未知のランサムウェアの分離に基づく検知・復元手法の提案

熊坂 賢人[2] 永野 凜太郎[2] 稲村 浩[1] 石田 繁巳[1]

概要：
近年，端末やそのネットワーク上のファイルを暗号化し，その復号と引き換えに身代金を要求する暗号化型ランサムウェアが流行している．ランサムウェアに感染した端末が，ファイルサーバ上のファイルを暗号化する場合もあり，大量のファイルが暗号化されるリスクがある．ランサムウェアを検知するために機械学習を用いる手法があるが，学習に用いられていない検体への対策は難しい．本稿では，ファイルサーバに対する未知のランサムウェアによる暗号化を復元するために，囮となるディレクトリである “囮空間” と，囮空間に対するランサムウェアの挙動をもとに暗号化によるファイル操作を特定し，行われたファイル操作をもとに戻す操作を行うことで，暗号化されたファイルを復元する手法を提案する．囮空間を設置したファイルサーバに対して暗号化攻撃と正規のユーザを模したファイル操作を並行して行い提案手法の実現性を評価した．評価のために 5003 個のファイルの暗号化と 500 個のファイルアップロード操作，500 個のファイルダウンロード操作を並行して行った結果，提案手法による検知率は 94.9%，誤検知率は 2.8% であり，提案方式について一定の実現性を示した．

1. はじめに

ランサムウェアの暗号化攻撃は世界中で増加傾向にあり，深刻な問題となっている．日本では，IPA の公開した「情報セキュリティ 10 大脅威」の組織のランキングにおいて，2021 年から 2023 年まで「ランサムウェアによる被害」が 1 位となった [1]．警察庁の発表した，企業・団体等におけるランサムウェア被害の件数は年々増加傾向にあり，令和 4 年度以降高い水準を保ち続けている [2]．令和 5 年上半期の被害報告件数，令和 2 年下半期の被害件数のおよそ 5 倍となっている．

ランサムウェアはファイルサーバ上のファイルを暗号化する場合もある [3]．ファイルサーバは通常，組織内にある複数の端末からのアクセスを可能としているため，ランサムウェアに感染した端末があった際，大量のファイルが暗号化されるリスクがある．ファイルサーバ上のファイルが暗号化されて復旧作業を行う場合，感染した端末のみを復旧するのに比べて業務に大きな影響が出る．このような組織へのランサムウェアの攻撃を防ぐために，それぞれの端末上で対策すると，導入や管理のコストが大きくなる．

一般に，ランサムウェアからの被害を防ぐためにウィルス対策ソフト等が用いられるが，その多くは機械学習による検知やランサムウェアのハッシュ値をあらかじめ用意したランサムウェアのハッシュ値のリストと比較して検知する [4,5]．しかしながら，これらの検知手法では，解析されていない未知のランサムウェアを検知することが難しい．実際，機械学習を用いた暗号化攻撃検知手法では，初出時点から時間を置いた時点での検体を用いた評価の際に大幅に検知率が下がっている [6,7]．

本稿では，機械学習やランサムウェア検体に関する事前の情報を使わず，未知のランサムウェアによる暗号化攻撃特有のファイル操作を特定し復元することを目的とする．機械学習を使わずにランサムウェアを検知する方法として，本研究では通常ユーザが操作しないという前提で作成されたディレクトリである「囮空間」を用いて暗号化攻撃の検知と挙動の観測を行う．ファイルサーバ上のファイルへの操作を監視することで，囮空間への操作が行われた際にランサムウェアによる攻撃として検知できる．クライアント端末のみで利用可能なデータは用いず，ファイルサーバ上で入手できるデータのみを用いて暗号化攻撃を特定し復元する．

提案手法の実現には永野らの提案したファイルサーバにおける細粒度バックアップと復元機能を利用する [8]．暗

[1] 公立はこだて未来大学システム情報科学部
Sch. Systems Information Science, Future Univ. Hakodate
[2] 公立はこだて未来大学大学院システム情報科学研究科
Grad. Sch. Systems Information Science, Future Univ. Hakodate

号化攻撃と正規のユーザを模したファイル操作を並行して行った後，提案手法によるファイル復元を行い，暗号化前後のファイルを比較することで評価する．

本稿の構成は以下の通りである．第 2 章では，本目的を達成するための関連研究について述べる．第 3 章で提案手法を述べた後，第 4 章で評価実験について述べる．最後に第 5 章でまとめとする．

2. 関連研究

2.1 細粒度バックアップを用いた復元手法

永野ら [8] は，ファイルサーバ上のファイルへの暗号化攻撃に伴うファイル拡張子の変更操作を利用し，ファイルを自動復元する手法を提案した．この手法では，拡張子を特有のものへ変更する検体の暗号化攻撃に対して，ファイルサーバ上でファイル操作列を細粒度に取得したバックアップを用いてロールバックする．

