静的解析と動的解析のハイブリッド解析への ファジー・ハッシュの導入

六反田 悠希[1]　向井 宏明[1]

概要：近年，マルウェアによるサイバー攻撃は深刻化しており，特に既存マルウェアを基に改変された亜種マルウェアが脅威となっている．これらは解析手法を回避する機構を備えることが多く，従来の静的または動的情報単独によるクラスタリングでは，類似関係の正確な把握が難しい．本稿では，ファジー・ハッシュ技術を用いた亜種マルウェアのクラスタリング手法を提案する．動的解析で得た API 呼び出し列にスライドウィンドウ法を適用し，局所的な動作パターンを抽出，ハッシュ化することで，類似挙動に基づくクラスタリングを行う．さらに，静的解析との情報統合により多面的な類似度評価を実現し，亜種マルウェアの検知性能向上を実現した．

キーワード：API 呼び出し系列，ファジー・ハッシュ，動的解析，静的解析，マルウェア検知

Introduction of Fuzzy Hashing into Hybrid Analysis of Static and Dynamic Malware Features

Yuki Rokutanda[1]　Hiroaki Mukai[1]

Abstract: In recent years, cyberattacks involving malware have become increasingly severe, and variant malware created by modifying existing malware has emerged as a major threat. These variants often incorporate mechanisms to evade analytical techniques, making it difficult to accurately capture similarity relationships through clustering based solely on static or dynamic information. In this paper, we propose a clustering method for variant malware using fuzzy hashing. By applying a sliding window technique to API call sequences obtained through dynamic analysis, we extract local behavioral patterns and convert them into hashed binary representations, enabling clustering based on behavioral similarity. Furthermore, by integrating information from static analysis, we realize multifaceted similarity evaluation and achieve improved detection performance for variant malware.

Keywords: API call sequence, fuzzy hashing, dynamic analysis, static analysis, malware detection

1. はじめに

近年，マルウェアによるサイバー攻撃は深刻さを増しており，個人・企業・政府機関など，多様なシステムが標的とされている[1]．中でも，新規に作られたマルウェアよりも，既存のコードを一部改変した亜種マルウェアが主な脅威となっている．これらの亜種は，静的解析や動的解析を回避する機構を備えることが多い．

静的解析は，プログラムを実行することなく，バイナリ構造や命令列，文字列，インポート関数などから特徴を抽出する手法であり，高速かつ安全に動作するため，無害化処理や大規模スキャンに適している．しかし近年のマルウェアは，難読化，暗号化，パッキング技術を用いて構造や挙動を巧妙に隠蔽しており，静的情報から有効な特徴を得ることが困難になっている．

動的解析は，マルウェアを仮想環境やサンドボックス上で実行し，API 呼び出し，ファイル操作，レジストリ変更などの挙動を時系列で観測する手法であり，実行時の本質的な動作を把握できる点で有効である．しかし，仮想環境を検出して挙動を変えるアンチ解析機構や，観測時間の制限により一部の挙動が記録されないこと，さらにマルウェアが自壊や潜伏によって解析を回避するケースもあり，解析の網羅性に課題が残る．また，環境構築や運用にかかるコストが高いため，大量解析には必ずしも適していない．このように，動的解析は高い柔軟性を持つ一方で，確実性やスケーラビリティには限界がある．

近年のマルウェアについては，従来の検出手法を回避するケースが増加しており，静的・動的解析のいずれか一方だけに依存するアプローチでは，亜種や改変型への網羅的な対応が困難である．このような背景から，マルウェア同士の「類似性」に着目した検出が重要視されている．コードや動作が完全には一致しなくとも，共通するパターンや構造を持つサンプルを同一のグループとして捉えることで，変化に強い検出が可能となる．特にファジー・ハッシュは，部分的な一致や構造的類似性を数値的に評価できる手法として有効であり，観測データから導出した類似度に基づき，マルウェアの識別やクラスタリングを行う枠組みを支える

1 金沢工業大学
Kanazawa Institute of Technology

技術となっている．

本稿では，亜種マルウェアの検出精度を高めることを目的として，動的解析によるクラスタ情報と静的解析で得られたバイナリのファジー・ハッシュ結果を統合することで，複数の観点からマルウェア同士の類似度を評価する手法を提案し，評価実験を行った結果を報告する．

2. ファジー・ハッシュについて

まず，提案手法で用いるファジー・ハッシュ技術について説明する．

2.1 ファジー・ハッシュと通常のハッシュ関数

ハッシュ関数は，任意長の入力データから固定長の出力値を生成する不可逆関数であり，データの整合性検証や電子署名，マルウェア検出など多くの場面で利用されている．特に SHA-1，SHA-256，MD5 などの暗号学的ハッシュ関数は，入力のわずかな変化でも大きく異なるハッシュ値を出力するため，完全一致の検出や改ざん防止には適している．しかしこの性質は，一部が異なるファイルの類似度評価には不向きであり，部分一致を捉えるには限界がある．そこで，ファジー・ハッシュが提案された．これは，入力データを小さなチャンクに分割し，それぞれに局所的なハッシュを適用して連結することで，部分的な一致や構造的な類似性を定量的に評価できるようにした手法である．この仕組みにより，柔軟な比較が可能となる．

