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

松澤 輝[1,a)]　久保 颯汰[2]　インミンパパ[3]　田辺 瑠偉[3,4]　吉岡 克成[3,5]

概要：マルウェアのサンドボックス解析では，サンドボックスと呼ばれる隔離された環境内で検体を実行することで，その挙動を詳細に分析する．しかし，マルウェアの一部には，サンドボックス環境とユーザ環境を判別し，サンドボックス環境の場合には本来の挙動を隠蔽することで解析の回避を試みるものが存在する．そのようなマルウェアは回避型マルウェアと呼ばれ，サンドボックス解析システムにおいては回避型マルウェアに対する耐性を向上させることが求められる．また，近年の機械学習技術の発展に伴い，マルウェア生成における大規模言語モデル（LLM）の悪用が懸念されており，実際に，マルウェアの生成に LLM が応用できることが既に知られている．加えて，LLM を用いたマルウェアのソースコードの書き換えがアンチウイルス製品の回避に影響を与えることを示す研究が行われており，今後はサンドボックス解析システムに対する回避型マルウェアの生成にも LLM が利用される可能性がある．そこで本研究では，そのような回避型マルウェアの生成における LLM の応用性，並びに，サンドボックス解析システムの回避耐性の向上のための評価に焦点を当てた．複数の回避技術を対象に，そのソースコードを LLM を用いて書き換えることで多数の解析回避検体を生成し，それらの検体が既存のサンドボックスの解析結果に与える影響を調査した．

キーワード：解析回避型マルウェア，サンドボックス解析，大規模言語モデル（LLM）

Investigating the Impact of Evasive Malware Created with LLM on Sandbox Analysis

Hikaru Matsuzawa[1,a)]　Sota Kubo[2]　Yin Minn Pa Pa[3]　Rui Tanabe[3,4]　Katsunari Yoshioka[3,5]

Abstract: Sandbox analysis enables detailed examination of malware behavior by executing malware samples within an isolated environment known as a sandbox. However, certain malware can evade analysis by distinguishing between a sandbox environment and a user environment, concealing its true behavior when executed in a sandbox. This type of malware is referred to as evasive malware, and it necessitates the enhancement of resistance in sandbox analysis systems. Moreover, with the advancement of machine learning technologies, there is growing concern over the misuse of large language models (LLMs) in malware generation. It has already been demonstrated that LLMs can be applied to create malware. Additionally, research indicates that modifying malware source code using LLMs can influence the evasion of antivirus products. This raises the possibility that LLMs could also be used in the future to generate evasive malware designed to bypass sandbox analysis systems. In this study, we focus on the applicability of LLMs in generating such evasive malware and evaluate methods for improving the resistance of sandbox analysis systems. We generated numerous variants of evasive malware samples by rewriting the source code of various evasion techniques using LLMs and examined the impact of these variants on the existing malware sandbox.

Keywords: Evasive Malware, Sandbox Analysis, Large Language Model (LLM)

1. はじめに

マルウェア (Malware) とは悪意のあるソフトウェア（Malicious Software）の総称であり，日々多数のマルウェアやその亜種が発見されている．PC やスマートフォンなどの電子機器がマルウェアに感染すると機器自体の破壊や情報漏洩など，個人や組織の利益・信頼の損失に繋がる可能性があり，この状況に対応するためにマルウェアがもつ特徴や挙動を迅速に把握する必要がある．

マルウェアの動的解析には主に，サンドボックスと呼ばれる解析環境内で実際に検体を実行し挙動を解析する技術が用いられる．しかし，実行環境が解析環境であることを検出し，無害な振る舞いを示すことで解析の回避を試みる回避型マルウェアが問題となっている [1]．先行研究ではサンドボックス環境とユーザ環境を識別する技術（以降では，解析回避技術/機能と呼ぶこととする）が体系的に調査されているが [2]，回避技術を実際に再現する方法は明らかにされておらず，企業や官公庁などに導入されている商用のサンドボックス解析システムにおいても回避型マルウェアに十分に対応できていない可能性がある [3]．

加えて，近年の大規模言語モデル（Large Language Model）の急速な発展に伴い，マルウェア作成における LLM の悪用が新たな脅威となっている [4]．悪用対策がされている商用の LLM に対して，脱獄 (Jailbreak) プロンプトと呼ばれる特殊な入力を与えることで対策を回避し，マルウェアのソースコードを生成可能であることが知られており [4]，今後は回避型マルウェアを作成するために攻撃者が LLM を悪用する可能性がある．

本研究では，サンドボックス解析システムの回避型マルウェアに対する耐性向上のため，解析回避技術のソースコードを LLM を用いて書き換えることで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を調査する．本研究の課題（Research Question: RQ）を以下に示す．

