5-4　悪性 USB デバイスに対する対策技術の研究

竹久達也　岩村　誠　丑丸逸人

USB インターフェースを利用したキーボードやマウスなど、USB デバイスが広く普及している。しかしながら、USB デバイスのファームウェアを書き換えることで、その USB デバイスを接続した PC 等の USB ホスト機器に、ユーザの意図しない不正な動作を起こさせることが可能であることが知られている。悪意を持った第三者が、悪性の USB デバイスを作成し、サイバー攻撃に用いることが現実的になってきており、その対策は急務である。本稿では、悪性 USB デバイスへの対策として、USB デバイスの構成情報を検査することにより、悪性 USB デバイスを検知し遮断する USB ハブを提案する。

1　まえがき

USB デバイスの利用が一般的となって久しい。USB デバイスは、特殊な用途のデバイスでなければ、USB コネクタへ挿すだけで利用できるため利便性が高く、広く普及しており、多くの PC などの装置に USB コネクタが搭載されている。

近年では、USB ファジング技術により、不正な USB コンフィギュレーション情報（構成記述子）を用いて、OS やデバイスドライバ実装の不備を突き、故意にシステムを停止させるような悪性な USB デバイス（以下、悪性 USB デバイス）の作成方法も発表されており [1]、USB デバイスの危険性が高まっている。

また、BadUSB といわれる、悪性 USB デバイスも発表されている [2]。これは、悪意の第三者により、USB デバイスのファームウェアを、悪性 USB デバイスへと書き換えられることで、情報の横取りや改ざん、意図しない動作を引き起こすデバイスである。また、近年普及著しい Android 携帯などが有する、USB ターゲット機能のファームウェアを書き換えることで、BadUSB と同様に、悪意のある動作を引き起こすことが可能であるとも示唆されている [3]。さらに、USB デバイスの製造過程で悪性 USB デバイスとして、マルウェアが混入され出荷されていたという報告もある [4]。

近年では、デジタル複合機など PC 以外にも USB コネクタが搭載されている装置が多く存在し、USB メモリに記録したデータの印刷や、スキャンした画像を USB メモリに記録することができる。また、デジタルカメラなどで撮影した写真データを店頭などで印刷するセルフプリント端末も同様である。これら装置に対しても、悪性 USB デバイスへの対策が求められている。

本来、USB デバイスは、製造者しか書き換えることができないよう対策を講じる必要があるにもかかわらず、ある一定以上のスキルを有する技術者であれば、いくつかの USB デバイスでは、ファームウェアの書き換えができることに問題がある [5]。しかし、目的の USB デバイスを、目的に応じて必要な機能を有するように書き換えるためには高度な知識を必要とする。この事柄は、組込み型ソフトウェアにおけるセキュリティ問題と酷似している。

本稿では、悪性 USB デバイスを以下の 4 種に分類する。

- 悪性組込み型
  悪意のある製造者が、製品本来の機能以外の悪性機能を組込み、悪性 USB デバイスとして機能させるもの。
- 不正記述子型
  USB 構成記述子を不正な記述子とすることで、悪性 USB デバイスとして機能させるもの。
- 機能追加型
  本来デバイスが有する USB 構成記述子を改変し、悪性 USB デバイスとしての機能を追加するもの。
- 機能改変型
  本来デバイスが有する USB 構成記述子を改変せず、悪性 USB デバイスとして機能を追加するもの。

本稿では、悪性 USB デバイスへの対策として、USB ホスト機器（例えば、PC やデジタル複合機、セルフプリント端末など）に対して、悪性 USB デバイスの接続を検知し遮断するための USB ハブを提案する。提案する USB ハブは、USB デバイスを PC などへ接続する前に、USB 構成記述子を検査し、悪性 USB デバイスと検知されれば PC への接続を遮断するデバイスである。

