CL-002

複数端末から成る横展開マルウェア解析システムの提案
Multi-machine Malware Analysis System for Lateral Movement Monitoring

藤井 翔太[1)] 都築 陽一[2)] 岡本 崇典[2)] 田村 悠[1)] 佐藤 隆行[1)]
Shota Fujii Yoichi Tsuzuki Takanori Okamoto Yu Tamura Takayuki Sato

概要

サイバー攻撃に用いられるマルウェアの機能の一つに横展開があり，感染拡大等の重要な役割を有する．他方で，既存の動的解析システムは 1 台のみによる構成が多く，横展開のような複数台に影響を及ぼす挙動の観測が困難である．そこで，横展開のように複数台に影響を及ぼす挙動を観測可能なシステムを提案する．本システムは複数の端末を起動し，マルウェア動作端末に加え，展開先への通信や展開後の挙動を観測する．また，展開後の端末において通常利用外のプロセスを抽出することで横展開後の挙動を特定し，解析を支援する．本稿では，システムの設計と実マルウェアを用いた予備評価を通した横展開や展開後の挙動に対する観測可能性を示す．

1 はじめに

マルウェアの数や巧妙さが年々増加していることを受け，自動化されたマルウェア動的解析が広く普及している．ここで，マルウェアの機能の一つに横展開があり，感染拡大等の重要な役割を有する．他方で，既存の動的解析システムは，1 台の解析環境のみによる構成が多く，横展開のような複数台に影響を及ぼす挙動や展開後の挙動を観測することが困難である．

そこで本稿では，横展開をはじめとした複数台に影響を及ぼすマルウェアの挙動を観測可能なシステムを提案する．同システムは，同一ネットワーク内に複数の端末を起動し，マルウェアの動作端末での観測に加え，展開先への通信や展開後のプロセスの挙動を観測する．この際，展開後の端末において，ホワイトリストを用いて通常利用外のプロセスを抽出することにより，横展開後の挙動を特定して解析を支援する．

また，デファクトの動的解析システムの一つである cuckoo sandbox [1] をベースとして，提案システムの実装を示す．さらに，提案システムのプロトタイプを実装し，疑似マルウェアや実マルウェアを用いた予備評価を通して，横展開や展開後の挙動に対する解析可能性を示す．加えて，処理時間を計測し，日々の利用での実用性を検証する．

本研究の主な貢献は以下の通りである．

- 複数端末から構成され，横展開等の複数端末に影響を及ぼすマルウェアの挙動を自動で観測可能なマルウェア動的解析システムを設計した．また，Virtual Machine Monitor（VMM）に ESXi，動的解析ソフトウェアに cuckoo sandbox を用いた場合の実装を具体的に示した．
- プロトタイプを実装し，疑似検体と実検体を用いた評価を通して，提案システムを用いることによってマルウェアの横展開や横展開後の挙動を自動で観測できることを実証した．

1） 株式会社 日立製作所. Hitachi Ltd.
2） 株式会社 FFRI セキュリティ. FFRI Security, Inc.

2 背景

2.1 横展開

攻撃者が攻撃対象の端末を侵害した後，より多くの端末を侵害するためにネットワーク上の別システムに侵入することは，横展開（lateral movement）として知られている．攻撃技術を定義するフレームワークの一つである *MITRE ATT&CK*® [2] においても TA0008 Lateral Movement として定義されている．防御側としては，横展開を許してしまうとさらに被害が拡大してしまうことから，特に検知・防御するべき攻撃手法の一つであると言える．

また，横展開の機能は，自動化されてマルウェアにも組み込まれており，効率的な攻撃の達成に寄与している．代表的なものとしては，telnet 等を介して感染拡大を図る mirai [3]，ファイル共有プロトコルの脆弱性やダンプした認証情報等を悪用して感染拡大を図る WannaCry，後発の NotPetya や BadRabbit [4] 等が挙げられる．こうした自動で感染拡大ならびに横展開を図るマルウェアは，依然として活動を続けており，亜種だけでなく新しい種別のマルウェアも継続的に観測されていることから，その挙動を解析して理解し，対策につなげることが重要である．

2.2 マルウェアの動的解析

マルウェアの動的解析は，解析対象のマルウェア検体を実際に動作させてその挙動を分析する手法であり，広く活用されている [5]．他方で，既存のマルウェア動的解析システムは，1 台の解析環境のみから構成されていることが多い．このため，前述した横展開のような複数台に影響を及ぼす挙動や展開後の挙動を観測することが困難である．

そこで，本稿では，横展開のような複数の環境に影響を及ぼすようなマルウェアを解析可能な解析システムの研究開発を目標とする．

3 横展開マルウェア解析システム

3.1 要件

本稿では，先述の通り，複数の環境に影響を及ぼすようなマルウェアを解析可能な動的解析システムの構築を目指す．同システムを実現するに際しての要件として，以下の 2 点が挙げられる．

（要件 1）動的解析システム内で複数端末を扱えること

横展開の挙動を誘発するとともに，横展開後の挙動を観測するために，複数端末を取り扱う必要がある．その際，マルウェアを起動する端末だけでなく，それ以外の端末でもマルウェアに関する挙動を観測し，記録できる必要がある．

