Computer Security Symposium 2013
21 - 23 October 2013

APIコール間のデータ依存関係を利用したマルウェア通信内容の特定

川古谷裕平† 塩治榮太朗† 岩村誠† 針生剛男†

†NTT セキュアプラットフォーム研究所
180-8585 東京都武蔵野市緑町 3-9-11
{kawakoya.yuhei,shioji.eitaro,iwamura.makoto,hariu.takeo}@lab.ntt.co.jp

あらまし マルウェアの通信内容を把握するため、事前に定義されたパターンをトラフィックから検知する手法が利用されている。しかし、この手法は通信内容が難読化されると適用することが難しい。そこで、本論文ではマルウェアの通信データに関わる端末内の挙動を抽出し、送受信されているデータの内容を特定する手法を提案する。本提案手法をマルウェアによく利用される遠隔管理ツールと MWS2013 データセットを利用して評価した。本提案手法を利用することで、マルウェア通信先の悪性判定等を実現することが可能である。

Identifying the Contents of Malware Communication Using Data Dependency Between API Calls

Yuhei Kawakoya† Eitaro Shioji† Makoto Iwamura† Takeo Hariu†

†NTT Secure Platform Laboratories
3-9-11 Midori-cho Musashino-shi Tokyo 180-8585, JAPAN
{kawakoya.yuhei,shioji.eitaro,iwamura.makoto,hariu.takeo}@lab.ntt.co.jp

Abstract Detecting specific patterns in traffic based on pre-defined signatures is a common technique to identify the contents of malware communication. However, it is difficult to apply this technique to encrypted or obfuscated traffic. In this paper, we propose a technique to identify the contents of sent or received data from in-host behaviors related to these data. We evaluated our technique with a remote administrative tool and MWS2013 dataset. Using our technique, it becomes possible to automatically detect malicious communication.

1 はじめに

標的型攻撃などの高度な攻撃への対策として、外部からの攻撃を防御する従来のセキュリティ対策に加え、自管理ネットワーク内でのマルウェア感染端末を特徴的なトラフィックや通信先から見つけるセキュリティ対策が注目を浴びている。その中でもマルウェアの通信先情報（URL、ドメイン、IP アドレス）は、様々なセキュリティ対策製品で利用されており、マルウェア感染端末を検出するための重要な情報源となっている。

このマルウェアの通信先情報を取得するため、仮想または実インターネットとマルウェアを通信させ、そのトラフィックや通信先を分析するシステムが提案されている [1]。

しかし、マルウェアが通信する相手は必ずしも悪性な通信先のみとは限らない。マルウェアの中には、接続先確認やサービス妨害攻撃のために正規の URL や IP アドレスと通信するものもある。このような場合、マルウェアの各通信の内容を正確に把握し、その通信の意図を理解

した上で、悪性な通信先のみをブラックリスト化する必要がある。

この通信内容の把握のために、通信トラフィックと予め用意したシグネチャとのパターンマッチングが行われている。例えば、Snort[8] のシグネチャにはマルウェアの実行ファイルを意味する特定のバイト列や文字列が含まれている。これらのパターンがマルウェア解析中のトラフィックに出現した場合、その通信内容にマルウェアの実行ファイルが含まれると判断し、その通信先をマルウェア配布サーバだと決定することができる。

しかし、このようなトラフィックに現れるパターンをシグネチャに基づいて検知する方法は通信内容の難読化には脆弱である。例えば、一旦通信データが暗号化されてしまうと、意図しているバイト列、文字列が出現しているかをネットワークから検知するのは非常に困難になってしまう。

2 既存研究とその問題点

悪性な通信先をネットワークからではなく、端末内の挙動から判定しようとする試みに関して、いくつかの既存研究が存在している。ここでは、それらの研究とその問題点について述べる。

Jacob らは、マルウェアが C&C サーバと通信する際の特徴的な Windows のシステムサービスコール (以後、システムコール) のパターンを学習し、マルウェア動作中にそのパターンが出現した場合、その際の通信先を C&C サーバとして検知する手法を提案している [3]。このパターンを抽出するため、システムコール間のデータ依存関係を各呼び出しの引数に対するテイント解析を実施することで構築している。