ファイル操作列とは，タイムスタンプ，実行されたファイル操作関数，ファイル操作関数の実行対象となるファイルやディレクトリの絶対パス，変更前のファイル情報などを出力したものである．ファイル操作列からのロールバックには，細粒度バックアップを用いる．細粒度バックアップは，「ファイルが作成されたら削除する」「書き込みが行われたら，行われる前のファイルを復元する」といった反転操作を可能にする情報を操作列と共に保存することで，ファイル操作列の情報から一列ごとでのファイル復元を可能とする．ファイル操作列から，ファイル拡張子の変更にあたる操作を検知した際に，そのファイル操作列の反転操作を行うことで，ファイルを自動復元する．

しかしながら，この手法は拡張子変更のみを行うランサムウェアに限定した手法であり，ベースネームを変更する検体には対応できない．ファイルを復元するためには，暗号化攻撃によって生成されるファイル操作列を特定する必要があるが，この手法では拡張子の変更のみを契機に特定する．

2.1.1 ファイル操作関数

本稿では，永野らの提案 [8] と同じくファイルサーバ上のファイルをクライアント端末から操作した際に実行される関数のことを「ファイル操作関数」と呼ぶ．実装には NFS ファイルシステムを用いているため具体的なファイル操作は NFS プロトコルの手続きとして定義できる．NFSv3 ファイルサーバに対してクライント端末からファイル操作を行うと，ファイルサーバ上で ONC RPC [9, 10] によってファイル操作関数が実行される．ONC RPC とは，Sun Microsystems によって開発されたシステムで，クライアントとサーバ間での関数の呼び出しを可能とする．このシステムを用いて，クライアントはサーバ上で関数を実行し，サーバから処理結果を受け取ることができる．ファイルサーバではこのシステムを用いて，クライアント端末からファイルサーバへリクエストを送信し，受け取ったサーバはそれに対応した処理を行いクライアント端末へと処理結果を返す．リクエストを受け取ったサーバが実行する関数をファイル操作関数と定義する．ファイル操作関数の一覧を**表 1** に示す．

実行された全てのファイル操作関数とその引数は，ファイルサーバ上で記録できる．これによって，クライアント端末に依存することなくシステムを構築できる．

表 1 ファイル操作関数一覧 [11]

Operation	Function
FSSTAT	Gets dynamic file system information
CREATE	Creates a file system node; may be a file or a symbolic link
GETATTR	Gets file or directory attributes such as file type, size, permissions, and access times
LINK	Creates a hard link in the remote file system
LOOKUP	Searches directory for file and return file handle
MKDIR	Creates a directory
NULL	Does nothing; used for testing and timing of server response
READ	Reads an 8-Kbyte block of data (32-KByte blocks). This can be raised beyond 64 KBytes for TCP.
READDIR	Reads a directory entry
READLINK	Follows a symbolic link on the server
RENAME	Changes the directory name entry
REMOVE	Removes a file system node
RMDIR	Removes a directory
SETATTR	Changes file or directory attributes
SYMLINK	Makes a symbolic link in a remote file system
WRITE	Writes an 8 Kbyte block of data (32-KByte blocks). This can be raised beyond 64 KBytes for TCP.
ACCESS	Check access permission
MKNOD	Create a special device
READDIR	Read from directory
READDIRPLUS	Extended read from directory
FSINFO	Get static file system information
PATHCONF	Retrieve POSIX information
COMMIT	Commit cached data on a server to stable storage

2.2 囮ファイルを使った検知手法

田中ら [12] は，既知のランサムウェアのファイルアクセスパターンから，それらのランサムウェアに対して有効な囮となるファイルの設置方法を提案した．WannaCry，Jigsaw，Cerber，TeslaCrypt の 4 種類のランサムウェアに対して，暗号化順をフォルダ単位とファイル単位で調査し，それらのフォルダの中に，最初にアクセ

スされるようなファイル名のファイル 90000 個を作成した．C:Users/USERNAME/Documents のフォルダ内へターゲットファイルを設置し，そのターゲットファイルが暗号化されるまでの時間を計測した．実験の結果 Jigsaw はエラーが発生し停止してしまったが，WannaCry，Cerber，TeslaCrypt による暗号化攻撃を 20 時間から 80 時間程度の間遮断できた．

しかしこの研究は，ランサムウェアが最初に暗号化するディレクトリやファイルが既知であることが前提であり，新しい検体に対しては効力を発揮しない可能性がある．また，このシステムは Windows10 上で動作し，ファイルサーバに関して議論していない．

荻原ら [13] は，囮となる暗号化対象のファイルを用いて，ランサムウェアによる暗号化を検知し，アクセス情報からランサムウェアを特定・停止するシステムを提案した．このシステムは，Windows10 上で動作し，デコイファイルが削除またはパスが変更されるのを契機にランサムウェアを検知した後，プロセスを特定・停止する．この手法ではプロセス ID を用いてデコイファイルを変更したプロセスを特定・停止しているが，ファイルサーバ上ではプロセス ID を取得できないため同様の特定手法を用いることが難しい．