2.2 ssdeep の構造と動作原理

ssdeep[2][3]は，Context Triggered Piecewise Hashing（CTPH）に基づくファジー・ハッシュアルゴリズムであり，データの一部が一致するような部分的類似性の検出を目的に設計され，入力ファイルに対して複数の段階的な処理を行うことで，構造的特徴を反映したハッシュ列を生成する．

まず，ファイル全体のサイズに応じて適切なブロックサイズが自動的に決定される．次に，スライディングウィンドウとローリングハッシュを用いてファイル全体を走査し，一定条件を満たす位置をコンテキストトリガーとして識別する．トリガーに基づき，データは複数のチャンクに分割され，構造的な区切りが形成される．各チャンクには Base64 に基づくハッシュ処理が施され，連結して最終的なファジー・ハッシュ列が構成される．生成されるハッシュは「ブロックサイズ：ハッシュ 1：ハッシュ 2」の形式を取り，局所的な一致に基づいた類似度評価を可能にする構造を備えている．

入力ファイルをチャンクに分割する際の最小単位がブロックサイズであり，ハッシュ列の粒度や比較精度に大きく影響する重要なパラメータである．小さすぎると微細な差異に敏感になるが，ハッシュ列が冗長になり，誤検出が増える．一方，大きすぎると局所的な変化を捉えにくくなり，類似度評価の精度が低下する． このため，ssdeep ではファイルのバイト長に応じてブロックサイズを自動調整する仕組みが導入されている．すなわち，大きなファイルには粗いブロックサイズを適用して，ハッシュ列の長さを抑制しつつ，処理の効率と安定性を確保している．また，同じブロックサイズで生成されたハッシュ同士でのみ比較を行うため，ブロックサイズの決定は構造分割だけでなく，ハッシュ比較の整合性を保つ前提条件でもある．

スライディングウィンドウおよびローリングハッシュを組み合わせた手法を用いて，局所的な特徴に基づく分割点（コンテキストトリガー）を決定する．この処理により，ファイルの構造的一貫性を保ちつつ，改変に対しても頑健なハッシュ列の生成が可能となる．スライディングウィンドウとは，固定長データの範囲をスライドさせながら，入力データ全体を走査する方式である（図 1 参照）．

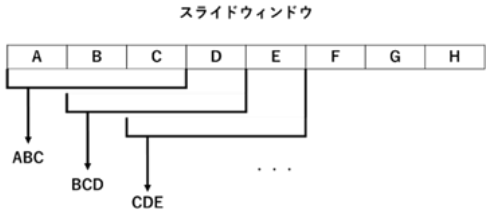

図 1　スライディングウィンドウの概念図
Figure 1 Sliding window over input data

各スライド位置において，ウィンドウ内のバイト列に対してローリングハッシュが適用され，現在位置における整数値（rh）が計算される．このローリングハッシュは，加算，累積加算，XOR などにより構成され，ウィンドウを 1 バイト移動するごとに高速に更新可能な設計となっている．得られた rh に対しては，式(1)の条件が評価される

$$rh \bmod block_{size} = block_{size} - 1 \qquad (1)$$

式(1)の条件を満たす位置がコンテキストトリガーとして選定され，チャンクの分割が実施される．たとえば，block_size = 48 の場合，rh % 48 = 47 を満たす値（383）はトリガーとして採用され，その位置でチャンクが分割される（図 2 参照）．

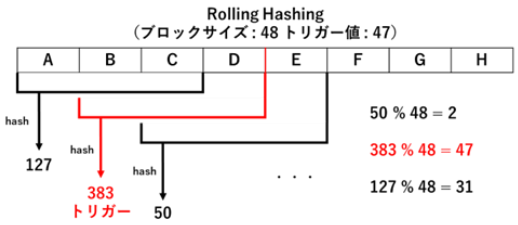

図 2　ローリングハッシュ値とトリガーの検出
Figure 2 Rolling hash and trigger detection

このようなトリガーベースの分割方式により，データ中

に挿入・削除・入れ替えといった局所的な変更が生じても，多くの分割点が維持されやすくなる．その結果，ファイル構造に部分的な類似性が存在する場合でも，それを反映したチャンク列が得られるため，類似性比較の精度が向上する．この特性は，特に亜種マルウェアや改変ファイルの検出において高い有効性を発揮する．

コンテキストトリガーに基づいて分割された各チャンクには，独立した文字列ベースのハッシュ処理が適用される．この処理はチャンク単位の局所的特徴を抽出することを目的とし，加算や XOR などの軽量演算による高速なハッシュアルゴリズムで実装されている．代表的な実装では，式(2)に示す更新式が用いられる

$$hash \leftarrow \left((hash \times p) \bmod 2^{32}\right) \oplus b \qquad (2)$$