この手法で、システムコール間の依存関係を構築する際、テイントタグの伝搬が切れてしまうと依存関係の構築に失敗する。Cavallaro らの論文 [2] でも報告されているように、既存のテイント解析の伝搬ルールは暗号化等の難読化処理に脆弱であるため、仮にある 2 つのシステムコールの間に暗号化などの処理を行うユーザランドの API（以後、単に API）が呼び出された場合、テイントタグの伝搬が途切れ、うまく依存関係を構築できずにパターンを見逃す可能性がある。また、システムコールは通常 API を経由して間接的に呼び出される。API からシステムコールを呼び出す過程で、API に渡される一部の引数情報が失われることがあり、マルウェアが送受信している詳細な情報を見逃す可能性がある。

Park らは、C&C サーバからボットが受信するコマンドのセマンティックスを解析するため API の呼び出し順序に着目している [6]。

この手法では、API の呼び出し順序のみに着目し、API 間の依存関係には着目していない。そのため、実際に送受信されているデータと、呼び出された API の関係性を正確に構築するのは難しい。例えば、データ送信の直前にマシン名を取得する API が呼ばれていたとしても、その取得したマシン名が送信されているかは、API の呼び出し順序だけからは断定できない。

3 提案手法

3.1 目的と対象範囲

本論文の目的は、マルウェアが外部と通信する際の送受信データに関わる端末内の挙動を抽出し、そのデータの内容を特定する手法を提案することである。また、端末内での送受信データの処理の過程で暗号化などの難読化処理が行われた場合でも、その影響を受けずに内容特定を行える手法も提案する。

本論文では、マルウェアの外部サーバとの送受信データの内容特定までを対象範囲とし、その特定した内容を利用した通信先の悪性度判定などは対象外とする。

3.2 手法概要

本提案手法は、テイント解析を用いてマルウェアが呼び出す API 間にデータ依存関係を構築し、送受信に関わる API と依存関係にある API を特定することでデータ内容を特定する。具体的には、ある API コールにより新たに生成されるデータに対して、その API コールを一意に特定できるテイントタグ（API-Call Tag:ACT）を設定する。また同時に API が呼び出される毎にその API に渡されるデータに設定されている

ACT を確認する。ある一定期間動作させた後、送信されたデータに設定されている ACT を辿ることで、どの API により生成されたデータが外部へ送信されたのかを特定し、その API に設定されている仕様情報によりデータの内容を特定する。また、受信されたデータにおいても、受信されたデータに ACT を設定し、各 API の呼び出しを監視することで、受信したデータをどの API、引数が利用したのかを特定し、そのデータの内容を特定する。

3.3 API に着目する効果

Jacob らの研究 [3] ではシステムコール間のデータ依存関係に着目しているのに対し、本論文では API 間のデータ依存関係に着目している。API に着目する効果は次の 2 点ある。

1. API を利用した暗号化等の難読化によるテイントタグの伝搬失敗を回避できる
2. より詳細な挙動情報を取得できる

1 に関して、システムコールの監視だけではテイントタグの伝搬が途切れる可能性のある API（暗号化などのデータ変換系 API）の呼び出しを検知することは難しい。その結果、これらの API 実行によりテイントタグの伝搬が途切れる可能性がある。一方、API レベルで監視していれば、これらの実行を捉えることができ、テイントの伝搬切れの問題に対処することができる。

2 に関して、システムコールは通常 API を介して呼び出される。API からシステムコールを呼び出す過程で一部の引数情報が失われたり、そもそもシステムコールを呼び出さない API も存在している。そのため、API レベルで監視することでより詳細なマルウェアの挙動情報を取得することができ、それらの情報を使ってより多くのマルウェアが利用するデータの内容を特定することができる。