（要件 2）横展開後のプロセスを特定できること

解析用端末においては，良性のプロセスやマルウェアの挙動に関連しないプロセスが無数に存在し，マルウェ

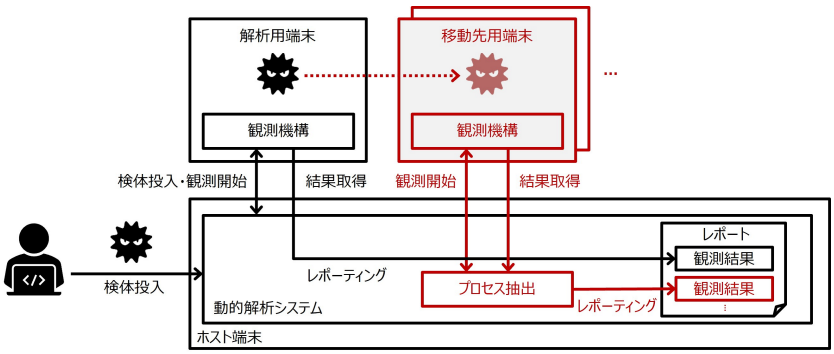

図 1　提案システムの全体像

アの解析においてはノイズとなる．そこで動的解析システムにおいては，実行したマルウェア検体のプロセスやその子プロセス等を抽出することによって，マルウェアに係る挙動を絞り込む．しかし，横展開先の端末においては起点となるマルウェア検体に直接紐づくプロセスが存在しない．このため，他の手法を用いて，マルウェアの挙動に関連するプロセスを抽出する必要がある．

3.2　基本アイデアと全体像

まず，（要件 1）を満たすために，同一ネットワーク内に複数の端末を起動し，マルウェアの動作端末での観測に加え，展開先への通信や展開後のプロセスの挙動を観測する．また，NTP サーバを用意し，両環境の時刻を同期することによって，複数環境にまたがった挙動を同一の時系列で取り扱うことができるようにする．これらの施策により，動的解析システム内で複数端末を取り扱う．

また，（要件 2）を満たすために，通常利用の範囲で起動されるプロセスを列挙し，許可リストを作成する．その後，マルウェアを実行した環境以外において，許可リストにないプロセスをマルウェアの横展開に起因する可能性があるものとして抽出する．

提案システムの全体像を図 1 に示す．なお，cuckoo sandbox 等の単一環境を対象とした一般的なマルウェア動的解析システムでの構成を黒色で記載し，複数端末を取り扱うに際して追加する箇所を赤色で記載している．

3.3　システムの構成

図 1 に示した通り，提案システムは大別して以下に示す 3 つの端末から構成される．

• ホスト端末

動的解析システムをホストする端末として動作し，後述の解析用端末や移動先用端末を制御する．また，実行する検体を解析者から受け取り，解析用端末に投入して実行する．さらに，検体の動作が完了した後，各端末から観測結果を取得してパースし，レポートとしてまとめる．

• 解析用端末

動的解析システムのゲスト端末として動作し，動的解析システムから渡されたマルウェア検体を実行する．また，検体が実行されている間，動的解析システムの機能として解析用端末における検体の挙動を観測する．具体的には，マルウェアにかかるネットワーク通信やプロセスの挙動等を観測する．検体の実行終了後，観測結果をホスト端末の動的解析システムに渡す．

• 移動先用端末

解析用端末と同様に，動的解析システムのゲスト端末として動作する．また，解析用端末からの横展開等を可能とするために，同一のネットワーク上に配置する．この際，解析用端末と同様に，ネットワーク通信やプロセスの挙動等を観測する．検体の実行終了後，観測結果をホスト端末の動的解析システムに渡す．なお，検体を最初に動作させる解析用端末は 1 台だが，移動先用端末は 1 台以上（複数台）から構成される．

3.4　観測対象

表 1 に，MITRE ATT&CK の TA0008 Lateral Movement として定義されている横展開のテクニック一覧を示す．また，MITRE ATT&CK では，各テクニックについてサブテクニックを定義してより詳細化している．例えば，T1563 Remote Service Session Hijacking については，T1563.001 で SSH のハイジャック，T1563.002 で RDP のハイジャックがサブテクニックとして定義されている．他方で，上位のテクニックが各サブテクニックを高い粒度で内包しているため，ここでは割愛した．

提案システムは，自動化されたマルウェアを解析するものであるため，横展開のテクニックのうち，自動化可能なものを対象とする．具体的には，表 1 の「対象」の列に示す通り，T1210 Exploitation of Remote Services のような自動化可能なものを対象とし，T1534 Internal Spearphishing や T1091 Replication Through Removable Media 等の人手でのメール送信や USB メモリの付け替えのように完全に自動化してマルウェアに組み込む事が困難なものは対象外としている．

また，表 1 に「対象」として示した横展開に係るテクニックは，基本的には同一ネットワーク上に端末が存在すれば達成可能である．例えば，T1021 Remote Service は，横展開先のクレデンシャルが判明していれば，Windows Management Instrumentation（WMI）のようなデフォルトで有効なサービス[1)]を介して横展開を達成できる．他方で，RDP を介した横展開や SMBv1 の脆弱性を悪用した横展開のように，一部デフォルトで有効化されていないサービスに依存するものもある．そこで，解析用端末や移動先用端末において RDP や SMBv1 等を有効化し，横展開に係る挙動の誘発を図る．

3.5　移動先用端末でのプロセスの特定

先述の通り，移動先用端末においては，実行した検体が別端末にあることから，検体に関するプロセスが親子

1）　Windows 2000 以降の OS ではデフォルトでインストールされる

表 1　横展開のテクニック一覧（TA0008 Lateral Movement）と提案システムの対象

ID	名称	説明	対象
T1210	Exploitation of Remote Services	脆弱なリモートサービス（SMB，RDP 等）を悪用した横展開	✓
T1534	Internal Spearphishing	組織内でのスピアフィッシング（電子メール）による横展開	-
T1570	Lateral Tool Transfer	横展開用のツールの転送	✓
T1563	Remote Service Session Hijacking	リモートサービス（RDP 等）の既存セッションの乗っ取りによる横展開	✓
T1021	Remote Services	有効なアカウントを用いたリモートサービス（RDP 等）による横展開	✓
T1091	Replication Through Removable Media	リムーバブルメディア（USB メモリ等）を用いた横展開	-
T1072	Software Deployment Tools	企業ネットワーク内の資産管理システム等を用いた横展開	-
T1080	Taint Shared Content	ネットワークドライブ等の共有ストレージを用いた横展開	✓
T1550	Use Alternate Authentication Material	パスワードハッシュ等を用いて認証プロセスを回避することによる横展開	✓