ここで，hash は中間的なハッシュ値，b は現在処理中の 1 バイト，p は FNV ハッシュ由来の定数（0x01000193）である．32 ビット幅の制限により，演算の簡素化とオーバーフローの抑制が図られている．得られたハッシュ値はバイナリ形式のままでは扱いにくいため，視認性と可搬性を高める目的でエンコードされる．ssdeep では，Base64 に類似した 64 進数表現（A-Z，a-z，0-9，+，/）を用いた符号化方式が採用されており，一貫性と機械処理性に優れたハッシュ列が生成される．エンコード済みの各チャンクのハッシュは順次連結され，入力ファイル全体の構造を反映したファジー・ハッシュ列となる．この列は，部分的改変を含むマルウェアの類似性評価において有効な識別情報となる．最終的な出力形式は以下のように表される（表 1 参照）．

表 1　ファジー・ハッシュの出力構造例

Table 1 Fuzzy Hash Output Format Example

384:BvXlsJLpZkgzy3wK2kJiWSGYWXiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLiiA:B/6LjQndJ5a		
ブロックサイズ	hash1	hash2
384	BvXlsJLpZkgzy3wK2kJiWSGYWXiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiLiiA	B/6LjQndJ5a

ここで，hash1 は決定されたブロックサイズ（384）で得られた主ハッシュ列であり，hash2 はその 2 倍（768）のブロックサイズによる補助的なハッシュ列である．この二重構造により，構造差やサイズ差のあるファイル間でも柔軟な比較が可能となる．

2.3 類似度評価の仕組み

ファジー・ハッシュは，ファイル間の部分的な構造や内容の一致を捉えることで，類似度を数値として評価できる．ssdeep はこの特性を活かし，完全に一致しないファイルであっても，その類似性の程度をスコアとして出力することができる．このため，たとえ難読化や軽微な改変が加えられていても，構造的に類似したマルウェア同士を検出・分類することが可能となる．また，マルウェアの亜種間における関係性を明らかにする手段としても有効であり，マルウェアのクラスタリングや分析に応用されている．

ssdeep における類似度評価は，Levenshtein 距離（編集距離）[4]に基づいて行われる．これは，ある文字列を別の文字列に変換するために必要な最小の編集操作（挿入・削除・置換）の回数を測るものであり，すべての操作には等しいコストが与えられる．ファイルごとに生成されたファジー・ハッシュ文字列同士を比較し，編集距離を文字列の合計長で正規化することで差異率を求める．類似度スコアは，数値が高いほどファイル間の構造や内容に類似性があることを示す．このスコアは，完全一致から部分的な一致，さらには無関係なデータまでを連続的に表現できるため，マルウェアの亜種検出や変異パターンの識別において有効な指標となる．また，比較前に共通チャンクの有無を確認し，それが存在しない場合は編集距離を計算せず，スコア 0 とすることで処理の効率化と誤検出の低減が図られている．

ssdeep では，1 つのファイルに対して 2 種類の異なるブロックサイズ（基準サイズとその 2 倍）を用いた 2 本のハッシュ列を生成する．この構成により，局所的な変化と大域的な構造の両方を捉えることが可能となり，改変ファイルや亜種マルウェアの検出に有効である．比較を行う際は，両ハッシュが同一のブロックサイズ，または一方が他方の 2 倍である必要がある．この条件が満たされない場合，チャンク構造が大きく異なり比較が意味を持たないため，スコアは自動的に 0 と判定される．

最終的なスコアは式(3)で定義され，0〜100 の範囲に変換される．ここで，d は編集距離，L は両文字列の長さの合計である．2 つの文字列に対し，距離行列を用いた動的計画法により最小編集距離を算出し，それを文字列長の合計で正規化することで差異率を得る．

$$S = \left(1 - \frac{d}{L}\right) \times 100 \qquad (3)$$

スコアが高いほど類似度が高く，完全一致で 100，差異が大きい場合は低い値となる． この評価法は，局所的な改変に対してはスコアの低下が抑えられ，一方で構造全体の違いには敏感に反応するため，マルウェア亜種の識別やファイルのクラスタリングにおいて有効である．

3. 提案手法

3.1 手法全体の概要と処理フロー

類似性に基づく検出アプローチを具現化するため，ssdeep アルゴリズムを拡張応用したマルウェアクラスタリング手法を提案する．本手法は，動的解析によって得られる API 呼び出し系列と，静的解析に基づくバイナリ情報の双方から類似度を算出し，統合的に分析することで亜種マ

ルウェアの識別精度を高めることを目的とする．

まず動的解析ログから API 呼び出し列を抽出し，スライドウィンドウ法によって連続する 3 つの API 呼び出し（3-gram）を単位とした特徴列を構成する．各 3-gram は API 名を連結したのち，ハッシュ関数によって 8 バイトの固定長バイナリ値に変換される．得られたハッシュ系列全体に対しては，ssdeep をベースとした類似度評価手法を適用し，動的挙動を要約したハッシュ列を生成する．なお，提案手法では後述するようにスライディングウィンドウの幅を従来の 7 バイトから 40 バイトに拡張するなど，オリジナルの ssdeep とは異なる設定を採用している（図 3 参照）．

