一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

サンドボックス回避実証コードのLLMを用いた分類の精度評価

東　志拓[†]　松澤　輝[†]　田辺　瑠偉[††,†††]インミンパパ[††]　吉岡　克成[††,††††]

† 横浜国立大学　〒240-8501 神奈川県横浜市保土ケ谷区常盤台 79-1
†† 横浜国立大学先端科学高等研究院　〒240-8501 神奈川県横浜市保土ケ谷区常盤台 79-5
†††† 横浜国立大学大学院環境情報研究院　〒240-8501 神奈川県横浜市保土ケ谷区常盤台 79-7
††† 順天堂大学　〒279-0013 千葉県浦安市日の出 6-8-1
E-mail: †{higashi-yukihiro-rd,matsuzawa-hikaru-nx}@ynu.jp, ††{yinminn-papa-jp,yoshioka}@ynu.ac.jp

あらまし　サンドボックス解析を回避する機能をもつマルウェアへの対策において，様々な回避機能のコンセプトを実証する PoC（Proof of Concept）コードを用いた対策技術の検証が重要である．特に近年では，高いプログラミング能力を有する AI の登場により容易にコード生成が可能となったことから，今後は利用可能な PoC コードの数は急速に増加することが予想される．しかし，作成される PoC コードの機能や特徴，使用言語等の属性には統一的な記述方法が存在せず，作成された PoC コード群を分類してその関係性を明らかにした研究は我々の知る限り行われていない．そこで，本研究では大規模言語モデル（LLM）を用いて，既存の機能定義群に基づき PoC コード群の自動分類を行いその精度を評価する．評価実験では，先行研究で紹介している回避機能のカテゴリに基づき，サンドボックス回避ツールやサンドボックス回避機能を公開しているウェブサイトから収集した 232 個の PoC コードを GPT-4o を用いて自動分類したところ，完全一致率は 0.77，マイクロ平均での適合率，再現率は共に 0.88 という結果が得られた．マイクロ平均での値が高く，LLM が高い精度で人間と同様の分類を行えることを確認した．
キーワード　マルウェア動的解析，サンドボックス回避，マルウェア分類，大規模言語モデル

Evaluating the Classification Accuracy of an LLM for Sandbox Evasion Proof-of-Concept Code

Yukihiro HIGASHI[†], Hikaru MATSUZAWA[†], Rui TANABE[††,†††], Yin MINN PA PA[††], and Katsunari YOSHIOKA[††,††††]

† Yokohama National University
†† Institute of Advanced Sciences, Yokohama National University
†††† Faculty of Environment and Information Sciences, Yokohama National University
††† Juntendo University
E-mail: †{higashi-yukihiro-rd,matsuzawa-hikaru-nx}@ynu.jp, ††{yinminn-papa-jp,yoshioka}@ynu.ac.jp

Abstract　In addressing countermeasures against evasive malware, it is essential to validate mitigation techniques using Proof of Concept (PoC) code that demonstrates various evasion techniques. Recently, the rise of AI with advanced programming capabilities has made code generation easier, and it is expected that the number of available PoC codes will increase rapidly in the future. However, there is no standardized method for describing the functions, features, and programming languages of the PoC codes created, and to our knowledge, no research has classified these PoC codes and clarified their relationships. In this study, we utilized a large language model (LLM) to automatically classify PoC codes based on existing evasion technique definitions and evaluate the accuracy of this classification. In the evaluation experiment, using categories of evasion techniques introduced in prior research, we collected 232 PoC codes from websites that publish sandbox evasion tools and techniques. These codes were classified using GPT-4o, resulting in a 0.77 exact match rate and a micro-average precision and recall of 0.88, confirming that the LLM can classify with high accuracy, similar to human classification.
Key words　Malware Dynamic Analysis, Sandbox Evasion, Malware Classification, Large Language Model

1. はじめに

サンドボックス内でマルウェア検体を解析してその挙動を明らかにする動的解析技術の普及に伴い，サンドボックス環境を検知して解析を回避する機能を搭載したマルウェアが大きな課題となっている．特に企業や官公庁といった多くの組織において，サンドボックス解析によりマルウェアの侵入を未然に防ぐセキュリティアプライアンスの導入が進んでいるが，サンドボックス解析が回避された場合には重大なセキュリティインシデントにつながる可能性があり対策が求められている．

