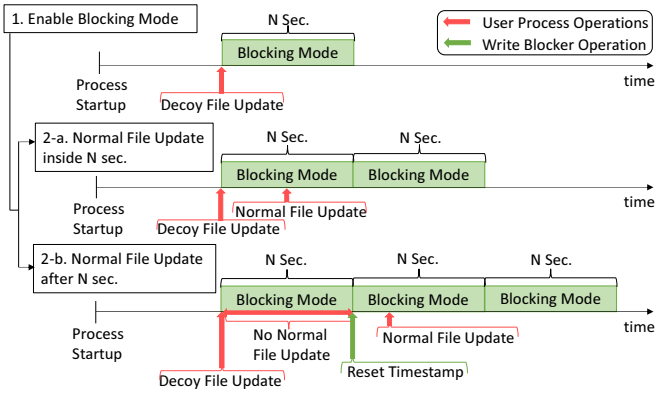


**図 4** Write Blocker による Blocking Mode の概観

### 5.4 通常ファイルへの変更に対するブロッキング

ユーザプロセスからデコイファイルに対し，書き込みや削除といった変更操作が生じた際には，Write Blocker により，ブロッキングモードの有効化がなされる．ブロッキングモードは時間限定的に動作し，ブロッキングモードが有効な期間にて通常ファイルに対し変更要求が生じた際には，ブロッキングモード期間が終了するまで，変更要求をしたスレッドをタスクスケジューラにて待ち状態にする．

デコイファイルに対する変更操作は，ブロッキングモードが有効な期間においても許容される．また，5.1 節から 5.3 節にて示した Decoy FS および Directory Entry Shuffler により，ランサムウェアは通常ファイルとデコイファイルの区別が困難なものと想定している．

#### 5.4.1 ブロッキングモードの概観

図 4 に Write Blocker によるブロッキングモードの概観を示す．ブロッキングモードは，既存のランサムウェア検出機構が検出までに要する $N$ 秒間に対し，$2 \times N$ 秒間の期間にて動作する．ブロッキングモード有効化後，$N$ 秒間以内に通常ファイルに対する変更要求が生じた際には，変更要求をしたスレッドを待ち状態にし，ブロッキングモード開始から $2 \times N$ 秒後に待ち状態を解除する．$N$ 秒間以内に通常ファイルに対し変更要求が生じなかった際には，タイムスタンプをリセットし，リセット時刻から $N$ 秒間以内の通常ファイルに対する変更要求を，再び監視する．

#### 5.4.2 ブロッキングモードの回避に対する検討

ブロッキングモードの回避を目的とし，ランサムウェア自身の実行を停止させた場合，$N$ 秒間以内における通常ファイルに対する変更要求は生じない．Write Blocker は $N$ 秒経過時にタイムスタンプをリセットし，ブロッキングモードを続行するため，ブロッキングモードの回避は成立しない．ランサムウェアがブロッキングモード有効化後から $N$ 秒目に到達する直前にデコイおよび通常ファイルに対する変更要求を行った場合，残りの $N$ 秒間のブロッキン

グモードが有効である．したがって，残りの $N$ 秒間にてデコイファイルに対する暗号化がなされ，既存のランサムウェア検出機構によるファイル暗号化の分析がなされる．

## 6. 実装

本稿における CryptoDelay プロトタイプの実装は, Linux Kernel Module として 2150 行程度の実装により実現した．CryptoDelay における Decoy FS，Directory Entry Shuffler および Write Blocker は，システムコールおよび Linux カーネル関数のフックにより実現する．

### 6.1 システムコールのフック

Decoy FS および Write Blocker は，システムコール呼び出し前のフック (Pre フック) 関数として動作する．例として，Decoy FS におけるディレクトリエントリの取得要求の監視は getdents システムコールの Pre フックにより実現，Write Blocker における通常ファイルに対する変更の監視は，write，unlink システムコールの Pre フックにより実現する．