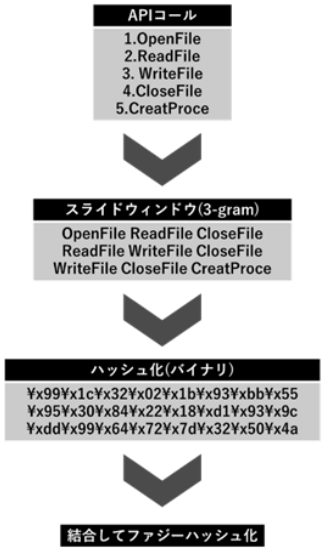

図 3　API 系列のファジー・ハッシュ化
Figure 3 Fuzzy Hashing of API Call Sequences

類似度の評価には，このハッシュ列間で最長共通部分列（LCS）[5]に基づくスコアリングを行い，順序や部分改変に対して頑健な類似性を測定する．

一方, 静的解析では, マルウェアのバイナリ全体に ssdeep を適用し，生成されたハッシュ列同士に対して，LCS に基づく類似度評価を行う．静的解析に基づくファジー・ハッシュの類似度行列からクラスタリングを実施し，構造的に類似するマルウェア群を抽出する．次に，同様の手法を動的解析によって得られた API 呼び出し系列に対して適用し，動作的な類似性に基づくクラスタを得る．

両者のクラスタリング結果を照合し，いずれかの解析において同一クラスタに属すると判断されたマルウェア群を統合することで，構造または動作のいずれかの観点で類似性を示すマルウェアのグループ化を行う．これにより，静的あるいは動的解析のいずれか一方では捉えにくい亜種や難読化されたマルウェアに対しても，高い識別性能を維持することができる．

3.2 動的解析ログの特徴抽出とバイナリ変換の設計

動的解析によって得られる API 呼び出し系列を対象に，マルウェアの挙動を定量的に表現する特徴を抽出する．API 呼び出しは実行時の動作を直接反映するため，その組み合わせにはマルウェア固有の傾向が現れやすい．

この特徴抽出には，連続する 3 つの API 呼び出しを 1 単位とする 3-gram を用いる．これは，API 呼び出しの組み合わせにおいて, 単なる 1 つの呼び出しや 2-gram では識別力が不十分であり，マルウェア固有の特徴を捉えるには情報が不足するためである．特に，API コールは類似の名称や機能を持つものが多く，部分的に一致しているだけでは同一系列であると判断するのは困難である．そのため，ある程度の連続性を持った 3 つの呼び出し単位で分析を行うことにより，より確実に類似の挙動を識別可能となる．

各 3-gram は, API 名を結合した文字列として表現される．しかし，類似した名称を持つ API（例：CreateFileA，CreateFileW など）の違いによって不適切に分類することや, API 名の長短によって特徴量のサイズに偏りが生じる問題がある．そこで， API 名列を固定長 8 バイトのバイナリデータへ変換するハッシュ処理を導入する．これにより，同様の構造を持つ API 呼び出し系列は，等長のバイナリ列に変換され，後続の処理において比較可能な形式を確保することができる．8 バイトを採用した場合のハッシュ衝突率について考えると $255^8 \approx 1.7 \times 10^{19}$ 通りであるが, API コールの種類は一般に約 1 万前後であり, それらの 3-gram 組み合わせは最大で $(10^4)^3 \approx 10^{12}$ 通り程度となるため, 十分に識別可能な空間を確保できると考えられる．

このようにして得られたバイナリ列は，マルウェアの挙動を圧縮かつ構造的に反映した表現となり，後続のファジー・ハッシュ処理によって振る舞いの類似性を効果的に評価するための基盤として機能する．

3.3 API 列に対するハッシュ化処理とウィンドウ設計

得られたハッシュ列全体に対しては，ssdeep の構造を参考にした独自の拡張アルゴリズムを適用する．従来の ssdeep ではスライディングウィンドウ幅として 7 バイトが用いられているが，提案手法ではこれを 40 バイト（8 バイト × 5）に拡張している．この理由は，API 呼び出し系列を 3-gram 単位で 8 バイトの固定長バイナリデータに変換しているためであり，7 バイトのウィンドウ幅では 1 つのハッシュ値すら完全に含めることができない．その結果，局所的な変化に過剰に反応してしまい，全体的な動作パターンを捉えにくくなるという問題がある．提案手法では，少なくとも 1 つ以上のハッシュ値を完全に含むこと，および API 列の連続的なパターンをより広い範囲で評価することを目的として，8 バイトより大きなウィンドウ幅とすることが望ましい．40 バイトという設定値は，現時点では 5 つのハッシュ値を同時に扱えるという点でバランスが良く，暫定的に採用したものであるが，今後の検討により最適な

