Vol.2020-CSEC-91 No.19
Vol.2020-SPT-40 No.19
Vol.2020-EIP-90 No.19
2020/11/26

Linuxゲスト向けCuckoo Sandboxへのファイル保存機能の実現

原田 隆成[1,a)]　鄭 俊俊[1]　毛利 公一[1]

概要：マルウェアの動的解析において，マルウェアによってダウンロードされたり書き込まれたファイル，および削除されたファイルを取得することで一連の攻撃の流れを解析することができるようになる．攻撃対象としては，Windows だけでなく，Linux を標的としたマルウェアも重要であるが，Linux マルウェアの動的解析システムにおいて，十分な解析能力を有し，かつファイルの保存機能を備えたものは存在しない．そこで我々は，広く使われているオープンソースの動的解析システムである Cuckoo Sandbox の Linux ゲストに対し，既に実装されているカーネル空間でのシステムコールトレースを拡張する形で新たにファイル保存機能を実装した．これにより，Linux マルウェアがダウンロードや書き込みを行ったファイル，永続化のために書き換えられた設定ファイル，攻撃の証拠隠滅のために削除されたファイルを取得できることを確認した．また，HiddenWasp を用いた動作検証により実際のマルウェア解析における提案手法の有効性についても確認した．

キーワード：Linux マルウェア，Cuckoo Sandbox，動的解析，ファイル

Implementation of file saving mechanism for Linux guests of Cuckoo Sandbox

RYUSEI HARADA[1,a)]　JUNJUN ZHENG[1]　KOICHI MOURI[1]

Abstract: In the dynamic analysis of malware, it is possible to analyze a series of attacks by obtaining the files that are downloaded, written or deleted by the malware. Like Windows, Linux also becomes a common attack target of malware so that the analysis of Linux malware is important. However, the existing dynamic analysis systems for Linux malware have no sufficient analysis capability and in particular a file saving function. Therefore, in this paper, we have implemented a new file saving mechanism for Linux guests of Cuckoo Sandbox, which is a widely used open-source dynamic analysis system, by extending the already-implemented system call trace in the kernel space. We confirmed that it is possible to obtain the files that were downloaded or written by Linux malware, the configuration files that were modified to persist, and the files that were deleted to destroy evidence of the attack. In addition, the effectiveness of the proposed approach in actual malware analysis was validated through operational verification with HiddenWasp.

Keywords: Linux malware, Cuckoo Sandbox, Dynamic analysis, File

1. はじめに

今日のクラウドサービス，IoT 機器，Docker に代表されるコンテナを活用した開発手法などの普及により，Linux を採用したシステムが増加傾向にある [1]．そのような中で，Linux ホストは攻撃のターゲットとして注目されており，日々多くのマルウェアが開発されている [2]．マルウェアへの対策技術の確立のためには，マルウェアの動作を解析し，明らかにする必要がある．マルウェアの解析手法は，マルウェアを逆アセンブルして解析する静的解析と，マルウェアをサンドボックスなどで動作させ，その挙動を観測する動的解析に大別される．本論文では，増加し続ける多くのマルウェアを効率的に解析する方式として有効な，動的解析に焦点を当てる．

[1] 立命館大学
Ritsumeikan University
[a)] rharada@asl.cs.ritsumei.ac.jp

Vol.2020-CSEC-91 No.19
Vol.2020-SPT-40 No.19
Vol.2020-EIP-90 No.19
2020/11/26

広く用いられている既存のマルウェア動的解析システムに，オープンソースソフトウェア（OSS）として開発が続けられている Cuckoo Sandbox[3] がある．Cuckoo Sandbox は，Windows をはじめとして，Linux，macOS，Android のマルウェアの動的解析のためのサンドボックスである．VirtualBox，VMWare，KVM などで構築されたサンドボックス内でマルウェアを実行，API トレースやシステムコールトレースを行うほか，マルウェアによる外部との通信のパケットログや，Web ベースのユーザインタフェースから解析作業を行うことができる．また，API による操作が可能なため自動化も可能である．そのため，オンラインのマルウェア解析サービスに採用されたり，マルウェアのデータセットの作成に用いられるなど，マルウェア解析者の中で広く使用されている．

しかし，Windows ゲストと Linux ゲストでは実装されている機能に差異があり，具体的には，マルウェアが生成するファイル（ドロップファイル）の保存や，マルウェアによって削除されるファイルの保存機能が Linux ゲストには実装されていない．マルウェアの中には，アンチウイルスソフトウェアからの検知を回避するため，ドロッパにより，ターゲットとなるホストにマルウェアファイルやその設定ファイルをダウンロードしたり，ドロッパ自身からマルウェアファイルを生成し，実行することがある．さらに，既存のコマンドを置き換えるマルウェアや，Linux マルウェア特有の動作として，永続化のために init や cron にマルウェアを実行するよう設定するものが存在する [4]．また，攻撃後にログファイルを削除することで証拠隠滅を図るケースもある．このような，マルウェアによって削除や書き込みが行われたファイルの取得が可能になることで，攻撃に関するより詳細な情報が取得できるようになる．