3. 提案手法

本稿では，ファイルサーバのみで実現可能な未知のランサムウェアによる暗号化攻撃からの回復手法を提案する．ランサムウェアによる暗号化攻撃中のファイル操作列をその場で取得して，パターンマッチのためのテンプレートを作成する．それを悪性と良性が混在する直近のファイル操作列にマッチングさせ，暗号化攻撃に由来するファイル操作列を抜き出すことで攻撃を受けたファイルを復元する．

3.1 キーアイデア

ランサムウェアの暗号化攻撃を検知しその悪性のファイル操作列を確実に含んだデータを取得するために，本研究ではファイルサーバ上に配置する囮空間と呼ぶディレクトリを提案する．ファイルサーバ上に配置することで端末からの隔離と，複数のファイルに対するランサムウェアの攻撃時の挙動の観測・データ化を可能にする．ファイルサーバを使用する際，ユーザは囮空間に対してファイル操作を行わないこととする．これは例えば実際の業務利用に見える無害なファイルを並べた，実在しないユーザの作業エリアを作成することで実現できる．これにより，囮空間に対するすべてのファイル操作はランサムウェアによって行われた暗号化攻撃によるものと想定できる．**図 1** に示すように，囮空間において観測したランサムウェアによるファイル操作列から暗号化攻撃特有のファイル操作列パターンを生成し，それをランサムウェア操作列パターンと定義する．

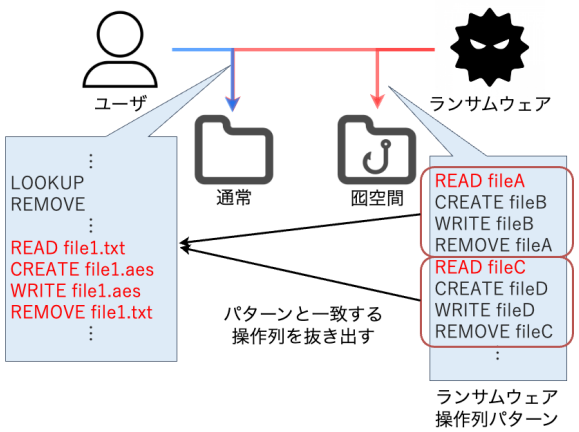

図 1 暗号化攻撃の検知と挙動の観測のための囮空間

ユーザとランサムウェアの操作が混在しているファイル操作列からランサムウェア操作列パターンと一致する操作列を抜き出すことで，暗号化攻撃によって生成されたファイル操作列を特定できる．

3.2 提案システムの概要

ランサムウェアによるファイルの暗号化に対する提案システムによるファイル復元までの流れを**図 2** に示す．提案システムは，ファイル操作列出力機能，ランサムウェア操作列パターン生成機能，操作列識別機能，ファイル操作列ロールバック機能の 4 つの機能で構成される．ファイル操作列出力機能においてファイルサーバに対して行われたファイル操作を監視し，「ファイル操作列」を記録する．囮空間に対するファイル操作はランサムウェアによる操作であることから，囮空間に対するファイル操作列をランサムウェア操作列パターン生成機能によってマッチングに適した形に抽象化してパターンを生成する．生成されたパターンを用い，操作列識別機能によって通常のディレクトリに対するファイル操作列の中から暗号化攻撃によるファイル操作列を抽出する．ファイル操作列ロールバック機能によって抽出された操作列に対して復元を行うための反転操作を適用することにより，暗号化攻撃の被害にあったファイルを復元する．

以降では各機能について詳述する．

3.3 ファイル操作列出力機能

ファイル操作列出力機能は行われたファイル操作に対してファイルサーバ内でファイル操作列を生成して出力する．ファイル操作列出力機能は，永野らの提案した機能をもとに構築する [8]．図 2 に示すように囮空間に対する操作列は分離され，ランサムウェア操作列パターン生成機能の処理に渡される．