### 6.2 カーネル関数のフック

Directory Entry Shuffler は，getdents システムコールにより生成されたディレクトリエントリについて，ユーザ空間にコピーがなされる前にシャッフルを行う必要があるため，getdents システムコール内の処理に対する介入を要する．CryptoDelay は，ファイルオープン時に呼び出される関数である do_dentry_open の呼び出し後のフック (Post フック) を行う．さらに，do_dentry_open の呼び出し後に対象のファイルの file 構造体に紐づく f_op にて格納される関数である iterate_shared の Post フックを行う．iterate_shared の Post フックにて，生成されたディレクトリエントリを検査，シャッフルを行い，通常の getdents システムコールによりユーザ空間に対しコピーがなされる．

## 7. 評価

CryptoDelay に対する評価として，ランサムウェアによるファイル暗号化に対する抑制の検証，ユーザープロセスの実行性能に与える影響の確認を行った．評価の目的と内容を以下に示す．

- **ファイル暗号化抑制の検証**
  CryptoDelay が適用された OS にてランサムウェア実検体を実行し，デコイファイルに対する暗号化を十分に許容しながら，通常ファイルに対する暗号化を抑制可能であるか，評価を行った．
- **パフォーマンスの計測**
  CryptoDelay が適用および未適用の OS にて，マイクロベンチマークならびに UnixBench [18] を動作させ，パフォーマンスを計測した．パフォーマンス結果を比較し，CryptoDelay がアプリケーション実行性能に与える影響を定量的に特定した．

**表 1** 実験環境

| | |
|---|---|
| Kernel | Linux 5.4.0-153-generic |
| File System | ext4 |
| OS Distribution | Ubuntu 24.04.2 LTS (Noble Numbat) |
| CPU | Intel(R) Core(TM) i7-8700K CPU @ 3.70GHz |
| CPU Core | 12 |
| Memory | 48 GiB |
| Storage | Samsung SSD 870 EVO 1TB |

**表 2** ランサムウェア起動からブロッキングモード ($N = 120$) 終了までに暗号化された通常ファイル数およびデコイファイル数

| ランサムウェア | 通常ファイルとデコイファイルの合計数 | 暗号化された通常ファイル数 | 暗号化されたデコイファイル数 |
|---|---|---|---|
| AvosLocker | 5015 | 0 | 67 (全体の 1.33 %) |
| HelloKitty | 5281 | 0 | 223 (全体の 4.22 %) |
| LockBit | 5064 | 0 | 330 (全体の 6.51 %) |
| RansomEXX | 5011 | 0 | 42 (全体の 0.83 %) |
| RansomEXX2 | 5025 | 0 | 41 (全体の 0.81 %) |
| RedAlert | 4971 | 0 | 22 (全体の 0.44 %) |
| ReEvil | 4943 | 0 | 81 (全体の 1.63 %) |

CryptoDelay の実験環境を表 1 に示す．表 1 の環境にて，Linux Kernel Module として CryptoDelay をインストールし，評価を実施した．

### 7.1 ファイル暗号化抑制の検証

CryptoDelay によるファイル暗号化抑制性能を確認するため，ランサムウェア起動から Write Blocker によるブロッキングモードが終了するまでの間の，通常ファイルおよびデコイファイルのファイル暗号化数の計測を行った．

2.2 節にて示した通り，既存のランサムウェア検出機構による検出時間は 30 秒から 120 秒程度であるため，ブロッキングモードの時間 $N$ について，$N = 120$ として設定を行った．ランサムウェアは 2.3 節にて示した 7 種類の実検体を使用し，1 KiB × 1000 ファイルに対するファイル暗号化を試みる．

結果を表 2 に示す．ランサムウェア起動からブロッキングモード終了までに暗号化された通常ファイル数は，全てのランサムウェアにおいて 0 ファイルであり，暗号化されたデコイファイル数は 22 ファイル (RedAlert) から 330 ファイル (LockBit) であった．