以上の背景から我々は，Linux ゲスト向け Cuckoo Sandbox に対し，動的解析においてマルウェアによって書き込みや削除が行われるファイルを保存する仕組みを実装する．また，Cuckoo Sandbox の特徴として，Windows マルウェアの解析をはじめとして広く使用されていることと，OSS であることが挙げられる．そこで実装にあたっては，Windows と Linux において解析操作を統一するため Windows ゲストの実装に近づけることを目標とした．

以下，本論文では，2 章で関連研究について述べ，3 章で各ゲストにおける動的解析と Windows ゲストの手法の Linux ゲストへの適用について述べる．4 章ではファイル保存機能について述べ，5 章で評価について述べたのち，6 章でまとめる．

2. 各ゲストにおける動的解析と Windows ゲストの手法の Linux ゲストへの適用

2.1 Cuckoo Sandbox における動的解析

ホスト OS の Cuckoo Sandbox（以下，Cuckoo ホストとする）によって検体ファイルに対応するゲスト OS が起動されると，Cuckoo ホストはゲスト OS に対応した Analyzer とよばれる動的解析プログラムとマルウェア検体を送信する．ゲスト OS では，Analyzer によりマルウェアを実行，動的解析結果を収集し，Cuckoo ホストに送信する．Analyzer の実装は解析するマルウェア検体の OS によって異なる．

2.1.1 Windows ゲストにおける Analyzer とファイル保存

Windows では，API フックのための DLL である monitor[5] を DLL インジェクションによりマルウェア検体に侵入させる．フックでは API トレースの情報として API 名と引数，返り値を取得し，逐次 Analyzer に送信する．また，API トレースとは別に，TransactNamedPipe[6] による monitor と Analyzer が連携した同期的なフック処理が行われる．

TransactNamedPipe は，名前付きパイプでの送信と受信をペアとして扱う Windows API である．TransactNamedPipe を用いて名前付きパイプに対しに書き込みを行うと，パイプの先の通信相手のプロセスによりそのパイプに書き込みが行われるまで待つ．monitor では，TransactNamedPipe をラップしたヘルパー関数が実装されている．これを利用して，monitor がフックポイント内でコマンド文字列を Analyzer に送信することで，同期的にフック処理を行う．同期的処理が行われるケースは，ファイル書き込み時の Analyzer への通知，ファイルの削除における保存処理，子プロセス生成時の DLL インジェクション，プロセスの終了におけるメモリダンプなどが挙げられる．

ファイル操作における TransactNamedPipe を用いた同期処理が行われる Windows API を表 1 に示す．例として，ファイル削除 API（FILE_DEL コマンド群）が実行されたときの Analyzer と連携したフック処理を図 1 と以下に示す．

(1) ファイル削除 API が実行される前にフックする．
(2) 関数を呼び出したプロセスの PID とコマンド，ファイルパスを TransactNamedPipe 関数を用いて名前付きパイプを経由して Analyzer に送信する．
(3) Analyzer が名前付きパイプからコマンド文字列を読み出す．
(4) ハンドラ関数により，ファイルの保存処理を行う．
(5) ハンドラ関数の実行が完了すると，名前付きパイプに”OK”を書き込む．
(6) 名前付きパイプに値が書き込まれると，TransactNamedPipe 関数は処理を終了する．
(7) フック処理を終了し，API を実行する．

ファイル書き込み API（FILE_NEW コマンド群）が実行されたときは，削除の場合とは異なり，API の実行後にフック処理が行われる．monitor は関数を呼び出したプロセスの PID とコマンド，ファイルパスを TransactNamedPipe 関数を用いて名前付きパイプを経由して Analyzer に送信し，

Vol.2020-CSEC-91 No.19
Vol.2020-SPT-40 No.19
Vol.2020-EIP-90 No.19
2020/11/26

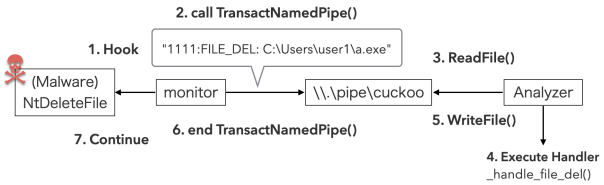

図 1 Windows ゲストにおけるファイル削除時の保存処理

表 1 ファイル操作に関連する フック対象 Windows API

Windows API	送信されるコマンド
URLDownloadToFileW, NtWriteFile	FILE_NEW
NtDeleteFile, DeleteFileW, NtSetInformationFile, MoveFileWithProgressW	FILE_DEL
NtSetInformationFile, MoveFileWithProgressW	FILE_MOVE

Analyzer のハンドラ関数では，送られてきたファイルパスをリストに保持する．ファイル書き込み時のフックで記録されたファイルパスは，マルウェアの実行終了後に一括で保存処理される．リスト内のファイルがマルウェアの実行中に削除された場合は，削除時の保存処理が行われたのち，リストからそのファイルパスを削除する．また，ファイル移動 API（FILE_MOVE コマンド群）が実行された場合は，ファイル書き込み API のときの処理と同じく API 実行後にフックし，ハンドラ関数ではリストで保持しているファイルパスを移動先のパスに置き換える．