対策技術を考える上でマルウェアの挙動を把握することは重要であり，実際のマルウェアに用いられるサンドボックス解析回避機能を分類して実態調査を行なった研究が存在する [1], [2], [5]．また，MITRE [9] では，回避機能を搭載したマルウェアのファミリー情報を公開しており，多くのマルウェアが始めに実行環境の情報を収集し，あらかじめ設定した条件と一致した場合に実行環境がサンドボックスであると判断することが知られている．実行環境の判断方法は様々であるが，典型的には仮想化技術やエミュレータなどのサンドボックスを構築する技術の検知などが挙げられる．このため，実マシンやユーザマシンと区別がつきにくいサンドボックス [6], [8] が研究開発されるなど，防御側と攻撃側で激しい攻防が繰り広げられている．

回避機能を搭載したマルウェアに対するサンドボックスの耐性を評価するためには，検証用のデータセットを用いてサンドボックス解析による検知の可否を調査する方法が一般的である．そのため，マルウェアに搭載された回避機能を実証する PoC（Proof of Concept） コードやツール [4], [7], [12], [13] が公開されており，特に近年では AI 技術の進展により容易にコード生成が可能となったことから，今後は PoC コードが急速に増加することが予想される [3], [14]．しかし，これらの PoC コードの機能や特徴，使用言語などの属性について統一された記述方法が存在せず，さらには，既存の PoC コードの機能や相互関係を明らかにした研究は我々の知る限り存在しない．

そこで本研究では，大規模言語モデル（Large Language Model: LLM） を活用し，回避機能を実証する PoC コードを既存の機能分類結果に基づき自動分類することで，サンドボックスの回避耐性評価用データセットの構築方法の実現を目指す．具体的には，先行研究 [5] で紹介されている 16 種類の回避機能のカテゴリに基づき，Web 上から収集した 232 個の PoC コードに対して GPT-4o [10] を用いて自動分類を行い，人手により分類した結果と比較することでその精度を評価する．

評価実験の結果，LLM による分類と人手による分類の完全一致率は 0.77 と高く，LLM が人間と遜色ないレベルで PoC コードの分類を行えることが確認された．特にマイクロ平均での適合率と再現率が共に 0.88 という高い値であったことから，LLM が多種多様な回避技術を効果的に分類できる能力を持つことが確認できた．カテゴリ毎に分類結果を確認したところ，”Table Descriptors”，”Traps”，”WMI”などのカテゴリでは，適合率，再現率ともに非常に高く，LLM がこれらのカテゴリを人と同様に分類できていることがわかった．一方で，”Memory Fingerprinting”や”Timing”などのカテゴリでは適合率が比較的低くなっており，これらのカテゴリに分類されるべきコードを LLM が他のカテゴリに誤分類していた．続いて，分類にかかる時間を調査したところ，ファイル一つ当たり平均 4.48 秒で分類が完了した．以上から，LLM は大量の PoC コードを正確に素早く分類できる点で有用であると言える．また，本研究の成果は，サンドボックス解析回避機能の体系的な理解を深め，効果的な対策技術の開発に貢献することが期待される．

2. 関連研究

これまでに，サンドボックス解析を回避する機能を搭載したマルウェアに関する研究が活発に行われている．Galloro [5] らは，Windows マルウェアが採用する回避行動に関する包括的な調査と分析を行い，仮想マシンやサンドボックスなどの実行環境を検知して回避するのに使用される 92 種類の回避技術を 16 種類のカテゴリに分類した．また，2010 年から 2019 年までの間に収集された 45,375 個のマルウェア検体を解析し，回避技術の分布や時系列推移を明らかにした．加えて，一般的な Windows プログラムとの比較分析を行い，回避行動の固有の特性を明らかにした．また，Barbosa [2] および Afianian [1] らは，マルウェアが用いる回避行動を調査し，仮想化検知，プロセス環境の偽装，タイミング攻撃，アンチデバッグ，API フックの回避，暗号化技術など幅広い手法を分類・整理した．