3.4 API 間の依存関係の構築

図 1 を用いて、API 間の依存関係の構築について説明する。

3.4.1 テイント伝搬に基づく依存関係

図 1 の (a) に関して、マルウェアが GetComputerNameA を呼び出した場合、lpBuffer 引数にはマシン名が保存される。本提案手法では、この lpBuffer に格納されたマシン名の文字列に対して、この GetComputerNameA の呼び出しを一意に特定できるテイントタグ (0xdeadbeef) を設定する。次に、このマシン名に対して XOR を実行し、マシン名の見た目上のバイト列が変更された場合を考える。ここでは、定数とテイント付データの XOR 処理の場合、テイント付データのタグを結果値に反映させる伝搬ルールを設定しているとする。すると、この XOR の実行結果に 0xdeadbeef のタグが伝搬する。そして、XOR されたマシン名が send に渡される際、buf に渡されたデータのテイントタグを確認すると、0xdeadbeef が設定されている。このテイントタグを設定したのは、GetComputerNameA であるので、この send と GetComputerNameA 間に依存関係を構築することができる。それにより、この send で送信されるデータにはマシン名が含まれていることがわかる。

3.4.2 特定 API に基づく依存関係

Windows API の中には、その API の実行中にテイントタグの伝搬が途切れてしまうものがいくつか存在している。そのような API をここでは特定 API として定義し、この API が呼び出された場合、その入力値と出力値に無条件に依存関係を構築する。

図 1 の (b) でその動作を説明する。(b) は (a) の場合と同様にマシン名が外部に送信される処理コードである。GetComputerNameA と send の間に特定 API である CryptEncrypt が実行されている。この場合、CryptEncrypt 内で pbData が指すマシン名に設定されていたテイントタグが消失するため、CryptEncrypt が書き込んだ pbData が指すデータに対しては新たなテイントタグ (0xbaadfood) が設定される。その結果、GetComputerNameA と CryptEncrypt、CryptEncrypt と send といった二つの異なる依存関係が構築され、send で送信されたデータにマシン名が含まれていることを特定することができない。

そこで、本提案手法では、特定 API に関しては無条件に入力されたデータと出力されたデータに依存関係を構築する。CryptEncrypt を場

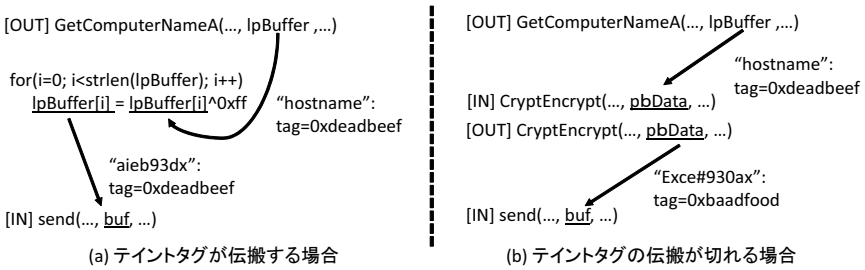

図 1: API 間の依存関係の構築

合を例にとると、CryptEncrypt を呼び出した時に渡される pbData の値と、CryptEncrypt から復帰する時に pbData が指している値に依存関係を構築する。その結果として、GetComputerNameA と CryptEncrypt と send の間に依存関係を構築でき、マシン名が外部に送信されたことを特定できる。

3.5 提案システムの動作

本提案手法を実装したシステム（以下、提案システム）は、動的解析ステップとデータ特定ステップの 2 つのステップにより動作する。動的解析ステップでは、API モニタとテイント解析を行える動的解析環境上でマルウェアを一定期間動作させ、外部のサーバと通信を行わせる。その間、マルウェアから呼び出される各 API に関して、その呼び出し時に渡される引数とそこに設定されているテイントタグの値、各 API からマルウェアへの復帰時に API によって書き込まれる値とその値に対して設定したテイントタグの値、をログとして出力する。

次にデータ特定ステップでは、外部通信を行う API を起点として、ログに記載されているテイントタグの値を元にして、ログの探索を行い外部と通信しているデータに関して、API の依存関係を辿ることで、送信されているデータを生成した API と引数の特定、受信したデータを利用している API と引数を特定する。

4 提案システムの実装

4.1 API Chaser

本提案システムでは、マルウェアを動的解析するための基盤として、著者らが開発した API Chaser[5] を利用する。API Chaser は Argos(Qemu) を基に開発した仮想マシンモニタベースのマルウェア動的解析システムであり、ゲスト OS 内で動作するマルウェアが実行する API を正確に捕えることができる。また、API Chaser はシステム全体を対象としたデータフロー解析のためのテイント解析機能を実装しており、メモリ、ディスク上への監視対象データの移動を追跡することができる。

