Vol.2021-DPS-186 No.50
Vol.2021-CSEC-92 No.50
2021/3/16

デコイファイルを用いた暗号化型ランサムウェアの検知とプロセス特定に関する検討

荻原拓海[1,a)]　小林良太郎[1]　加藤雅彦[2]

概要：近年ランサムウェアは大きな脅威となっている．IPA が毎年公開している情報セキュリティ 10 大脅威では，ランサムウェアの脅威は 2016 年から 2021 年まで 6 年連続でランクインしておりその重大さを表している． このような情勢下でも様々な理由でパッチが適用できず，バックアップも頻繁に取れない環境で重要なデータを保存せざるを得ないケースもある．このような場合ランサムウェアに対して有効な対策を取ることは非常に困難である．そこで，本研究では暗号化型ランサムウェアのファイル操作に着目し，囮となるファイルを利用することで暗号化の検知とファイルの保護を行い，プロセスのファイル操作からランサムウェアの特定と動作停止を行うシステムの提案をする．また，本研究のシステムの有効性を確認するために複数のランサムウェアと良性ソフトウェアを実行し評価する．

A Study on Detection and Process Identification of Encrypting Ransomware Using Decoy Files

Abstract: Ransomware has become a major threat in recent years. According to the IPA's "10 Major Security Threats", the ransomware threat has been ranked from 2016 to 2021. This indicates that ransomware has been recognized as a major threat. Even under these circumstances, there are cases where patches cannot be adapted and backups cannot be made frequently for various reasons. In such a case, it is very difficult to take effective countermeasures against ransomware. Therefore, in this study, we focused on file operation of ransomware. We implement a system that detects encryption and protects files by using decoy files, and identifies and stops ransomware from manipulating files. In addition, we run and evaluate several ransomware and benign software to confirm the effectiveness of our system.

1. はじめに

近年ランサムウェアは大きな脅威となっている．IPA が毎年公開している情報セキュリティ 10 大脅威 [1] では，ランサムウェアによる脅威が 2016 年から 2021 年まで 6 年連続でランクインしており，2021 年では組織向けの脅威として 1 位となっている．また，Cyber Attack Trends: 2020 Mid-Year Report[2] によると，ランサムウェアは Cyber Attack Categories by Region において全地域で第 6 位となっている．ランサムウェアとはコンピュータに特定の制限をかけ，元通りにする代わりに身代金を要求する不正プログラムである．その種類は大まかに分類すると，コンピュータに保存されているデータを暗号化する「暗号化型ランサムウェア」とコンピュータの操作ができないようにロックする「画面ロック型ランサムウェア」があり [3]，現在は「暗号化型ランサムウェア」が主流となっている [2]．

暗号化型ランサムウェアの Wannacry は 2017 年に猛威を振るったランサムウェアである．Wannacry はデータを暗号化する機能に加えて自己増殖する機能を持ち，ネットワークを介して他のコンピュータに感染をさせることで広範囲に影響を及ぼした．このランサムウェアは SMBv1 の脆弱性を利用しユーザーが気が付かない間に感染を広げていた [4]．2017 年の時点で Wannacry が利用している脆弱性は修正されていたが，パッチを適用していないコンピュータが数多く存在し，大規模な被害が発生した．特に顕著だったのは，英国の公的医療制度を担う NHS トラストを標的とした攻撃で，全体の 3 分の 1 のトラスト団体が被害を受け，被害総額は 9200 万ポンドに及んだと推計さ

[1] 工学院大学
Kogakuin University
[2] 長崎県立大学
University of Nagasaki
[a)] j117058@ns.kogakuin.ac.jp

Vol.2021-DPS-186 No.50
Vol.2021-CSEC-92 No.50
2021/3/16

れている [5]．このように修正済みの脆弱性を利用した攻撃でも大きな被害を与えるケースが存在する．一般的なランサムウェアの対策として，システムへの迅速なパッチ適用や定期的なバックアップが推奨されているが，様々な理由でこれらを実施しないユーザーも存在する．このような場合ランサムウェアに対して効果的な対策を取ることは非常に困難である．

Windows 10 にはランサムウェア対策機能としてコントロールされたフォルダーアクセスというオプションが存在する．これは許可されていないプログラムのファイルやメモリへのアクセスを防止するものである．しかし，標準ではオフになっており，その機能を利用するには適切な設定を行う必要がある．また，この機能はホワイトリスト形式のため良性プログラムであってもブロックされる可能性がある．そのため手動でホワイトリストに登録できる機能が搭載されている．さらに，デフォルトで設定されているフォルダ以外の保護も行う場合は，手動で追加する必要がある．このようなホワイトリスト形式の対策は強力な一方，ユーザーの負担が大きい．

