Vol.2021-DPS-186 No.26
Vol.2021-CSEC-92 No.26
2021/3/15

クロック分解能を用いたRaspberry Pi仮想マシンの検出

鈴木 克弥[1]　大山 恵弘[1]

概要：近年の IoT デバイスの普及に伴って，IoT デバイスを攻撃対象とするマルウェアが増加している．マルウェアの挙動を解析することは非常に重要なトピックの 1 つであり，その中でもマルウェアの解析回避処理を解明することは大きな意味を持つ．IoT デバイスで採用されている Arm アーキテクチャは一般的なコンピュータで採用されている x86 アーキテクチャとは異なる特徴を持つ．また，OS も Windows ではなく Linux が採用されることが多い．しかし，解析回避処理を扱う既存研究の多くが x86 アーキテクチャや Windows を対象としている．したがって，Arm アーキテクチャや Linux を対象とするマルウェアの解析回避処理の研究が必要である．これまでに発見された解析回避処理の 1 つに仮想マシンの検出がある．本研究では，Arm アーキテクチャと Linux を採用している代表的な IoT デバイスとして Raspberry Pi を対象とし，実マシンと仮想マシンでクロックの分解能に差があることを利用し，仮想マシンを検出する手法を提案する．また，提案手法に基づいて仮想マシン検出プログラムを作成し，実験の結果，その有効性を確認した．

キーワード：仮想マシン検出，サンドボックス検出，Arm，Raspberry Pi，IoT

Virtual Machine Detection for Raspberry Pi with Clock Resolution

Abstract: In recent years, IoT malware has been increasing along with the widespread use of IoT devices. Analyzing the behavior of malware is one of the most important topics. In particular, we focused on elucidating the analysis evasion process of malware. The Arm architecture used in IoT devices has different characteristics from the x86 architecture used in general computers. However, most of the existing researches dealing with analysis evasion process target the x86 architecture. Therefore, it is necessary to study the analysis evasion process for malware targeting the Arm architecture. The virtual machine detection is one of the analysis evasion processes discovered so far. In this paper, we propose a new method for detecting virtual machines. The method focuses on the difference in clock resolution between real and virtual machines, and targets the Raspberry Pi as a typical machine that uses the Arm architecture. We have implemented a virtual machine detection program based on the proposed method and confirmed its effectiveness through experimental results.

Keywords: Virtual machine detection, Sandbox detection, Arm, Raspberry Pi, IoT

1. はじめに

近年，IoT デバイスが急速に普及している [6]．それに伴って，IoT デバイスを攻撃対象とするマルウェアが多数出現している [1,10,15,16]．広く普及している IoT デバイスの 1 つに，Raspberry Pi がある．Raspberry Pi は主に教育目的で開発されたシングルボードコンピュータである．Raspberry Pi OS [5] という Linux ベースの OS が公式に配布されており，Raspberry Pi を使用する多くのユーザが OS にこれを選択する．Raspberry Pi は OS が Linux ベースであるため扱いやすいことが利点であるが，同時に利用者数が多いのでマルウェアの標的になりやすいという側面を持つ．また，Raspberry Pi をはじめとするいくつかの IoT デバイスは Arm アーキテクチャの CPU を採用しており，従来の一般的なコンピュータで CPU に採用している x86 アーキテクチャとは特性や命令セットなどが大きく異なっている．これまでに x86 アーキテクチャの CPU や Windows を対象としたマルウェアに関しては様々な研究が行われてきた．しかし，既存研究の手法を必ずしもそ