上記のような回避機能を搭載するマルウェアへの対策を講じるため，サンドボックス解析を回避する技術を実装したツールやそのソースコードが研究開発されている．例えば，*al-khaser* [4]，*checkpoint evasion techniques* [7]，*unprotect project* [13]，*pafish* [12] などのツールがウェブ上で公開されている．しかしながら，これらの研究にはいくつかの課題が存在する，第一に，回避技術の PoC コードを分類する際に，統一された記述方法が存在しないため，コードの機能や特徴，使用言語の違いにより分類基準が異なる場合がある．第二に，マルウェアがどのような回避技術を持つかの分析に重点を置いており，既存の PoC コードの包括的な分類は行われていない．

本研究では，サンドボックスの回避耐性評価用データセットの構築方法の実現を目指しており，先行研究 [5] で紹介された回避機能の分類カテゴリに基づき，サンドボックス回避機能を実証する PoC コードの LLM を用いた自動分類を試みる．

3. LLM を用いた回避実証コードの分類

本章では，LLM を用いてサンドボックス回避機能を実証する PoC コードを分類する方法と，その分類結果を評価する方法を 4 つのステップに分けて説明する．図 1 にサンドボックス回避実証コードの分類と評価の流れを示す．初めに，ステップ 1 では PoC コードの収集を行う．次にステップ 2 では収集した PoC コードを人手で分類し，どの PoC コードがどの回避機能を有するか明らかにする．ステップ 3 では LLM に回避機能のカテゴリとその説明を与えて PoC コードの自動分類を行う．最後に，ステップ 4 では分類結果を比較してその精度評価を行う．

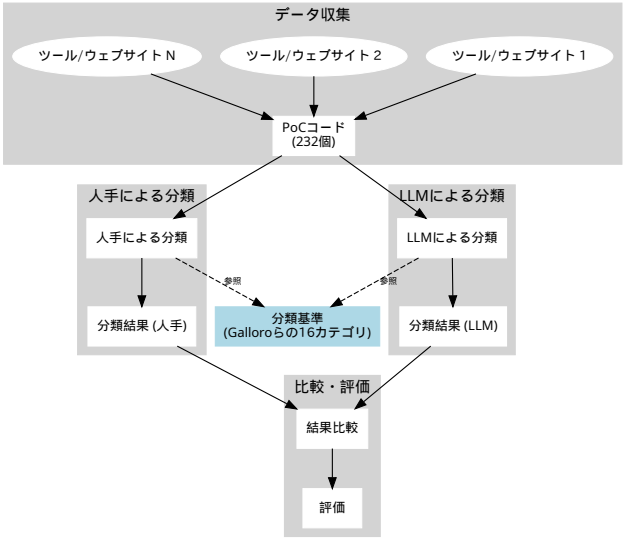

図 1: LLM を用いた PoC コードの自動分類と精度評価の流れ．

Listing 1: Poc コードの例：マウス動作の監視

```
int gensandbox_mouse_act() {
    POINT position1, position2;

    GetCursorPos(&position1);
    Sleep(2000);
    GetCursorPos(&position2);

    if ((position1.x == position2.x) && (position1.y ==
         position2.y))
        return TRUE;
    else
        return FALSE;
}
```

3.1 ステップ 1：PoC コードの収集

はじめに，PoC コードをインターネット上から収集する．ここで，Listing1 に実際の PoC コードの例を示す．ただし，PoC コードは収集したプログラムの一部である可能性があるため，一部のコードについては，サンドボックス解析回避機能に関連する部分（関数，コードブロックなど）のみを抽出する．

3.2 ステップ 2：人手による分類

続いて，収集した各 PoC コードに対して人手で分類ラベルを付与する．ここで，ラベルは様々考えられるが，評価実験では先行研究 [5] で紹介されている 16 種類のサンドボックス解析回避技術のカテゴリを採用した．このカテゴリでは，マルウェアが実際に利用する回避技術を網羅的に捉えており，他の分類法と比較してより実践的な分類が可能であると考えられる．表 1 にサンドボックス回避機能のカテゴリとその説明をまとめる．

3.3 ステップ 3：LLM による自動分類

LLM を用いて PoC コードの自動分類を行う．評価実験では，OpenAI の gpt-4o-2024-08-06 モデルを使用した．gpt-4o は，膨大なテキストデータとコードデータで事前学習されており，自然言語処理とプログラムコード理解の両方において優れた能力を持つ．分類を実行する際には，各 PoC コードのソースコードと，先行研究 [5] の分類定義をプロンプトとして gpt-4o に入力する．その際，出力フォーマットに合わない場合は 3 回まで再試行し，5 回中 3 回以上出てきたものを分類結果とする．プロンプトの設計には，OpenAI API [11] における `system` と `user` の 2 つのロールを用いた．`system` ロールでは，LLM に与える指示や前提条件を定義し，`user` ロールでは，分類対象となる PoC コードのソースコードを与える．具体的には，`system` ロールには以下の 2 つの内容を記述した．

（1） **分類指示**:

> Please classify the code to be entered according to the following categories (selection is optional, duplicates allowed). Output the classification result as a list only (e.g., [”Category1”, ”Category2”]).