そこで，本研究ではランサムウェアのファイルアクセスに着目し，囮となる暗号化対象のファイル（以下デコイファイルと呼ぶ）を用いてランサムウェアによる暗号化を検知し，ファイルのアクセス情報からランサムウェアを特定・停止させるシステムの提案を行う．また，有効性を確認するために 14 種のランサムウェアを仮想マシンで動作させ，検知・特定・停止の可不可を検証する．さらに良性プログラムが誤って検知されプロセスを強制的に終了させることが無いか，いくつかのソフトウェアを動作させ検証する．

2 章では関連研究について述べる．3 章では提案手法を示し，4 章の実装ではプログラムの動きなどを示し，5 章で提案システムの検証を行う．6 章では検証の過程や結果から考察を行い，7 章で本稿のまとめを行う．

2. 関連研究

J. A. Gómez-Hernández らは，FIFO 形式のデコイファイルを利用しファイルにアクセスしたプロセスをブロックする仕組みの実装と検証を行っており，このシステムを R-Locker と呼んでいる [6]．FIFO 形式のデコイファイルを使う手法ではその特性からファイルサイズを小さくすることができる．また多数のファイルを監視する必要がないため，CPU 使用率も低く抑えることができる．論文内の有効性の検証では，3 つのランサムウェアをそれぞれ動作させ，暗号化が開始される前に停止できたことが示されている．[6] では Unix 用のシステムを実装しているが，Windows 用のシステムも実装しており，後者は GitHub で公開されている [7]．

Shagufta Mehnaz らは，デコイファイルを用いた手法に加え，機械学習やファイルアクセス，プロセスの監視を組み合わせたシステムの実装と検証を行っている [8]．この論文では 14 種のランサムウェア[*1]に対して有効性の検証を行っており，偽陰性率は 0%，偽陽性率は 0.1%未満であり，メモリの使用量が約 68.6MB，CPU 使用率が約 1.9% であると示されている．また CryptAPI を利用している暗号化型ランサムウェアに限定されるものの，暗号化されたファイルの修復も可能としている．

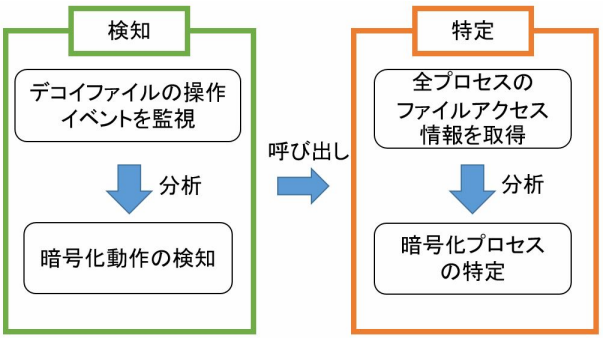

図 1 提案システムのイメージ
Fig. 1 Image of the proposed system

3. 提案手法

3.1 概要

システムの概要図を図 1 に示す．提案手法では，ランサムウェアが行うファイル操作の特徴を利用する．まず，暗号化対象となるデコイファイルを設置する．次にデコイファイルに対するファイル操作イベントを監視し，暗号化動作を検知する．その後，全プロセスのファイルアクセス情報を取得し，その情報を元にランサムウェアの特定を行う．最後に特定されたランサムウェアの動作を停止させる．

暗号化動作を検知した後にプロセスの特定を行うことで，誤検知を減らすことができ，かつ CPU 負荷を軽減させることができる．また，本システムはデコイファイルを設置する必要があるが，それらの総容量は十分に抑制可能である．

デコイファイルとは本システムを導入する際に新たに生成するファイルであり，ユーザーや他のシステムによって，変更が加えられないことを前提としている．

3.2 暗号化動作の検知

本節では，暗号化動作の検知について述べる．暗号化動作の検知を行うため，本研究ではランサムウェアが暗号化を行う際のファイル操作に着目する．ランサムウェアがファイルの暗号化を行う際の手順は大別すると約 3 種類存在する．1 つ目は暗号化対象となるファイルのデータを読み込み，暗号化を行ったデータを同一のファイルに上書きする方法である．2 つ目は暗号化対象となるファイルのデータを読み込み，暗号化を行ったデータを別ファイルに

[*1] 本研究の検証に使用したランサムウェアとは異なる

保存した後，元のファイルを削除する方法である．3 つ目は暗号化対象となるファイルのデータを読み込み，暗号化を行ったデータを別ファイルに保存した後，元のファイルを別のデータで上書きし破壊する方法である [9]．いずれの手法においても，暗号化されたファイルは foo.txt から foo.lock のようにファイル名が変更される．

このようなランサムウェアの特徴から，提案手法では暗号化の対象となるデコイファイルを設置し，デコイファイルの削除とパスの変更イベントを取得することで，ランサムウェアの暗号化動作を検知する．