4.2 API フックによるテイント処理

本提案システムでは、マルウェアから呼び出される API の引数に対してテイントタグを設定、確認するために API Chaser の API フック機構を利用している。具体的にはマルウェアがある API を呼び出した場合、API を呼び出した時と API から復帰する時の 2 回実行をフックする。API を呼び出した時には、その API に渡される引数のテイントタグを確認し、ログに出力する。また、実行が API からマルウェアに復帰する時には、その API 内で書き込まれた引数データを確認し、その API 呼び出しを一意に特定できるテイントタグを設定する。各 API の引数の型や入出力に関する情報は Windows の SDK 内に含まれるヘッダファイルから取得した。

ここで、API Chaser の API フックの仕組みを説明する。API Chaser ではマルウェアから呼び出される API を正確に監視するため、監視対象のプログラムコード部分にテイントタグを設定し、仮想 CPU でフェッチした命令に当該タグが設定されている否かで監視対象コードとするかを決定するコードテイント手法 [5][10] を応用した API フックを行っている。3 種類のテイントタグ（マルウェアタグ、正規タグ、API タグ）を用意し、監視対象のマルウェアとそのマルウェアが書き込みを行った領域に対して、監

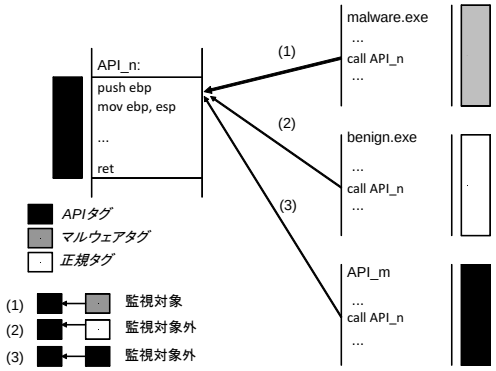

図 2: 3 種のテイントタグによる API フック

視対象を示すマルウェアタグを設定する。また、Windows のシステムライブラリ内の各種 API に関しては、API タグを設定する。さらに、上記以外のゲスト OS 内にデフォルトで存在する正規プログラムのコード部分には正規タグを設定する。

図 2 のようにマルウェアタグを持ったコードから API タグが呼び出された場合を監視対象 API コールとし、その他の場合を監視対象外とする。これにより、マルウェアのプログラムコード部分から呼び出される API のみを監視対象とすることができ、API が他の API を呼び出す場合や監視対象以外の正規プログラムが API を呼び出す場合などを除外することができる。

4.3 ACT:API-Call Tag

提案システムでは、API の各呼び出し毎に新たに生成されるデータに対して、その API 呼び出しを一意に特定できるテイントタグ、API-Call Tag(ACT) を設定する。この ACT や、API Chaser が利用する複数種類のテイントタグを扱うため提案システムでは図 3 のようなテイントタグフォーマットを利用している。提案システムでは、システム上の 1 バイトあたり 4 バイトのテイントタグ保存領域（シャドウメモリ）を用意している。この 4 バイトの使い方として、テイントなしを除くと、即値型とポインタ型が存在する。

即値型の場合は、タグの 4 バイト自体に各データの識別子の値が格納され、ポインタ型の場合は、データの識別子、関連情報を格納したデータ構造体へのポインタが格納される。先

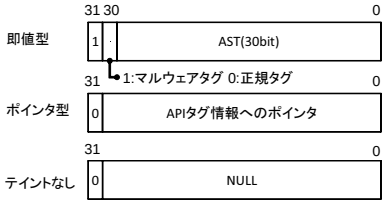

図 3: 提案システムのテイントタグフォーマット

頭 1 ビット目で即値型とポインタ型の区別を行う。即値型の場合、2 ビット目でマルウェアタグか、正規タグかの区別を行う。残り 30 ビットを利用して ACT を表現する。そのため、$2^{30} = 1,073,741,824$ 個分の API が出力する値に関して区別することが可能である。ポインタ型は、API タグの情報を格納したデータ構造体へのポインタが格納される。