（2） **分類カテゴリ**: あらかじめ定義したサンドボックス解析回避技術のカテゴリとその説明．

このように，`system` ロールで分類の指示とカテゴリを明確に定義し，`user` ロールで分類対象のコードを与えることで，LLM に対して適切なコンテキストを提供し，精度の高い分類結果を得ることを目指す．なお，LLM の設定は temperature や top_p などのパラメータを含め，全てデフォルト値を使用した．

3.4 ステップ 4：分類結果の評価

最後に，LLM による分類結果を人手による分類結果と比較し，マルチラベル分類タスクにおける性能評価に一般的に用いられる以下の評価指標を用いて分類性能を評価する．

- **完全一致率（Exact Match Ratio）**：LLM による分類結果が，人手による分類結果と完全に一致した PoC コードの割合を示す．LLM の分類結果が人手による分類結果とどれだけ一致しているかを評価する．
- **適合率（Precision）**：LLM があるカテゴリに分類した PoC コードのうち，人手での分類でもそのカテゴリに分類した PoC コードの割合を示す．割合が高いほど，LLM がそのカテゴリに分類したという判断の信頼性が高いことを示す．
- **再現率（Recall）**：人手で分類した PoC コードのうち，LLM がそのカテゴリに分類した PoC コードの割合を示す．割合が高いほど，LLM がそのカテゴリの PoC コードを見逃すことなく見つけられていることを示す．
- **マイクロ平均適合率（Micro-averaged Precision）**：全てのカテゴリの分類結果をまとめて計算した適合率を示す．カテゴリの大きさ（サンプル数）の差を考慮した，全体の適合率を評価する．
- **マイクロ平均再現率（Micro-averaged Recall）**：全てのカテゴリの分類結果をまとめて計算した再現率を示す．カテゴリの大きさの差を考慮した，全体の再現率を評価する．

4. 評価実験

本章では，インターネット上から収集した 232 個の PoC コードを入力として，LLM を用いて自動分類した結果の精度評価を行う．表 2 に PoC コードの入手元と収集数を示す．ただし，収集した PoC コードが既に収集した PoC コードと同じ回避機能である場合には除外する．このため，入手元が実際に公開している PoC コードの種類数は表中の数とは異なる．

表 1: 本研究で扱うサンドボックス回避機能のカテゴリと説明.

カテゴリ	概要
Memory Fingerprinting	実行中のプロセスのメモリ領域にアクセスしてデバッガを検出する
Exception Handling	例外を発生させ，その例外がデバッガによって処理されるかどうかで，デバッガの存在を検出する
CPU Fingerprinting	物理 CPU と仮想化された CPU の違いを検出する
Table Descriptors	テーブルディスクリプタのアドレスの変化を監視することで仮想環境を検出する
Traps	トラップ命令を使用し， デバッガや仮想環境を検出する
Timing	命令実行時間を計測し，仮想環境やデバッガ，DBI ツールによるオーバーヘッドから，分析環境を検出する
Stalling	長時間スリープなどの遅延行為を挿入することで，サンドボックスの分析時間制限を超え検出を逃れる
Human Interaction	マウスの動きや最終入力時刻などを監視し，人間の操作がない場合に，自動化された分析環境であると判断する
Registry	Windows レジストリ内の特定のキーや値の有無を確認することで，分析環境を特定する
System Environment	MAC アドレスやディスク容量，メモリサイズなど，システム情報から仮想環境やデバッガを識別する
WMI	WMI を利用してシステム情報を取得し，分析環境やインストールされているソフトウェアを検出する
Process Environment	実行プロセスに関する情報を収集・操作することで，分析環境を検出する
File System	特定のファイルやディレクトリの有無を確認することで，分析環境を検出する
List Processes	実行中のプロセス一覧を取得し，既知の分析ツールを特定する
List Services	実行中のサービス一覧を取得し，仮想環境やデバッガに関連するサービスを特定する
Drivers	ドライバ一覧を取得し，仮想化されたデバイスに関連するドライバを特定する