3.3 暗号化プロセスの特定

本節では，暗号化プロセスの特定に関して述べる．3.2 で示したように，ランサムウェアが暗号化を行う際にはデコイファイルを読み込み，上書きまたは新規ファイルへの書き込みを行う．そのため本システムでは，暗号化動作の検知後，プロセスごとのファイルアクセス情報を一定時間取得して分析することでランサムウェアの特定を行う．分析に必要な情報は，プロセス ID とファイル操作命令，アクセス先のパスである．ファイルアクセス情報の取得後，プロセスごとに以下の情報を抽出する．

(1) デコイファイルを読み込んだ回数
(2) 書き込みを行った回数
(3) プロセスによるファイルへの読み書き量 (byte)

以上 3 つの値を乗算した値が最も大きくなるプロセスをランサムウェアとして扱う．

次に乗算する目的に関して説明する．ランサムウェアがデコイファイルの暗号化を行う際には必ずファイルを読み込む必要がある．そこでデコイファイルを読み込んだ回数を記録している．しかしデコイファイルを読み込むプロセスはセキュリティ関連ソフトや Windows のファイル検索用プロセスなど他にもあり，この情報のみで判断することは出来ない．そこでファイル書き込みを行った回数を取得し，それを乗算することで，ファイルの書き込みを行わないプロセスはその値が 0 になり除外できる．少量の書き込みを行った場合でも，その値の差が大きなものとなる．さらにプロセスによるファイルへの読み書き量を乗算することで，ランサムウェアと他プロセスの違いを明確に判別することが可能になる．

この手法はランサムウェアが動作していることを前提として設計している．そのため 3.2 で示した暗号化動作の検知が行われた後にプロセスの特定を行う必要がある．

3.4 デコイファイル

ランサムウェアは，種類によって暗号化対象としているファイルとサイズが異なる [10]．そこで本研究では，サイズの異なるデコイファイルを複数設置することで，ランサムウェアに暗号化されるデコイファイルを生成している．

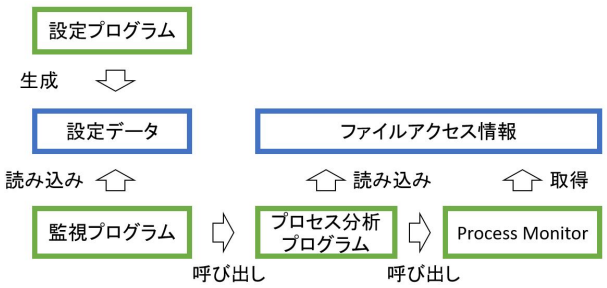

図 2 プログラムの関連イメージ
Fig. 2 Relevance of the program

生成するデコイファイルのサイズは文献 [10] を参考に，1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB の 6 種類とする．

デコイファイルを大量に生成することで，ランサムウェアが暗号化に要する時間が長くなり，暗号化開始から本システムが動作を停止させるまでに，他のファイルが暗号化される数が少なくなると考えられる．しかし，ストレージの容量には限りがあり無制限に使用できるものではない．したがって本研究では，ランサムウェアの検知後，事前に生成した 2 KiB のデコイファイルを暗号化の進行度に合わせて 65MiB のデコイファイルに変更することで，他のファイルが暗号化されるまでの時間を確保している．またハードリンクを用いて単一のデータを示す複数のリンクを作成することで，多くのフォルダを少ない容量で守ることが可能になる．しかし，暗号化するファイルのデータを確認し，一度暗号化したものを再度暗号化しない様に設計されているランサムウェアも存在する [11]．そのため本システムでは通常にデコイファイルを生成する場合とハードリンクで作成する場合を用意に切り替えられるように実装し，その両方で検証を行う．

4. 実装

本章では，システムの実装に関して説明する．本システムは主に以下 4 つのプログラムによって構成されている．

(1) 設定プログラム
(2) 監視プログラム
(3) プロセス分析プログラム
(4) Process Monitor

各プログラムの関わりを図 2 に示す．設定プログラムは他のプロセスが動作するために必要な設定データの生成とデコイファイルの生成を行う．監視プログラムは設定プログラムが生成した設定データを読み込み，暗号化動作の監視を行う．暗号化動作の検知後，プロセス分析プログラムに必要な情報を受け渡して呼び出す．プロセス分析プログラムは，受け取った情報を元に Process Monitor を起動する．Process Monitor はファイルアクセス情報を一定時間取得し，csv 形式で保存する．Process Monitor 終了後，プロセス分析プログラムはファイルアクセス情報を読み込