2.1.2 Linux ゲストにおける Analyzer

Linux ゲストでは，SystemTap[7] によるシステムコールトレースが行われる．マルウェアプロセスが呼び出したシステムコールのプロローグポイントのプローブでシステムコール名と引数を，エピローグポイントのプローブで返り値をそれぞれフックすることにより取得する．Windows のような Analyzer と連携したフック処理は実装されていない．

2.2 Windows ゲストの手法の Linux ゲストへの適用

Windows ゲストにおける TransactNamedPipe 関数を用いることによる Analyzer と連携したフック処理は，マルウェアのフック処理がユーザ空間で完結するため，同期的な処理を行うことができる．一方で Linux で同期的な処理を実現する場合，SystemTap によるフックがカーネル空間で行われることから，カーネルと Analyzer の間で同期を取る必要がある．しかし，SystemTap のバックエンドである kprobe が割り込み無効なコンテキストで動作するためことから，待機ができないため，同期的な処理は不可能である．そのため，同期的な方式でフックを行う場合はユーザ空間で行う必要がある．

ユーザ空間のフックとして考えられるものとして，環境変数 LD_PRELOAD を変更することによる libc 関数のフックが挙げられる．ファイル操作に関する libc 関数に，本来の libc 関数の実行前後でコマンドを Analyzer に送信する処理を追加した共有ライブラリをマルウェア実行時に LD_PRELOAD に指定する．システムコールのトレースは既存の SystemTap のものを使用することでシステムコールトレースと libc 関数のフックが両立できる．この方式は Windows ゲストにおける DLL インジェクションに近い方式のため，モジュールの再利用性が高まるメリットがある．しかし，Linux マルウェアにはスタティックリンクされたバイナリが多い．その場合，LD_PRELOAD で指定された libc 関数を経由しないため，フックができないという問題が発生する．また，LD_PRELOAD を検出するマルウェアも存在することが示されている [4].

他のユーザ空間のフック手法として，ptrace によるフックが考えられる．ptrace はプロセスのトレースとデバッグを行うシステムコールであり，プロセスのメモリ，レジスタの値の書き換え，システムコールのトレースなどのデバッグに関する機能を有する．この方式では，システムコール実行前後で停止する動作モードである PTRACE_SYSCALL を用い，マルウェア実行時に ptrace をアタッチする．システムコールの実行前後の停止時にレジスタにアクセスすることでシステムコールの引数や返り値を取得してから実行を再開する．これに加え，ファイル操作に関するシステムコールの場合はシステムコールの引数や返り値を取得を取得後，Analyzer にコマンドを送信する処理を実行する．この方式もシステムコールの実行前後に処理を挟み込む点では Windows ゲストに近いため，モジュールの再利用が容易となるメリットがある．しかし，Linux マルウェアのアンチデバッグ手法として ptrace がアタッチされていることの検出方法があることが示されており，実際に ptrace の検出を行うマルウェアが存在している [4]．1 つのプロセスにアタッチできる ptrace は 1 つだけという制約があるため，マルウェア自身が ptrace を呼び出すことで自身に ptrace がアタッチされているかを判定することができる．

以上の問題から，Linux ゲストにおけるユーザ空間でのフックは同期的な処理が行いやすいというメリットがあるが，一方でマルウェアの持つアンチデバッグ機能によりフックに失敗する可能性が存在する．そのため，フック処理はカーネル空間で行うことが適している．具体的には，SystemTap によるフックを用いる，詳細は 3 章で述べる．

3. Cuckoo Sandbox の Linux ゲストに対するファイル保存機能の実現

本章では，Cuckoo Sandbox の Linux ゲストに対してファイル保存機能を実現するにあたり，Windows ゲストでフックされるファイル操作に関する Windwos API を参考に Linux ゲストでフックするシステムコールの検討を行ったのち，提案手法である Analyzer と連携したファイル保存機能につ

Vol.2020-CSEC-91 No.19
Vol.2020-SPT-40 No.19
Vol.2020-EIP-90 No.19
2020/11/26

表 2　フック対象 Windows API に対応するシステムコール

Windows API	システムコール	コマンド
URLDownloadToFileW	該当なし	-
NtWriteFile	write, writev	FILE_NEW
NtDeleteFile, DeleteFileW, NtSetInformationFile, MoveFileWithProgressW	unlink, unlinkat	FILE_DEL
NtSetInformationFile, MoveFileWithProgressW	rename, renameat, renameat2	FILE_MOVE

いて述べる．

3.1　フックする必要のあるシステムコール

Linux ゲストにおいてフックする必要のあるシステムコールは，表 1 の Windows ゲストでフックされる Windows API と等価な動作をするシステムコールである．表 2 に該当するシステムコールを示す．

URLDownloadToFileW は，URL が示すデータをファイルとして保存する API であるが，Linux には等価な動作をするシステムコールは存在しない．