### 7.2 パフォーマンスの計測

#### 7.2.1 マイクロベンチマーク

ユーザープロセスからファイルシステムに対する操作において，CryptoDelay による主な介入が行われるのは，ディレクトリエントリ取得時におけるデコイファイル配置およびディレクトリエントリのシャッフルである．ユーザープロセスによるディレクトリエントリ取得時にて，CryptoDelay 適用による性能およびストレージに対する影響について調査した．

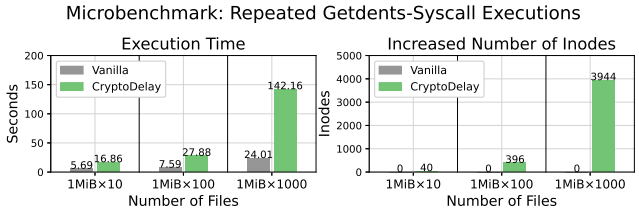


**図 5** マイクロベンチマーク実行結果の比較

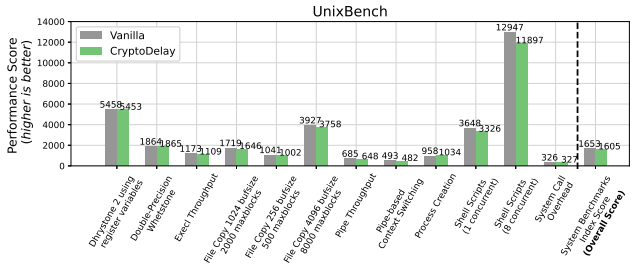


**図 6** UnixBench スコアの比較

ベンチマークについては，専用の計測プログラムを用いた．専用の計測プログラムは，カレントディレクトリに対し getdents システムコールを $10^5$ 回発行し，完了にかかる時間，およびストレージ追加消費量，inode 追加消費数の計測を行う．カレントディレクトリには 1 MiB サイズのファイルを配置するものとし，10, 100, 1000 ファイルを配置したパターンをそれぞれ用意し，各パターンにおいて実行，結果の比較を行った．

結果を図 5 に示す．ファイル数が 10, 100, 1000 ファイルの配置パターンにて，それぞれ 2.96，3.67，5.92 倍程度の実行時間増加が確認された．ストレージ追加消費量はいずれの配置パターンにおいても 0 バイトであり，inode 追加消費数はそれぞれ 40，396，3944 であった．

#### 7.2.2 UnixBench

UnixBench 5.1.3 を用いて，ディレクトリエントリ取得時に限ることのない幅広いワークロードにおける，CryptoDelay 適用による性能およびストレージに対する影響について調査した．

結果を図 6 に示す．スコアダウンは最大 8.82 % (Shell Scripts, 8 concurrent) であり，ベンチマーク全体の評価スコアである System Benchmarks Index Score のスコアダウンは 2.90 % 程度であった．ストレージ追加消費量は 0 バイトであり，inode 追加消費数は 188 であった．

## 8. 考察

### 8.1 評価結果に関する考察

#### 8.1.1 ファイル暗号化抑制の検証

ランサムウェア起動から Write Blocker によるブロッキングモードの終了までの間に，通常ファイルにおけるファイル暗号化被害は生じなかった．ファイル暗号化がなされたデコイファイルは 22 から 330 ファイルであり，ファイル数全体に対する 0.44 から 6.51 % 程度にあたる．

ランサムウェアの動的検出を行うための既存研究 [7] では，検出までの間に 0.12 から 2.79 % 程度のファイルが暗号化の被害を受けることを報告している．したがって，本稿における検証結果は，既存のランサムウェア検出機構が検出を行うにあたり，十分なファイル暗号化がデコイファイルに対してなされているものと考えられる．

評価に使用したランサムウェアは，全ての検体においてマルチスレッドに対応しており，マルチコア CPU のホストにて CryptoDelay がファイル暗号化被害を抑制可能であることを示した．