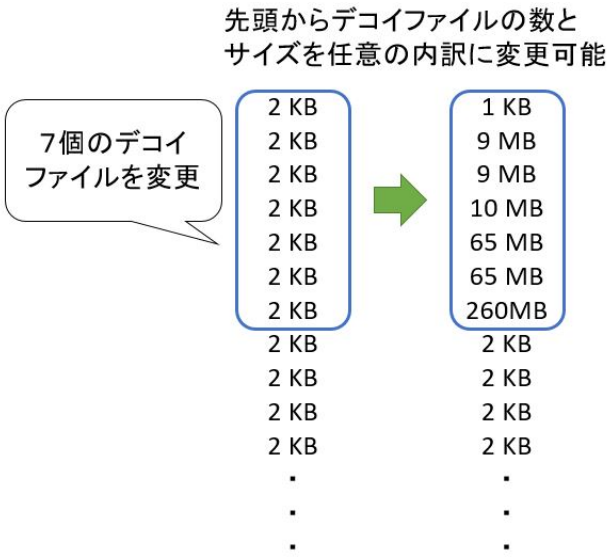

図 3 サイズ内訳のイメージ
Fig. 3 Image of file size breakdown

み，ランサムウェアの特定を行う．

プログラムは python で記述し，pyinstaller と cx_freez を用いて実行形式に変換している．

4.1 設定プログラム

このプログラムは，他のプログラムを動作させるために必要な設定ファイルとデコイファイルを生成するプログラムである．このプログラムを通してパラメータの変更が可能となっている．以下にこのプログラムで変更・追加可能な項目を示す

(1) デコイファイルを設置するフォルダの追加
(2) 1 フォルダに生成するデコイファイルの数の変更
(3) デコイファイルのサイズ内訳の変更
(4) デコイファイルの削除
(5) ハードリンク活用の有無
(6) 監視プログラムの状態確認周期の変更
(7) Procass Monitor で情報を取得する時間の変更
(8) プロセス終了時の確認の有無を変更

デコイファイルファイルのサイズ内訳を行った際のイメージを図 3 に示す．

4.2 監視プログラム

このプログラムは，デコイファイルの削除とパスの変更イベントを監視することでランサムウェアの検知を行う．検知後はユーザーに通知を行い，プロセス分析プログラムを呼び出す．また，デコイファイルの上書きを行い，暗号化に要する時間を増加させる．このプログラムの動作を図 4 に示す．監視プログラムで生成するスレッドの動作を図 5 に示す．N は設定プログラムで設定したデコイファイルのサイズ内訳を変更する数である．デコイファイルの削除

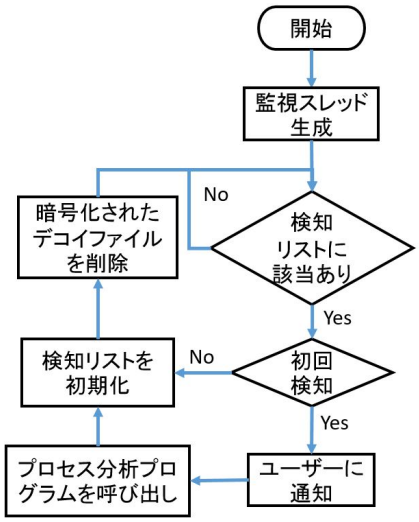

図 4 監視プログラムのメインループ
Fig. 4 The main loop of the monitoring program

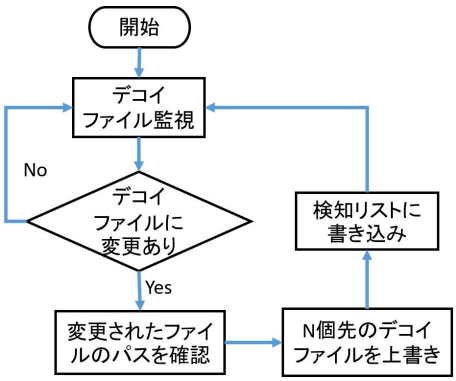

図 5 監視プログラムの監視スレッド
Fig. 5 The thread of the monitoring program

とパスの変更イベントを監視するため，本システムでは watchdog.events.FileSystemEventHandler クラスを使用している [12]．監視フォルダ内のファイル変更イベントが発生した場合，変更前のパスを取得しデコイファイルであることを確認する．デコイファイルであった場合には検知状況を記録するリストに情報を書き込む．その後，N 個先のデコイファイルを上書きしサイズを増加させる．検知情報を管理しているリストを定期的に確認し，検知を示す情報が確認された場合にユーザーへ通知を行い，プロセス分析プログラムを呼び出す．その後暗号化されたデコイファイルをすべて削除する．なお通知とプロセス分析プログラムの呼び出しは初回検知時かランサムウェアの動作を停止させた後の状態のみ実行する．プロセス分析プログラムの終了が確認された場合，プロセス分析プログラムの動作結果が保存されたデータを読み込み内容を出力する．

4.3 プロセス分析プログラム

このプログラムは，監視プログラムから呼び出され，Process Monitor を用いて指定された時間ファイルアクセス情報を取得し，その情報を用いてプロセスの特定と停止