NtWriteFile はファイルに書き込みを行う API であり，Linux においてファイルへの書き込みに用いられるシステムコールは write，writev である．monitor における NtWriteFile のフックでは，通常ファイルへの書き込みのみをフックの対象としている．そのため，write，writev のフックの際も同じく通常ファイルに対する書き込みのみをフックする．

NtDeleteFile，DeleteFileW はファイルを削除する Windows API であり，Linux においてファイルを削除するシステムコールは unlink, unlinkat の 2 つである．また，ファイル操作に汎用的に用いられる Windows API として，MoveFileWithProgressW と NtSetInformationFile がある．MoveFileWithProgressW はファイルを移動する Windows API であるが，MOVEFILE_DELAY_UNTIL_REBOOT フラグが指定され，かつ移動先のファイル名（lpNewFileName）が NULL の場合は再起動後にファイルが削除される．そのため，monitor ではこの条件に合致した場合にファイル削除としてフックしている．NtSetInformationFile はファイルに関する情報を変更するための Windows API である．渡されるオプションに応じて振る舞いを変えるが，monitor ではファイルの削除（FileDispositionInformation）とファイル名変更（FileRenameInformation）のオプションが渡された場合にそれぞれフックしている．

NtSetInformationFile と MoveFileWithProgressW は，前述のファイル削除の他にファイル移動にも使用される API である．Linux においてファイルの移動は rename，renameat，renameat2 で行われることから，これらをフックする．

3.2　ファイル削除における非同期的なファイル保存処理

ファイルの削除を行うシステムコールが呼び出されたとき は，ファイルが削除される前，つまりシステムコールが実行される前にファイルの保存処理を行う必要があるが，割り込み不可能コンテキストで実行される SystemTap のプローブハンドラの中でファイルの読み出しと書き込みが発生する保存処理はできない．そこで，非同期的なファイル保存処理を提案する．SystemTap によりファイルを削除するシステムコールが実行される前に短時間で処理が完了するファイルのリネームにより削除を回避し，次にそれにより失敗するシステムコールの返り値をシステムコール実行後に書き換えることで，あたかも成功したかのようにマルウェアに見せかける．図 2 にファイル削除時の Analyzer と連携した保存処理について示し，以下でその流れについて述べる．（1）～（4）は SystemTap により行われる処理で，（5′）～（7′）が Analyzer により行われる処理である．

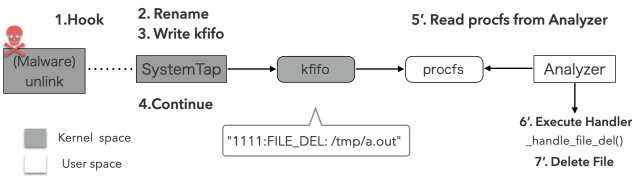

図 2　提案手法におけるファイル削除時の保存処理

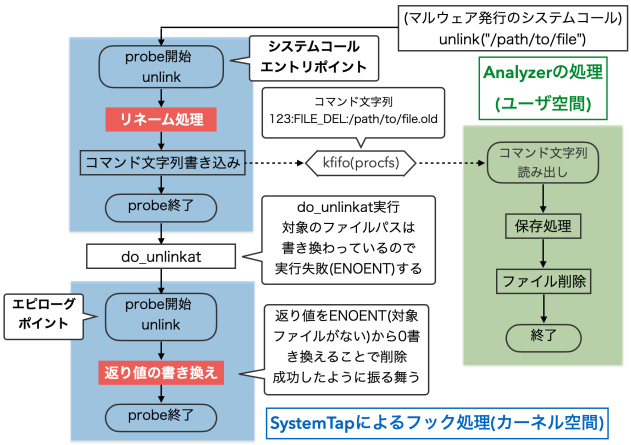

図 3　ファイル削除時のリネーム処理と Analyzer との連携

（1）　ファイルの削除を行うシステムコールが実行される前にフックする．
（2）　削除されるファイルをリネームする．
（3）　システムコールを呼び出したプロセスの PID とコマンド (FILE_DEL)，ファイルパスを kfifo に書き込む．
（4）　システムコールを再開する．

（5′）　Analyzer が proc ファイルを介して kfifo に書き込まれたコマンド文字列を読み出す．
（6′）　ハンドラ関数により，ファイルの保存処理が行われる．
（7′）　保存処理が完了すると，リネームされたファイルを削除する．

これらの処理の詳細について，以降で述べる．

3.2.0.1 削除対象ファイルのリネーム処理

削除対象のファイルのリネーム処理は，図 3 の左上部分に示すとおり，ファイルを削除するシステムコールのエントリポイントに設置したプローブハンドラの中で行う．リネーム処理は，削除ファイルのファイル名を，サフィックスを付与したファイル名に変更する．リネーム処理の実装については，rename システムコールの処理を一部移植することで実現している．ファイルのリネームはディスク I/O が発生せず，極めて短い時間で処理が完了することから，割り込み不可能コンテキストで実行するのに適切である．リネーム処理が完了すると，kfifo に対してコマンド (FILE_DEL) とシステムコールを実行したプロセスの PID，ファイルパスを文字列として書き込む．その後，システムコールを再開する．