#### 8.1.2 パフォーマンスの計測

ディレクトリエントリ取得時における CryptoDelay による介入処理は，Decoy FS によるデコイファイル配置，ならびに Directory Entry Shuffler によるディレクトリエントリのシャッフルにより構成される．デコイファイル配置ならびにディレクトリエントリのシャッフルは，対象のディレクトリにおけるファイル数に依存し，ファイル数の増加に伴い，オーバーヘッドが増加する傾向にある．

CryptoDelay の適用により増加が確認された inode は，いずれもデコイファイルによるものであった．デコイファイルはサイズ 0 のファイルとして配置がなされるため，ファイルシステムにおける inode テーブルのエントリは消費されるが，データブロックの消費はなされない．ext4 においては，inode テーブルの領域は事前に割り当てがなされていることから，デコイファイル配置によるストレージ消費は生じなかったものと考えられる．

### 8.2 正規のアプリケーションに対するブロッキングの影響

デコイファイルを含めた，ディレクトリ内のファイル全体に対して変更を行うようなアプリケーションにおいては，CryptoDelay における Write Blocker よる影響を受ける．例として，zip [19] においては，-m オプションの指定により，zip ファイル生成後にオリジナルファイルの削除を試みた場合にて，Write Blocker による一定時間のブロッキングを受ける．類似の例として，rsync [20] においても，–remove-source-files オプションの指定時において，Write Blocker による一定時間のブロッキングを受ける．

今後の展望として，サーバホストで使用される正規アプリケーションにおける，Write Blocker によるブロッキングの影響について，包括的に調査を行う予定である．

### 8.3 既存研究との比較

振る舞いの監視による検出 [4–8] では，検出までの間にファイル暗号化の被害を受けるファイルが存在し，CryptoDelay の適用によってそれらのファイル暗号化の被害を抑制可能である．一方で，ランサムウェアの検出および停止は CryptoDelay の機能要件外であり，振る舞いの監視による検出機構と CryptoDelay は，補完関係にある．

バックアップによるファイルリカバリ [9] は，ファイル

書き込みのたびに専用のストレージ領域に対するファイルコピーを要する．CryptoDelay はデコイファイルに対する変更がなされていない状況下での，通常ファイルに対する I/O においては介入処理を要さない．一方で，ディレクトリエントリの取得時に CryptoDelay はデコイファイルの配置およびディレクトリエントリのシャッフルを要する．

## 9. おわりに

サーバホストを標的として開発された近年のランサムウェアは，サーバ上における実行にて，ファイル暗号化にかかる時間を短縮するための設計を備える．ランサムウェアの振る舞いに基づいた既存の検出機構は，強力な検出を実現するが，検出までの間に暗号化の被害を受けるファイルが生じうる．ファイル暗号化の被害発生後にファイルリカバリを可能とするストレージシステムは，I/O におけるランタイムオーバーヘッドならびにストレージオーバーヘッドが高く，高性能なアプリケーション実行が要求されるサーバホストに対して適さない．

本稿では，ファイル暗号化の被害を抑制しながら，既存のランサムウェア検出機構による検出を可能とするための新たな手法である CryptoDelay を提案した．CryptoDelay はユーザープロセスによるディレクトリエントリの取得に応じて，対象のディレクトリに対しデコイファイルを配置，デコイファイルへのアクセス監視を行う．デコイファイルに対し変更を行ったユーザープロセスを特定し，デコイファイルへの変更を許容しながら，通常ファイルへの変更をブロックすることで，既存のランサムウェア検出機構による検出までの間のファイル暗号化抑制を実現する．

CryptoDelay を Linux Kernel Module として実装し，既存のランサムウェアによるファイル暗号化被害を十分に抑制可能であることを確かめた．加えて，CryptoDelay がユーザープロセスの実行性能やストレージの消費量に与える影響について，定量的に明らかにした．

**謝辞** 本研究の一部は，JSPS 科研費 JP25K07749, JP23K03847 の助成を受けたものです．

### 参考文献

[1] Cybersecurity and Infrastructure Security Agency (CISA): StopRansomware Guide, https://www.cisa.gov/stopransomware/ransomware-guide. (accessed 2025-08-03).