以下、**2** では前提知識を示し、**3** では提案手法を述べ、**4** では考察を述べ、**5** にてまとめる。

2 前提知識

本章では、簡単な USB 仕様の概略及び本稿で特に重要となるコントロール転送と、USB デバイスの保護について記述する。

2.1 USB 仕様とコントロール転送

USB 仕様では、PC など USB デバイスを制御する装置のことをホストと定義し、USB デバイスのことをターゲットと定義している。

図 1 の USB システム構成例に示すように、ホストには、ホスト・コントローラが有り、配下の USB 通信を全体的に管理している。ホスト・コントローラの直下にルート・ハブが存在し、ルート・ハブのポートに、実際の USB デバイスが接続されるツリー型のトポロジを採用している。

ハブは、5 段までカスケード接続が可能で、ハブを接続することで USB ポートを増やすことができ、1 つのホスト・コントローラで、最大 127 台まで USB デバイスを接続することが可能である。

USB は、転送モードとして、コントロール転送、バルク転送、インタラプト転送、アイソクロナス転送が規定されている。

コントロール転送は、USB デバイスのアドレスを設定、USB デバイスの構成情報を取得し、USB デバイスが有する機能を利用するために、デバイスのリクエストと呼ばれる通信を行う。コントロール転送にて行う USB デバイスの簡単な初期化手順を、図 2 に示す。

図 2 に示したように、USB ホストは、どのような USB デバイスが USB コネクタに挿入され、その USB デバイスがどのような機能を有するのかをコントロール転送におけるリクエストを用いて調べる。

リクエストには、標準リクエスト、クラス固有リクエスト、ベンダ固有リクエストの 3 種類のリクエストがあり、図 3 で示すフォーマットとなっている。標準リクエストには、以下で示す GetDescriptor, SetAddress, SetConfiguration などがある。

- GetDescriptor リクエスト
  デバイス、コンフィグレーション、ストリングなどの記述子（ディスクリプタ）を要求することができる。要求する記述子タイプの値は、表 1 のとおりである。USB デバイスは、GetDescriptor リクエストを受け付けると、リクエストの記述子タイプで要求されたリクエストに応じて、表 1 の長さ列で示した長さの記述子タイプごとに決められた記述子を返送する。