RQ1) サンドボックス解析回避技術を実装したソースコードの LLM による書き換えはどの程度可能か？
RQ2) LLM を用いて書き換えた解析回避検体がサンドボックス解析の結果にどのような影響を与えるか？

[1] 横浜国立大学大学院環境情報学府
Graduate School of Environment and Information Sciences, Yokohama National University
[2] 横浜国立大学
Yokohama National University
[3] 横浜国立大学先端科学高等研究院
Institute of Advanced Sciences, Yokohama National University
[4] 順天堂大学
Juntendo University
[5] 横浜国立大学大学院環境情報研究院
Faculty of Environment and Information Sciences, Yokohama National University
[a)] matsuzawa-hikaru-nx@ynu.jp

検証実験では，20 種類の解析回避技術について，その機能を実現するプログラムのソースコードを作成し（以降では，ベース回避コード），LLM を用いて当該コードを書き換えることで各解析回避技術ごとに 15 個，計 300 個のソースコード（以降では，回避コード）を生成した．加えて，サンドボックス解析システムに与える影響を調査するため，生成した 300 個の回避コードから動作確認と実装方法の重複を取り除くためのフィルタリングにより 105 個の回避コードを選別し，ベース回避コードと合わせた計 125 個の回避コードを用いて 125 個の解析回避検体を作成した．そして，マルウェア解析サービスである VirusTotal[5] 上で動作する 5 種類のサンドボックスで解析し，その結果を分析・比較した．

本研究における貢献は以下の通りである．

1). 20 種類の解析回避技術を実装した 20 個の回避コードに対し，LLM を用いたソースコードの書き換えにより各 15 個，計 300 個の回避コードを生成した結果，平均して 3 割程度が同様の機能を維持しつつ元の回避コードとは異なる実装となることを確認した．
2). LLM による書き換えにより作成した 125 個の解析回避検体について，VirusTotal のサンドボックス上で動的解析した結果，同一の解析回避技術であっても実装方法により解析回避の成否，ならびに，解析回避挙動の観測可否に影響を与えることを確認した．

2. 関連研究

回避型マルウェアの解析回避機能に関する研究は多岐に渡るが，Galloro らは Windows における回避型マルウェアが持つ 92 個の解析回避技術を動的に分析し，その普及と進化を定量的かつ体系的に整理した [2]．2010 年から 2019 年までに収集した 45,375 のマルウェア検体を解析し，回避機能を有するマルウェアは過去 10 年間で約 12%増加していることを示すとともに，回避型マルウェアが解析環境を検出する際に一般的に用いるマシンのシステム情報に関して，その情報の取得方法，ならびに，識別に用いるブラックリストや閾値をまとめており，本研究でも参考とした．

一方で，LLM を用いたマルウェア生成に関する研究が増加しており，Yin Minn Pa Pa らは OpenAI が提供する ChatGPT，gpt-3.5-turbo，AutoGPT を利用し，LLM によるマルウェア生成の可能性に言及した [6]．多くの一般的な LLM においては，その社会実装に際し，マルウェアの生成等を始めとした倫理に反する内容の出力を低減するためにフィルタリングなどの対策が取られている．しかし，自然言語のみで構成される特殊なプロンプトを与えることで，ワーム，キーロガー，ランサムウェア，ファイルレスマルウェア等の 7 種のマルウェアと 2 つ攻撃ツールのソースコードを生成可能であることを示した．