[1] 筑波大学
University of Tsukuba

のまま IoT マルウェアに対して適用できるわけではない．したがって，IoT デバイスを対象とするマルウェアについて新たに研究し，知見を深める必要がある．

マルウェアの解析者はサンドボックス上で悪意のあるプログラムを実行することがある．主な利点として，マルウェアが実行する命令やアクセスするファイル，インターネットの通信先などを特定するのに役立つこと，もしマルウェアに感染した場合でも感染前の状態に復旧することが可能であることが挙げられる．そのため，マルウェアの作者はサンドボックス上でプログラムが実行された場合は直ちに実行を中断するようにしてマルウェアの挙動を隠蔽する場合がある．このように自身の挙動を隠蔽するためにマルウェアが行う処理を解析回避処理 [7] と呼ぶ．解析回避処理には，逆アセンブルされるときに容易に解読できないような命令の構成にしたり，デバッガを使用してリバースエンジニアリングをされないようにすること [8] などが挙げられる．このような解析回避処理の 1 つにサンドボックスの検出がある．マルウェアの解析に用いられるサンドボックスは一般的に仮想マシンの技術が使われているため，サンドボックスの検出は仮想マシンの検出に置き換えて考えることが可能である．したがって，マルウェアが仮想マシンを検出する手法を明らかにすることはマルウェアの挙動を解明する上で非常に有効である．

近年，IoT デバイスを攻撃対象とするマルウェアを解析するためのサンドボックスが利用され始めている．代表的なものに IoTPOT [12] がある．このようなサンドボックスの多くは内部で IoT デバイスの仮想マシンが用いられているため，IoT マルウェアが行う仮想マシン検出手法について理解することは重要である．

本研究では，IoT デバイスを攻撃対象とするマルウェアの解析回避処理の解明を目的として，時間情報を用いた Raspberry Pi 仮想マシンの検出手法を提案する．具体的には，計時デバイスの分解能に Raspberry Pi の実マシンと仮想マシンとの間で差があることに注目し，その違いを利用して仮想マシンを検出プログラムを作成する．

そのプログラムによる検出の精度を評価し，提案手法が仮想マシンを検出するための 1 つの有効な手法になる可能性を示すことで，マルウェアの解析回避処理の解明に貢献する．

対象として Raspberry Pi を選出した理由は 3 つある．1 つ目は Raspberry Pi が最も広く普及している IoT デバイスの 1 つであることである．2 つ目は IoT デバイスで広く普及している Arm アーキテクチャを採用していることである．3 つ目は Raspberry Pi 用に開発されている OS である Raspberry Pi OS が Linux を基にしており，多くの人が利用している点である．

本研究の特徴として，複雑なアルゴリズムを用いずに実装することが可能であること，実行の際に管理者権限が必要ないことなどが挙げられる．

2. 関連研究

Shi らの研究 [14] では，Windows 環境での主要なデバッガの 1 つである WinDbg の拡張機能として *Apate* を作成している．Apate はマルウェアからデバッガの存在を悟られないようにする機能を実装している．アンチデバッガ機能を有しているマルウェアのサンプルの中からランダムに 20 個選択し，様々なデバッガ上で解析しようとしたところ，20 個全てに対して Apate はアンチデバッガ技術を回避することができた．2 番目に精度の良かったデバッガでも 20 個中 9 個のサンプルは自身の挙動を隠蔽しており，また Apate は他のデバッガと同等かそれ以上の性能を示したので，この研究によって Apate の有効性とともに実用性が明らかになった．この研究はマルウェアの解析回避処理の 1 つであるデバッガ検出に注目しており，Windows 環境を対象としている．これに対し本研究は，サンドボックスの検出についての研究であり，対象としている環境は Linux ベースの OS である．

Raffetseder らの研究 [13] では，CPU のバグや CPU モデル特有のレジスタの値を比較したり，マシンごとの性能の違いからエミュレータを検出する手法を提案している．この研究では QEMU と VMware を実マシンと比較しており，OS は Linux を使用している．この研究では本研究と同じ QEMU を使用した環境でのエミュレータ検出を行っているが，エミュレートしているのは x86 アーキテクチャのプロセッサであり，本研究が対象としている Arm アーキテクチャとは異なる．

Colin の研究 [11] では，CPUID 命令と NOP 命令をそれぞれ同じ回数実行したときに要する時間や，同じ時間が経過するまでのそれぞれの実行回数から，実マシン上か仮想マシン上かを識別し，さらに仮想マシンの種別やバージョンまでも区別できることを明らかにした．この研究は様々な仮想マシンモニタを用いているが，本研究で用いている QEMU が対象とされておらず，すべて x86 アーキテクチャの Linux 環境を対象としている．したがって，Arm アーキテクチャを対象とする本研究とは異なる．