[2] TREND MICRO: Analysis and Impact of LockBit Ransomware ' s First Linux and VMware ESXi Variant, https://www.trendmicro.com/en_us/research/22/a/analysis-and-Impact-of-lockbit-ransomwares-first-linux-and-vmware-esxi-variant.html. (accessed 2025-08-03).

[3] VMware: ESXi-Targeting Ransomware: The Threats That Are After Your Virtual Machines (Part 1), https://blogs.vmware.com/security/2022/09/esxi-targeting-ransomware-the-threats-that-are-after-your-virtual-machines-part-1.html. (accessed 2025-08-03).

[4] Amin Kharaz et.al: UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware, *Proceedings of the 25th USENIX Security Symposium (Security '16)*, USENIX Association, pp. 757–772 (2016).

[5] Kharraz Amin et.al: Redemption: Real-Time Protection Against Ransomware at End-Hosts, *Proceedings of the 20th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID '17)*, Springer, pp. 98–119 (2017).

[6] Moussaileb Routa et.al: Ransomware's Early Mitigation Mechanisms, *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES '18)*, ACM, pp. 1–10 (2018).

[7] Wang Shenao et.al: CanCal: Towards Real-time and Lightweight Ransomware Detection and Response in Industrial Environments, *Proceedings of the 31st on ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*, ACM, pp. 2326–2340 (2024).

[8] Md. Ahsan Ayub et.al: RWArmor: a static-informed dynamic analysis approach for early detection of cryptographic windows ransomware, *International Journal of Information Security*, vol. 23, no. 1, Springer, pp. 533–556 (2024).

[9] Continella Andrea et.al: ShieldFS: a self-healing, ransomware-aware filesystem, *Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC '16)*, ACM, pp. 336–347 (2016).

[10] McIntosh Timothy et.al: Ransomware Mitigation in the Modern Era: A Comprehensive Review, Research Challenges, and Future Directions, *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, ACM, pp. 1–36 (2021).

[11] Trend Micro: Ransomware Spotlight: RansomEXX, https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-ransomexx. (accessed 2025-08-03).

[12] WatchGuard Technologies: Ransomware - RansomExx2, https://www.watchguard.com/wgrd-ransomware/ransomex2. (accessed 2025-08-03).

[13] Trend Micro: Ransomware Spotlight: AvosLocker, https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-avoslocker. (accessed 2025-08-03).

[14] Intel: Intel® Advanced Encryption Standard Instructions (AES-NI), https://www.intel.com/content/www/us/en/developer/articles/technical/advanced-encryption-standard-instructions-aes-ni.html. (accessed 2025-08-03).

[15] Trend Micro: Ransomware Spotlight: REvil, https://www.trendmicro.com/vinfo/us/security/news/ransomware-spotlight/ransomware-spotlight-revil. (accessed 2024-08-19).

[16] Trend Micro: Ryuk ransomware attack, https://www.trendmicro.com/en_us/what-is/ransomware/ryuk-ransomware.html. (accessed 2025-08-03).

[17] WatchGuard Technologies: Ransomware - RedAlert, https://www.watchguard.com/wgrd-ransomware/redalert. (accessed 2025-08-03).

[18] kdlucas: byte-unixbench, https://github.com/kdlucas/byte-unixbench. (accessed 2025-08-17).

[19] linux.die.net: zip(1) - Linux man page https://linux.die.net/man/1/zip (accessed 2025-08-19).

[20] linux.die.net: rsync(1) - Linux man page https://linux.die.net/man/1/rsync (accessed 2025-08-19).