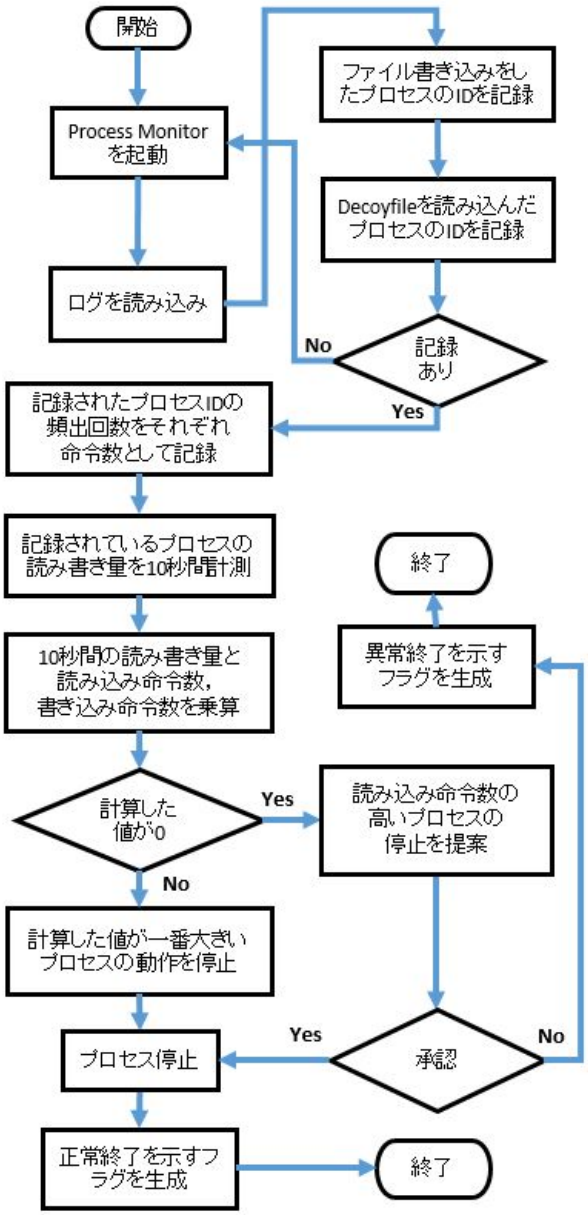

図 6 プロセス分析プログラムのイメージ
Fig. 6 Image of a process analysis program

を行うプログラムである．プログラムの動きを図 6 に示す．このプログラムは呼び出し後，Process Monitor を呼び出し設定プログラムで指定されている時間分のファイルアクセス情報を収集する．収集した情報は csv 形式に変換し，プロセス分析プログラムで読み込む．読み込んだ情報を元に 3.3 節で示した情報を抽出する．取得した 3 つの情報を乗算した値が最も大きくなるプロセスをランサムウェアとみなしプロセスの停止を行う．プロセスの停止が正常に実行された場合，停止させたプロセスの情報を動作結果として保存する．また，乗算した値がすべて 0 の場合，デコイファイルを読み込んだ回数が多いプロセスの情報をユーザーに通知し，動作の停止か継続を選択可能にする．

4.4 Process Monitor

Process Monitor は Microsoft が提供する Windows 用の高度な監視が行えるツールである [13]．Process Monitor にはフィルタ機能があり，それを用いて必要な情報を取得している．本システムでは，ファイルアクセス情報の取得に活用するため，Show File System Activity を有効化し，Operation is WriteFile Include, Operation is ReadFile Include をフィルタとして使用している．また取得する情報を，Process Name, PID, Operation, Path の 4 つに削減している．

5. 検証

本章では，提案システムの有効性を確認するために 14 種のランサムウェアを仮想マシンで動作させ，検知・特定・停止の可不可を検証する．また，本システムには変更可能なパラメータが複数存在する．したがって検証では変更可能なパラメータの内，システムの有効性に深くかかわるパラメータを変化させ，その影響を確認する．さらに良性プログラムが誤って検知されプロセスを強制的に終了させることが無いか，いくつかのプログラムを動作させ検証する．

提案システムの検証に加え，GitHub 上で公開されている R-Locker[7] を動作させ，14 種のランサムウェアに対しての有効性を確認し，比較する．

5.1 検証環境

仮想環境で動作させるランサムウェアの種類とハッシュ値を表 1 に示す．検証に使用するコンピュータの環境は表 2 に示すとおりである．今回の検証ではデコイファイルを設置するフォルダ（以下では監視フォルダと呼ぶ）を表 3 の通りに設定する．監視フォルダの名前は!としているが，これは Unicode 順にフォルダ・ファイルを取得した際に初めに取得されるようにするためである．デコイファイルのファイル名は!100000decoyfilename.txt, !100001decoyfilename.txt, !100002decoyfilename.txt ... としている．また，一部のランサムウェアは.Net Framework 3.5 (2.0 および 3.0 を含む) を必要としているため，「Windows の機能の有効化または無効化」から有効にした．