パラメータの再評価も可能である．

このようにスライディングウィンドウ幅を拡張することで，API 系列の局所的な変化に対して頑健なハッシュ列が得られ，マルウェア間の動作類似性をより安定的かつ包括的に比較することが可能となる．

3.4 最長共通部分列（LCS）による類似度評価

動的解析により得られた API 呼び出し列の類似度を評価するために，LCS に基づく手法を採用する．LCS は，2 つの文字列に対して，順序を保ちながら共通して出現する部分列の最大長を求めるアルゴリズムである．この評価指標は，一致している部分の長さに注目する点で，差異の大きさを測る Levenshtein 距離よりも保守的な一致評価を実現できるという特徴がある．LCS の計算は動的計画法により実現され，2 つの文字列 s, t に対して $(|s|+1) \times (|t|+1)$ の距離表を構築し，順次更新することで両者の最長共通部分列の長さを求める．得られた LCS 長は以式(4)によりスコアとして正規化される．

$$S = \frac{2 \times LCS(s,t)}{|s|+|t|} \times 100 \qquad (4)$$

このスコアは 0 から 100 の範囲で示され，完全一致の場合は 100，一部が共通する場合は中程度となり，類似性がほとんど見られない場合には 0 に近づく．

LCS を用いることで，関数の一部改変や無意味な API 挿入といった変異に対しても頑健な比較が可能となり，検知逃れが施された亜種マルウェアの動的挙動を比較する上で有効な指標となる．

3.5 類似度に基づくクラスタリング手法

クラスタリング処理においては，マルウェア間の類似度スコアに基づき，各サンプルのグループ化を段階的に行う．今回，類似度のしきい値を 70 に設定しており，これは「高い構造的または動作的な一致が認められる」と判断する基準値として採用されている．具体的には，すべてのマルウェアサンプル対について類似度スコアを算出し，スコアが 70 以上であるペアを同一グループに属するものとして扱う．この操作を全サンプルに対して逐次的に適用し，ペアごとの関係に基づいてクラスタを構成していく．一方で，いずれの既存クラスタとも 70 以上のスコアを持つサンプルが存在しない場合，そのサンプルは新たな独立クラスタとして割り当てられる．この手法により，全体として類似性に基づく適切なクラスタ構造を構築することが可能となる．

すでに形成された複数のクラスタ間においても，クラスタ A 内の任意のサンプルとクラスタ B 内の任意のサンプルの間で 70 以上の類似度が確認された場合，これら 2 つのクラスタは同一のマルウェア群として統合される．このグループ間伝播による統合処理により，間接的な類似性も反映された柔軟なクラスタリングが実現される．このような処理は，マルウェアにおける多段階的・連鎖的な変異にも対応可能であり，亜種の検出やファミリ単位での整理において有効な基盤となる．

3.6 静的・動的クラスタ情報の統合手法

動的解析および静的解析から得られたクラスタ情報は，それぞれ独立に構築された後，両者を統合することでマルウェア群のより包括的な関係性を抽出する．統合においては，各クラスタに共通して含まれるマルウェアサンプルの存在に着目し，クラスタ同士を伝播的に結合する処理を行う．動的解析のクラスタ内に含まれるサンプルと，静的解析のクラスタ内に含まれるサンプルの重なりを探索することで，間接的に関連するマルウェア同士を同一グループへと統合する．たとえば，静的解析において A と B が同じクラスタに属し，動的解析において B と C が同じクラスタに含まれている場合，A・B・C はすべて同一のマルウェアクラスタとして統合される．このような処理により，一方の解析結果では見逃されていた関係も，他方の視点から補完的に抽出されることとなり，部分的にしか類似しない亜種マルウェアに対しても，高い識別精度を維持したクラスタリングが可能となる．

一方で，異なるマルウェアが偶然同一クラスタに含まれてしまうことによる誤統合の可能性についても考慮する必要がある．本手法では，動的解析において API 呼び出し列の 3-gram を 8 バイトの固定長ハッシュに変換し，その系列に基づいてクラスタを構築している．このハッシュ値は局所的な動作パターンを反映しており，3-gram と 8 バイトの固定長ハッシュを用いることで，わずかな変更であってもハッシュ値が大きく変化する性質を持つ．そのため，異なるマルウェア間でのハッシュ列の一致は起こりにくく，誤統合のリスクは限定的である．さらに，統合処理は静的・動的解析という異なる観点から得られたクラスタを併用しており，一方で生じた誤クラスタが他方の解析結果により補正される設計とした．以上により，本手法は見逃された関係性の補完と誤分類の抑制を両立し，変異や難読化を含む亜種マルウェアの高精度な識別を実現する．

4. 評価実験

4.1 データセットの概要

マルウェアの動的解析ログを用いた評価実験を行うために，MWS 2022 において提供された「Soliton Dataset 2022」を使用した［6］．本データセットは，株式会社ソリトンシステムズが提供する EDR 製品「InfoTrace Mark II」を用いて取得されたマルウェアの実行ログや振る舞い情報を含んでおり，標的型攻撃に利用されるマルウェアの解析や挙動の分類に適している．加えて，Cuckoo Sandbox による解析結果も付与されており，マルウェアが実行時に呼び出す