3.2.0.2 返り値の書き換え

返り値の書き換えは，図 3 の左下部分に示すとおり，ファイル削除を行うシステムコールのエピローグポイントに設置したプローブハンドラの中で行う．削除対象ファイルのリネームが行われたあとに実行される unlink(at) システムコールの実際の処理である do_unlinkat 関数の返り値は，対象のファイルが存在しないことによる-ENOENT エラーになっている．そのため，SystemTap によりエラーとなっている返り値を 0 に書き換えることで，ユーザ側には削除できたかのように見せる．

3.2.0.3 kfifo と proc ファイルシステムによる Analyzer への通知

SystemTap のプローブハンドラは割り込み無効な状態で実行されるため，プローブハンドラと Analyzer の連携においてスリープやロックが発生する可能性のある方式は使用できない．そこで，書き込みがロックフリーな kfifo（Kernel FIFO）[8] を通信手段として用いる．kfifo は，カーネル内で FIFO（リングバッファ）を扱うための API で，数値，文字列，構造体など任意のデータ構造を FIFO で扱うことができる．また，proc ファイルシステムの file_operations 構造体の read メソッドで kfifo から値を読み出すように実装することで proc ファイルシステムを介してユーザ空間から kfifo にアクセスできることから，カーネルとユーザ空間の通信手段として扱うことができる．提案手法では，Analyzer が proc ファイルを read することで，kfifo に書き込まれたコマンド文字列を読み出している．

3.2.0.4 保存処理

Analyzer は proc ファイルを読み出すことで，コマンド文字列を受け取る．コマンド文字列に含まれるファイルパスは，リネーム後のパスとなっているので，Analyzer はそのファイルを読み出して，ファイルパスやファイルを削除したプロセスの PID とともに Cuckoo ホストに送信する．以上の保存処理が完了すると，リネームされたファイルを Analyzer 自身が削除する．

3.3 ファイルの書き込みと移動に対する実装

ファイルへの書き込みと移動のフックは，当該システムコールの実行完了後（エピローグポイント）で行い，システムコールを呼び出したプロセスの PID と書き込まれたファイルのパスを含むコマンド文字列を，kfifo に書き込む．Analyzer は proc ファイルを介して書き込まれたコマンド文字列を読み出し，書き込みが行われたファイルのパスと PID をリストとして保持する．また，マルウェア実行途中でリストに含まれるファイルが移動された場合はリストのパスを移動後のパスに更新し，移動を行ったプロセスの PID をリストに追加する．マルウェアの実行終了後に，リスト内のファイルパスからファイルを順に読み出し，ファイルパスや PID とともに Cuckoo ホストに送信する．

4. 評価

4.1 機能検証

機能検証として，提案手法を実装した Linux ゲストの Cuckoo Sandbox において，表 2 に示すフック対象システムコールをすべて呼び出すサンプルプログラムを実行することで，提案手法が正しく動作することを確認する．

4.1.1 サンプルプログラムの概要

図 4 に示す，実行可能ファイルや設定ファイルのダウンロードと永続化処理，証拠隠滅を行うマルウェアを模したシェルスクリプトを動作させ，ファイルが保存されることを確認する．シェルスクリプトは，実際に運用していたハニーポットで取得したコインマイナー XMRig によるマイニングを行うマルウェアの初期展開ファイルを参考に記述している．（ハッシュ値: 7d0ba2ba4ed8357e3c23f2a1847192ac73a3306bc237ccc0ed4f40c7efd02061）また，シェルスクリプト部分と，図 5 に示すスクリプト中でダウンロードする実行可能ファイル（a.out）によりフック対象システムコールを網羅する．シェルスクリプトのコマンドでは実行されない writev，renameat，renameat2 システムコールを実行可能ファイルの中で呼び出している．また，実行はオリジナルの Cuckoo Sandbox の Linux ゲストと同様に root ユーザで行う．テストに用いる模擬マルウェアを以下の図に示す．検証に用いた環境は表 3 の通りである．

4.1.2 検証方法

予めホスト OS 上において Web サーバを動作させ，シェルスクリプト中でダウンロードされる a.out と config.json を配置する．次に Cuckoo Sandbox の Web ユーザインタフェース上から，マルウェアを模したシェルスクリプトの解析を行う．シェルスクリプトの実行が終了後，Cuckoo Sandbox から出力されるログファイルである report.json の”dropped”に，a.out，cron 設定ファイル，test3.txt，syslog，config.json の 5 つのファイルの情報が存在することを確認する．

4.1.3 検証結果

図 6 に，模擬マルウェアを実行した際に Cuckoo Sand-

```
#!/bin/sh
wget http://192.168.56.1/upload/a.out -O /tmp/a.
  out
curl -s http://192.168.56.1/upload/config.json >
  config.json
echo "1 */1 * * * wget http://192.168.56.1/upload
  /a.out -O /tmp/a.out; chmod +x /tmp/a.out; sh /
  tmp/a.out" | crontab -
chmod +x /tmp/a.out
/tmp/a.out
rm -f /tmp/a.out
rm -fr /var/log/syslog
```