ファイル操作列は “実行タイムスタンプ”，“ファイル操作関数名”，“ロールバックに必要なパラメータ” で構成さ

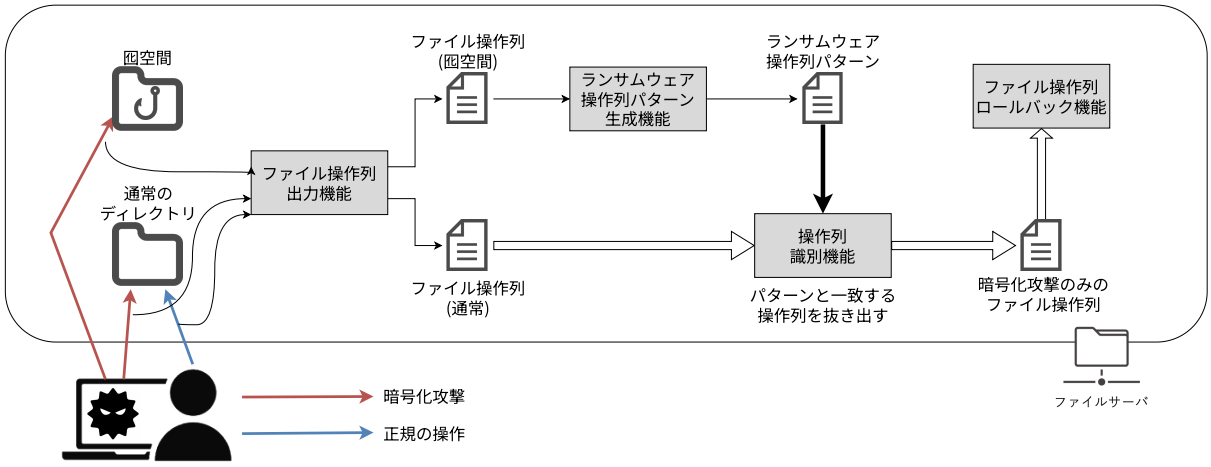

図 2　提案システムによるファイル復元までの流れ

れるレコードの列として定義される．例えば write 関数が実行される際，ロールバックに必要なパラメータとして対象となるファイルパスと，変更前のファイル情報が出力される．

ファイル操作列の 1 レコードの例を**表 2** に示す．この例では，時刻 “1698147037.362647” に，“WRITE 関数” が “/exports/file1.aes” を対象に呼び出され，その変更前のファイルを “/backup/˜362810-before_write” に保存している．変更前のファイルを保存しているディレクトリである “/backup” は保護のためにプライベートとしサーバ外に共有しないため，ランサムウェアによってクライアント端末からこれらのファイルへの操作を行うことは不可能である．

3.4　ランサムウェア操作列パターン生成機能

ランサムウェア操作列パターン生成機能では，囮空間に対するファイル操作列からランサムウェアによる暗号化攻撃の操作列を識別するためのランサムウェア操作列パターンを生成する．ランサムウェア操作列パターンは，ファイル操作関数と対象となる抽象化されたファイルパス名で構成されるレコードの列として定義される．

本稿の手法は，悪意の暗号化操作をパターンマッチして特定するために，冗長性の少ない最小限のランサムウェア操作列パターンを生成するアプローチをとる．最小限のランサムウェア操作列パターンの生成には，マッチング対象の個々のファイルの差異を吸収するための適切な抽象化が必要である．具体的には，暗号化攻撃の対象となったファイルのサイズ，サイズに応じて複数回実行された同一のファイル操作の数，複数のファイルの暗号化での繰り返し操作からのファイルひとつ分の操作列の抜き出しと，具体的なファイル名を抽象化，生の操作列に含まれる識別に不要な操作の除外を行う．

3.4.1　ファイルのサイズと複数回実行されたファイル操作の抽象化

暗号化の対象となったファイルのサイズに応じて生成されるファイル操作列の数が変わることがある．提案手法で用いたファイルサーバでは，ペイロードサイズを越える大きさのデータによるリクエストは複数個のリクエストに展開されるため繰り返しとして記録される．この仕様のため，同じ対象となるファイルとファイル操作関数を持つファイル操作列がファイルサイズに応じた回数分連続して生成される．

このようにして展開された複数のファイル操作列を識別し，ひとつのファイル操作に併合する．一度のリクエストによるファイル操作列のみを識別するために，ファイル操作列のファイル操作関数と対象となるファイルパスを利用する．連続して同じファイル操作関数が同じファイルパスに対して実行された際，それらは同じリクエストによって実行されたファイル操作列と見なす．

3.4.2　ファイルひとつ分の操作列の抜き出し

次に，複数のファイルの暗号化での繰り返し操作からのファイルひとつ分の操作列の抜き出しを行う．ファイル操作列をひとつのファイルに対するファイル操作列ごとに分割するための指標として，ファイル操作関数の READ 関数を用いる．囮空間に対する複数のファイル操作列を，一度の暗号化操作で生成される操作列ごとに分割する．これは，ランサムウェアが暗号化操作を行う前，必ず暗号化対象のファイルを読み取る必要があるからである．READ を含む操作列からつぎの READ 関数が呼び出される操作列または最後尾の操作列から先にみつかった操作列までをファイルひとつ分の暗号化操作による操作列として記録する．

3.4.3　ファイル名の抽象化

ファイルひとつ分の操作列の抜き出しを行った後，具体的なファイル名を抽象化する．ファイル名を抽象化することにより，特定のファイルに依存しないランサムウェア操

表 2 ファイル操作列 1 レコードの例

タイムスタンプ	ファイル操作関数	対象ファイル	編集前ファイルの保存先
1698147037.362647	WRITE	/exports/file1.aes	/backup/˜362810-before_write