API やファイルアクセス，プロセス生成といった詳細な動的情報を取得可能である．さらに，本データセットには，マルウェアの実行環境として AWS（EC2）ベースのクラウド環境と KVM ベースのローカル仮想環境の 2 種類の解析環境で取得されたログが含まれており，それぞれ異なる実行コンテキストにおけるマルウェアの挙動を比較・分析することが可能である．

このデータセットに含まれるマルウェアログから API 呼び出し系列を抽出し，提案手法によるファジー・ハッシュを用いたクラスタリング処理を実施した．ただし，クラスタリング精度や解析対象の信頼性を確保するため，使用するマルウェアデータには以下の条件を設けて選別を行った．まず，各マルウェア種別に対して動的解析環境ごとの API 呼び出しログを整理し，以下の基準に基づいて選別および前処理を実施した．

① API 呼び出し数によるフィルタリング

取得された API 呼び出しログについて，API 呼び出し数が 100 以下のものは，実行に失敗や，短時間で終了した可能性があると判断し，信頼性の観点から除外した．

② 例外発生ログの除外

API 呼び出し系列内に__exception__という特定の文字列を含むファイルは，動的解析中に異常終了した可能性があるため除外した．これにより，正常に実行された検体に基づくログのみを分析対象とした．

③ サンプル数が極端に少ないマルウェア種の除外

一つのマルウェア種別に含まれるログファイル数が 5 件以下のものは，統計的に有意な分析が困難であると判断し，除外対象とした．

④ マルウェア種別の手動除外によるノイズ排除

偽装インストーラー型のマルウェアや，APT グループに由来する特殊な検体（例：Blind Eagle 関連）については，本研究の目的がマルウェアの“亜種間”の類似性を評価することにあるため，これらのように明確に異なる系統や目的を持ち，亜種とは見なしにくい検体は分析対象から除外した

⑤ 解析環境に基づく信頼性の考慮

評価実験で使用したデータには，KVM ベースのローカル環境および AWS（EC2）ベースのクラウド環境という 2 種類の解析環境において取得されたログが含まれている．これらはそれぞれ異なる実行コンテキストに基づくマルウェアの挙動を反映しており，多様性のあるデータを構成している．一方で， AWS 環境では，AWS 独自の常駐プロセスに起因するログも記録対象となっており，監視除外がなされていないことから，マルウェア本体の挙動とは無関係なノイズが混入する可能性が指摘されている［6］．これに対して，KVM 環境では背景プロセスの影響が少なく，マルウェア本体の API 呼び出しをより純粋に観測できる傾向がある．

このような観点から，評価実験では KVM 環境由来のログを主たる解析対象とし，AWS 環境のログは補完的に位置付けて使用する方針を採用した．両環境で同一のマルウェア種に対するログが取得されている場合には，KVM 側のログを優先的に用い，AWS 環境でしか取得されなかったログのみを統合対象とした．

上記の前処理により，動的解析ログの信頼性と統計的妥当性が確保された，ファジー・ハッシュによるクラスタリングに適したデータセットを構築した．評価実験で使用したマルウェア種別と解析件数の内訳を示す（表 2 参照）．

表 2　使用したマルウェア種別と解析件数の内訳

Table 2 Breakdown of Malware Types and Number of Analyses

マルウェア名	種別	検体数
ALPHV	ランサムウェア	22
BitRAT	RAT	10
Conti	ランサムウェア	7
Emotet	トロイの木馬	23
Karma Ransomware	ランサムウェア	8
LockBit2.0	ランサムウェア	6
LokiBot	トロイの木馬	11
PoVlsomware	ランサムウェアスクリプト	9
RURansom	ワイパー型マルウェア	6
SigLoader	マルウェアローダー	12
Warzone RAT	RAT	7
合計		121

表 2 中には，種別ごとに静的解析および動的解析に用いた検体数を記載しており，多様な種別が含まれている．動的解析に使用された検体の総数は 121 件であり，いずれも複数のマルウェア種別にわたってバランスよく分布している．

4.2 静的解析情報に基づくファジー・ハッシュの取得

マルウェアファイル本体に対する静的解析情報も取得し，クラスタリングにおける類似度計算に活用した．「Soliton Dataset 2022」に含まれる各マルウェアサンプルの情報には，対応する VirusTotal のスキャン結果を参照するための URL が含まれている．この URL を利用して，各マルウェアの VirusTotal ページにアクセスし，そこで表示される ssdeep のファジー・ハッシュ値を取得した．

4.3 クラスタリング手法ごとの分類結果の比較

静的解析，動的解析，および両者の統合手法により得られたクラスタリング結果を比較した．各マルウェアファミリについてのサンプル総数と，各手法によって得られたクラスタ数を示す（表 3 参照）．

表 3　マルウェアの総数と手法別クラスタ数
Table 3 Malware Sample Totals and Cluster Counts by Analysis Method