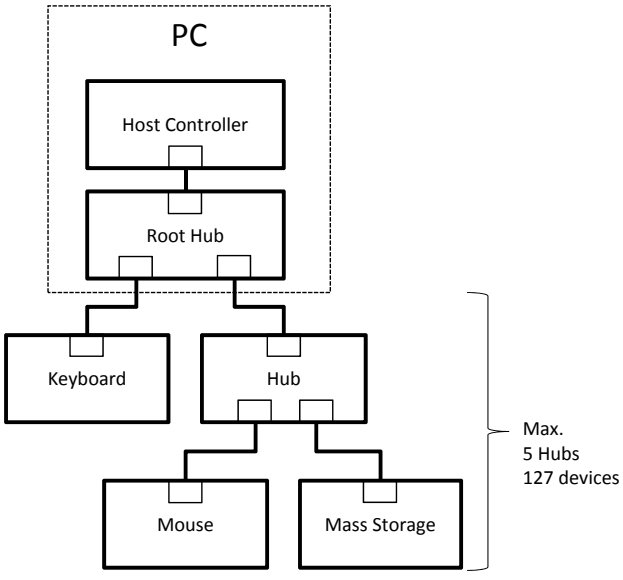


図 1　USB システム構成例

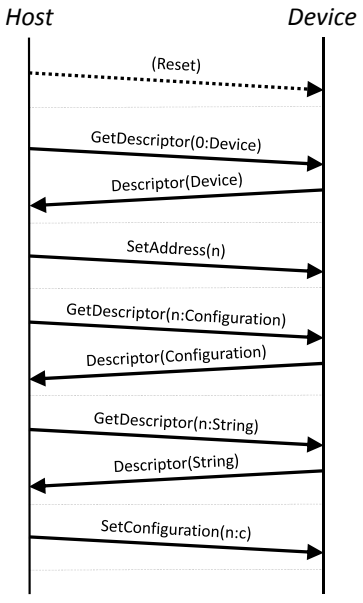


図 2　USB エニュメレーション・シーケンス例

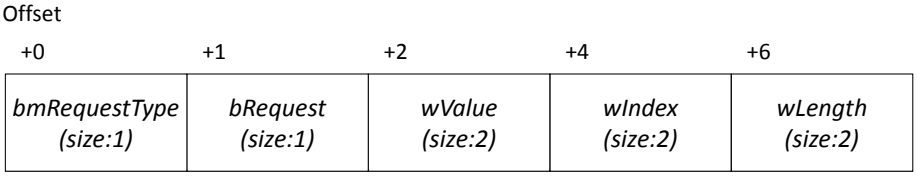


図 3　USB Device Requests フォーマット

表 1　記述子タイプ (抜粋)

記述子タイプ	値	長さ
DEVICE	1	18
CONFIGURATION	2	9
STRING	3	2 +N
INTERFACE	4	9
ENDPOINT	5	3

表 2　USB 各規格の速度

規格	モード	最大転送速度
USB 1.0	LS (LowSpeed) FS (FullSpeed)	1.5 Mbps 12 Mbps
USB 1.1	同上	同上
USB 2.0	HS (HighSpeed)	480 Mbps
USB 3.0	SS (SuperSpeed)	5 Gbps
USB 3.1	SSP (SuperSpeedPlus)	10 Gbps

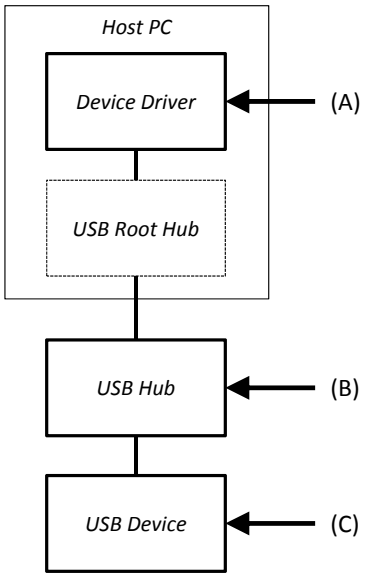


図 4　USB デバイスの検査場所

- SetAddress リクエスト
  USB デバイスの USB バス上のアドレスを設定する。
- SetConfiguration リクエスト
  USB デバイスが有する機能を選択する。これは、USB デバイスが機能を複数有する場合に、USB ホストが利用する最適な機能を選択するために利用される。

USB は、通信速度の向上と機能の追加により仕様が更新されている。表 2 に、各 USB 規格での通信速度を示す。USB は、USB 1.1 以降 USB 3.1 まで上位互換性が保たれている。このため、USB デバイスが接続された際、USB デバイスの情報を取得するためコントロール転送は、まず、互換性のため USB 1.1 規格の速度で通信する。

2.2　USB デバイスの保護

悪性 USB デバイスからシステムを保護するために検査する場所として、図 4 が考えられる。

図 4 (A) では、USB ホストを OS やデバイスドライバにより、悪性 USB デバイスから保護する。この方法では、Bouyat らの手法 [1] のような、デバイスドライバの脆弱性を利用した攻撃方法に対して効果を発揮できない。

図 4 (B) では、USB ハブは、USB デバイスが接続されたタイミングで、USB デバイスが悪性かを判定できるため、(A) のように USB ホスト側の脆弱性に影響しない。また、Tonder らの手法 [6] のように、USB ホストと USB デバイスの間に、透過型の双方向調停デバイスを中間的に構成し、調停デバイスが、USB に対するファジング攻撃をブロックすることも可能である。しかし、Tonder らの手法では、速度面で遅延が発生する。また、透過型の双方向調停デバイスは多くのコンポーネントが必要なため高価となる。