4.4 ACT の探索

データ特定ステップでは、動的解析で生成したログを探索することで、API 間の依存関係を構築する。データ送受信系の API と依存関係を持つ API は、データ内容を特定できるかの観点で次の 3 種類に分類できる。

1. その API とその引数の型、名前からデータの内容を特定できるもの
2. その API と関連する他の API の情報と組み合わせることで、データの内容や関連情報を特定できるもの
3. それだけではデータの内容を特定できないもの（主にデータを変換する役割を担うもの）

タイプ 1 に関して、送信データの場合は、GetComputerName や GetVersionEx などがある。これらの API は情報をシステムから取得するものであり、これらの API の特定の引数と依存関係を持っていることがわかれば、すぐに送信データの内容を特定することができる。また、受信データの場合は、connect の name 引数 (sockaddr 構造体）などがある。受信したデータが、connect の name 引数として渡されていた場合、受信したデータの内容は接続先を示すデータだと特定することができる。

タイプ2に関して、送信データの場合は、ReadFileやRegQueryValueExなどがある。例えば、ReadFileの場合は、ReadFileに第一引数として渡されているハンドルのタグを辿ることで、このハンドルを生成したAPI（例えばCreateFileA）を特定し、そこからファイル名を特定することができる。受信データの場合は、WriteFileやRegSetKeyValueExなどの他にCreateProcess、WinExecといったAPIからは、受信したデータが実行ファイルであるといったファイルタイプ情報の特定などを行うことができる。

タイプ3に関しては、送受信データ共に、暗号化を行うCryptEncryptや圧縮、解凍を行うRtlCompressBuffer、RtlDecompressBufferなどがある。3.4.2で説明したように、このようなAPIは予め特定APIとして把握しておき、これらのAPIがACTを辿る過程で現れた場合、これらAPIの入力データと出力データの間で依存関係を無条件に構築し、ACTの探索を引き続き行うことで、これらの特定APIにおけるテイントタグの伝搬切れの問題を回避する。

また、送信データの場合、そのデータの元がAPIではなく、マルウェアの実行ファイル自身の場合や、正規のファイルの中身の場合もある。このような場合は、ACTではなく、マルウェアタグや正規タグから、マルウェア実行ファイル内のデータや正規ファイルのデータが送信されたことのみを特定することができる。

5 実験

5.1 実験1:Poison Ivy

5.1.1 実験概要

この実験では、Poison Ivy(バージョン2.3.2)[7]の実行ファイル（以下PIクライアント)を本提案システム上のゲストOS(Windows XP SP2)内で動作させ、同一ネットワーク上に予め用意しておいたPoison Ivyのサーバ（以下PIサーバ）と通信させる。PIサーバからPIクライアントへ端末情報の取得命令を送り、その結果、PIクライアントからPIサーバに送られ、PIサーバのGUI画面に表示される情報と、提案システムで得られた通信データ生成の挙動とを比較し、提案手法が送信されたデータの大部分を特定できることを示す。

5.1.2 実験結果

表1に、情報取得命令により得られた、PIクライアントが動作している端末情報の一覧を示す。また、提案システムにより、それらのデータ生成に関わったAPIとその引数も示す。図4には、PIクライアントからPIサーバへデータ送信する挙動の一つを示している。複数のAPIにより生成されたデータが、RtlCompressBufferに渡され、sendで送信されている様子を示したものである。

提案システムではRtlCompressBufferを特定APIとして扱っているため、RtlCompressBufferのUnCompressedBuffer引数（入力）とCompressedBuffer引数（出力）に無条件の依存関係を構築している。そのため、表1で示している通り、PIクライアントから送られる情報のほぼすべてに対して、そのデータ生成の挙動の特定に成功している。

表1の中で、Account TypeとProcessorの性能情報に関しては、そのデータ生成の挙動が観測できなかった。Acount Type情報に関しては、PIクライアントの実行中にアカウントの特権を取得するAPIが呼ばれていたが、その情報が直接送られたわけではなく、いったん記号等に変換されたため、テイントタグの伝搬が途切れてしまったものと考えられる。Processorに関しては、rdtscなどのAPIを利用しないでクロック数を計測する命令を用いて性能情報を取得したためだと考えられる。