表 2: PoC コードの入手元とその内訳

入手元	分類に用いたコード数
ayoubfaouzi/al-khaser	101
evasions.checkpoint.com	77
unprotect.it	50
a0rtega/pafish	4

4.1 人手による分類結果

収集した PoC コードを人手で分類した．表 3 の 2 列目にカテゴリ毎の分類結果を，表 4 と表 5 に複数のカテゴリに分類される場合の内訳をまとめる．人手による分類結果を詳しく見ると，全てのコードがいずれかのカテゴリに分類されており，その中でも”System Environment”のカテゴリに分類されるコードが多かった．また，187 個のコードが単一のカテゴリに分類され，44 個のコードが 2 つのカテゴリに，1 個のコードが 3 つのカテゴリに分類された．これは，多くの PoC コードが特定の回避技術に特化して実装されている一方で，複数の回避技術を組み合わせたコードも一定数存在することを示している．特に，”System Environment”と”WMI”の組み合わせが多く，これらの回避技術が頻繁に併用されていることを示唆している．

4.2 LLM を用いた PoC コードの分類結果

続いて，収集した PoC コードを LLM を用いて自動分類し，人手による分類結果との比較を行った．表 6 にその評価結果をまとめる．この結果から，マイクロ平均 F 値は 0.88 と高い値を示し，マイクロ平均は，データセット全体の個数を用いて平均化されるため，カテゴリごとのサンプル数の不均衡の影響を受けにくいため，優れた分類性能を有しているといえる．また，LLM の分類結果が人手の分類結果と完全に一致した PoC コードは全体の 77% であった．このため，LLM が多くの PoC コードに対して，人間と概ね同様の分類を行えると言える．

表 3: 人手と LLM での PoC コードの分類結果の比較

	人手での分類	LLM での分類	適合率	再現率
Memory Fingerprinting	6	11	0.55	1.00
Exception Handling	5	5	0.80	0.80
CPU Fingerprinting	21	21	0.95	0.95
Table Descriptors	6	6	1.00	1.00
Traps	3	3	1.00	1.00
Timing	13	21	0.62	1.00
Stalling	12	15	0.67	0.83
Human Interaction	13	9	1.00	0.69
Registry	18	20	0.90	1.00
System Environment	102	91	0.91	0.81
WMI	35	33	1.00	0.94
Process Environment	4	3	1.00	0.75
File System	21	22	0.91	0.95
List Processes	13	13	0.92	0.92
List Services	3	3	1.00	1.00
Drivers	3	3	0.67	0.67
マイクロ平均	-	-	0.88	0.88

表 4: 複数のカテゴリに分類される PoC コードの内訳

カテゴリ数	人手での分類	LLM での分類
1	187	187
2	44	43
3	1	2

続いて，カテゴリ毎の分類結果を比較する．表 3 の 3 列目にカテゴリ毎の分類結果を示す．人手による分類結果と同じように，System Environment カテゴリが多い一方で，Traps や Drivers などのカテゴリは少なく，カテゴリごとにサンプル数が大きく異なっている．カテゴリごとの分類性能には差が見ら

表 5: 複数のカテゴリに分類される PoC コードの詳細

カテゴリの組み合わせ	人手での分類	LLM での分類
System Environment, WMI	29	16
Human Interaction, System Environment	2	0
Exception Handling, Traps	2	2
Memory Fingerprinting, System Environment	2	0
Stalling, Timing	0	10
CPU Fingerprinting, Timing	1	1
System Environment, Timing	0	3
File System, System Environment	1	2
Registry, System Environment	0	2
Drivers, List Services	1	1
Drivers, System Environment	1	0
File System, Human Interaction	1	0
Human Interaction, Registry	1	0
List Processes, Process Environment	1	0
Timing, WMI	1	0
CPU Fingerprinting, System Environment, WMI	1	0
Exception Handling, Stalling	1	0
CPU Fingerprinting, Exception Handling, Traps	0	1
CPU Fingerprinting, Memory Fingerprinting	0	1
Drivers, File System	0	1
Exception Handling, Memory Fingerprinting	0	1
File System, List Processes	0	1
Human Interaction, Stalling	0	1
List Processes, Process Environment, System Environment	0	1
List Services, Registry	0	1