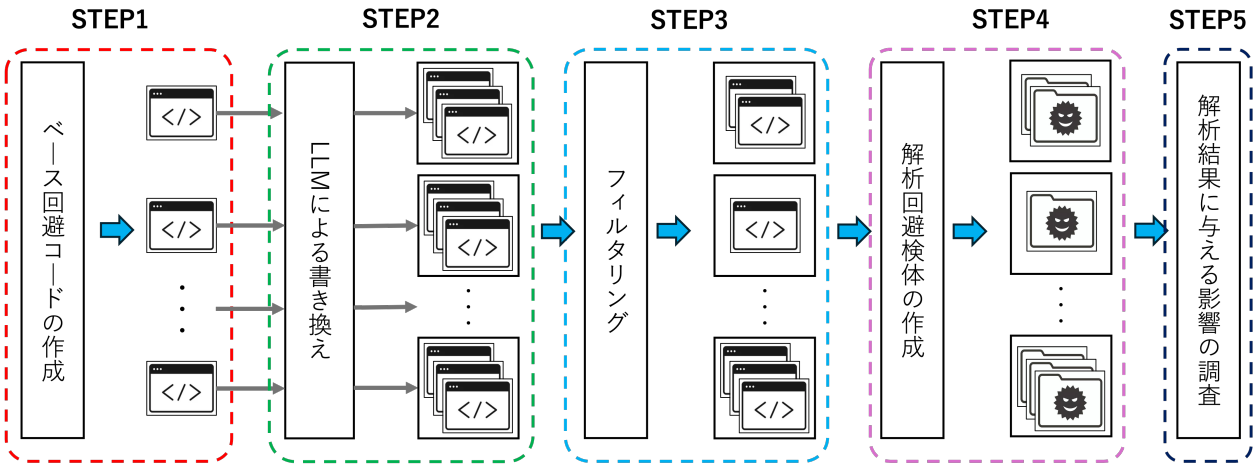

図 1: LLM を用いて作成した解析回避検体がサンドボックス解析に与える影響の調査の流れ

また，黄らは ChatGPT を利用したマルウェア亜種の開発，ならびに，それらの亜種検体がアンチウイルスによる検知に与える影響を調査した [4]．そして，4 種のマルウェア検体をベースとして，そのソースコードを ChatGPT を用いて書き換えることで計 400 個の亜種ソースコードを生成した．そのうち，約半数が実行可能ファイルに変換可能かつ元検体と同じ機能を維持していたことに加え，それらの亜種検体の中には既存のアンチウイルス製品によって検知されないものがあったことから，マルウェアソースコードを ChatGPT 等の LLM を用いて書き換えることによるマルウェア亜種の生成の実現性を示している．

このように，マルウェア自体のソースコードの書き換えがアンチウイルスによる検知結果に与える影響を言及している研究は存在するものの，高度なマルウエアが持つ対解析機能に関する調査は我々の知る限り存在しない. そのため, 様々な耐解析検体と LLM によるそれらの亜種がサンドボックス解析に与える影響の調査は本研究の新規性である．

3. 調査手法

本研究における調査は主に 5 つのステップで構成される．その概要を図 1 に示す．STEP1 では，調査対象の解析回避技術を選定し，ベース回避コードを作成する．次に STEP2 では，ベース回避コードを LLM により書き換える．STEP3 では，LLM により書き換えた回避コードについて，動作確認によるフィルタリングを実施する．STEP4 では，フィルタリング後の各回避コードを用いて解析回避機能を有する検体を作成する．最後に STEP5 では，作成した検体を VirusTotal にアップロードし，複数のサンドボックス上で実行して解析結果を分析・比較する．なお，今回はサンドボックス解析のみに焦点を当て，静的解析の結果は考慮しない．また，サンドボックス環境は Windows OS を想定しており，回避コード等の作成には Python を利用した．以降では，各 STEP の詳細を述べる．

3.1 STEP1: ベース回避コードの作成

調査対象の解析回避技術を選定し，ベース回避コードを作成する．ベース回避コードの作成に際しては，回避型マルウェアに関する先行研究やインターネット上から収集した解析レポートを参照する．解析回避技術のソースコードが含まれる文献 [7], [8] については Python コードに変換し，解析回避技術の概念のみを述べた文献 [9], [10] についてはその記述を再現するコードを手動で実装する．

3.2 STEP2: LLM によるベース回避コードの書き換え

本研究における回避コードの書き換えとは，回避のために取得するシステム情報 (例:CPU コア数) を元の回避コードとは異なる方法で取得するように回避コードの実装を変更することと定義する．回避コードは実行環境の何らかのシステム情報を取得して評価することで解析環境とユーザ環境を判定するが，その際，Windows 環境においては異なるライブラリやコマンドの利用等により複数の実装方法が存在する．例として，図 2 および図 3 に回避コードの書き換え例を示す．2 つの図のコードは，実行環境の CPU コア数を取得する回避コードの一部分を表している．図 2 が LLM の入力として与えるベース回避コードの本質部分であり，図 3 が本調査における定義に合致する書き換えの例である．書き換え前のベース回避コードでは psutil (Python System and Process Utilities) と呼ばれるシステム情報を取得・管理するためのモジュールを利用して CPU コア数を取得する．一方で，書き換え後の回避コードは，Python スクリプトから外部プログラムやコマンドを実行するための subprocess モジュールを利用し，Windows のシステム情報を取得するツールである wmic (Windows Management Instrumentation Command-line) のコマンドを実行することで CPU コア数を取得する．

LLM の種類は多岐に渡るが，検証実験では，OpenAI 社が提供する gpt-4o-2024-05-13[11] を API 経由で使用し，

```
import psutil

physical_cores = psutil.cpu_count(logical=False)
```

図 2: 回避コードの書き換え例：ベースの回避コード

```
import subprocess

cores = (
    subprocess.check_output(["wmic", "cpu", "get", "
        NumberOfCores"])
    .decode()
    .split("\n")[1]
)
physical_cores = int(cores.strip())
```

図 3: 回避コードの書き換え例：書き換え後のコード

```
Rewrite the following source code, keeping the same
    functionality.

### Request for Conversion:
- The output code should be a different code style
    from the input code.
- Use different algorithms, data structures,
    libraries, etc. as appropriate.
- Do not change the names of variables.

### Input Souce code
# - Place base evasion code here -

### Examples
# - Place examples of the rewriting -
```