表 2　実装環境

役割	OS/VMM	CPU	メモリ	主なソフトウェア
VMM	ESXi 7.0	Intel Xeon Silver 4316 2.3GHz（20C/40T）	256GB	
ホスト端末	Ubuntu 18.04 LTS 64bit	同上（vcpu 2 コア）	2GB	cuckoo sandbox 2.0.7，chrony 3.2
解析用端末	Windows 10 Pro 64bit	同上（vcpu 1 コア）	1GB	Wireshark 3.4.0，Process Monitor 3.87，DTrace 1.1.0
移動先用端末	Windows 10 Pro 64bit	同上（vcpu 1 コア）	1GB	同上

関係の形で直接紐づかない．このため，別の方法を用いて実行した検体に係るプロセスを特定する必要がある．そこで，提案システムでは，通常利用の範囲で生成されるプロセスをまとめた許可リストを作成し，移動先用端末内で観測したプロセスのうち許可リストにないものを検体に係るプロセスである可能性のあるものとして抽出する．

プロセスの許可リストは，検体を動作させない状態で起動した移動先用端末上で，一定時間生成されるプロセスを観測して作成する．このプロセス許可リストを作成したうえで観測を開始し，観測完了後に移動先用端末で取得したプロセスの中から許可リストにないものを抽出してレポートに記載する．なお，インストールしたソフトウェアやバージョンに依り，端末ごとに動作するプロセスは異なる可能性がある．このため，移動先用端末が複数存在する場合は，端末ごとにプロセスの許可リストを作成する．

4　実装

4.1　概要

本章では，3 章で示した設計について，デファクトの動的解析システムの一つである cuckoo sandbox（バージョン 2.0.7）を用いた場合の実装について述べる．Cuckoo sandbox を含む，本章で取り扱う実装に係る各種ソフトウェアや環境を表 2 に示し，その全体像を図 2 に示す．以降の記述は，同表に示した環境，ソフトウェア，およびそれぞれのバージョンを前提とする．また，以降では，cuckoo sandbox のインストールディレクトリ（`/usr/local/lib/python2.7/dist-packages/cuckoo`）を`$CUCKOO`，cuckoo sandbox の working directory（`~/.cuckoo`）を`$CWD` と表記する．

本稿での提案システムの実装では，各端末は仮想環境上に構築し，VMM には ESXi を用いる．このため，設定ファイル（`$CWD/conf/cuckoo.conf`）中の VMM の指定（`[cuckoo] machinery`）をデフォルト値の `virtualbox` から `esx` に変更する．ホスト端末，解析用端末，および移動先用端末はローカルネットワーク（解析環境用ネットワーク）で接続する．同ネットワークは，解析環境の操作，検体のやり取り，端末間の感染の誘発のために用いる．この際，NTP サーバとして chrony [6] をホスト端末にインストールし，端末間の時刻を同期する．また，ホスト端末は後述の通り VMM の ESXi に接続して解析用端末や移動先用端末を操作するため，ホスト端末と ESXi が接続する用途のネットワークを管理用ネットワークとして設ける．

提案システムを用いた観測においては，まず検体に係るプロセスを抽出するためのプロセス許可リストを構築する．その後，前述の環境下において，標準の cuckoo sandbox が担う観測用端末の制御に加えて，移動先用端末の操作，同端末内での検体の挙動の観測，および結果のレポーティングを行う．以降の節では，cuckoo sandbox における機能追加について説明した後，上述の追加実施事項や追加機能について述べる．

4.2　Cuckoo sandbox における機能追加

Cuckoo sandbox では，カスタマイズ用のモジュールが幾つか用意されており，これらを活用することにより，機能の拡張や追加が可能である．本節では，提案システムの実装に活用する auxiliary modules と processing modules について説明する．

• Auxiliary modules

Auxiliary modules では，検体の解析開始前と解析完了時に実行する処理を定義できる．同モジュールは，ホスト側の版とゲスト側の版が存在する．ホスト側で実行したい処理は，Python プログラムとして記載し，ホスト側の`$CUCKOO/auxiliary/`に格納することで，ホスト上で解析開始前と解析終了時に実行される．ホスト側の auxiliary modules の記法は，リスト 1 に示す形で定義されている．Auxiliary クラスを継承し，起動時の処理を start() 以下に，終了時の処理は stop() 以下に記載することで，cuckoo sandbox がそれぞれのタイミングで各処理を実行する．

ゲスト側で実行したい処理は，ホスト側の`$CWD/analyzer/windows/modules/auxiliary/` に Python プログラムとして格納することで，ホスト側からゲスト側に送付され，ゲスト上で検体実行前に実行される．ただし，ゲスト側の auxiliary modules には，終了時の処理に係る定義はないため，別の手法を用いて終了時の処理を開始する必要がある．提案システムでは，開始時の処理を実行した後にプログラムを終了せずに待機する．その後，ホスト側の auxiliary modules の終了処