宮本らの研究 [18] では，x86 アーキテクチャの命令である RDTSC 命令を用いて仮想マシンモニタを検出する手法を提案している．Arm アーキテクチャの CPU にも RDTSC に相当する命令は存在するが，その命令には管理者権限が必要である．

Olivier の研究 [9] では，サンドボックスである Cuckoo Sandbox と仮想マシンモニタである VirtualBox を検出する方法を明らかにした．それらの解析環境の検出には，Cuckoo Sandbox などのプログラムにハードコーディングされているファイルの名前や実行するプロセスの名前などを使用している．この研究ではそのような資源の特徴や有

無の情報を用いて検出する artifact ベースの手法で検出しているが，ほとんどの技術は Windows を対象としているものである．これに対し本研究は，Linux ベースの OS である Raspberry Pi OS を対象としている．

これまでに Paranoid Fish (Pafish) [4] や Al-Khaser [2], InviZzzible [3] など，デバッガやサンドボックスを検出するツールがいくつか開発されている．しかし，これらは Windows 及び x86 アーキテクチャを対象としており，Raspberry Pi 環境では動作しない．

本研究に先行する研究として，Raspberry Pi 環境のためのサンドボックス検出ツール，Gosanta が開発されている [17]．この研究は Raspberry Pi 仮想マシンを検出するためのいくつかの有効なテスト項目を明らかにしているが，有効であると予想される多くのテスト項目のうちの一部を提案，評価したに過ぎない．したがって，IoT デバイスを対象とする解析回避処理を十分網羅的に解明するためにはさらなる研究が必要である．

3. 予備実験

予備実験として，実マシンと仮想マシンそれぞれにおける時間の流れの違いを可視化することを考える．予備実験のために，時刻を取得してメモリ上の配列に書き込むことだけを 1000 万回繰り返し，その差分を算出するプログラムを実装した．作成したプログラムの時刻の差分値を計算する関数を図 1 に示す．

時刻の取得には C 言語の clock_gettime 関数を使用した．clock_gettime 関数の実体はシステムコールである．clock_gettime 関数は引数として渡す値を変えることでクロックの種類を変更することができる．今回はシステムで一意な実時間を計測する CLOCK_REALTIME を採用した．他に選択可能なクロックとして，任意の時点から計測し，システムが休止している間も計測できる CLOCK_MONOTONIC や高速だが精度の低い CLOCK_REALTIME_COARSE が挙げられる．1000 万回時刻を取得したあと，隣接する値との差分を計算する．得られた差分値を可視化するため，中間の 100 万回分の差分値を抽出してファイルに書き出し，グラフ化した．中間部分の値を使用する理由は，プログラムの実行開始直後の CPU 使用率が安定せず，取得する時刻の誤差が大きくなるからである．

本実験の実験環境を表 1，2 に示す．エミュレータには QEMU を使用した．Raspberry Pi 仮想マシンを構成している CPU は QEMU によってエミュレートされた仮想 CPU の仕様を表記している．

実マシンと仮想マシン上でそれぞれ時刻取得プログラムを実行し，その結果をグラフ化した様子を図 2 に示す．

本実験では，プロセスを特定の CPU コアに張り付けて実行するということはしていない．そのため，検出プログラムを実行する CPU コアが変わると，それに伴う時間が時刻差分値に影響する可能性があると推測される．時刻の差分値のプロットを細かく観察すると，いくつもの層が重なっているように見える．ここで観察される層の幅がグラフの縦軸の目盛幅と一致するようにパラメータを調整してプロットした結果が図 3，4 である．

実マシンの結果のプロットは縦軸が 208〜728 ns の区間のみを拡大し，メモリ幅が 52 ns になるように調整してある．同様に，仮想マシンの方は縦軸が 15〜25 µs の区間のみを拡大し，メモリ幅が 1 µs になるように調整してある．