図 4 マルウェアを模したシェルスクリプト

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <sys/uio.h>
#include <linux/fs.h>
#include <fcntl.h>
#include <sys/syscall.h>

int main(void){
  int fd;
  fd = syscall(SYS_openat, AT_FDCWD, "test.txt",
    O_WRONLY|O_CREAT|O_TRUNC, 0666);

  char *str0 = "Cuckoo";
  char *str1 = "Sandbox";
  struct iovec iov[2];

  // test for writev
  iov[0].iov_base = str0;
  iov[0].iov_len = strlen(str0);
  iov[1].iov_base = str1;
  iov[1].iov_len = strlen(str1);
  syscall(SYS_writev, fd, iov, 2);
  syscall(SYS_close, fd);
  // test for renameat, renameat2
  syscall(SYS_renameat, AT_FDCWD, "test.txt",
    AT_FDCWD, "test2.txt");
  syscall(SYS_renameat2, AT_FDCWD, "test2.txt",
    AT_FDCWD, "test3.txt", RENAME_NOREPLACE);
  syscall(SYS_unlink, "test3.txt");

  return 0;
}
```

図 5 模擬マルウェア内で実行される a.out プログラム

box から出力された解析結果ファイル（report.json）に含まれる，”dropped”のログを示す．4.1.2 項に示した 5 つのファイルが記録されていることが確認できる．また，それぞれファイルの操作元プロセスを示す pids キーを見ると，/home/cuckoo/test3.txt は PID6957 に，/tmp/a.out は PID6959 に，/var/log/syslog は PID6961 に，root ユーザの cron 設定ファイルは PID6918 に，config.json は PID6880 よって操作されたことが記録されていることから，システムコールフックで取得した PID とファイルパスが正常に Analyzer から読み出されたのち，ハンドラ関数によって保存処理が行えていることが確認できる．これにより，実装したすべてのシステムコールフックが動作していると言える．Linux ゲストの Analyzer のファイル保存機能は Windows ゲストで実装されているモジュールを再利用しつつ実装したので，出力される dropped ログも Windows ゲストと同様であり，Cuckoo Sandbox の Web ユーザインタフェースの解析結果ページから Dropped Files の閲覧も同様に行えることを確認した．

表 3 検証環境

ホスト OS	Ubuntu18.04(Linux4.15)
物理メモリ	16GB
仮想マシンモニタ	VirtualBox
ゲスト OS	Ubuntu18.04(Linux4.15)
仮想 CPU コア数	2
割当てメモリ	2GB
ネットワーク	ホストオンリーアダプタ

表 4 検体として使用する HiddenWasp

サイズ	767.3KB
タイプ	ELF 64-bit LSB executable, x86-64 dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.9, with debug_info, not stripped
MD5	898c087702899394ebf5bf2d7c94e195

4.2 実マルウェアを用いた動作検証

次に，実際の Linux マルウェアを用いた動作検証を行う．マルウェア検体の選定にあたっては，レポートが公開されており，ファイル操作を行うことが判明しているマルウェア検体を選定した．今回，検証に用いたマルウェアは HiddenWasp で，ファイルの詳細は表 4 に示す．検証に用いた環境は表 3 の通りである．この検体が接続するサーバは停止していたため，インターネットに接続しない状態で動作させた．

4.2.1 HiddenWasp の動作

HiddenWasp の挙動について，レポート [9] をもとに説明する．HiddenWasp はトロイの木馬型マルウェアであり，Linux ホストに対する標的型攻撃に用いられる．マルウェアは初期展開スクリプト，ユーザモードルートキット，トロイの木馬で構成される．初期展開スクリプトでは，ルートキットやトロイの木馬などを含む圧縮ファイルをダウンロードしたり，ルートキットのパスを LD_PRELOAD に追加，その他動作に必要な環境変数を設定する．また，トロイの木馬の実行や/etc/rc.local への動作の永続化の設定を