5.2 検証方法

本システムをゲストの Windows 10 上で動作させ，14 種のランサムウェアをそれぞれ 1 つずつ実行し，本システムによる動作停止の可不可を確認する．また，本システムの有効性に深く影響するパラメータであるデコイファイルのサイズの内訳を表 4 に示すように変更しそれぞれの条件に対して動作停止の可不可を確認する．デコイファイルの内訳は，5.3 節の予備実験の結果から設定する．表 4 では条件 1〜5 まで記述しているが，同様の内訳でデコイファイルをハードリンクを用いて生成した場合をそれぞれ条件 1′，条

表 1　各検体のハッシュ値
Table 1　hash value

種類	ハッシュ値
Cerber	8b6bc16fd137c09a08b02bbe1bb7d670
Clop	a04eb443870896fbe9a0b6468c4844f7
Dharma	ba67dd5ab7d6061704f2903573cec303
GandCrab	49eb4afe5b817ed56eacf504c80106d1
Jigsaw	2773e3dc59472296cb0024ba7715a64e
Katyusha	7f87db33980c0099739de40d1b725500
Keypass	6999c944d1c98b2739d015448c99a291
Lockbit	a7637dfb6b9408fe020d9333d0ade6dc
Phobos	e59ffeaf7acb0c326e452fa30bb71a36
TeslaCrypt	209a288c68207d57e0ce6e60ebf60729
Thanos	be60e389a0108b2871dff12dfbb542ac
Vipasana	2aea3b217e6a3d08ef684594192cafc8
Wannacry	84c82835a5d21bbcf75a61706d8ab549
Zerolocker	bd0a3c308a6d3372817a474b7c653097

表 2　検証環境
Table 2　Verification environment

	ホスト	ゲスト
OS	Ubuntu 18.0.4LTS	Windows 10 home
CPU	Intel® Core™ i5-9400	仮想プロセッサ 8
RAM	16GiB	8GiB
Strage	256GiB	100GiB

表 3　監視フォルダ
Table 3　Folders to monitor

ラベル	パス
0	C:\Users\%USERNAME%\Desktop\!\
1	C:\Users\%USERNAME%\Downloads\!\
2	C:\Users\%USERNAME%\Favorites\!\
3	C:\Users\%USERNAME%\OneDrive\!\
4	C:\Users\%USERNAME%\Pictures\!\
5	C:\Users\%USERNAME%\Documents\!\
6	C:\Users\%USERNAME%\Videos\!\
7	C:\Users\%USERNAME%\Public\!\
8	C:\Windows\!\
9	C:\Users\!\
10	C:\Program Files\!\
11	C:\Program Files (x86)\!\
12	C:\!\

件 2′，条件 3′，条件 4′，条件 5′ として計 10 種の異なる条件のもと検証を行う．さらに，5 種の良性プログラムをランサムウェアが動いていない状態で動作させ，誤検知や動作の停止が起こらないか確認する．検証に使用する良性プログラムは Cristal Disk Mark, OneDrive, Windows Diffender フルスキャン, Windows 10 ドライブ最適化, 7Zip である．

5.3　予備実験

本節では，1 フォルダあたりのファイル数内訳を決定するために各ランサムウェアの暗号化対象とするファイルサイズを実験によって確認する．結果は表 5 に示す．

表 4　1 フォルダあたりのファイル数
Table 4　Number of files per folder

サイズ	条件 1	条件 2	条件 3	条件 4	条件 5
1 KiB	0	0	0	0	0
9 MiB	100	80	50	40	0
10 MiB	0	0	0	0	0
65 MiB	0	18	0	9	0
260 MiB	0	2	0	1	0
2 KiB	9900	9900	9950	9950	10000
総数	10000	10000	10000	10000	10000
総容量	0.92GiB	2.43GiB	0.47GiB	1.23GiB	20MiB

表 5　暗号化対象のファイルサイズ
Table 5　File size to be encrypted

種類	対象サイズ
Cerber	2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Clop	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Dharma	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
GandCrab	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Jigsaw	2 KiB, 9 MiB
Katyusha	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Keypass	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Lockbit	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Phobos	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
TeslaCrypt	2 KiB, 9 MiB, 10 MiB, 65 MiB
Thanos	2 KiB, 9 MiB, 10 MiB, 65 MiB
Vipasana	1 KiB, 2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Wannacry	2 KiB, 9 MiB, 10 MiB, 65 MiB, 260 MiB
Zerolocker	2 KiB, 9 MiB, 10 MiB

表 6　結果の定義
Table 6　Definition of results

記号	意味
◎	暗号化が開始された監視フォルダ数が 1 以下
○	暗号化が開始された監視フォルダ数が 2 以上
△	1 監視フォルダの暗号化完了後，他の監視フォルダを無視
×	検知または停止が不可能
—	動作を停止すべきプロセスが存在しない

5.4　結果

ランサムウェアの暗号化動作の可不可に関して検証の結果を示す．結果はランサムウェアの実行状況により，5 段階に分けて示す．表 6 に結果の分類に用いる記号とその意味を示す．なお，◎，○，△はともに暗号化の動作を停止できたことを示している．検証の結果は表 7，表 8 に示す．