作列パターンを生成する．抽象化のため，新しいファイル名が現れるたびにそれぞれのファイルに対して A，B のようにラベル付けをして置換する．

3.4.4 識別に不要な操作の除去

ファイル操作列の抽象化を行った後，生の操作列に含まれる識別に不要な操作を除外する．提案手法では，ファイル操作関数が “COMMIT”，“ACCESS”，“FSINFO”，“FSSTAT”，“LOOKUP”，“GETATTR”，“READDIR”，“READDIRPLUS”, “PATHCONF”, “NULL”, であるファイル操作列を除外した．これは，内部状態に依存し必ずしもクライアントの挙動によって動作しないことや，ユーザが意図せずに囮空間に対してこれらの操作を行うなどの理由により本手法では必要のない情報であると考えられるからである．

それぞれの操作列を除外した理由について**表 3** に示す．表 3 の 1 について，この操作関数に相当するリクエストをユーザが囮空間に対して送信してしまうケースがある．Windows OS であれば Windows Explorer，Mac OS であれば Finder といったファイルマネージャが，アクセス可能なすべてのディレクトリやファイルに対して表 3 の 1 に相当するリクエストを送信することが原因である．ランサムウェアによる操作として断定するために，本手法では囮空間を対象とするファイル操作列を抽出した後，これらの操作関数を含むファイル操作列を除外する．

表 3 の 2 は，READ 関数や WRITE 関数が呼ばれる際に同時に呼ばれ，これら単体で呼ばれるケースはほとんどない．表 3 の 3 について，本手法ではファイル単位での暗号化を検知することによりディレクトリ単位の情報はノイズとなるため除外する．表 3 の 4 はファイルサーバに接続した際に呼ばれ，それ以外で呼ばれることはほぼ無い．表 3 の 5 は RPC における疎通性の確認目的であり，ファイル操作に関係ないと判断した．

カテゴリ	説明	操作関数
1	単体で呼ばれるケースは ほとんどない	COMMIT ACCESS FSINFO FSSTAT
2	ファイルマネージャ等によって 記録されることがある	LOOKUP GETATTR
3	ディレクトリの読み出し操作は パターンに含まない	READDIR READDIRPLUS
4	マウントした際に呼ばれるのみ	PATHCONF
5	対応するファイル操作がない	NULL

表 3 除外するファイル操作関数と説明

複数のランサムウェア操作列パターンを生成した後，完全一致するランサムウェア操作列パターンがあればそれらを併合する．

3.5 操作列識別機能

操作列識別機能の提案と実装について示す．操作列識別機能は，ランサムウェア操作列パターンを用いて，ランサムウェアによって生成されたファイル操作列と正常なユーザによって生成されたファイル操作列とを識別する．

操作列識別機能の主要部について，擬似コードにしたものを Algorithm1 に示す．このアルゴリズムは，ファイル操作列中でランサムウェア操作列パターンと一致する部分を見つけ出力する．それぞれの操作列を入力として Fseq と Rpat とに格納する．Fseq と Rpat を要素ごとに比較し，Fseq に同一の操作が繰替えされる場合は 1 つと見なして一致判定を行っている．Rpat の要素との一致が最後まで到達したら，その操作列をランサムウェアによる操作列であるとして foundmatch 関数に出力する．Rpat の最後まで到達しても Fseq に余りがある場合はパターンをオーバーランしたものとして Fseq に与えられたファイル操作列はランサムウェアによる操作列ではないと判定する．すべてのファイル操作列に対して上記の操作を完了し，foundmatch 関数によって出力されたファイル操作列をすべてランサムウェアによって生成された操作列とする．

Algorithm 1 操作列識別機能

```
Fseq[Flen] ← FileOperationSequence
Rpat[Rlen] ← RansomOperationPattern
MatchSeq[]                          ▷ 一致した操作列を格納する変数
Fidx, Ridx ← 0
extension ← 10
Function foundmatch(MatchedSeq[]) ▷ 識別後のランサムウェア操作部分列を出力するための関数
function RIDENTIFY
    while Findx ≦ Flen do
        for i = 0; i ≦ Ridx + extension; i++ do
            if Fseq[Fidx + i] == Rpat[Ridx] then
                MatchSeq[Ridx] ← Fseq[Fidx + i]
                Ridx++
                if Ridx == Rlen then
                    FOUNDMATCH(MatchSeq)
                    break
                end if
            end if
        end for
        MatchSeq[] ← NULL
        Fidx++
    end while
end function
```

3.6 ファイル操作列ロールバック機能

暗号化攻撃のみのファイル操作列を識別した後，ファイル操作列ごとのロールバックによる暗号化されたファイルの復元を行う．ファイルの復元作業では，細粒度バックアップシステム [8] を用いる．