図 4 (C) では、USB デバイス側で、何らかの検知ロジックが必要であり、USB デバイス用 IC や CPU として高機能なデバイスを必要とする。

上記事柄から、悪性 USB デバイスを検査する場所として考えた場合、(A) (C) では、様々な OS やデバイスドライバ、USB デバイスそれぞれに検知ロジックの開発や脆弱性改善の必要がある。

しかし、(B) で検査する場合、前述の (A) (C) の問題を意識する必要が無いため、(B) の USB ハブで検査する方法は、非常に効果的と言える。

本稿では、(B) の場所の特性を利用し、可能な限り USB デバイスの性能を低下させない悪性 USB デバイス対策の手法について提案する。

3　提案手法

本章では、**3.1** で、提案する検査機能付き USB ハブ (以下、提案 USB ハブ) の構成と機能説明を行い、**3.2** で、提案 USB ハブで検査する手法について説明する。

本提案手法は、悪性 USB デバイスを検知するために、USB プロトコルにおけるコントロール転送のみを利用することで、悪性 USB デバイス検査を行う。このことは、**2** で述べたように、USB の上位互換性を利用し、USB デバイスの構成や機能を確認するためのコントロール転送は、USB 1.1 規格のみで可能ある。そのため、検査する USB ハブは、USB 2.0 や USB 3.x の様な高速な通信を必要とせず、安価に実現できる。

3.1　検査機能付き USB ハブ

提案 USB ハブは、図 5 で示すように、検査用 USB ホスト（組込み型 CPU）、検査用 USB ホストと接続するための USB ハブ（USB ハブ A）、本来のホスト（PC など）と接続するための USB ハブ（USB ハブ B）、USB デバイスを接続するための各ポートに対応した、USB スイッチ IC（#A ～ D）と、プッシュボタン（#A ～ D）で、構成される。

提案 USB ハブは、以下に示す 3 つの動作モードがある。

- USB デバイス登録：
  プッシュボタン #n を押しながら、それに対応する USB ポートに USB デバイスを挿入したとき。
- USB デバイス削除：
  プッシュボタン #n を押しながら、それに対応する USB ポートから USB デバイスを抜いたとき。
- USB デバイス認証：
  プッシュボタンを押さずに、USB デバイスを挿入したとき。

USB デバイス登録及び認証時、検査用 USB ホストは、USB デバイスが挿入された USB ポートの USB スイッチを制御し、USB デバイスを USB ハブ A に接続する。

USB デバイスが USB ハブ A に接続されると、USB ハブ A は、検査用 USB ホストにデバイスが接続されたことを通知し、検査用 USB ホストは、そのタイミングで接続された USB デバイスを検査する。

検査が OK の場合、登録時は接続された USB デバイスから取得したデバイス記述子に含まれる VID、PID のペアをキーとして、検査時に取得した記述子の Hash 値を登録デバイス情報として記録する。

その後、検査用 USB ホストは検査が完了した USB ポートの USB スイッチを制御し、USB デバイスを USB ハブ B に接続する。

このようにして、提案 USB ハブは USB デバイスをホストに接続する前に検査を行う。また、検査が NG の場合は、USB デバイスの機能を停止させたままにしておく。