次に良性プログラムの誤検知についての結果を表 9 に示す．なおこのプログラムの他にも，Windows 10 の標準機能など複数のプロセスが動作しており，それらのプロセスの誤検知を避けるためのホワイトリストなどは存在しな

表 7 条件 1 から条件 5 における動作停止の可不可
Table 7 Results from No.1 to No.5

種類	条件 1	条件 2	条件 3	条件 4	条件 5
Cerber	◎	◎	◎	◎	◎
Clop	◎	◎	◎	◎	◎
Dharma	◎	◎	◎	◎	○
GandCrab	◎	◎	◎	◎	◎
Jigsaw	◎	◎	◎	◎	○
Katyusha	◎	◎	◎	◎	—
Keypass	◎	◎	◎	◎	○
Lockbit	○	○	○	○	○
Phobos	◎	◎	◎	◎	○
TeslaCrypt	◎	◎	◎	◎	◎
Thanos	◎	◎	◎	◎	◎
Vipasana	◎	◎	◎	◎	◎
Wannacry	◎	◎	◎	◎	○
Zerolocker	◎	◎	◎	◎	◎

表 8 条件 1′ から条件 5′(ハードリンク) における動作停止の可不可
Table 8 Results from No.1′ to No.5′

種類	条件 1′	条件 2′	条件 3′	条件 4′	条件 5′
Cerber	◎	◎	◎	◎	◎
Clop	◎	◎	◎	◎	◎
Dharma	◎	◎	◎	◎	△
GandCrab	◎	◎	◎	◎	△
Jigsaw	◎	◎	◎	◎	△
Katyusha	◎	◎	◎	◎	△
Keypass	◎	◎	◎	◎	○
Lockbit	○	○	○	○	○
Phobos	◎	◎	◎	◎	○
TeslaCrypt	◎	◎	◎	◎	○
Thanos	◎	◎	◎	◎	◎
Vipasana	◎	◎	◎	◎	◎
Wannacry	◎	◎	◎	◎	◎
Zerolocker	◎	◎	◎	◎	○

表 9 良性プログラム誤検知の有無
Table 9 Results of false positives

プログラム名	誤検知	誤停止
Cristal Disk Mark	無	無
OneDrive	無	無
Windows Diffender フルスキャン	無	無
Windows 10 ドライブ最適化	無	無
7Zip	無	無

い．また，本システムが使用するリソースに関して，プログラムごとに検知前，検知後の CPU 使用率，メモリの使用量の最大値を測定する．検知前の結果を表 10，検知後の結果を表 11 に示す．

表 10 検知前のリソース (最大値)
Table 10 Resources used before detection (maximum value)

プログラム名	CPU 使用率	メモリ使用量
提案システム	0.0 %	8.1 MB
Process Monitor	—	—

表 11 検知後のリソース (最大値)
Table 11 Resources used after detection (maximum value)

プログラム名	CPU 使用率	メモリ使用量
提案システム	25.7 %	29.8 MB
Process Monitor	16.8 %	7.9MB

表 12 R-Locker のリソース (最大値)
Table 12 Resources used by R-Locker (maximum value)

状態	CPU 使用率	メモリ使用量
検知前	54.8%	73.8 MB
検知後	64.2 %	87.6MB

5.5 R-Locker の結果

Windows 版の R-Locker を動作させ，14 種のランサムウェアに対して，本システムと同じ環境においての有効性を検証した．R-Locker はプロセスの停止の判断をユーザーにゆだねる方式のため，検知したプロセスがランサムウェアである場合のみ，即座に手動で停止させた．その結果，動作の停止が可能であったランサムウェアは 14 種中 Wannacry, Zerolocker, Dharma, Phobos の 4 種のみであった．それ以外のランサムウェアを実行した際には，explorer.exe や haskhost.exe などが誤って検知された．なお，誤検知されたプロセスについては動作の停止を拒否するとホワイトリストに登録された．R-Locker は暗号化の開始から検知までの時間は非常に短く 1 秒未満であるが，検知後停止するまではユーザーが判断する必要があるため，その間暗号化が継続されてしまう．また，R-Locker 動作時に使用したリソースを計測した．その最大値を表 12 に示す．なお，検知前の CPU 使用率の平均値は約 38 %であった．

6. 考察

5.4 節で示した通り，提案システムは 14 種すべてのランサムウェアに対して暗号化動作の検知，プロセスの特定，停止が可能であった．表 7，表 8 を見ると，総容量が最も小さい条件 5 の時には，ランサムウェアの動作を停止するまでに暗号化されるフォルダの数が大きくなることが分かる．特にハードリンクを用いた条件 5′ の場合には，ランサムウェアの種類によって 1 フォルダの暗号化完了後は，既に暗号化されたファイルであると判断され，デコイファイルの暗号化が行われないという結果が得られた．一方，総容量が大きい条件 1 の場合には，暗号化されるフォルダ数