図 4: 書き換えに用いたプロンプトの例

書き換え対象のベース回避コードの機能を維持した書き換えを指示するプロンプトを併せてモデルへ入力した．図 4 にプロンプトの一例を示す．プロンプトは書き換えの条件，入力回避コード，書き換え例の 3 つで構成され，生成される回避コードの多様性を向上させるため，書き換えの例を含まない Zero-Shot，例を 1 つ含む One-Shot，例を 3 つ含む Few-Shot の 3 種類のプロンプト作成・利用した．加えて，1 回の応答につき 5 つの書き換え後の回避コードを出力するよう設定したうえで，1 つのベース回避コードにつき，前述の 3 種類のプロンプトをそれぞれ用いた生成を 1 回ずつ試行した．また，$max\_tokens = 4000$，その他の生成に伴うパラメータにはデフォルト値の $temperature = 1, top\_p = 1, frequency\_penalty = 0$ 等を使用し，1 つの解析回避技術あたり 15 個，調査対象の解析回避技術全体で計 300 個の回避コードを生成した．

3.3 STEP3: 生成された回避コードのフィルタリング

前節で生成した回避コードの中には，正常に動作しないコードや，解析回避に用いるシステム情報の取得方法が同じだが変数などが異なるコードが含まれる可能性がある．そのため，以下に示すフィルタリングルールを実施した．

1). 生成が途中で途切れてしまったコードや実行時エラー等，正常に実行できないものを除外する．
2). 生成された回避コードを動作確認用の複数のテスト環境で実行し，LLM による書き換えの前後で同じ機能を有すること，加えて，実行環境のシステム情報を正しく取得できることを確認する．その際，書き換えにより解析回避機能に変化が生じたものは除外する．
3). 解析回避に用いるシステム情報を取得する際に使用するライブラリや関数，コマンドを確認することで重複するものを除外する．

なお，回避コードの実行可否は Windows 10 pro (22H2) と Windows 11 pro (23H2) の 2 種類の環境で動作確認を実施した．また，回避コードの書き換え成否はソースコードを人手で確認することで異なる実装であるかを判断した．

3.4 STEP4: 解析回避検体の作成

フィルタリング実施後の各回避コードとベースの回避コードに対し，マルウェア検体（以降，ペイロードと呼ぶ）を結合することで，解析回避技術ごとに解析回避機能を有する検体を複数作成した．具体的には，マシンの画面をロックするとともに解除のためのパスワードを要求するロッカー型マルウェア検体を使用し，回避コードが実行環境をユーザ環境と判定した場合に限りペイロードを実行するよう条件分岐を設定した．加えて，サンドボックス解析時に検体の挙動を把握できるよう，攻撃者視点での解析回避の成否，すなわち，ペイロードが実行されたか否かと，利用したシステム情報を別途用意した調査用サーバに送信する機能を追加した．結合後のコードは PyInstaller[12] により実行可能ファイルへと変換した．

3.5 STEP5: サンドボックス解析に与える影響の調査

サンドボックス解析システムが回避型マルウェアに対応する方法は大きく 2 種類考えられる．サンドボックスをユーザ環境に似せることで回避型マルウェアの環境判定を騙し，マルウェア本来の挙動を観測する方法と，マルウェア検体が解析環境検知のためにシステム情報を取得する挙動を捉えることで回避型マルウェアの可能性を警告する方法である．そこで，以下の 2 点に着目して解析回避機能を有する検体がサンドボックス解析に与える影響を調査する．

1). **システム情報の取得による解析回避の成否**
解析回避の成功・失敗に差異が生じ得るかを調査用サーバに対して送信された情報に基づき判断する．また，取得した解析環境のシステム情報に実装方法ごとに差異が生じるかを確認する．

表 1: 調査対象の回避機能

Evasion Method	Criteria	LOC
CPU のコア数の確認	4 以上	20
CPU のプロセッサ数の確認	4 以上	20
HDD の全容量の確認	256GB 以上	20
HDD の残り容量の確認	100GB 以上	25
MAC アドレスプレフィックスの確認	ブラックリスト非該当	40
仮想 HDD からの起動か確認	ブラックリスト非該当	40
インストール済みソフトウェア数の確認	50 以上	30
オーディオデバイスの有無を確認	1 個以上	30
クリップボードの有無を確認	値が存在するか	20
モニターの有無を確認	1 個以上	25
スクリーン解像度の確認	ブラックリスト非該当	30
Downloads フォルダの要素数の確認	100 個以上	30
特定プロセスの有無を確認	ブラックリスト非該当	45
特定レジストリキーの有無を確認	ブラックリスト非該当	40
メモリサイズの確認	4GB 以上	20
ユーザ名の確認	ブラックリスト非該当	20
キーボード入力の有無を確認	操作があるか	60
ダイアログボタンの押下を確認	操作があるか	30
マウス移動の有無を確認	操作があるか	30
マウスクリックの有無を確認	操作があるか	50