```
"dropped":[
  {"yara":[],"name":"c81db75b0ead9a9a_syslog","filepath":"/var/log/syslog","type":"ASCIItext","crc32
    ":"6A748279","path":"/home/user/.cuckoo/storage/analyses/347/files/c81db75b0ead9a9a_syslog","ssdeep
    ":null,"size":30157,"pids":[6961],"md5":"267bafd44abc1720ce6c64a94f70d9f8"},
  {"yara":[],"name":"79a265d02b8b62f9_a.out","filepath":"/tmp/a.out","type":"ELF64-bitLSBsharedobject,
    x86-64,version1(SYSV),dynamicallylinked,interpreter/lib64/ld-linux-x86-64.so.2,forGNU/Linux3.2.0,
    BuildID[sha1]=421d8ca04d22f7737dd4b8ce59deee72a54f41db,notstripped","urls":[],"crc32":"2BDEE10A","
    path":"/home/user/.cuckoo/storage/analyses/347/files/79a265d02b8b62f9_a.out","ssdeep":null,"size
    ":8400,"pids":[6959],"md5":"ab5acc40d11b5913b789e122e0d45054"},
  {"yara":[],"name":"fc7dd29d5648033e_test3.txt","filepath":"/tmpNQhI0z/test3.txt","type":"ASCIItext,
    withnolineterminators","crc32":"6DC1A95A","path":"/home/user/.cuckoo/storage/analyses/347/files/
    fc7dd29d5648033e_test3.txt","ssdeep":null,"size":13,"pids":[6957],"md5":"
    a6f678a8d14b01f762e904aa7f82b887"},
  {"yara":[],"name":"6f4b6aec4216f5a1_config.json","filepath":"/tmpNQhI0z/config.json","type":"
    ASCIItext","crc32":"85CD9B66","path":"/home/user/.cuckoo/storage/analyses/347/files/6
    f4b6aec4216f5a1_config.json","ssdeep":null,"size":26,"pids":[6880],"md5":"7
    be402d425707ba4f08882e14b2904c2"},
  {"yara":[],"name":"d36e931628414b0e_root","filepath":"/var/spool/cron/crontabs/root","type":"
    ASCIItext","crc32":"A4400FF0","path":"/home/user/.cuckoo/storage/analyses/347/files/
    d36e931628414b0e_root","ssdeep":null,"size":253,"pids":[6918],"md5":"2
    be037aeb1e2f5f57c41a75653dcbb87"}
],
```


図 6 模擬マルウェアの解析結果ファイルの Dropped Files のログ（抜粋）

行う．

ルートキットは共有ライブラリとしてインストールされ，いくつかのライブラリ関数をフックすることで HiddenWasp を隠蔽したり，マルウェア内で使用する暗号化された文字列のデコード機能を有する．特に，fopen 関数のフックでは/proc/net/tcp のオープンを監視し，C2 サーバとの接続を隠蔽する挙動をとることが示されている．

トロイの木馬は，C2 サーバと通信し，サーバのリクエストに応じて処理を行う．また，root で実行されている場合のみ，初期展開スクリプトでダウンロードしたルートキットのパスを書き込んだファイル/sbin/.ifup-local を作成し，動的リンカのバイナリに含まれる/etc/ld.so.preload という文字列を/sbin/.ifup-local に置換する．

4.2.2 HiddenWasp の動作確認結果

図 7 にマルウェアを実行した際に Cuckoo Sandbox から出力された解析結果ファイル（report.json）の，保存処理が行われたファイルの情報を示す．ルートキットの共有ライブラリのパスが記述された.ifup-local ファイル，バイナリに含まれる文字列が置換された共有ライブラリである/lib/x86_64-linux-gnu/ld-2.27.so, 6830 という文字列が書き込まれた/tmp/691d90704...（中略）...aa89292b.0 ファイルが確認できた．.ifup-local の中身を確認すると，/lib/se1inux.so というパスが書き込まれいたが，そのパスにファイルは存在しなかった．今回実行した検体はトロイの木馬のみであったため，本来であれば初期展開スクリプトによりこのパスに se1inux.so ファイルが設置されていたものと考えられる．また，共有ライブラリファイル/lib/x86_64-linux-gnu/ld-2.27.so は，一旦は/lib/x86_64-linux-gnu/ld-2.27.so.tmp として書き込まれたのち，rename システムコールで/lib/x86_64-linux-gnu/ld-2.27.so に書き換えられていたことをシステムコールトレースログから確認した．pids キーには 2 つの PID が含まれるが，PID6808 は/lib/x86_64-linux-gnu/ld-2.27.so.tmp に書き込みを行い，PID6817 によりによりファイル名が変更されたことが示されている．.ifup-local ファイルに関しても PID が 2 つ含まれるが，異なる 2 つのプロセスが同一の/lib/se1inux.so という文字列を書き込んでいたことをシステムコールトレースログから確認した．以上により，レポートに示されていたトロイの木馬のファイルへの書き込みを網羅することができていたことから，実マルウェアに対しての一定の有用性を確認できたといえる．

5. 関連研究

5.1 Limon Sandbox

Limon Sandbox[10] は，2015 年の BlackHat Europe で発表された，オープンソースの Linux マルウェアの動的解析システムである．Limon Sandbox では，VMWare 上のゲスト OS でマルウェアを動作させ，strings や yara ルール，VirusTotal による表層解析を行う．動的解析では，Sysdig によるシステムコールトレースと，Volatility によるメモリダンプ，tcpdump によるパケットキャプチャを実行する．ファイル操作に関しては，Sysdig によるシステムコールトレースで，ファイルパスと書き込み内容の一部，削除されるファイルパスを取得する．しかし，削除や書き込みが行われるファイルそのものを保存する機能はない．