表 6: LLM による PoC コード分類結果の評価

指標	EMR （Accuracy）	Micro 平均
Accuracy	0.77	—
Precision	—	0.88
Recall	—	0.88

れ，”Table Descriptors”や”Traps”，”WMI”などのカテゴリでは，適合率，再現率ともに非常に高く，LLM がこれらのカテゴリを人と同様に分類できていることがわかる．これは，これらのカテゴリに属する PoC コードの特徴が明確であり，LLM にとっても比較的容易に識別できるためと考えられる．一方，”Memory Fingerprinting”や”Timing”，”Stalling”では適合率が比較的低く，”Human Interaction”では再現率が比較的低くなっている．また，”Drivers”では適合率と再現率のとちらも比較的低くなっているが，サンプル数が 3 と極めて少なく，この結果だけで性能を結論づけることはできない．

”Memory Fingerprinting”，”Timing”，”Stalling”，”Human Interaction” のカテゴリにおける誤分類の要因としては，以下のようなものが考えられる．

- **Memory Fingerprinting**: 適合率が 0.55 と比較的低くなっている．これは，LLM がメモリ関連の操作を含むコードを過剰に ”Memory Fingerprinting” と判断してしまう傾向があるためである．例えば，Listing2 に示す NumberOfProcessors 関数のコードは，実際にはプロセッサ数をチェックしているだけであるが，誤って ”Memory Fingerprinting” に分類される．
- **Timing/Stalling**: time 関数など時間関連の関数が使われているコードを”Timing”と”Stalling”の両方に分類する傾向がある．例えば，Listing3 に示すコードは，”Timing”と”Stalling”の両方に分類された．このコードは，曜日が月曜日かどうかを判定し，それによって異なるメッセージを表示するものである．コード内では time 関数と localtime 関数を用いて現在時刻を取得し，strftime 関数で曜日を取得している．これらの関数は時間に関連する処理を行っているため，LLM はこのコードを ”Timing” に分類したと考えられる．一方で，このコードの主目的は，特定の曜日（月曜日）にのみ特定の動作を実行することである．つまり，実行を遅延させる，あるいは特定のタイミングまで待機するという意味で，”Stalling” の特徴を持っている．このように，”Timing”と”Stalling”の分類は，コードの文脈に大きく依存する．時間関連の関数が使用されていても，それが時間経過の計測を目的としているのか，実行の遅延を目的としているのかを判断するには，周辺のコードや処理の流れを総合的に理解する必要がある．LLM は個々の関数や命令は理解できるものの，その文脈を完全に把握することが難しく，このような場合に両方のカテゴリに分類してしまう可能性がある．
- **Human Interaction**: 再現率が 0.69 と低く，LLM がユーザーインタラクションを検出するコードを完全には捉えきれていないことを示している．例えば，マウスの移動やクリック，キーボード入力などの明示的なユーザー操作を検出するコードは正しく分類できる可能性が高いが，デフォルトの壁紙から変更がないや Chrome の検索履歴数が少ないなどの，人手では”Human Interaction”カテゴリに該当するが，LLM にとっては単なる環境情報の取得と区別がつきにくい可能性がある．

Listing 2: Memory Fingerprinting

```
BOOL NumberOfProcessors()
{
#if defined (ENV64BIT)
  PULONG ulNumberProcessors = (PULONG)(__readgsqword(0x30)
       + 0xB8);
#elif defined(ENV32BIT)
  PULONG ulNumberProcessors = (PULONG)(__readfsdword(0x30)
       + 0x64);
#endif

    if (*ulNumberProcessors < 2)
        return TRUE;
    else
        return FALSE;
}
```

Listing 3: Timing/Stalling

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
     hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
    time_t rawtime;
    struct tm * timeinfo;
    char buffer[100];

    time(&rawtime);
    timeinfo = localtime(&rawtime);

    strftime(buffer, sizeof(buffer), "%A", timeinfo);

    const char * str(buffer);

    if (str == "Monday")
    {
        cout << "Wait!" << endl;
        MessageBox(NULL, (LPSTR)str, (LPSTR)str, MB_OK);
```