2). **解析回避に対する警告の有無**
解析回避の成否にかかわらず，解析の回避を試みたことがサンドボックス解析システムに観測されるかを確認する．具体的には，MITRE ATT&CK [13] における解析回避関連のタグである Virtulization/Sandbox Evasion が解析結果に付与されるかを確認し，付与された場合には解析回避の試みが検出されたとする．

4. 検証実験

はじめに，計 20 種類の解析回避技術を LLM による書き換えの対象として選定し，それぞれに対応するベース回避コードを手動で作成した（表 1）．なお，表中の Criteria は各解析回避技術が解析環境とユーザ環境を識別する際の条件であり，実行環境から取得したシステム情報が条件を満たす場合にはユーザ環境，満たさない場合には解析環境として判定する．また，LOC（Lines Of Code）は各回避コードの行数を 5 行刻みの切り上げで表した値である．

4.1 LLM による回避コードの書き換え結果

本調査の対象とした回避コードの ChatGPT による書き換え，ならびに，フィルタリング後の結果を表 2 に示す．ChatGPT により計 300 個の回避コードを生成し，フィルタリングを実施した結果，約 3 割である 105 個の回避コードの書き換えに成功した．このことから，サンドボックス解析回避技術の LLM による書き換えは一定の精度で可能であることが確認でき，LLM により多様な回避型マルウェアを攻撃者によって作成され得ることが示唆される．一方，各解析回避技術に着目すると，HDD の全容量と残り容量，メモリサイズの取得には 8 通りの書き換えが成功したのに対して，マウスクリックの検出については 2 通りしか成功しなかったことから，解析回避技術の中でも書き換えの難易度には大きな差があるといえる．このような差が生じた理由としては，Windows OS のシステム上，回避コードの複数の実装自体が困難である場合や，使用するプログラミング言語により実装可能な方法が限定されるためだと考えられる．

表 2: ChatGPT による回避コードの書き換え結果

回避技術	生成数	フィルタリング後
CPU のコア数の確認		4
CPU のプロセッサ数の確認		5
HDD の全容量の確認		8
HDD の残り容量の確認	各	8
MAC アドレスプレフィックスの確認	15	4
仮想 HDD からの起動か確認	個	7
インストール済みソフトウェア数の確認	の	5
オーディオデバイスの有無を確認	回	4
クリップボードの有無を確認	避	3
モニターの有無を確認	コ	6
スクリーン解像度の確認	｜	7
Downloads フォルダの要素数の確認	ド	6
特定プロセスの有無を確認	を	6
特定レジストリキーの有無を確認	生	4
メモリサイズの確認	成	8
ユーザ名の確認		6
キーボード入力の有無を確認		4
ダイアログボタンの押下を確認		4
マウス移動の有無を確認		4
マウスクリックの有無を確認		2
合計	300	105

4.2 サンドボックス情報の取得結果

フィルタリング後の 105 個の回避コードと 20 個のベース回避コードを用いて作成した回避型マルウェア 125 検体を VirusTotal 上で動作する 5 つのサンドボックス解析システムにより解析した結果を表 3 に示す．表中の ● は検体が実行環境を解析環境と識別したこと，○ はユーザ環境と識別したことを表す．すなわち，● は攻撃者視点での解析の回避に成功したことを表し，○ は解析の回避に失敗し，ペイロードが実行されたことを表す．また，続く括弧内は解析回避試行時に解析環境から取得したシステム情報を表す．例として，CPU のプロセッサ数を利用する解析回避検体を Sandbox F にアップロードした際の結果は 5 検体のうち 3 検体が，実行環境の CPU プロセッサ数として 2 を取得し，解析環境と判別したため回避に成功したことを表す．残り 2 検体は CPU プロセッサ数として 8 を取得し解析環境をユーザ環境と識別した結果，ペイロードが実行されたため，解析の回避に失敗したことを表す．なお，表中の Error は実行時エラーが発生したことを表す．

4.3 サンドボックス解析への影響の調査結果

作成した解析回避検体を 2024 年 8 月 12 日から 8 月 19 日の間に VirusTotal 上で動作する 5 種類のサンドボックス解析システム A〜F により解析し，その結果を分析・比較することで，LLM による回避コードの書き換えが解析結果に与える影響を調査した．