```
"dropped":[
  {"yara":[],"name":"fc4ab2c819a06f01_.ifup-local","filepath":"/sbin/.ifup-local","type":"ASCIItext,
    withnolineterminators","crc32":"8CEF59DF","path":"/home/user/.cuckoo/storage/analyses/345/files/
    fc4ab2c819a06f01_.ifup-local","ssdeep":null,"size":18,"pids":[6795,6830],"md5":"00
    fa7a096c00ae633c6e3a9a3b302905"},
  {"yara":[],"sha1":"c611e945a2d346b4c7d73379514bf80d706591b4","name":"
    fc421db97eaa6741_691d907043bcea63f9974681e1feabb1bc8c781a1a007c4dc3c1af1faa89292b.0",
    "filepath":"/tmp/691d907043bcea63f9974681e1feabb1bc8c781a1a007c4dc3c1af1faa89292b.0","type":"
    ASCIItext","crc32":"C462EE51","path":"/home/user/.cuckoo/storage/analyses/345/files/
    fc421db97eaa6741_691d907043bcea63f9974681e1feabb1bc8c781a1a007c4dc3c1af1faa89292b.0","ssdeep":null
    ,"size":5,"pids":[6830],"md5":"dc70820624202bf9a1c7e56ee30aaff0"},
  {"yara":[{"meta":{"description":"ContainsanembeddedPE32file","author":"nex"},"name":"embedded_pe","
    offsets":{"b":[[139068,0]]},"strings":["VGhpcyBwcm9ncmFt"]}],"name":"0ebf64a0d409395a_ld-2.27.so",
    "filepath":"/lib/x86_64-linux-gnu/ld-2.27.so","type":"ELF64-bitLSBsharedobject,x86-64,version1(SYSV
    ),dynamicallylinked,BuildID[sha1]=64df1b961228382fe18684249ed800ab1dceaad4,stripped","crc32":"368
    E2F05","path":"/home/user/.cuckoo/storage/analyses/345/files/0ebf64a0d409395a_ld-2.27.so","ssdeep":
    null,"size":170960,"pids":[6808,6817],"md5":"316a33fc8d17de1605fb37eda8c40375"}
],
```


図 7 HiddenWasp の解析結果ファイルの Dropped Files のログ（抜粋）

5.2 哈勃分析系統 (HaboMalHunter)

HaboMalHunter[11] は，中国 Tencent 社によって開発が続けられているオープンソースの Linux マルウェア動的解析ツールである．ハッシュ値，ELF ファイル，yara ルール，strings コマンドなどによる表層解析，ELF 形式のマルウェアの動的解析をサポートしている．Virtualbox 上で Python スクリプトから strace をアタッチした ELF マルウェアを動作させ，そのログを出力する．strace の結果を解析することによりファイルへの操作を検出するが，ファイルの保存は行うことができない．

6. おわりに

本論文では，Cuckoo Sandbox の Linux ゲストにおけるファイル保存機能の実現について述べた．Linux マルウェアの耐解析機能を考慮し，カーネル空間でシステムコールをフックすることと，ファイル削除における対象のファイルのリネームと返り値の書き換えにより，非同期にファイルの保存を行う．疑似マルウェアを使用した動作確認では，ファイルの書き込み，移動，削除すべてにおいてテストケースを満たすことができ，Cuckoo ホスト側でも保存処理されたファイルを確認できた．また，HiddenWasp を用いた動作確認では，マルウェアによって書き換えられた共有ライブラリや，書き込まれた設定ファイルを取得できたことで，実マルウェアの解析における有効性を示すことができた．

しかし，現在の Linux ゲスト向け Cuckoo Sandbox では，実行したマルウェアプロセスのみをトレースする仕様となっているため，マルウェアプロセスが子プロセスを生成後に wait せず終了し，子プロセスが孤児プロセスとなるケースにおいてトレースができなくなるという問題がある．そのため今後は，本論文で実装した SystemTap と Analyzer の通信方式を応用し，孤児プロセスの追跡に対応できるよう改良する．

参考文献

[1] IDC Japan: 国内サーバーオペレーティングシステム市場予測を発表, https://www.idcjapan.co.jp/Press/Current/20180821Apr.html (2018).
[2] カスペルスキー公式ブログ：Linux を標的とするサイバー脅威, https://blog.kaspersky.co.jp/threats-targeting-linux/29222/ (2020).
[3] Cuckoo Foundation: Cuckoo Sandbox, https://cuckoosandbox.org/ (2019).
[4] Cozzi, E., Graziano, M., Fratantonio, Y. and Balzarotti, D.: Understanding Linux Malware, *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 161–175 (2018).
[5] Cuckoo Foundation: monitor, https://github.com/cuckoosandbox/monitor (2019).
[6] Microsoft: TransactNamedPipe function (named-pipeapi.h), https://docs.microsoft.com/en-us/windows/win32/api/namedpipeapi/nf-namedpipeapi-transactnamedpipe (2020).
[7] SystemTap: SystemTap, https://sourceware.org/systemtap/ (2019).
[8] kernel.org: Chapter 6. FIFO Buffer, https://www.kernel.org/doc/htmldocs/kernel-api/kfifo.html (2020).
[9] Intezer: HiddenWasp Malware Stings Targeted Linux Systems, https://www.intezer.com/blog/linux/hiddenwasp-malware-targeting-linux-systems/ (2020).
[10] monnappa22: Limon, https://github.com/monnappa22/Limon (2019).
[11] Tencent: HaboMalHunter, https://github.com/Tencent/HaboMalHunter (2019).