この手法では，それぞれのファイル操作関数に対応したファイル操作を行うことでファイル操作列 1 列ごとのロールバックを行う．ファイル操作列のタイムスタンプ情報と逆順にロールバックを実行することで，ファイル操作列を全て正常にロールバックする．

ロールバックを行う際のファイル操作関数と，その反転の操作について [8] から引用した図を**図 3** に示す．例えば WRITE 関数が呼ばれた際には，書き込みが行われる前のファイル情報を書き込むことでロールバックする．他にも，REMOVE 関数が呼ばれた際には，削除前のファイルを生成することでロールバックする等の操作により任意のファイル操作列のみを復元する．

ファイル操作関数	操作	ロールバック操作
WRITE	ファイルへの書き込み	書き込みが行われる前のバックアップを保存しておきファイル内容を復元する
SETATTR	ファイル属性の指定	変更前のファイル属性を記録しておき，変更する．
MKDIR	ディレクトリの作成	作成したディレクトリの名前を記録しておき，作成したディレクトリを削除する
SYMLINK	シンボリックリンクの作成	作成したシンボリックリンクの名前を記録しておき，作成したシンボリックリンクを削除する
MKNOD	特殊ファイル (名前付きパイプ，デバイスファイル) の作成	作成した特殊ファイルの名前を記録しておき，作成した特殊ファイルを削除する
REMOVE	ファイルの削除	削除したファイルのバックアップを保存しておきバックアップをもとにファイルを再生成する
RMDIR	ディレクトリの削除	削除されたディレクトリの名前を記録しておき，ディレクトリを再生成する．
RENAME	ファイル名の変更	変更される前のファイル名を保存しておき，変更されたファイル名を変更される前のファイル名に再変更する
LINK	ハードリンクの作成	生成したハードリンクの名前を記録しておき，ハードリンクを削除する

図 3 ファイル操作関数の意味と反転操作によるロールバック手順 [8]

3.7 本手法で対象とするランサムウェアの挙動

前項までのパターン構成手法の中でランサムウェアの挙動についていくつかの前提を置いており，提案手法の適用において制約となるためここで明確化する．提案手法で対象とするランサムウェアは，逐次的にファイルを暗号化するランサムウェアである．ランサムウェアは，ファイルを暗号化する際にひとつのファイルに対しての操作を完了した後に次のファイルへの操作を始める．本稿で対象としないランサムウェアの挙動の例として，ディレクトリ内のファイルすべてを読み込んだ後にそれぞれのファイルを暗号化する挙動が挙げられる．

4. 提案手法による検知・復元手法の評価

本稿では，提案手法の実現性を評価するために，正常なユーザによるファイル操作の一例から疑似ランサムウェアによるファイル操作の復元可能性の実験的評価を行う．提案手法の実現可能性について，

1) ランサムウェアによる暗号化攻撃を受けたファイルを復元できたか
2) 正常なユーザによるファイル操作を誤って復元することがないか

の 2 点を評価する．

本稿では評価にあたって，暗号化攻撃を受けたファイルの総数のうち正しく復元できた割合を復元率，正常なユーザによるファイル操作の総数のうち誤って復元した割合を誤検知率と定義する．

4.1 実験

疑似ランサムウェアによる暗号化と正常なユーザを想定したファイル操作を並行して行った後，提案手法によってファイルを復元する．ファイルサーバへ行う操作について第 4.3 項に，提案手法による復元について第 4.4 項に示す．

実験環境を**図 4** に示す．本実験では，ファイルサーバ用とクライアント用に 2 台の PC を使用し，ファイルサーバに対してクライアント用 PC から暗号化攻撃を行った．ファイルサーバ用の PC には，ファイル操作列を出力するよう UNFS3 を拡張したファイルサーバを導入し，その共有ディレクトリ内に囮空間用フォルダと暗号化対象のフォルダを，非共有ディレクトリにロールバック用データの保存先フォルダとファイル操作列の保存先フォルダを設置した．クライアント PC には，ファイルアップロード用のフォルダとファイルダウンロード先のフォルダを設置し，疑似ランサムウェアを設置した．それぞれの PC の仕様を

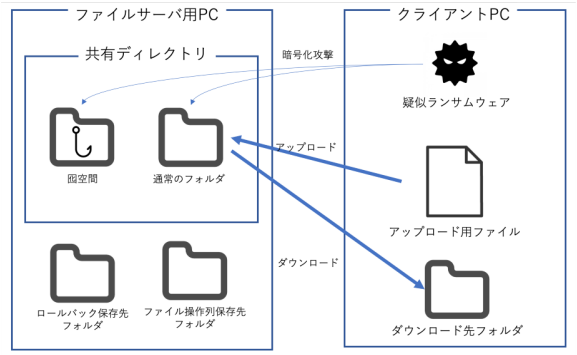

図 4 実験環境

表 4 と**表 5** に示す．

表 4 ファイルサーバ用 PC の仕様

製品名	G2 mini pc n100
CPU	Intel(R) N100
メモリ	12GB
OS	Debian GNU/Linux 10 (buster)