マルウェア名	静的解析	動的解析	統合後
ALPHV	9	2	2
BitRAT	10	1※	1※
Coint	6	7	6
Emotet	5	7	5
Karma Ransomware	2	7	2
LockBit2.0	3	6	3
LokiBot	5	8	2
Povlsomware	4	1※	1※
RURansom	4	1※	1※
SigLoader	11	1	1
Warzone RAT	1	5	1
合計	60	46	23
※BitRAT, Povlsomware, RURansomは一つのグループ			

クラスタ数が少ないほど，同一マルウェア内のサンプルが効果的にグループ化されていると解釈でき，分類の精度が高いとみなすことができる．

静的解析では，マルウェアのバイナリ構造に基づく情報をもとにクラスタリングを行うため，構文的特徴が安定している場合には有効に機能するが，難読化やパッキングの影響を受けやすい特性がある． SigLoader, ALPHV, BitRAT といったマルウェアでは，構造の差異によりサンプルが分散し，それぞれ多数のクラスタに分類された（図 4 参照）．

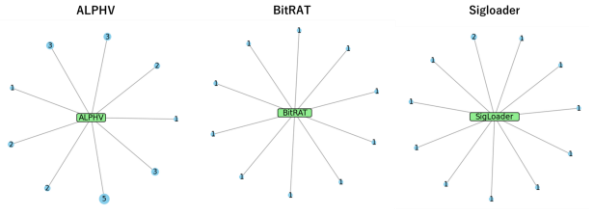

図 4 静的解析に基づくクラスタ構造の可視化 1
Figure 4 Static Analysis Clustering Visualization 1

マルウェアファミリに対する静的解析のクラスタリング結果をネットワーク構造で可視化したものである．各図において，中央の緑色ノードはクラスタの代表（中心）を示し，その周囲に配置された青色ノードは，そのクラスタに属する個別サンプルを表す．ノード（丸）の大きさは，そのクラスタに含まれるサンプル数に比例しており，大きな中心ノードは多くのサンプルが 1 つのクラスタにまとまっていることを示す．一方，周囲に存在する小さな多数の青ノードがそれぞれ異なるクラスタから分離されている場合，そのファミリ内でクラスタリングがばらけていることを意味する．特に BitRAT は 10 件のサンプルがすべて異なるクラスタに分類され，静的解析単独での安定した分類は困難であったことが示された．一方，Karma Ransomware や Warzone RAT は，バイナリ構造に一貫性が見られ，静的解析でも少数のクラスタに分類されるなど，比較的高い精度を示した（図 5 参照）．

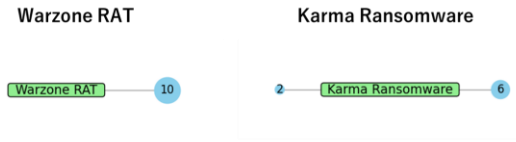

図 5 静的解析に基づくクラスタ構造の可視化 2
Figure 5 Static Analysis Clustering Visualization 2

動的解析は，実行時に観測される API 呼び出しやファイルアクセス，プロセス生成といった挙動に基づいてクラスタリングを行うため，コード構造が異なっていても類似した振る舞いを持つ検体を効果的にまとめることができる．特に, ALPHV は 22 サンプル中 2 つのクラスタに集約され，また SigLoader も動的解析では 12 サンプルが 1 つのクラスタにまとまり，高い分類精度を示した(図 6 参照).

図 6 動的解析に基づくクラスタ構造の可視化 1
Figure 6 Dynamic Analysis Clustering Visualization 1

一方，Warzone RAT や Karma Ransomware では実行時の挙動にばらつきがあり， 分散するなど，挙動の多様性による分類の不安定さが確認された（図 7 参照）．

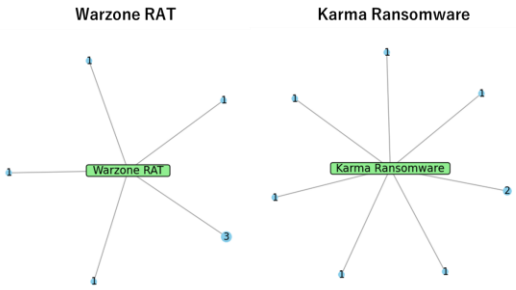

図 7 動的解析に基づくクラスタ構造の可視化 2
Figure 7 Dynamic Analysis Clustering Visualization 2

また，BitRAT，Povlsomware，RURansom については，それぞれ静的には異なるクラスタに分類される一方で，動的解析では挙動の類似性から 1 つのクラスタに統合されていた（図 8 参照）．

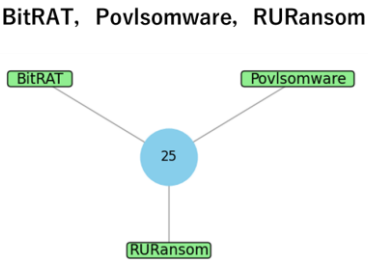

図 8 異種マルウェアの動作に基づくクラスタ統合
Figure 8 Behavior-Based Cluster Integration

統合後のクラスタ数は 23 と，静的解析単体（60）および動的解析単体（46）よりも大幅に削減されており，これは両手法を組み合わせることで，より集約度の高い分類が実現されていることを示す．