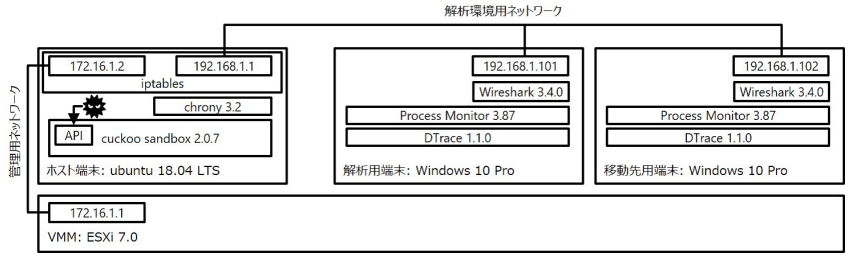

図 2　実装環境の全体像

Listing 1　Auxiliary modules の記法（ホスト側）

```
from cuckoo.common.abstracts import Auxiliary

class MyAuxiliary(Auxiliary):

    def start(self):
        # 起動時の処理

    def stop(self):
        # 終了時の処理
```

理時にホスト側から waitfor シグナルを送付する．ゲスト側では同シグナルを受け取ったタイミングで処理を再開することにより，解析終了のタイミングでゲスト上でも任意の処理を実行できるようにした．

• Processing modules

Processing modules は，検体の実行を通して得られたデータを分析し，グローバルコンテナと呼ばれるデータ構造に格納する．グローバルコンテナに格納された解析結果は最終的に reporting modules で JSON 等のレポート形式に変換される．この processing modules にも，処理を追加することができる．具体的には，解析終了後に processing modules で実行したい処理は，Python プログラムとして記載し，ホスト側の`$CUCKOO/processing/`に格納することで，ホスト上で解析終了後に実行される．

以降で述べる各機構は，解析とは独立したプロセスの許可リストの構築（4.3 節）を除き，上記の何れかのモジュールを用いて実装する．

4.3　プロセス許可リストの構築

解析に先んじて，3.5 節で述べた移動先用端末におけるプロセスの許可リストを構築する．具体的には，移動先用端末の Windows 10 を起動した後，Process Monitor [7] を起動した状態で 24 時間動作させ，その間に起動したプロセスを取得し，許可リストとする．また，上記の方法で取得したプロセスは，Process Monitor を起動した後のプロセスのみであるため，それ以前のプロセスは Windows 起動直後のプロセスリストを別途確認し，許可リストに追加する．今回の環境では，winlogon.exe，services.exe，sihost.exe，explorer.exe，および MsMpEng.exe の 5 つが確認されたため，これらのプロセスを加えた合計 108 のプロセスから成る許可リストを用いた．

4.4　移動先用端末の操作

解析用端末は，デフォルトの cuckoo sandbox が解析前に起動やスナップショットの復元を，解析後に停止を実行する．しかし，移動先用端末はデフォルトの cuckoo sandbox は操作しないため，別途実装する必要がある．

Cuckoo sandbox は，VMM が ESXi の場合は先述の解析用端末の操作を libvirt [8] を用いて実施する．提案システムもこれに倣い，libvirt を用いて移動先端末の停止，起動，およびスナップショットの復元の操作を実行する．具体的には，`$CWD/conf/esx.conf` から移動先用端末の IP アドレスやスナップショット名を取得し，管理用ネットワーク経由で libvirt を介して ESXi の機能によって移動先用端末を操作する．なお，解析開始時の起動やスナップショットの復元はホスト側の auxiliary modules の起動処理部分に，解析終了時の停止処理は同モジュールの終了処理部分に実装する．

4.5　観測

提案システムは，一般的な動的解析システムに倣う形でネットワーク通信，プロセス，および Windows API の観測を行う．本節では，各観測項目について，それぞれの実装方法を述べる．

なお，上述の 3 項目については，解析用端末に関しては標準の cuckoo sandbox が同様の観測を行う．ただし，端末間での観測結果の一貫性を確保するため，今回は移動先端末だけでなく解析端末にも本節で述べる観測機構を実装した．

また，後述の通り何れの観測においても，観測結果を解析用端末/移動先用端末からホスト端末に送付する．この観測結果の送付には，cuckoo sandbox がホスト端末にファイルを送付する関数である`lib.common.results.upload_to_host()` に倣い，INET ドメインソケット通信を利用する．

4.5.1　ネットワーク通信の観測

ネットワーク通信の観測には，パケットキャプチャツールの一つである Wireshark に付属し，同等のキャプチャがコマンドラインベースで実行可能な tshark [9] を使用する．

まず，解析用端末/移動先用端末での解析開始前に，tshark を起動する．具体的には，ゲスト側の auxiliary modules を用いてゲスト上でコマンドを実行することで tshark を起動し，解析中に発生したネットワーク通信を観測する．

解析終了後，ゲスト側の auxiliary modules を用いて tshark を停止するとともに，tshark の機能を用いて観測結果ファイルを pcap ファイルとして出力する．その後，出力した pcap ファイルを前述の手法でホスト側端末に送付する．