20 種類のうち 10 種類の解析回避技術において，その実装が異なる亜種検体間で解析回避の成否に差が生じた．具体的には，同等の解析回避機能を有する実装の異なる亜種検体群に関して，VirusTotal 上で動作する 5 つのサンドボックス解析システムのうち少なくとも 1 つで解析回避の成否が異なっていた．例として，HDD の全容量を利用する解析回避技術では，サンドボックス E において，100GB を取得し解析環境と識別した検体と 1024GB を取得しユーザ環境と識別した検体が存在し，解析回避の成否が異なっていることがわかる．このように，同一の解析回避技術でも異なる実装がなされた検体は実行環境のシステム情報として異なる値を取得する場合があり，その結果として回避の成否が変化する場合があることが確認された．一方，HDD の全容量に基づく解析回避技術のサンドボックス B における結果のように，検体によって取得する値が 40GB と 64GB のように異なる場合であっても，解析回避の成否は変わらない例が確認できる．これらの結果から，LLM による回避コードの書き換えにより作成した解析回避検体はサンドボックス解析の結果に影響を与える場合が存在すると捉えることができる．

実装方法により回避コードが取得するシステム情報に差異が生じ，解析回避の成否に影響を与えた理由としては，主に 2 点が考えられる．1 点目は，実行環境の変化に伴う回避コードの動作の変動である．本調査においては，Windows 10，ならびに，Windows 11 にて LLM により生成した回避コードの動作確認を行い，正しく動作することを確認した上で解析回避検体を作成した．しかし，実行環境によっては使用するライブラリなどの挙動が変化する可能性や参照する情報や計算方法に差異が生じる可能性がある．また，一部の解析環境ではユーザ環境を仮想的にエミュレートするため，実マシンと仮想マシンの間に生じるシステム上の相違が取得するシステム情報の差異に関係している可能性も考えられる．

2 点目は，解析環境におけるフッキングの影響である．フッキングとは，解析のために特定のシステムや API の動作を監視・変更する技術であり，回避型マルウェアに対する対策としても使用される．例えば，回避型マルウェアが CPU のコア数の取得を試みた場合，実際の解析環境ではコア数が 2 であるのに対して，6 などの予め設定した偽の情報を返すようにフックすることで，回避型マルウェアに対して解析環境をユーザ環境と誤認させることが可能となる．そのため，サンドボックス解析システムの中には，主要な API やシステムに対してフッキングが設定されているものが存在するが，1 つのシステム情報を取得する場合でも多数の方法・ルートが存在するため，その全てがフックにかかるとは断言できない．具体的には，本調査では 1 つの解析回避技術について，実装の異なる複数の回避型マルウェアの亜種検体を作成した．これらの検体は異なるライブラリやコマンド等を使用するため，同一のシステム情報を取得する場合でもシステム上のどのような情報源からどのように取得するのがそれぞれ異なる場合がある．その結果として，解析環境に設置されたフックにかかり偽のシステム情報を取得する検体と，フックを通らず解析環境の真のシステム情報を取得する検体が生じ，取得するシステム情報に差異が生じた可能性がある．

また，回避コードにより取得されるシステム情報の差異の大小に着目すると，微小なものとそうでないものが確認できる．例えば，HDD の全容量を取得した際の結果を見ると，サンドボックス C において 208GB と 224GB の 2 通りに分かれ，その差は比較的小さいが，サンドボックス D では 80GB と 238GB，サンドボックス E では 100GB と 1024GB のように取得した値に大きな差が確認できる．前者の場合，HDD 容量により解析環境を検出する回避型マルウェアへの対策としては不十分であるため，別の要因により取得した値に差異が生じたものと推測されるが，後者は回避型マルウェアへの対策である可能性がある．加えて，表全体よりサンドボックス解析システムにより取得するシステム情報間の差異の生じやすさにも違いがあることが確認できる．Sandbox A では実行時エラーを除けば異なる実装の検体間で取得するシステム情報に差異が存在しないのに対して，Sandbox C，Sandbox D では比較的多くの解析回避技術で差異が生じており，サンドボックス解析システムにより解析回避機能の異なる実装が影響しやすいものとそうでないものがあることがわかる．

一方で，マウスの移動やクリックの有無，キーボード入力の有無を確認する解析回避技術など，実装による解析結果の変化が見られないものも確認できる．これは，実行環境の影響を受けず，すべての検体が正しくシステム情報を取得した場合と，フックが設定された上ですべての検体が偽の情報を取得した場合の 2 パターンの可能性が考えられる．また，解析回避の成否に注目すると，半数以上のサンドボックスではマウスの移動やクリックは解析時に操作するよう対策されており解析の回避を阻止できているが，キーボード入力はすべてのサンドボックス解析システムで考慮されておらず，解析の回避が成功してしまっていることがわかる．加えて，表全体を通して，いずれのサンドボックス解析システムにおいても半数程度の解析回避技術により回避が成功しており，既存のサンドボックス解析システムにおける回避型マルウェアへの耐性が十分ではないと捉えることができる．

表 3: VirusTotal 内の 5 種類のサンドボックス解析システムによる評価結果
●：解析の回避に成功　○：解析の回避に失敗　※括弧内は取得したシステム情報を表す