USB デバイス削除時、検査用 USB ホストは、接続されていた USB デバイスに対応する、VID、PID ペアで登録されている記述子情報を抹消する。

このことで、次回 USB デバイス接続時は、再登録しない限りホストへ接続されない。

次節では、提案 USB ハブが行う検査について述べる。

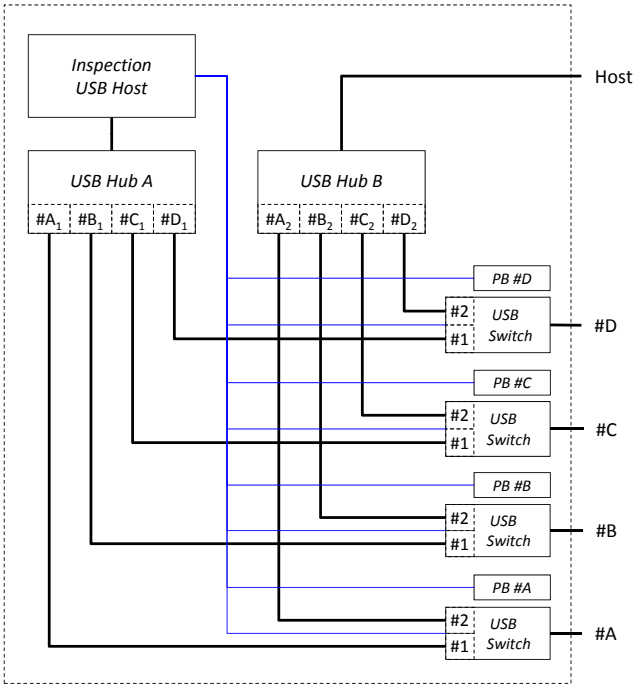


図 5　検査機能付き USB ハブブロック図

3.2　検査手法

本節では、検査用 USB ホストが行う検査について説明する。

検査には、ホワイトリスト検査、ブラックリスト検査、不正記述子検査があり、どの検査においても、USB デバイスの記述子を取得し、その内容を検査する。

各検査の方法を、次節以降にて詳しく述べる。

3.2.1　ホワイトリスト検査

ホワイトリスト検査は、図 6 で示すフローにて検査する。

（A）GetDescriptor リクエストにて、デバイス、コ

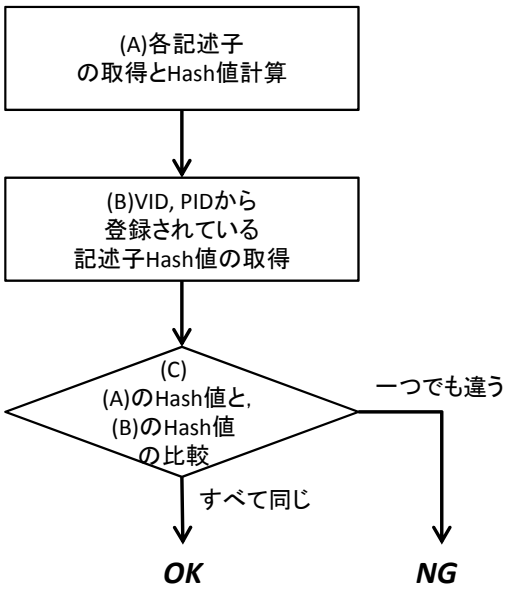


図 6　ホワイトリスト検査フロー

ンフィギュレーション、ストリング、インターフェース、エンドポイント記述子を取得し、記述子ごとに Hash 関数（ここでいう Hash 関数は、入力値を一定の決まった範囲の数値に変換する関数で、同じ入力値に対して常に同じ出力を返すという性質を持つ必要があり、可能な限り SHA-1 などの暗号学的 Hash 関数を使用することが望ましい）にて計算した、確認用デバイス情報を作成する。

（B）（A）で取得したデバイス記述子に格納されている PID, VID から、登録してある登録デバイス情報を取得する。

（C）確認用デバイス情報と登録デバイス情報を比較する。そして、すべての情報が同じであれば検査 OK とし、1 つでも違う場合は検査 NG と判定する。

これにより、登録されたときのデバイス、コンフィギュレーション、ストリング、インターフェース、エンドポイント記述子と同じ USB デバイスであると判定でき、PID, VID が同じ悪性 USB デバイスを挿入されたとしても検査できる（悪性 USB デバイスとして機能するためには、必ず本来の機能以外の機能を追加するために記述子を変化させる必要があるため）。