最後に，processing modules の中で pcap ファイルをパースし，その結果をグローバルコンテナに追加して処理を終了する．

4.5.2　プロセスの観測

プロセスの観測には，プロセスについてファイル操作やレジストリ操作等を監視できるツールである Process Monitor [7] を使用する．

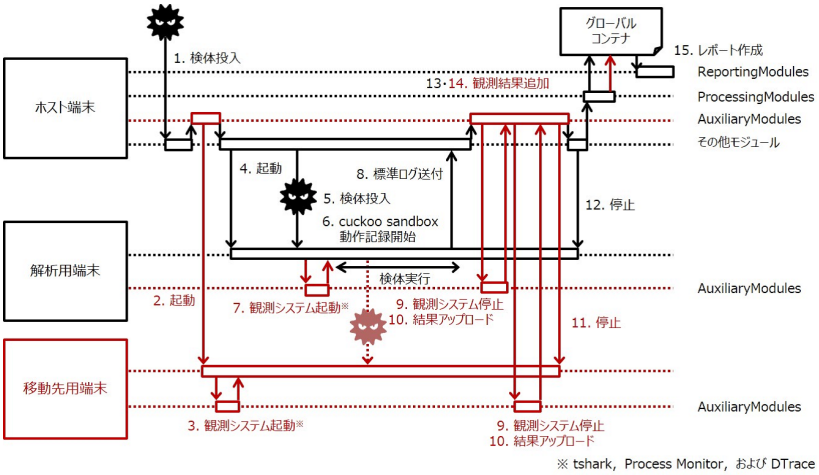

図 3　cuckoo sandbox を用いた提案システムの処理フロー

まず，解析用端末/移動先用端末での解析開始前に，ホスト側の auxiliary modules を用いて 4.3 節で構築したプロセスの許可リストをホスト端末から送付する．ホスト端末からのファイルの送付には，cuckoo sandbox に実装されている `cuckoo.core.guest.upload_analyzer()` を利用する．同関数は，特定のディレクトリ（解析用端末が Windows の場合は，ホスト側の`$CWD/analyzer/windows/`）以下に格納されているファイルを送付するため，プロセスの許可リストは同ディレクトリに配置しておく．その後，ゲスト側の auxiliary modules を用いて Process Monitor を起動し，解析中に発生したプロセスの挙動を観測する．この際，プロセスの許可リストを読み込むことによって，同リストに含まれないプロセスのみを検体に関連するものとして観測する．

解析終了後，ゲスト側の auxiliary modules を用いて Process Monitor を停止するとともに，Process Monitor の機能を用いて観測結果ファイルを pml ファイルとして出力した後 csv ファイルに変換する．その後，csv ファイルを前述の手法でホスト側端末に送付する．

最後に，processing modules の中で csv ファイルをパースし，その結果をグローバルコンテナに追加して処理を終了する．

4.5.3　Windows API の観測

Windows API の観測には，トレーサの一つである DTrace [10] を使用する．DTrace は，プロセス ID を指定することで，対象のプロセスが発行する Windows API をトレースすることができるものである．

まず，解析用端末/移動先用端末での解析開始前に，ゲスト側の auxiliary modules を用いて DTrace 起動用の自作プログラムを起動する．DTrace では，前述の通りプロセス ID を指定する必要がある．このため，上述の自作プログラムは，プロセスの起動を検知して起動したプロセスの PID を指定して DTrace を起動することにより，プロセスのトレースを取得する．また，プロセスの起動の監視は，`wmic process` コマンドによってプロセスリストを一定時間ごとに取得し，その差分を新規に起動したプロセスとして取得することにより達成する．この際，4.5.2 節で送付したプロセスの許可リストを活用し，同リストに含まれないプロセスのみを検体に関連するものとしてトレースする．

解析終了後，ゲスト側の auxiliary modules を用いて DTrace を停止し，観測結果ファイルを csv ファイルとして出力した後，前述の手法でホスト側端末に送付する．

最後に，processing modules の中で csv ファイルをパースし，その結果をグローバルコンテナに追加して処理を終了する．

4.6　システムの処理フロー

Cuckoo sandbox を用いた場合の処理フローを図 3 に示し，以下で説明する．なお，標準機能にはなく今回追加した箇所について，以下の説明では **[追加]**，図 3 では赤色として示している．

1. ホスト端末の cuckoo sandbox に cuckoo API で検体を投入する．
2. **[追加]** 移動先用端末を全て起動し，スナップショットを復元する．
3. **[追加]** 移動先用端末で tshark，Process Monitor，および DTrace による観測を開始する．
4. 解析用端末を起動し，スナップショットを復元する．
5. 解析用端末に検体を投入する．
6. 解析用端末にて cuckoo sandbox 標準の動作記録を実施する．
7. **[追加]** 解析用端末で tshark，Process Monitor，および DTrace による観測を開始する．
8. 検体の解析を終了し，解析用端末から cuckoo sandbox 標準の動作記録をホスト端末にアップロードする．
9. **[追加]** 解析用・移動先用端末で tshark，Process Monitor，および DTrace による観測を終了する．
10. **[追加]** 解析用・移動先用端末で観測した tshark，Process Monitor，および DTrace の結果ファイルをホスト端末にアップロードする．
11. **[追加]** 移動先用ゲストを停止する．
12. 解析用ゲストを停止する．
13. ホストで cuckoo sandbox 標準の動作記録をグローバルコンテナに追加する．
14. **[追加]** アップロードされた tshark，Process Monitor，および DTrace の結果をパースし，グローバルコン

テナに追加する．
15. グローバルコンテナを基にレポートを作成する．

上記のフローによって，cuckoo sandbox の各種機能を活用して複数端末から成る解析環境を制御しつつ，横展開等の複数端末に影響を及ぼす動作を含むマルウェアの挙動を観測する．また，各種観測結果を手順 10 にてホスト側にアップロードしているが，それに加えて手順 14 にて各種観測を cuckoo sandbox 標準のレポートに同一フォーマットで追記することによって，従前のレポートと透過的に活用できるようにした．

5 予備評価

5.1 予備評価に係る環境設定

予備評価に際して，検体の動作を妨げないようにするため，以下の設定を解析端末と移動先用端末に対して実施した．

- ユーザアカウント制御の無効化
- Microsoft Defender ファイアウォールの無効化
- Microsoft Defender ウイルス対策の無効化
- Windows Update の無効化
- 自動スリープの無効化