ここで，描かれた層の幅が一定の値を取っていることに着目する．これに対する考察として，プロットした点が層になって見えるのは clock_gettime 関数の分解能が関係していると考えられる．clock_gettime 関数は分解能の倍数を返すので，それより細かい値を返すことは出来ない．したがって，この層の幅がプログラムを実行したマシン上でのクロックの分解能を表していると考えられる．この分解能は時刻取得命令の実行にかかる CPU サイクル数と強く関係していると考えられる．また，実マシンと仮想マシンとで層の幅に大きな違いが見られた．ここで，この違いを

```
1: timestamps ← []
2: for i = 0, ··· , 9, 999, 999 do
3:     timestamps[i] = clock_gettime()
4: end for
5: diffs ← []
6: OFFSET ← 5, 000, 000
7: for i = 0, ··· , 999, 999 do
8:     diffs[i] = timestamps[i+1+OFFSET] − timestamps[i+
       OFFSET]
9: end for
10: return diffs
```

図 1 時刻を取得して差分値を計算するアルゴリズム

表 1 予備実験で使用する Raspberry Pi の仕様

	実マシン	仮想マシン
CPU	Arm Cortex-A7	Arm1176JZF-S
コア数	4	1
メモリ	1 GB	256 MB
OS	Raspberry Pi OS GNU/Linux 9 (stretch)	
エミュレータ	QEMU 2.11.1	

表 2 予備実験で使用するホストマシンの仕様

CPU	Core i9-9900
コア数	8
メモリ	32 GB
OS	Ubuntu 18.04.3 LTS

Vol.2021-DPS-186 No.26
Vol.2021-CSEC-92 No.26
2021/3/15

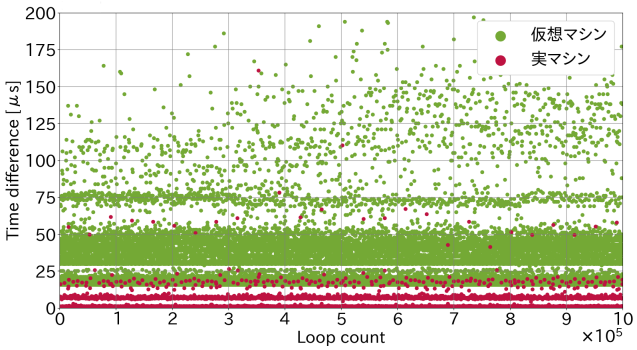

図 2 時刻取得プログラムの実行結果

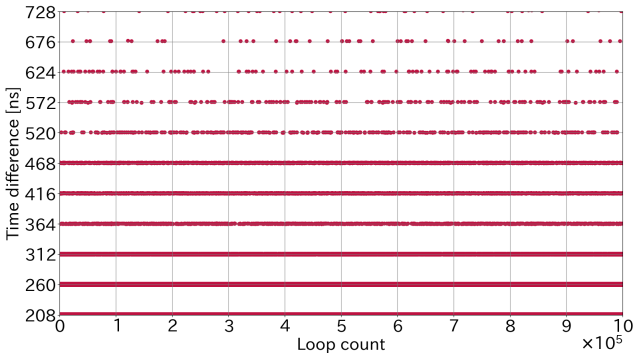

図 3 実マシン上での時刻取得プログラムの実行結果 (拡大図)

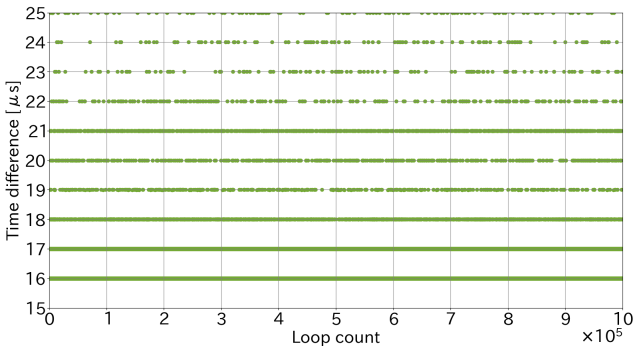

図 4 仮想マシン上での時刻取得プログラムの実行結果 (拡大図)

仮想マシンの検出に適用できるのではないかという仮説が立つ．

4. 提案手法

本研究では，予備実験で立てた仮説に基づき，クロック分解能を計算することによって Raspberry Pi の仮想マシンを検出する手法を提案する．クロック分解能を計算するアルゴリズムを図 5 に示す．