4.4 考察

評価実験の結果より，静的解析ではバイナリ構造が安定しているマルウェアに対しては高い精度で分類が行えた一方，難読化やパッキングの影響を受けやすく，クラスタが細分化される傾向が見られた．BitRAT や SigLoader はその典型であり，多数のクラスタに分散し，一貫した分類が困難であった．一方で，Karma Ransomware や Warzone RAT のように構造が比較的一様なファミリでは，静的解析のみでも効果的なクラスタリングが可能であった．また，Warzone RAT のように挙動のばらつきが大きいファミリでは，クラスタリングの精度が低下する傾向も確認された．この要因として，Warzone RAT がリモートアクセス型トロイの木馬（Remote Access Trojan, RAT）である点が挙げられる．RAT は，感染後に攻撃者の遠隔操作に応じて多様なコマンドを実行するため，サンプルごとの実行内容や API 呼び出しの順序が固定化されにくいという特性を持つ．このような動作の不定性により，動的解析においてもサンプル間の共通性が捉えづらく，結果としてクラスタが分散する傾向が生じた可能性がある．

BitRAT，Povlsomware，RURansom は，静的解析ではそれぞれ異なるバイナリ構造を有しており，正確な分類が困難であった．しかし，動的解析ではこれらすべてのサンプルが同一クラスタに分類された．これは，動的解析ログ中の API 呼び出し系列が極めて高い一致を示していたためであり，それらのマルウェアが共通してサンドボックス環境や解析ツールの存在を検出するための API コールを多く含んでいたことが原因と考えられる．環境検知系 API が複数サンプルに共通して出現しており，動的解析の検出に特化したロジックが似通っていた可能性が高い．ただし，このように同一クラスタに分類されたからといって，必ずしもマルウェアの本質的な機能や目的が同一であるとは限らない点には注意が必要である．共通の解析回避手法が類似した動作パターンとして現れたに過ぎず，それ以外の攻撃挙動が大きく異なる可能性もあるためである．したがって，クラスタリング結果を解釈する際には，その背景にある動作の意味を慎重に評価することが求められる．

総括すると，静的解析ではバイナリ構造に起因する情報の類似性を，動的解析では実行時の挙動によるパターンを捉えることができるため，両者を統合することで，従来の単一手法では難しかったマルウェアの高精度なグルーピングが実現されることが示唆された．また，ファジー・ハッシュの類似度閾値として設定した 70 という値については，誤クラスタリングの発生は観測されなかったことから，分類の網羅性と精度のバランスを確保する上で有効であることを示唆している．

5. おわりに

ファジー・ハッシュ技術を活用し，マルウェアの静的解析および動的解析から得られる情報に基づくクラスタリング手法を提案した．動的解析ログに記録された API 呼び出し系列と，バイナリ構造から得られる静的情報をそれぞれファジー・ハッシュ化することで，マルウェアの動作傾向と構文的類似性の両面から分析を行い，高精度な分類を実現した．評価実験では，各手法がマルウェアの特性に応じた有効性を示し，両者を統合することで分類の安定性と一貫性が向上することが確認された．とくに，個別の手法では分類が困難であったサンプル群に対して，統合手法が有効に機能する例が得られたことは，本手法の有用性を示すものである．

今後は，API 呼び出しに依存しないマルウェアにも対応可能な，アセンブリレベルの命令列を用いたファジー・ハッシュ化の応用を進めることで，解析環境に依存しない高精度なクラスタリング技術の確立を目指す．

参考文献

[1] IPA：情報セキュリティ 10 大脅威 2025，< https://www.ipa.go.jp/security/10threats/10threats2025.html >，令和 7 年 2 月 1 日参照
[2] Jesse Kornblum, “Identifying almost identical files using context triggered piecewise hashing”, Digital Investigation, Vol.3, Supplement, pp.91-97, 2006
[3] Fuzzy hashing の利用に関する検討及び評価，< https://www.ffri.jp/assets/files/monthly_research/MR201403_Consideration_and_evaluation_of_using_fuzzy_hashing_JPN.pdf>，令和 7 年 1 月 21 日参照
[4] Vladimir Iosifovich Levenshtein, “Binary codes capable of correcting deletions, insertions and reversals”, Soviet Physics Doklady, Vol.10, No.8, pp.707-710, 1966
[5] D. S. Hirschberg, “A linear space algorithm for computing maximal common subsequences”, Communications of the ACM, Vol.18, No.6, pp.341–343, 1975.
[6] 株式会社ソリトンシステムズ，”Soliton Dataset 2022”，マルウェア対策研究人材育成ワークショップ（MWS）2022 プレミーティング資料, 2022 年 7 月 12 日.