また，各端末はインターネットに接続しない状態で解析を実施した．横展開等の解析用/移動先用端末間での影響を観測するために，両端末間の通信は制限していない．それ以外の通信については，cuckoo sandbox の通信等，必要最低限のもの以外はすべて通信を許可しない設定とした．

5.2 予備評価項目

4 章で述べた実装に沿ったプロトタイプを用いて，以下に示す予備評価を実施した．

（評価 1）解析可能性

本予備評価では，疑似検体と実検体を用いて，横展開を図るマルウェアの解析可能性を検証した．

横展開に係るマルウェアの挙動は，主に展開に際してのネットワーク通信と展開後のプロセス等からなる悪性挙動の 2 つに大別できる．そこで，それぞれについて疑似検体を実装し，その解析可能性を検証した．具体的には，近傍の端末を ICMP Echo で探索する疑似検体を用いてネットワーク通信に係る複数端末にまたがる挙動の解析可能性を評価した．また，PsExec によって別端末上でプログラムを実行する疑似検体を用いて，プロセスに係る複数端末にまたがる挙動の解析可能性を評価した．さらに，実検体として，横展開を図る事が知られている BadRabbit[2)] [11] を用いて，その解析可能性を評価した．

（評価 2）処理時間

提案システムは，解析者が日々のマルウェア動的解析に係る業務で活用することを想定している．そこで，提案システムの処理時間を測定し，実用の範囲であるかを評価した．

また，文献 [5] の調査において，多くのマルウェア解析者が動的解析時の実行時間は 5 分を基準としていることが明らかにされている．そこで，各予備評価においても，検体の各実行時間を 5 分間とした．

2） MD5: fbbdc39af1139aebba4da004475e8839

5.3 予備評価結果:（評価 1）解析可能性

5.3.1 疑似検体 1: ネットワーク通信に係る挙動

本評価に際して，同一ネットワークに所属する別端末に ICMP Echo（ping）を送付するプログラムを疑似検体として実装した．Ping は，ネットワーク上の別端末の探索，稼働中サービスの確認，および脆弱性有無の推定等に利用されることが知られている [12] ため，ネットワーク通信に係る挙動の解析可能性を検証するための疑似検体で活用した．

本疑似検体を実行したところ，解析用端末から移動先要端末に対して ping が送付されることが確認できた．また，提案システムでの観測結果より，解析用端末と移動先用端末の両方に，src が解析用端末（192.168.1.101），dst が移動先用端末（192.168.1.102）の ICMP Echo Request/Reply がネットワーク通信として記録されていることが確認できた．

以上より，ネットワーク通信に係る挙動を顕現させること，および観測できることが確認できた．

5.3.2 疑似検体 2: プロセスに係る挙動

本評価に際して，電卓（`calc.exe`）をマルウェアと見立て，同一ネットワークに所属する別端末上の電卓を PsExec によって起動するプログラムを疑似検体として実装した．PsExec は，遠隔の端末を操作・管理するための良性ツールであるものの，その利便性から横展開にも悪用される事が知られている [13] ため，プロセスに係る挙動の解析可能性を検証する疑似検体で活用した．

本疑似検体を実行したところ，移動先端末で電卓が起動することが確認できた．また，提案システムでの観測結果より，解析用端末と移動先用端末の両方に，ネットワーク通信として SMB2 プロトコルによる `Create Request File: PSEXESVC.exe` 等から成る PsExec を用いた横展開の痕跡が確認された．加えて，移動先用端末に `CreateFile()` API によって `C:\Windows\PSEXESVC.exe` が作成されていることやその後 `calc.exe` が実行されていること等，上記のネットワーク通信に加えてプロセスの粒度での横展開およびその後の挙動に係る痕跡が確認された．

以上より，5.3.1 項で述べたネットワーク通信に加えて，プロセスに係る挙動に関しても顕現させること，および観測できることが確認できた．

5.3.3 実検体

本評価では，前述の通り横展開を図ることが知られている BadRabbit の実検体を利用した．

本検体を実行したところ，解析用端末に加えて，移動先用端末も暗号化されることが確認できた．また，提案システムでの観測結果より，BadRabbit のコンポーネントの一つである `infpub.dat` が解析用端末から移動先用端末にコピーされて実行されていることが確認できた．具体的には，デフォルトで公開状態になっている共有フォルダである管理共有（`admin$`）に対して，`NtCreateFile()`・`NtWriteFile()` で `UNC\192.168.1.102\admin$\infpub.dat` としてコピーされることによって横展開が図られていた．また，解析用端末から svcctl リモートプロシージャコール（RPC）を使用して，移動先用端末上にコピーした `infpub.dat` を実行することで，展開先での感染活動を行っていた．これらの一連の感染

拡大に係る挙動は，上述のファイルやプロセスの挙動に加えて，ネットワーク通信として SMB2 プロトコルによる `Create Request File: infpub.dat` や `Create Request File: svcctl` 等としても痕跡が確認された．

以上より，実検体についても，その横展開に係る挙動を顕現させること，および観測できることが確認できた．

5.3.4 まとめ

今回用いた各種検体，およびそれに類する検体は，単一環境のみから成る解析環境では動作が顕現しない可能性があり，顕現したとしても展開後の別端末での挙動の観測は困難であると言える．これに対し，本章での予備評価を通して，解析用端末に加えて移動先の端末を同一ネットワーク内で用意することによって横展開に係る挙動を顕現させる事が可能なことを示した．予備評価の内容は，直接的には 5.3.2 項の横展開が表 1 中の T1570 Lateral Tool Transfer や T1021Remote Services に，5.3.3 項の横展開が T1080 Taint Shared Content 等に該当する．ただ，それ以外の本研究が対象としている横展開に係るテクニックも上述のテクニック群と同様にネットワーク上に別端末が存在する際には自動で実行され得るものであるため，定性的には何れのテクニックも顕現させることが可能と推察される．