5〜7 行目では，時刻差分値を 10 個取得するために時刻を 11 回連続で取得する．8〜11 行目では，隣り合う時刻との差分値を計算し，差分値をキー，出現回数を値とするような連想配列に代入する．この差分値計算を，最頻区間が 1 つに決まるまで繰り返す．ここで言う最頻区間とは，時刻差分値の最頻値と 2 番目に頻出の値が隣り合っている区間のことである．最頻区間を計算する際に，分解能の小数点以下の誤差を考慮して計算する [19]．その後，最頻区間の時刻差分値同士の差を計算し，これをクロック分解能の実測値とする．最後に，クロック分解能の実測値を理想値と比較し，誤差が 1 ns よりも大きければ仮想マシンであると判定する．

```
1: finished ← false
2: timestamps ← []
3: diffFreq ← {}
4: repeat
5:     for i = 0, ··· , 10 do
6:         timestamps[i] = clock_gettime()
7:     end for
8:     for i = 0, ··· , 9 do
9:         diff = timestamps[i + 1] − timestamps[i]
10:        diffFreq[diff] = diffFreq[diff] + 1
11:    end for
12:    if the most frequent interval can be determined to be one
       then
13:        finished ← true
14:    end if
15: until not finished
16: max ← 0
17: for iter of diffFreq do
18:    sum ← diffFreq[iter] + diffFreq[iter + 1]
19:    if sum ≥ max then
20:        max = sum
21:        resolution = diffFreq[iter + 1] − diffFreq[iter]
22:    end if
23: end for
24: return resolution
```

図 5 クロック分解能を計算するアルゴリズム

クロック分解能の理想値は，事前に実マシン上で `dmesg` コマンドで取得する．理想値は，Raspberry Pi の水晶発振器の周波数の逆数に等しい．仮想マシンかどうかを判定するためのクロック分解能の取得に `dmesg` を使用せず算出する理由は，`dmesg` は Arm のデバイスツリーから情報を取得するからである．デバイスツリーの内容を書き換えることは可能であるため，`dmesg` で偽の情報を表示されてしまう可能性がある．そのため，実際のクロック分解能を計算する必要がある．

5. 主実験

提案手法を実装した仮想マシン検出プログラムを複数の Raspberry Pi 上で実行する．対象マシンとして 2 世代の Raspberry Pi を選択し，それぞれ実マシンと仮想マシンを用意する．エミュレータには QEMU を使用し，コマンドラインオプションとして`-M raspi2` や`-M raspi3` を与えることでハードウェアの仕様を指定した．仮想マシンにおけるクロック分解能の実測値はホストマシンの処理能力に依

表 3 Raspberry Pi 実マシンの仕様

	2B	3B+
CPU (Arm)	Cortex-A7	Cortex-A53
CPU 周波数	900 MHz	1.4 GHz
クロック周波数	19.2 MHz	19.2 MHz
コア数	4	4
メモリ	1 GB	1 GB
OS	Raspberry Pi OS GNU/Linux 10 (buster)	

表 4 Raspberry Pi 仮想マシンの仕様

	RasPi2VM	RasPi3VM
仮想 CPU (Arm)	Cortex-A7	Cortex-A53
クロック周波数	62.5 MHz	62.5 MHz
コア数	4	4
メモリ	1 GB	1 GB
OS	Raspberry Pi OS GNU/Linux 10 (buster)	
エミュレータ	QEMU 5.1.0	

表 5 仮想マシンを動作させるホストマシンの仕様

	HOST1	HOST2
CPU	Core i9-9900	Celeron G4930
CPU 周波数	3.6 GHz	3.2 GHz
コア数	8	2
メモリ	32 GB	8 GB
OS	Ubuntu 18.04.3 LTS	

存する．そのため，予備実験で使用したホストマシンと性能差が出るようにもう 1 台ホストマシンを用意する．本実験の実験環境を表 3〜5 に示す．Raspberry Pi 1, 4 を使用しなかった理由は，本論文執筆時点で QEMU がそれらマシンのエミュレーションに対応していなかったからである．

RasPi2VM, RasPi3VM はそれぞれ Raspberry Pi 2, 3 を仮想化したものと同等とみなす．これらの仮想マシンを HOST1，2 上でそれぞれ実行し，実マシンと合わせて計 6 台の Raspberry Pi マシンを用意する．