また、同じ USB デバイスであっても、シリアル番号がストリング記述子で与えられる USB デバイスの場合、登録したデバイスしか検査 OK にならない。（同じ USB デバイスであっても、シリアル番号が違えば検査 NG となる）

3.2.2　ブラックリスト検査

ブラックリスト検査は、図 7 で示すフローにて検査する。

（A）GetDescriptor リクエストにて、デバイス記述子を取得する。

（B）取得したデバイス記述子に含まれる VID, PID がブラックリスト登録されているかチェックする。

ブラックリスト登録されている VID, PID と取得した VID, PID が一致すれば、検査 NG とし、一致しなければ検査 OK と判定する。

さらに、VID, PID 以外にも、特定のストリング記述子内の文字列や、記述子の組み合わせによる検査も可能である。

これにより、既に悪性 USB デバイスと認知しているデバイスの判定を行う。

ブラックリスト検査は、実装としては比較的単純であるため、まず大まかにチェックする場合に有効である。

3.2.3　不正記述子検査

不正記述子検査は、図 8 で示すフローにて検査する。

（A）GetDescriptor リクエストにて、デバイス、コンフィギュレーション、ストリング、インターフェース記述子を取得する。また、必要に応じて、インターフェース記述子の bFunctionClass, bFunctionSubClass, bFunctionProtocol から [7]、インターフェース記述子で指定されているクラスを得る [8]。

（B）取得した各記述子に不正な値が無いかチェックする。チェックする項目は、例えば、Bouyat の報告 [1] の様に、インターフェース記述子の bNumEndpoints を 0 にすることで、Windows を BSOD（Blue Screen Of Death）させるような不正記述子をチェックする。

3.2.4　動作モード別検査

本節では、提案 USB ハブの各動作モードにおける

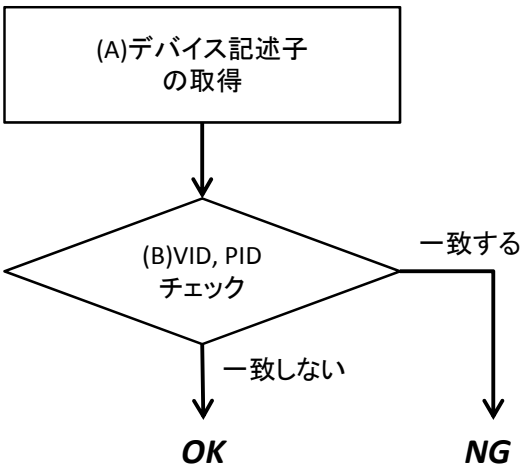


図 7　ブラックリスト検査フロー

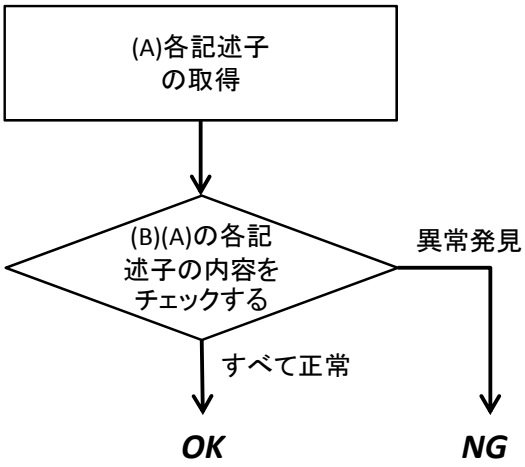


図 8　不正記述子検査フロー

表 3　動作モード別の実施する検査内容

動作モード	チェック
USB デバイス登録	ブラックリスト検査 不正記述子検査
USB デバイス認証	ブラックリスト検査 ホワイトリスト検査
USB デバイス削除	なし

検査内容について記述する。

各動作モードにおける検査内容を表 3 に示す。動作モードが、USB デバイス登録の場合は、ブラックリスト検査と、不正記述子検査を行う。これにより、USB ホストへの既知の悪性 USB デバイスの接続と、不正記述子をもつ悪性 USB デバイスの接続を遮断することができる。