が小さくなることが分かる．上記のことから，システムの有効性を保ちつつ，ストレージに占めるデコイファイルの容量を削減するためには，条件 3 または条件 3′ に示した通りにデコイファイルを生成することが適切であると考えられる．条件 3′ ではハードリンクを用いているが，総容量が適切であるため，条件 5′ で確認されたデメリットが発生していない．なお，ハードリンクを利用した場合デコイファイルのサイズはハードリンクを用いない場合に比べ，1/＜監視フォルダ数＞ に削減可能である．

提案手法では，暗号化動作の検知を行った後にプロセスの特定を行うことで動作の軽減を実現すると述べた．実際に実装し負荷を確認したところ表 10 に示した通り，通常時では非常に負荷が少ないことが確認された．

また，本システムの検証に加え GitHub 上で公開されている R-Locker の検証も同様に実施した．その結果は 5.5 節に示した通りである．R-Locker に対して，本システムは検知の精度と CPU 使用率，メモリの使用量に関して優れていることが示された．一方でストレージの使用量と検知のリアルタイム性に関しては R-Locker が優れていることが確認された．また R-Locker に関しては自動ですべてのフォルダにデコイファイルのリンクを生成するため，他のソフトウェアが正常に動作しなくなるといった問題も発生した．

7. まとめ

本稿では，近年脅威となっている暗号化型ランサムウェアに対して，デコイファイルを用いて暗号化動作の検知とプロセスの特定を行うシステムを提案した．そして，14 種のランサムウェアと 5 種の良性プログラムを用いて有効性の評価を行った．さらに検知前と検知後に分けて使用するリソースを確認した．また，デコイファイルのサイズ内訳を変えた 5 つの条件でハードリンクを使用する場合としない場合でそれぞれ検証し，条件の違いによる有効性への影響を示した．その結果，高精度かつ低負荷で動作可能なシステムであることを示した．

謝辞 本研究の一部は，JSPS 科研費 20K11818, 19K11968, 19H04108 の支援により行った．

参考文献

[1] IPA：情報セキュリティ 10 大脅威 2021，available from ⟨https://www.ipa.go.jp/security/vuln/10threats2021.html⟩ (参照 2021-01-19).
[2] Check Point： Check Point Rearch Cyber Attack Trends: 2020 Mid-Year Report, available from ⟨https://www.checkpoint.com/downloads/resources/cyber-attack-trends-report-mid-year-2020.pdf⟩ (参照 2020-12-9).
[3] IPA： ランサムウェアの脅威と対策〜ランサムウェアによる被害を低減するために〜, available from ⟨https://www.ipa.go.jp/files/000057314.pdf⟩ (参照 2021-1-21).
[4] IPA： 情報セキュリティ白書, available from ⟨https://www.ipa.go.jp/files/000070313.pdf⟩ (参照 2021-2-10).
[5] Kaspersky： ランサムウェア（身代金要求型ウイルス）の多様な手口と対策方法, available from ⟨https://www.kaspersky.co.jp/resource-center/threats/ransomware-examples⟩ (参照 2020-12-9).
[6] J. A. Gómez-Hernández, L. Álvarez-González, P. García-Teodoro：R-Locker: Thwarting ransomware action through a honeyfile-based approach, Computers & Security, Vol.73, pp.389-398, 2018.
[7] J. A. Gomez-Hernandez： R-Locker, available from ⟨https://github.com/JA-Gomez-Hernandez/R-Locker⟩ (参照 2021-01-19).
[8] S. Mehnaz, A. Mudgerikar, E. Bertino, RWGuard：A Real-Time Detection System Against Cryptographic Ransomware, The 21st International Symposium on Research in Attacks, Intrusions and Defenses, pp.114-136, 2018.
[9] 本多俊貴, 向山浩平, 白井丈晴, 大木哲史, 西垣正勝：ユーザのコンテンツ編集操作を考慮した暗号化型ランサムウェア検知の検討, 情報処理学会論文誌, Vol.60, No.9, pp.1477-1488, 2019.
[10] 田中 智也，小池 一樹，小林 良太朗，加藤 雅彦：ダミーファイルを利用した暗号化型ランサムウェア対策システムの実装，コンピュータセキュリティシンポジウム 2019 論文集，pp.163-169，2019.
[11] MBSD Blog：標的型攻撃ランサムウェア「Ryuk」の内部構造を紐解く, available from ⟨https://www.mbsd.jp/blog/20191211.html⟩ (参照 2021-2-9).
[12] Watchdog 0.8.2 Documentation： API Reference, available from ⟨https://pythonhosted.org/watchdog/api.html⟩ (参照 2021-2-5).
[13] Microsoft Docs： Process Monitor v3.60, available from ⟨https://docs.microsoft.com/en-us/sysinternals/downloads/procmon⟩ (参照 2020-12-9).