用意した 6 台のマシンのそれぞれにおいて仮想マシン検出プログラムを 10000 回実行した結果を表 6 に示す．左の列は実行したマシンを表している．中央の列はプログラムを実行した環境が実マシンか仮想マシンかを正しく判定した割合を表している．右の列はクロック分解能を計算するために必要な時刻差分値の取得回数の平均値を表している．

仮想マシン環境においては 100%の正答率を達成することが出来た．実マシン環境においても高い精度で仮想マシンかどうかを判別することが出来た．また，時刻差分値の取得回数も数十回程度で済むため，少ない計算量で仮想マシンの判定ができることがわかる．今回の実験で誤答した際のクロック分解能の実測値は，ほとんどが理想値の定数倍であった．これは，最頻区間が一意に定まったものの，

表 6 主実験の結果 (通常時)

	正答率	時刻差分値の平均取得回数
2B	0.9894	15
3B+	0.9998	16
RasPi2VM (on HOST1)	1.0000	12
RasPi3VM (on HOST1)	1.0000	19
RasPi2VM (on HOST2)	1.0000	13
RasPi3VM (on HOST2)	1.0000	22

表 7 主実験の結果 (高負荷時)

	正答率	時刻差分値の平均取得回数
2B	0.9933	15
3B+	0.9994	17
RasPi2VM (on HOST1)	1.0000	12
RasPi3VM (on HOST1)	1.0000	19
RasPi2VM (on HOST2)	1.0000	23
RasPi3VM (on HOST2)	1.0000	42

取得した時刻の数が少なかったために正しい計算ができなかったものと考えられる．

さらに，仮想マシンを高負荷にした状態で検出プログラムを実行した．結果を表 7 に示す．仮想マシン検出プログラムを実行するときにマシンのコアの数だけ `yes > /dev/null` というコマンドをバックグラウンドで実行する．これによって，CPU 使用率を限りなく 100%に近い状態にすることができ，マシンに高い負荷がかかった状態を作ることができる．Raspberry Pi を実利用する場合，サーバプログラムやカメラなどのセンサ情報を処理するプロセスが稼働していることが考えられる．したがって，より実用的な場面を想定して仮想マシン検出プログラムを実行することを意味する．

結果より，マシンに高い負荷がかかっている状態でも高精度に仮想マシンの判別ができることがわかった．また，時刻差分値を取得する回数も通常時と比較して大きな変化はなかった．このことから，提案した手法は実用的な場面でも脅威となりうる手法であると考えられる．

6. 議論

本研究で使用した clock_gettime 関数はシステムコールである．一般的に，マルウェア解析に用いられるツールはシステムコールを捕捉する機能を有する．そのため，解析回避処理にシステムコールを使用するのは憚られる．しかし，clock_gettime 関数を含めたいくつかの計時のためのシステムコールは vDSO (*virtual dynamic shared object;* 仮想動的共有オブジェクト) というライブラリで実装されていることが多い．Linux カーネルは設定や状態によっては自動的にこの vDSO をユーザ空間にマッピングするため，clock_gettime 関数を呼ぶ場合は，実際にはカーネル空間に遷移することなく時刻情報を取得することができる．した

がって，vDSO の clock_gettime 関数を呼ぶ場合はシステムコールとしてマルウェア解析ツールに捕捉されることがないため，ステルス性を発揮することができる．1 つ注意すべき点は，Arm における clock_gettime 関数の vDSO 化は，アーキテクチャの実装上 Armv7 以降でのみ対応している．Armv7 は Raspberry Pi 2 系に相当するため，マルウェア解析ツールに対してステルス性を発揮するためには Raspberry Pi 2 系以降で提案手法のプログラムを実行する必要がある．

7. まとめ

本研究では，Raspberry Pi の実マシンと仮想マシンでクロック分解能の値に差があることを利用して Raspberry Pi の仮想マシンを検出する手法を提案した．また，提案手法を実装したプログラムを用いて実験を行い，脅威となりうる手法であることを示した．提案した手法は，複雑なアルゴリズムを必要とせず，管理者権限が必要な命令を使用していないという特徴がある．また，Raspberry Pi 2, 3 系においてはセキュリティツールに捕捉されにくいというステルス性を有している．