解析回避技術	回避挙動の検知	Sandbox A	Sandbox B	Sandbox C	Sandbox D	Sandbox E	Sandbox F
CPU のコア数の確認	√ (3/4)	● (2CPUs)×2 ● (Error) ×2	● (1CPUs)×4	○ (4CPUs)×3 ○ (8CPUs)×1	● (2CPUs)×1 ○ (4CPUs)×2 ○ (8CPUs)×1	● (2CPUs)×4	● (2CPUs)×4
CPU のプロセッサ数の確認	なし (0/5)	● (2threads)×5	● (1threads)×5	○ (4threads)×5	○ (4threads)×5	● (2threads)×5	● (2threads)×3 ○ (8threads)×2
HDD の全容量の確認	なし (0/8)	○ (750GB)×8	● (40GB)×3 ● (64GB)×5	● (208GB)×3 ● (224GB)×5	● (80GB)×3 ● (238GB)×5	● (100GB)×2 ○ (1024GB)×6	● (95GB)×8
HDD の残り容量の確認	√ (2/8)	○ (716GB)×8	● (26GB)×8	○ (109GB)×2 ○ (169GB)×6	● (46GB)×8	○ (307GB)×8	● (39GB)×8
MAC アドレスプレフィックスの確認	√ (1/4)	○×4	●×2 ○×2	○×4	○×4	○×4	○×4
OS が仮想 HDD から起動されたか確認	√ (3/7)	○×7	○×7	○×7	○×7	○×7	○×7
インストール済みソフトウェア数の確認	なし (0/5)	● (36apps)×5	● (46apps)×5	● (35apps)×3 ○ (56apps)×2	○ (73apps)×5	● (5apps)×5	● (23apps)×5
オーディオデバイスの有無を確認	√ (3/4)	● (0pcs)×4	● (0pcs)×2 ○ (1pcs)×2	● (0pcs)×4	● (0pcs)×1 ○ (1pcs)×3	○ (1pcs)×4	○ (1pcs)×4
クリップボードの有無を確認	なし (0/3)	○×3	●×3	●×3	○×3	○×3	●×3
モニターの有無を確認	なし (0/6)	○ (1pcs)×6	○ (1pcs)×6	●(0pcs)×1 ○ (1pcs)×5	○ (1pcs)×6	○ (1pcs)×6	●(0pcs)×3 ○ (1pcs)×3
スクリーン解像度の確認	√ (1/7)	● (900, 1440)×7	○ (1024, 1280)×7	○ (768, 1024)×7	● (960, 1280)×2 ● (1050, 1400)×5	○ (1024, 1280)×7	○ (600, 800)×7
Downloads フォルダの要素数の確認	なし (0/6)	● (1items)×6	● (3items)×6	● (13items)×2 ● (19items)×2 ● (25items)×2	● (1items)×6	● (8items)×6	● (8items)×6
特定のプロセスの有無を確認	√ (4/6)	○×6	○×6	○×6	○×6	○×6	○×6
特定のレジストリキーの有無を確認	√ (3/4)	●×4	●×4	●×4	●×4	●×4	○×4
メモリサイズの確認	√ (3/8)	○ (4GB)×8	● (1GB)×8	○ (8GB)×8)	○ (4GB)×8	● (2GB)×8	● (2GB)×6 ○ (16GB)×2
ユーザ名の確認	なし (0/6)	● (John)×6	○ (Abby)×6	○ (george)×6	○ (Bruno)×6	● (Administrator)×5 ○ (ADMIN～1)×1	● (admin)×6
キーボード入力の有無を確認	なし (0/4)	●×4	●×4	●×4	●×4	●×4	●×4
ダイアログボタンの押下を確認	なし (0/4)	●×2 ○×2	●×4	●×2 ○×2	●×2 ○×2	●×3 ○×1	●×2 ○×2
マウス移動の有無を確認	なし (0/4)	○×4	●×4	○×4	○×4	○×4	○×4
マウスクリックの有無を確認	なし (0/2)	○×2	●×2	○×2	○×2	●×2	○×2

4.4 解析回避のシグネチャに与える影響

サンドボックス解析システムにおいては，解析回避の成否に加えて，回避型マルウェアの解析回避の挙動を観測することも重要であると言える．本調査では，解析結果に Virtulization/Sandbox Evasion と呼ばれる解析回避に関連する MITRE ATT&CK[13] の挙動タグが付与されるかを基に，解析回避技術の実装方法が解析回避の挙動の観測に与える影響を調査した．その結果を表 3 の 2 列目に示す．√のついている解析回避技術は 5 種類のサンドボックス解析システムの少なくとも 1 つで解析回避の挙動を観測し該当するタグが付与されたことを表す．また，その後に続く括弧内は，分母が全検体数，分子がタグが付与された検体数を表す．例として，CPU のコア数を利用する解析回避技術では，解析回避機能部分の実装が異なる 4 つの検体のうち，3 検体に解析回避のタグが付与され，残り 1 検体にはタグが付与されなかったことを表す．表より，同一の解析回避技術であっても，実装方法によりタグが付与されるものとそうでないものが多数存在していることから，解析回避挙動の検知についても，回避コードの実装が影響を与え得ることがわかる．すなわち，解析回避の挙動として予め設定されたパターンに当てはまる検体とそうでないものが存在し，サンドボックス解析システムが解析回避技術の実装に依存する一面があると捉えることができる．