動作モードが、USB デバイス認証の場合は、ブラックリスト検査と、ホワイトリスト検査を行う。ブラックリスト検査を行うのは、USB デバイスの登録後に、登録済み USB デバイスが悪性 USB デバイスとして見つかった場合、ホストへの接続を遮断するためである。

動作モードが、USB デバイス削除の場合は、とくに検査する必要はない。

上記のように、各動作モードにて、各検査を行うことで、USB ホストへ接続する前に、悪性 USB デバイスを検知し遮断することができる。

4　考察

提案 USB ハブでは、ホストに接続された USB デバイスが悪性 USB デバイスとして書き換えられることを遮断することはできない（図 9 に示す（A）の方向）。これは、USB デバイスが持つファームウェア書き換え方法が、USB デバイスに内蔵された IC の仕様により変わるため、市場に出回っているすべての IC への対応が難しく、また、仕様が公開されていない IC も多いためである。しかし、提案 USB ハブをホストとの間に接続し、USB デバイスが、USB デバイスとして機能するために必要な記述子を、ホワイトリスト検査、ブラックリスト検査、不正記述子検査すれば、多くの場合、提案 USB ハブは悪性 USB デバイス化させられた USB デバイスを検知し遮断することができる（図 9 に示す（B）の方向）。

以下では、**1** にて分類した悪性 USB デバイスについて考察する。

- 悪性組込み型

  悪性組込み型に関しては、提案 USB ハブでは、主にブラックリスト検査にて検知し遮断する。ブラックリスト検査であるため、アンチウィルスソフトウェアと同様、悪性組込み型の悪性 USB デバイスが発見された後、ブラックリスト検査情報を更新する必要がある。

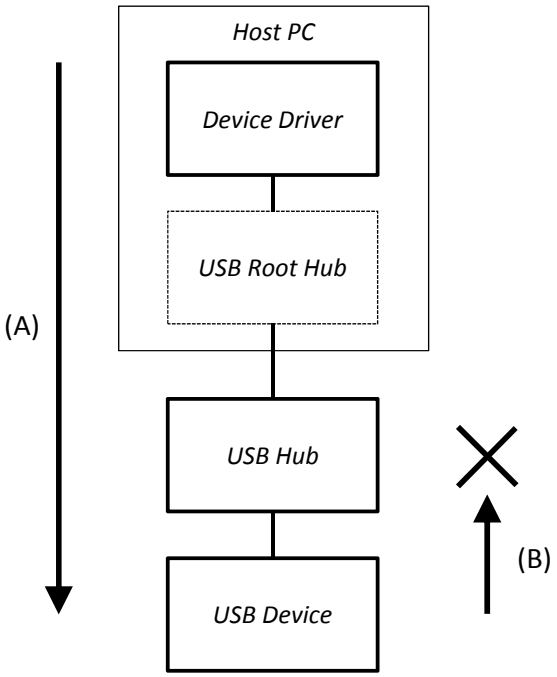


図 9　提案 USB ハブの検査可能方向

- 不正記述子型

  不正記述子型に関しては、提案 USB ハブでは、不正記述子検査を行うことで検知し遮断する。これは、USB ファジングなどの攻撃に対して有効である。

- 機能追加型

  機能追加型に関しては、提案 USB ハブでは、ホワイトリスト検査にて検知し遮断する。USB メモリが、USB メモリ + USB キーボードの様に、機能が追加された場合において、ホワイトリスト検査は有効である。

- 機能改変型

  機能改変型に関しては、提案 USB ハブでは、検知することが難しい。これは、USB デバイスが、悪性 USB デバイスとして書き換えられた後にも、書き換え前の記述子と同じ記述子とすることも可能である。例えば、ファームウェアを巧妙に改変した悪性 USB キーボードでは、ある一定時間キー入力が無い時や、設定されたタイミングで（ユーザが PC の前にいないタイミングなど）、不正なキーストロークを送るような悪性 USB デバイスは、記述子を変化させなくても USB キーボードとして機能すればよい。このような悪性 USB デ