また，移動先端末でネットワーク通信やプロセス挙動を観測することで，従前の単一環境からなる動的解析システムでは困難な横展開後の挙動の観測が可能となることを示した．

5.4 予備評価結果:（評価 2）処理時間

提案システムの処理時間を評価するため，5.3.2 項で用いた疑似検体を 5 分間解析した．この際，提案システムの実装に際して追加した機能の前後で時間をログへ出力し，処理の開始時間と終了時間の差分をとることにより，処理時間を算出した．

提案システムの処理時間を図 4 に示す．標準の cuckoo sandbox の処理を青色，提案システムの実装に際して追加した処理を橙色で示している．全体の処理時間は，合計で 1,026 秒（17 分 6 秒）である．このうち，観測終了後に移動先用端末からホスト端末へ結果をアップロードして移動先用端末を終了するまでの処理に時間を最も要している．これは，各観測システムからの結果出力，ファイル形式の変換，および結果のアップロードをゲスト上で実施していることによる．特に，ネットワーク通信の観測結果（pcap）のサイズが大きく，各処理に時間を要している．ただし，図 2 に示した通り，移動先用端末には最低限の計算資源のみを割り当てており，割り当てるコア数やメモリを追加することにより，処理時間を短縮することが期待される．また，同処理は移動先用端末ごとに実施するため，仮に移動先用端末を増やしたとしても各端末で並列に行われることから，処理時間は長大化しないと推察される．さらに，文献 [5] の調査において，マルウェア解析者が 1 週間に解析するマルウェアは 20 件前後であると述べられている．提案システムは自動的な解析システムであることから，同調査の数よりも大量に解析することが期待されることを考慮しても，処理時間は実用の範囲であると推察される．

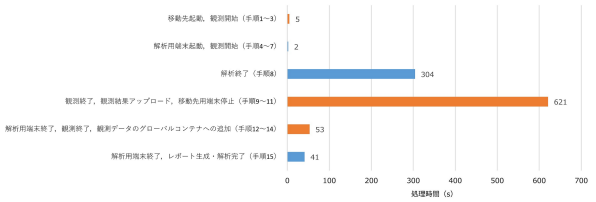

図 4　提案システムの処理時間（秒）

6 議論

6.1 移植可能性

本稿では，cuckoo sandbox を用いた場合の実装を述べた．他方で，本稿で述べた実装は，解析用ゲストでの電源操作，ファイルの授受，およびプログラム実行といった動的解析システムに標準で備わっている機能と公開されているプログラム群の組み合わせから構成されている．このため，cuckoo sandbox 以外のシステムをベースとした場合であっても実装することは可能である．実際に，cuckoo sandbox は，2023 年 6 月現在において最終更新が 3 年間停止しており，実用に際しては移植についても検討する必要があると考えられる．例えば，cuckoo sandbox をベースとした動的解析システムである後発の cape sandbox [14] に関しては，上述の理由に加えてシステム的にも共通部分が存在することから，提案システムの移植が比較的容易であると推察される．

6.2 解析回避

本稿で述べた提案システムは，横展開に係る挙動の誘発とその観測に焦点を当てて設計や実装を述べたが，マルウェアの解析環境を検知することによる解析回避は考慮していない．このため，マルウェアが有する解析回避機能によって解析環境であることを検知され，横展開に係る挙動が顕現しない可能性がある．解析回避機能を回避する手法としては，ファイルやブラウザの使用履歴等を蓄積してユーザが使用している状況を模擬する手法 [15]，解析環境検知のために参照されるレジストリエントリや各種命令の応答を偽装する手法 [16, 17]，および解析環境としてベアメタル環境を用いる手法 [16, 18] 等が知られている．これらの手法を取り込むことにより，本課題を緩和できると推察される．

6.3 制限事項

本稿における提案システムの実装では，マルウェアを動作させる環境は，解析用端末 1 台と移動先端末 1 台の合計 2 台のみから構成されている．このため，2 台以上の環境に展開を図る挙動やユーザ端末以外に影響を及ぼす挙動を観測しきれない可能性がある．例えば，Active Directory の認証情報を活用して横展開する検体 [19] が報告されており，こうした検体の動作を観測しきれない可能性がある．この課題に関しては，解析環境に別の移動先用端末や認証サーバのような別途狙われ得るシステムを追加することによって緩和できると考えられる．

6.4 研究倫理

提案システムは，マルウェアを取り扱うため，攻撃や感染拡大の通信が外部に発生する可能性がある．このため，今回の実験は，評価の章で述べた通り，外部に接続できない閉塞環境で実施した．また，実験に活用するマ

ルウェア検体については，限られたメンバのみがアクセスできる領域で管理した．

7 関連研究

提案システムと同様に，マルウェアの自動解析を図る研究やシステムは，cuckoo sandbox [1] をはじめとして多数存在する．しかし，前述の通りこれらの研究やシステムは単一の環境を前提としており，横展開等の複数環境に影響を及ぼす挙動を観測することは難しい．

利用者の環境を模したような，より高度な解析環境もある [20, 21]．これらの解析環境は，複数の端末に加えて，DNS や Active Directory 等，一般的な組織のネットワークに存在するシステムを内包していることが多く，複数環境に影響する挙動を観測することも可能である．しかし，これらの解析環境は，人手での解析を前提にしていることや高度な解析環境の構築に計算機資源が大量に必要なことから，大量の検体を自動で解析することが目標である本研究とはスコープが異なる．例えば，提案システムで解析したものの中から横展開を試みる可能性がある検体やより不審な検体を選定し，これらの高度な解析で解析するといった連携方法が考えられる．