4.5 研究倫理

本研究では回避型マルウェア作成における LLM の悪用可能性とその影響について調査を行った．LLM のサイバー攻撃への悪用については既に多くの検討がなされているが，回避型マルウェアなどの高度なマルウェア作成における悪用可能性と現行のセキュリティ対策技術に与える影響の分析は不十分であり，これらを正確に把握することはサイバー攻撃の高度化への適切な対応に資するものであると考え，論文として報告する次第である．研究成果の悪用を防ぐため，調査に用いた解析回避技術のソースコードは論文内では開示しないが，調査に関する詳細情報を希望する研究者との情報共有は適切な手続きに従い行う予定である．また，サンドボックス解析システムについては，一般的な製品への影響を示すことを目的とし，特定製品への言及は避けることとした．

5. まとめと今後の課題

回避型マルウェアの持つ解析回避技術に着目し，そのソースコードを LLM を用いて書き換えることで作成した解析回避検体がサンドボックス解析の結果に与える影響を調査した．検証実験の結果から，攻撃者は回避型マルウェアの作成に LLM を悪用することで多種多様なマルウェアを効率的に作成できる可能性が示唆された．加えて，実験に用いたサンドボックス解析システムは一定の解析回避耐性を持つ一方で，作成した多数の解析回避検体が解析の回避に成功したことから対策は十分ではないと言える．

今後の展望として，実装方法ごとの呼び出す API の違いや順序の違いを調査することで，解析結果に影響を与える原因を分析する．加えて，本調査で使用しなかった LLM やプロンプト，解析回避技術，サンドボックス解析システム，プログラミング言語についても調査を行うことで，回避型マルウェア作成における LLM の悪用性の検証，ならびに，サンドボックス解析システムの回避型マルウェアに対する耐性向上のための検討を進める．

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（JPNP22007）の結果得られたものです．本研究の一部は JSPS 科研費 JP23K16879 の助成を受けて行われた．

参考文献

[1] Watch Guard, “Watchguard’s threat lab analyzes the latest malware and internet attacks,” https://www.watchguard.com/wgrd-resource-center/security-report-q4-2023.

[2] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, “A systematical and longitudinal study of evasive behaviors in windows malware,” *Computers & security*, vol. 113, p. 102550, 2022.

[3] A. Yokoyama, K. Ishii, R. Tanabe, Y. Papa, K. Yoshioka, T. Matsumoto, T. Kasama, D. Inoue, M. Brengel, M. Backes, and R. Christian, “Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion,” in *19 the Research in Attacks, Intrusions, and Defenses (RAID ’16)*, 2016, pp. 165–187.

[4] 黄哲偉, インミンパパ, 吉岡克成, 松本勉, “Chatgpt を用いたソースコードの書き換えがアンチウイルスの検知に与える影響,” in **電子情報通信学会** *ICSS* **研究会**, 2024.

[5] Google Inc., “Virustotal,” https://www.virustotal.com.

[6] Y. M. Pa Pa, S. Tanizaki, T. Kou, M. Van Eeten, K. Yoshioka, and T. Matsumoto, “An attacker’s dream? exploring the capabilities of chatgpt for developing malware,” in *Proceedings of the 16th Cyber Security Experimentation and Test Workshop (CSET’23)*, 2023, pp. 10–18.

[7] Check Point Software Technologies LTD, “cp¡r¿ windows: Evasion techniques,” https://evasions.checkpoint.com/src/Evasions/index.html.

[8] O. Alberto, “Pafish,” https://github.com/a0rtega/pafish?tab=readme-ov-file.

[9] G. Jiaxuan, W. Junfeng, F. Zhiyang, Z. Yingjie, W. Di, and G. Wenhan, “A survey of strategy-driven evasion methods for pe malware: Transformation, concealment, and attack,” *Computers & Security*, vol. 137, p. 103595, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823005059

[10] UNIT 42, “Navigating the vast ocean of sandbox evasions,” https://unit42.paloaltonetworks.com/sandbox-evasion-memory-detection/.

[11] OpenAI, “Models - openai api,” https://platform.openai.com/docs/models/gpt-4o#4ofootnote.

[12] C. David and C. William, “Pyinstaller,” https://pyinstaller.org/.

[13] MITRE, “MITRE ATT&CK,” https://attack.mitre.org/.