バイスは、提案 USB ハブでは検知することができない。しかしながら、提案 USB ハブの応用として、提案 USB ハブに LCD など表示装置を構成することで、USB キーボードを提案 USB ハブに接続後、表示装置にワンタイムパスワードなどを表示し、一定時間以内に表示された内容を入力しなければ検査 NG とするように拡張することも考えられる。この場合、悪性 USB キーボードが、USB コネクタに接続後直ぐに悪性な動作を行うようなデバイスに対しては有効である。

提案 USB ハブとは異なり、記述子をチェックする以外にも、USB の伝送信号を IDS/IPS 装置の様に、Deep Inspection することでチェックすることも考えられるが、表 2 の様に、USB 3.1 では、線路上に流れるデータは、10 Gbps と高速な通信であるため、Deep Inspection するには、多くのリソースと技術が必要である。また、市販化された際には、高額な装置となると考える。

5　まとめ

本稿では、悪性 USB デバイスの対策として、PC などの USB ホストに対して、悪性 USB デバイスが接続されたことを検知し遮断するための検査機能付き USB ハブを提案した。

今後、悪性 USB デバイスを利用した攻撃がますます増加すると考えられるため、本提案 USB ハブは、悪性 USB デバイス対策の一手法として効果があると考えられる。

今後の課題としては、提案 USB ハブを実際に回路設計し、動作するデバイスとして作成し効果を評価することと、ホワイトリスト検査や不正記述子検査をパスするような、悪性 USB デバイスに対する検査手法が挙げられる。

謝辞

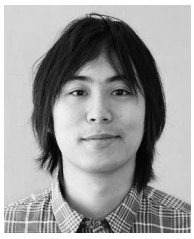
サイバーセキュリティ研究室の金谷延幸氏には、本稿に関する貴重な意見を頂いた。ここに改めて感謝の意を表したい。

【参考文献】

1　J. Bouyat, “USB Fuzzing Basics: From fuzzing to bug reporting,” Quarkslab's blog, http://blog.quarkslab.com/usb-fuzzing-basics-from-fuzzing-to-bug-reporting.html
2　K. Nohl, and J. Lell, “BadUSB -- On accessories that turn evil,” https://srlabs.de/blog/wp-content/uploads/2014/07/SRLabs-BadUSB-BlackHat-v1.pdf, 2014.
3　BadAndroid-v0.1, https://srlabs.de/blog/wp-content/uploads/2014/07/BadAndroid-v0.1.zip
4　“Now e-cigarettes can give you malware,” Guardian News and Media Limited, http://www.theguardian.com/technology/2014/nov/21/e-cigarettes-malware-computers
5　Turning USB peripherals into BadUSB, SRLabs, https://srlabs.de/badusb/
6　V. Tonder, Rijnard, and Herman Engelbrecht “Lowering the USB fuzzing barrier by transparent two-way emulation,” Proceedings of the 8th USENIX conference on Offensive Technologies. USENIX Association, 2014.
7　Universal Serial Bus 3.1 Specification, Revision 1.0, July 26, 2013, http://www.usb.org
8　Usb.org, USB Class Codes, Aug. 11, 2014, http://www.usb.org/developers/defined_class

竹久達也　（たけひさ　たつや）
サイバーセキュリティ研究所
サイバーセキュリティ研究室
招へい専門員
サイバーセキュリティ、組み込みデバイスセキュリティ

岩村　誠　（いわむら　まこと）
元サイバー攻撃対策総合研究センター
元サイバー防御戦術研究室
協力研究員
博士 (工学)
マルウェア解析



丑丸逸人　（うしまる　はやと）
サイバーセキュリティ研究所
サイバーセキュリティ研究室
専門研究技術員
マルウェア解析