今後の課題として，本研究で扱った Raspberry Pi 以外の IoT デバイス（Jetson Nano や Arduino など）を対象とした仮想マシン検出手法であったり，特定のデバイスに依存しない汎用的な手法を研究する必要がある．また，提案手法に対抗する手法として，クロック分解能を利用する解析回避処理を無効化する手法について研究することも重要である．

謝辞 本研究の一部は JSPS 科研費 20K11741 の助成を受けている．

参考文献

[1] : 2019 Internet Security Threat Report, Symantec (online), available from ⟨https://www.symantec.com/content/dam/symantec/docs/reports/istr-24-2019-en.pdf⟩ (accessed 2020-01-20).
[2] : Al-Khaser, https://github.com/LordNoteworthy/al-khaser. 2020-11-28.
[3] : InviZzzible, https://github.com/CheckPointSW/InviZzzible. 2020-11-28.
[4] : Paranoid Fish, https://github.com/a0rtega/pafish. 2020-11-28.
[5] : Raspberry Pi OS, https://www.raspberrypi.org/software/. 2021-02-11.
[6] : 令和元年版情報通信白書，総務省（オンライン），入手先 ⟨http://www.johotsusintokei.soumu.go.jp/whitepaper/whitepaper01.html⟩
[7] Botacin, M., Rocha, V., De Geus, P. and Gregio, A.: Analysis, Anti-Analysis, Anti-Anti-Analysis: An Overview of the Evasive Malware Scenario, *17th Brazilian Symposium in Information Security and Computational Systems (SBSeg 2017)* (2017).
[8] Chen, P., Huygens, C., Desmet, L. and Joosen, W.: Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware, *IFIP International Conference on ICT Systems Security and Privacy Protection*, Springer, pp. 323–336 (2016).
[9] Ferrand, O.: How to detect the Cuckoo Sandbox and to Strengthen it?, *Journal of Computer Virology and Hacking Techniques*, Vol. 11, No. 1, pp. 51–58 (2015).
[10] Mikhail Kuzin, Y. S. and Kuskov, V.: New trends in the world of IoT threats, Kaspersky (online), available from ⟨https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/⟩ (accessed 2020-01-20).
[11] Murray, C. S.: Classifying Virtual Machine Managers by Overhead, *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, IEEE, pp. 77–82 (2016).
[12] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T. and Rossow, C.: IoTPOT: Analysing the Rise of IoT Compromises, *9th USENIX Workshop on Offensive Technologies (WOOT 15)* (2015).
[13] Raffetseder, T., Kruegel, C. and Kirda, E.: Detecting System Emulators, *International Conference on Information Security*, Springer, pp. 1–18 (2007).
[14] Shi, H. and Mirkovic, J.: Hiding Debuggers from Malware with Apate, *Proceedings of the Symposium on Applied Computing*, pp. 1703–1710 (2017).
[15] Vladimir Kuskov, Mikhail Kuzin, Y. S. D. M. and Grachev, I.: Honeypots and the Internet of Things, Kaspersky (online), available from ⟨https://securelist.com/honeypots-and-the-internet-of-things/78751/⟩ (accessed 2020-01-20).
[16] TrendLabs フィリピン：新しい Linux マルウェア、CGI の脆弱性を利用，トレンドマイクロ（オンライン），入手先 ⟨https://blog.trendmicro.co.jp/archives/14577⟩（参照 2020-01-20）．
[17] 大山恵弘：Raspberry Pi 環境におけるステルス性の高い仮想マシン検出，情報処理学会研究報告コンピュータセキュリティ (CSEC)， Vol. 2018, No. 2 (2018).
[18] 宮本久仁男，田中英彦：特徴データベースを用いない効率的な仮想マシンモニタ検出方式の提案，情報処理学会論文誌，Vol. 52, No. 9, pp. 2602–2612 (2011).
[19] 山口健二，中村勝洋：計算機の時刻取得関数に関する性質とその解析法に関する考察，全国大会講演論文集，No. アーキテクチャ, pp. 7–8 (2010).