表 5 クライアント PC の仕様

製品名	MacBook Air (M1，2020)
CPU	Apple M1
メモリ	16GB
OS	macOS Ventura 13.0

4.2 使用するファイルサーバ

本稿ではオープンソースファイルサーバである UNFS3 [14] を拡張して提案システムを構築する．UNFS3 は，NFSv3 (Network File System ver3.0) プロトコル [15] をユーザスペース上で動作させるファイルサーバ実装である．提案システムを構築する際に使用した UNFS3 のリリースバージョンは 0.9.22 である．

4.3 特定すべきファイル操作と操作対象ファイル

実験ではファイルサーバに対して疑似ランサムウェアを用いた暗号化と正規のユーザを模したファイル操作を行う．本項では，疑似ランサムウェアによる暗号化手順と正規のユーザを模したファイル操作，操作対象のファイルについて述べる．

4.3.1 疑似ランサムウェア

本稿では疑似ランサムウェアを用いて評価実験を行う．疑似ランサムウェアは暗号化攻撃の挙動をエミュレートするように制作した評価用ソフトウェアである．疑似ランサムウェアを用いることで，暗号化タイミングの制御が容易になり，他のファイル操作と同期しながら暗号化することができる．しかしながら，C&C サーバを介した操作を行う等の操作を行っていないため，実際のランサムウェアと挙動が異なる部分がある．

疑似ランサムウェアによる暗号化は以下の手順で動作する．

(1) 暗号化対象ファイルを読み取る
(2) 暗号化対象ファイルを削除する
(3) 暗号化後データの格納先ファイルを作成する
(4) 暗号化後データを書き込む

暗号化方式は AES を用いる．対象ファイルの読み出し・削除を行った後に暗号化データ用ファイルの作成と書き込みを行う一連の操作を，指定したディレクトリ内のサブディレクトリを含むすべてのファイルに行う．

4.3.2 操作対象のファイル

ファイルサーバ上に暗号化するディレクトリと暗号化しないディレクトリを設置し，クライアント PC 上にアップロード用のディレクトリを設置する．暗号化するディレクトリ内に囮空間となるディレクトリを設置し，囮空間内に 15kB のテキストファイルと 32kB, 153kB のバイナリファイルの計 3 個のファイルを，囮空間でない通常のディレクトリ内に 8Byte のテキストファイルを 5000 個設置する．さらに，暗号化しないディレクトリ内に 8Byte のテキストファイルを 500 個設置する．囮空間内のファイルは，ファイルサイズが変わることによるファイル操作列の変化を観測するためにファイルサイズを変化させた．通常のディレクトリ内のファイルは，提案手法の実現可能性を評価するためにサイズを変化させる必要がないと判断し，同じファイルサイズのテキストファイルを設置した．

4.3.3 正規のユーザを模したファイル操作

正規のユーザを模した良性のファイル操作として，テキストファイルの送受信を行う．ファイルサーバ上の暗号化しないディレクトリ内のファイル 500 個をダウンロードするとともに，クライアント PC からファイルサーバ上の暗号化しないディレクトリへアップロードする．ダウンロードとアップロードの操作はクライアント PC 上で別のプロセスとして動作し，それぞれ 3 秒おきに 1 個のファイルをダウンロードまたはアップロードする．クライアント PC にはアップロード用のテキストファイルとして 8Byte のファイル 500 個をあらかじめ用意した．

4.4 提案手法による復元結果

評価のために，暗号化が行われる前のディレクトリを複製し，複製先に正規のユーザによるファイル操作のみを行う．複製元のディレクトリへ暗号化と並行して正規のユーザによるファイル操作を行い，提案手法により復元を行う．その後，複製元のディレクトリと複製後のディレクトリを比較することで評価する．

提案手法による復元結果を**表 6** に示す．暗号化したファイル 5003 件に対して，提案手法による復元を試みた結果，復元率は 94.9%であった．正常なユーザによるファイル操作 1000 件に対して提案手法を適用した結果，誤検知率は

表 6 提案手法による復元結果

評価対象のファイル操作	回数	割合
ランサムウェアに暗号化されたファイル	5003 件	-
正しく復元した	4746 件	94.9%
復元されなかった	257 件	5.1%
良性なユーザ操作の対象ファイル	1000 件	-
ダウンロード操作: 正しく検知しなかった	500 件	50.0%
ダウンロード操作: 暗号化として誤検知した	0 件	0.0%
アップロード操作: 正しく検知しなかった	472 件	47.2%
アップロード操作: 暗号化として誤検知した	28 件	2.8%