```
    }
    else
    {
        cout << "Time of attack!" << endl;
        MessageBox(NULL, (LPSTR)str, (LPSTR)str, MB_OK);
    }
    return 0;
}
```

4.3 考　　察

評価実験では，LLM を用いたサンドボックス解析回避機能を持つ PoC コードの自動分類を試みた．実験の結果，LLM は PoC コードの分類タスクにおいて，人間による分類結果と概ね一致する結果を示した．特に，”Table Descriptors”，”Traps”，”WMI” のような，コードの特徴が明確なカテゴリでは，人手による分類と高い一致率を示した．これは，LLM がコード内の特定のパターンやキーワードを効果的に捉える能力を持っていることを示唆している．一方で，”Memory Fingerprinting”，”Timing”，”Stalling”，”Human Interaction”などのカテゴリでは，人手による分類との一致率に課題が残った．これらのカテゴリに共通する特徴として，コードの表面的な特徴だけでは分類が難しく，文脈や人間の判断基準への依存度が高いことが挙げられる．この結果は，LLM がコードの構文的な特徴は捉えられるものの，その意味や文脈，さらには人間特有の判断基準を完全に理解することは難しいことを示唆している．

5. まとめと今後の課題

本研究では，LLM を用いてサンドボックス解析回避機能を持つ PoC コードを自動分類する手法を提案し，その有効性を実験的に検証した．具体的には，232 個の PoC コードを収集し，人手で分類したデータセットを用いて gpt-4o による自動分類を行い，その性能を評価した．実験の結果，完全一致率は 0.77 と LLM は全体として人間による分類結果と概ね一致する結果を示した．特に，マイクロ平均で適合率と再現率が共に 0.88 と，カテゴリ間のサンプル数の不均衡があるデータセットにおいて，LLM が全体として優れた分類性能を発揮していることが確認された．今後は，データセットの拡充と分類基準の精緻化を行う．また，他の LLM（例えば，Gemini，LLaMA など）との性能比較を行うことで，モデルの選択や改善を行う予定である．

謝辞　この成果の一部は，国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP24003）の結果得られたものです．また，本研究の一部は JSPS 科研費 JP23K16879 の助成を受けて行われた．

文　　献

[1] Amir Afianian, Salman Niksefat, Babak Sadeghiyan, and David Baptiste. Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, 52(6):1–28, 2019.

[2] Gabriel Negreira Barbosa and Rodrigo Rubira Branco. Prevalent characteristics in modern malware. `https://kernelhacking.com/rodrigo/docs/blackhat2014-presentation.pdf`, 2014.

[3] Marcus Botacin. Gpthreats-3: Is automatic malware generation a threat? In *2023 IEEE Security and Privacy Workshops (SPW)*, pages 238–254. IEEE, 2023.

[4] Ayoub Faouzi. al-khaser. `https://github.com/ayoubfaouzi/al-khaser`.

[5] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. A systematical and longitudinal study of evasive behaviors in windows malware. *Computers & Security*, 113:102550, 2022.

[6] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 287–301, 2014.

[7] Check Point Software Technologies LTD. Evasion techniques. `https://evasions.checkpoint.com/`.

[8] Najmeh Miramirkhani, Mahathi Priya Appini, Nick Nikiforakis, and Michalis Polychronakis. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 1009–1024. IEEE, 2017.

[9] MITRE. Mitre att&ck - technique t1497: Virtualization/sandbox evasion. `https://attack.mitre.org/techniques/T1497/`.

[10] OpenAI. Hello gpt-4o. `https://openai.com/index/hello-gpt-4o/`.

[11] OpenAI. Openai api reference. `https://platform.openai.com/docs/api-reference/chat/create`.

[12] Alberto Ortega. Pafish - a tool to detect malware sandbox analysis environments. `https://github.com/a0rtega/pafish`.

[13] Unprotect.it. Unprotect - the ultimate source of information about malware techniques. `https://unprotect.it/`.

[14] 松澤輝, 久保颯汰, 田辺瑠偉, and 吉岡克成. Llm を用いて作成した解析回避検体がサンドボックス解析に与える影響の調査. コンピュータセキュリティシンポジウム *2024* 論文集, pages 1148–1155, 2024.