一方で、誤検知も観測された。図4のGlobalAllocの戻り値（ハンドル型）がRtlCompressBufferに渡されている挙動は誤検知だと考えられる。この誤検知の原因として、RtlCompressBufferのUnCompressedBufferに渡されるデータサイズが実際データサイズよりも大きいかったため、バッファの未使用（未初期化）部分に残っていたGlobalAllocの戻り値が一緒に送られたためだと考えられる。

[1]レジストリキーは SOFTWARE/Microsoft/Windows/CurrentVersion/ProductId

表 1: 実験 1 の結果

取得情報	送信データ取得 API/取得元	
	API	引数
OS	GetVersionExA	lpVersionInfo→dwMajorVersion
Edition	GetVersionExA	lpVersionInfo→dwMinorVersion
Service Pack	GetVersionExA	lpVersionInfo→zCSDVersion
Build	GetVersionExA	lpVersionInfo→dwBuildNumber
Windows ID	ReQueryValueExA[1]	lpData
Logged on User	GetUserNameA	lpBuffer
Account Type	-	-
Computer Name	GetComputerNameA	lpBuffer
WordGroup	NetWkstaGetInfo	servername
System Uptime	GetTickCount	ret
Processor	-	-
RAM	GlobalMemoryStatus	lpBuffer→dwTotalPhys
Install Path	GetModuleFileNameA	lpFileName
Process Mutex	マルウェア実行ファイル	

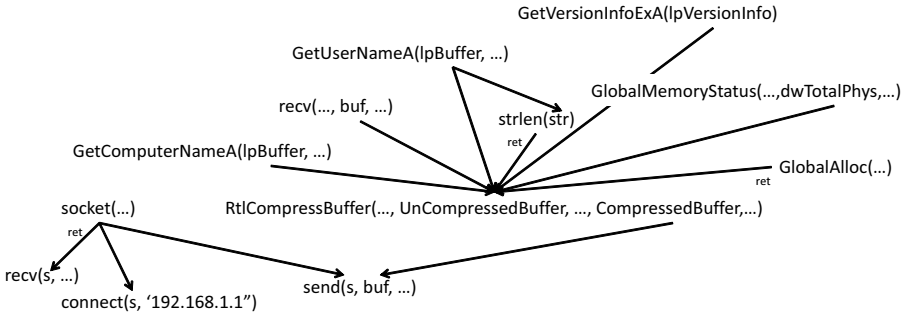

図 4: Poison Ivy のデータ送信に関わる API 挙動

5.2 実験 2:ZeroAccess.C

5.2.1 実験概要

この実験では、提案手法が実際のマルウェアを利用した場合でも有効であることを示すため、MWS2013 データセットの D3M[9] に含まれるマルウェア 16 検体のうち、本論文執筆時点で正常に外部サーバと通信を行った 3 検体（アンチウィルス名は全て ZeroAccess.C）を実インターネットに接続した提案システム上で実行し、送受信データの特定とそのときの通信先の特定を行った。なお、他の 13 検体は、DNS エラーや接続先が存在しなかったため、意味のある通信を行わなかった。

5.2.2 実験結果

11b7 検体が通信を行った相手と、それぞれの送信データ元、受信データ利用 API の情報を表 2 に記載する。jmaxmind.com に対しては、マルウェア実行ファイル内に含まれていたデータを送信した。この返信として受信したデータを利用する API は観測されなかった。download.microsoft.com に対しては、マルウェア実行ファイル内に含まれていたデータを送信し、受信したデータは WriteFile にて一旦ファイル (ファイル名は IE8-WindowsXP-x86-EN.exe) に保存した後、CreateProcessAsUser にて実行された。その際の動作を図 5 に示す。受信したデータは CreateProcessAsUser にて実行されているため、実行ファイルだったことがわかる。つまり、download.microsoft.com は実行ファイルを配布しているサーバだということがこの挙動から読み取れる。

複数の IP アドレスに対してポート 16471/UDP でマルウェア実行ファイル内に含まれていたデータを送信したが、どの通信先からも応答はなかった。

6 考察

ここでは提案手法の制限事項について述べる。

本提案手法では、API による暗号化や圧縮等の難読化処理に関しては、無条件に依存関係を

[2]URL のフルパスは download.microsoft.com/download/C/C/0/CC0BD555-33DD-411E-936B-73AC6F95AE11/IE8-WindowsXP-x86-EN.exe