2.8%であった．誤検知したファイル操作の内訳はダウンロード操作が 0 件，アップロード操作が 28 件であった．この誤った検知により，正常なユーザが送信したファイル 500 個の内，28 個を誤って削除した．

正しく復元されなかったファイルに関するファイル操作列を観察すると，暗号化中のファイル操作列中に正常なユーザによるファイル操作列が含まれていた．このように，検知したい操作列中に他のファイル操作列が暗号化中の操作列を正しく検知できなかった．正規のユーザによる操作について，提案手法によって誤って削除してしまったファイルはあるが，正規のユーザによる操作全体に対する割合としては低い．

これらの結果より，提案手法による復元の実現可能性を示せた．

5. おわりに

本稿では，囮空間を用いることで未知のランサムウェアによる暗号化攻撃から攻撃を受けたファイルを復元する手法を示した．

関連研究では，ランサムウェア検体に関する事前の情報を使って，ファイルサーバ上で復元を行う手法や囮となるファイルを用いる手法の提案がされていたが，本稿では機械学習やランサムウェア検体に関する事前の情報を使わず，未知のランサムウェアによる暗号化攻撃特有のファイル操作を特定し復元することを目的とした．目的を達成するため，本稿では囮空間を用いて攻撃中のファイル操作列をその場で取得し，それを直近のファイル操作列に適用して暗号化によって生成されたファイル操作列の識別をした後，攻撃を受けたファイルを復元する手法を提案した．

本稿では，提案手法の実現性を評価するために，正常なユーザによるファイル操作の一例から疑似ランサムウェアによるファイル操作の復元可能性の実験的評価を行った．5003 個のファイルの暗号化と 500 個のファイルアップロード操作，500 個のファイルダウンロード操作を並行して行い，提案手法による検知率は 94.9%，誤検知率は 2.8%であった．評価の結果，提案方式について一定の実現性を示した．

参考文献

[1] 独立行政法人情報処理推進機構 (IPA)：情報セキュリティ 10 大脅威 (2023). https://www.ipa.go.jp/security/10threats/10threats2023.html(2023-07-18 アクセス).

[2] 警察庁：令和 5 年上半期におけるサイバー空間をめぐる脅威の情勢等について (2023). https://www.npa.go.jp/publications/statistics/cybersecurity/data/R05_kami_cyber_jousei.pdf(2024-01-23 アクセス).

[3] 日本情報通信株式会社：ファイルサーバーを狙うランサムウェアの脅威とその対策 (2021). https://www.niandc.co.jp/tech/20211227_2120/(2024-01-24 アクセス).

[4] ESET: EDR とアンチウイルスの違い. https://www.eset.com/jp/topics-business/edr-antivirus-comparison/(2024-01-23 アクセス).

[5] トレンドマイクロ：不正プログラム対策機能のファイルハッシュ計算 (File Hash Calculation) について (2019). https://trendmicro.my.site.com/dcx/s/solution/1116309?language=ja(2024-01-23 アクセス).

[6] Continella, A., Guagnelli, A., Zingaro, G., De Pasquale, G., Barenghi, A., Zanero, S. and Maggi, F.: Shieldfs: a self-healing, ransomware-aware filesystem, *Proceedings of the 32nd annual conference on computer security applications*, pp. 336–347 (2016).

[7] Gupta, A., Prakash, A. and Scaife, N.: Prognosis negative: Evaluating real-time behavioral ransomware detectors, *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, IEEE, pp. 353–368 (2021).

[8] 永野凜太郎，稲村　浩，石田繁巳：拡張子変更に注目したファイルサーバにおけるランサムウェアの暗号化攻撃からの回復手法の検討，マルチメディア，分散，協調とモバイル シンポジウム論文集 (DICOMO 2023) (2023).

[9] Sun Microsystems: RPC: Remote Procedure Call Protocol specification: Version 2, RFC 1057 (1988).

[10] Corporation, I.: ONC RPC concepts (2024). https://www.ibm.com/docs/en/cics-ts/5.6?topic=rpc-onc-concepts.

[11] Oracle Corporation and/or its affiliates: NFS Version 3 Features and Operations (2010). https://docs.oracle.com/cd/E19620-01/805-4448/z4000027624/index.html(2024-01-16 アクセス).

[12] 田中智也，小池一樹，小林良太郎，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， Vol. 2019, pp. 163–169 (2019). https://cir.nii.ac.jp/crid/1050574047071091968.

[13] 荻原拓海，小林良太郎，加藤雅彦ほか：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，研究報告マルチメディア通信と分散処理 (DPS)， Vol. 2021, No. 50, pp. 1–8 (2021).

[14] Schmidt, P.: UNFS3, https://unfs3.sourceforge.net/ (参照 2023-10-28).

[15] Staubach, P., Pawlowski, B. and Callaghan, B.: NFS Version 3 Protocol Specification, RFC 1813 (1995).