また，認証ログやエンドポイントのログを活用して横展開の検知を図る研究 [22] やマルウェアの挙動を横展開を含む *MITRE ATT&CK techniques* にマッピングする研究 [17] もある．これらの技術を提案システムで観測したログに対して応用することにより，横展開に係る挙動の検出をより高度に実施できる可能性がある．

8 おわりに

本稿では，横展開等の複数端末に影響を及ぼすマルウェアを観測するシステムを提案した．本システムは，解析用端末に加えて複数の端末を起動し，展開先への通信や展開後の挙動を観測する．また，展開後の端末において通常利用外のプロセスを抽出することで横展開後の挙動を特定し，解析を支援する．本稿では，提案システムのプロトタイプを実装し，予備評価によって横展開を図るマルウェアの解析可能性を示した．今後の課題として，より多くのマルウェアを用いた評価が挙げられる．

謝辞 本研究は国立研究開発法人情報通信研究機構事業「サイバー攻撃に関する解析作業」で実施したものです．本研究を進めるにあたって有益な助言と協力をいただいた関係各位に深く感謝致します．

参考文献

[1] Cuckoo Sandbox. Automated Malware Analysis. https://cuckoosandbox.org/.

[2] MITRE. ATT&CK. https://attack.mitre.org/.

[3] Manos Antonakakis, Tim April, Michael Bailey, Matthew Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, Deepak Kumar, Chaz Lever, Zane Ma, Joshua Mason, Damian Menscher, Chad Seaman, Nick Sullivan, Kurt Thomas, and Yi Zhou. Understanding the mirai botnet. In *26th USENIX Conference on Security Symposium*, SEC '17, pp. 1093–1110, 2017.

[4] Maxat Akbanov, Vassilios Vassilakis, G., and Logothetis Michael, D. Wannacry ransomware: Analysis of infection, persistence, recovery prevention and propagation mechanisms. *Journal of Telecommunications and Information Technology*, Vol. 1, No. 1, pp. 113–124, 2019.

[5] Miuyin Yong Wong, Matthew Landen, Manos Antonakakis, Douglas M. Blough, Elissa M. Redmiles, and Mustaque Ahamad. An inside look into the practice of malware analysis. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pp. 3053–3069, 2021.

[6] chrony. Introduction. https://chrony.tuxfamily.org/.

[7] Microsoft Learn. Process Monitor - Sysinternals. https://learn.microsoft.com/en-us/sysinternals/downloads/procmon.

[8] libvirt. The virtualization API. https://libvirt.org/index.html.

[9] tshark. tshark(1) Manual Page. https://www.wireshark.org/docs/man-pages/tshark.html.

[10] Microsoft Learn. DTrace on Windows. https://learn.microsoft.com/en-us/windows-hardware/drivers/devtest/dtrace.

[11] Cisco Talos. Threat Spotlight: Follow the Bad Rabbit. https://blog.talosintelligence.com/bad-rabbit/.

[12] Zhihong Tian, Wei Shi, Yuhang Wang, Chunsheng Zhu, Xiaojiang Du, Shen Su, Yanbin Sun, and Nadra Guizani. Real-time lateral movement detection based on evidence reasoning network for edge computing environment. *IEEE Transactions on Industrial Informatics*, Vol. 15, No. 7, pp. 4285–4294, 2019.

[13] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. Advanced persistent threats: Behind the scenes. In *2016 Annual Conference on Information Science and Systems*, CISS '16, pp. 181–186, 2016.

[14] CAPE Sandbox. CAPE: Malware Configuration And Payload Extraction. https://github.com/kevoreilly/CAPEv2.

[15] 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*, S&P '17, pp. 1009–1024, 2017.

[16] Jialong Zhang, Zhongshu Gu, Jiyong Jang, Dhilung Kirat, Marc Stoecklin, Xiaokui Shu, and Heqing Huang. Scarecrow: Deactivating evasive malware via its own evasive logic. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN '20, pp. 76–87, 2020.

[17] Md Sajidul Islam Sajid, Jinpeng Wei, Basel Abdeen, Ehab Al-Shaer, Md Mazharul Islam, Walter Diong, and Latifur Khan. Soda: A system for cyber deception orchestration and automation. In *Annual Computer Security Applications Conference*, ACSAC '21, pp. 675–689, 2021.

[18] Dhilung Kirat, Giovanni Vigna, and Christopher Kruegel. Barecloud: Bare-metal analysis-based evasive malware detection. In *23rd USENIX Conference on Security Symposium*, SEC '14, pp. 287–301, 2014.

[19] Recorded Future. Targeting of Olympic Games IT Infrastructure Remains Unattributed. https://www.recordedfuture.com/olympic-destroyer-malware.

[20] 津田侑, 遠峰隆史, 金谷延幸, 牧田大祐, 丑丸逸人, 神宮真人, 高野祐輝, 安田真悟, 三浦良介, 太田悟史, 宮地利幸, 神園雅紀, 衛藤将史, 井上大介, 中尾康二. サイバー攻撃誘引基盤 stardust. コンピュータセキュリティシンポジウム 2017, CSS '17, pp. 472–479, 2017.

[21] Muhammad Aminu Ahmad, Steve Woodhead, and Diane Gan. The v-network testbed for malware analysis. In *2016 International Conference on Advanced Communication Control and Computing Technologies*, ICACCCT '16, pp. 629–635, 2016.

[22] Grant Ho, Mayank Dhiman, Devdatta Akhawe, Vern Paxson, Stefan Savage, Geoffrey M. Voelker, and David Wagner. Hopper: Modeling and detecting lateral movement. In *30th USENIX Conference on Security Symposium*, SEC '21, pp. 3093–3110, 2021.