表 2: 実験 2 の結果

URL/IP アドレス	ポート番号	送信データ取得 API/取得元	受信データ利用 API
jmaxmind.com/app/geoip.js	80	マルウェア実行ファイル	-
download.microsoft.com[2]	80	マルウェア実行ファイル	WriteFile,CreateProcessAsUser
14.97.XXX.20	16471	マルウェア実行ファイル	-
147.46.XXX.200	16471	マルウェア実行ファイル	-
188.233.XXX.200	16471	マルウェア実行ファイル	-
79.253.XX.48	16471	マルウェア実行ファイル	-
…	16471	マルウェア実行ファイル	-

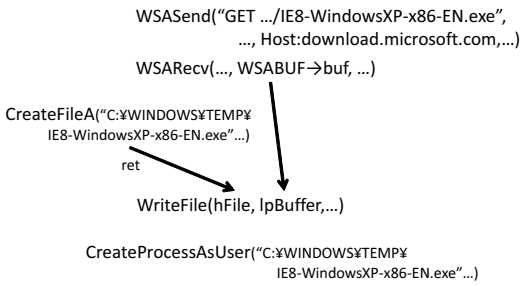

図 5: 11b7 検体の受信データに関わる API 挙動

構築することで対処しているが、静的リンクされた暗号、圧縮ライブラリによる処理が行われた場合は、テイントタグの伝搬が切断されてしまう可能性がある。これらの問題を回避するためには、Kang らの研究 [4] のようなテイントタグの伝搬機構自体を強固にする取り組みが必要である。

本提案手法は、API 毎に定義された仕様に基づき、送受信されたデータの内容特定を行っている。そのため、API を利用しないで生成された送信データや、API を利用しないで処理される受信データに関しては、内容を特定することはできない。これらを行うためには、より詳細な命令単位のテイントタグの設定、例えば rdtsc の戻り値に対してテイントタグを設定するなど、を行うことでより細かい送受信データの内容特定を行える可能性がある。

7 まとめ

本論文では、マルウェアが外部と通信する際の送受信データに関わるマルウェアの端末内の挙動をテイント解析を用いた API 間のデータ依存関係を利用することで抽出し、その送受信されたデータの内容を特定する手法を提案した。

参考文献

[1] Aoki, K., Yagi, T., Iwamura, M. and Itoh, M.: Controlling malware HTTP communications in dynamic analysis system using search engine., *proceedings of 3rd IEEE International Workshop on Cyberspace Safety and Security 2011* (2011).

[2] Cavallaro, L., Saxena, P. and Sekar, R.: On the Limits of Information Flow Techniques for Malware Analysis and Containment, *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08 (2008).

[3] Jacob, G., Hund, R., Kruegel, C. and Holz, T.: JACKSTRAWS: picking command and control connections from bot traffic, *Proceedings of the 20th USENIX conference on Security* (2011).

[4] Kang, M. G., McCamant, S., Poosankam, P. and Song, D.: DTA++: Dynamic Taint Analysis with Targeted Control Flow Propagation, *NDSS* (2011).

[5] Kawakoya, Y., Iwamura, M., Shioji, E. and Hariu, T.: API Chaser:Anti-analysis Resistant Malware Analyzer, *Proceedings of 16th International Symposium on Research in Attack, Intrusions and Defenses* (2013).

[6] Park, Y. H. and Reeves, D. S.: Identification of Bot Commands by Run-Time Execution Monitoring, *ACSAC* (2009).

[7] Poison Ivy: Remote Administration Tool, *http://www.poisonivy-rat.com/*.

[8] Sourcefire Inc: Snort, *http://www.snort.org/*.

[9] 神園雅紀, 他.:マルウェア対策のための研究用データセット ～MWS Datasets 2013～, *MWS* (2013).

[10] 川古谷裕平, 岩村 誠, 針生剛男.:テイント伝搬に基づく解析対象コードの追跡方法, 情報処理学会論文誌, Vol. 54, No. 8, pp. 2079–2089 (2013).