PreOS 環境における
アプリケーションとセキュリティ

研究者　　金津 穂
指導教員　原　元司
松江工業高等専門学校
情報工学科

平成 26 年 2 月 7 日

目次

1. はじめに 2

2. PreOS 環境 2
2.1 PreOS 環境とは . . . 2
2.2 BIOS . . . 2
2.3 UEFI . . . 3

3. UEFI における開発フレームワーク 3
3.1 gnu-efi . . . 3
3.2 EDK とそのプログラミング例 . . . 3

4. UEFI における関連研究 4
4.1 Rootkit Detection Framework for UEFI . . . 4
4.2 Dreamboot . . . 4
4.3 Measured Boot . . . 5

5. 未知の脆弱性の発見と検証 5

6. 対策の提案 5

7. まとめ 6

PreOS 環境におけるアプリケーションとセキュリティ

研究者: 金津 穂

指導教員: 原 元司

Abstract

近年，パーソナルコンピューターにおけるファームウェアについて，BIOS から UEFI への置き換えが進んでいる．さまざまな制限のある BIOS と違って柔軟な環境である UEFI は，柔軟であるかわりに悪意のあるプログラムのインストール実行を容易にしてしまった．そこで，本研究では UEFI にて新たに発生したセキュリティ上の問題について，現時点で未知の脆弱性の発見とその解決法の提案を行なった．

Keywords: BIOS，UEFI，RootKit

1. はじめに

近年，一般的なパーソナルコンピュータにおける PreOS 環境として BIOS から UEFI の置き換えが進められている．BIOS は，現在パーソナルコンピュータのアーキテクチャとして一般化した PC/AT 互換機のためのファームウェアとして開発された．この BIOS は開発当時と互換性を保つために制限された仕様になっている．このことは，BIOS や OS，ブートローダーといったソフトウェア開発者にとっては大きなデメリットである．

また，BIOS の乏しい拡張性が実用上限界にきていたため，代替のファームウェアとして UEFI が開発され，一般に普及するようになった．UEFI は BIOS と異なり標準化団体によって策定された仕様に基づいて開発されている．また，シェルを内蔵し柔軟な開発フレームワークを備え誰にでも OS が起動する以前の環境においてモジュールをインストール・実行できるようになった．しかしデメリットとして，悪意のあるプログラム，とくに Rootkit や Bootkit*1 の開発や実行も容易になってしまった．

そこで，本研究では UEFI にて指摘されつつあるセキュリティ上の問題について調査し，現在のところ知られていない未知の脆弱性を発見した．また，この問題について実マシンでの実証実験を行ない，さらに対策を提案・検討する．

2. PreOS 環境

2.1 PreOS 環境とは

OS を搭載した汎用コンピュータは，(1) 電源の投入，(2) デバイスの初期化，(3) ブートローダーの実行，(4) デバイスの制御を OS に引き渡し OS の起動を行ない，初めて OS が起動し，使用状態に入ることができる．この (1)〜(4) の手順の間の環境を，PreOS 環境と言う．PreOS 環境は普通カーネルモード*2 で動作する．この時起動しているプログラムは，コンピュータのボード上の ROM に実装されたファームウェアであることが多い．BIOS も UEFI もファームウェアの一種である．

2.2 BIOS

IBM が 1984 年に発売した PC/AT については，多くの互換機が作成され，今現在のパーソナルコンピュータの

*1 Rootkit は root 権限を奪取した悪意あるユーザーが自らの痕跡を隠すツールを指す．また，Bootkit はブートローダー領域やファームウェアに潜みカーネルモードで実行される Rootkit のことを指す．

*2 またはスーパーバイザーモードとも．プロセッサがもつ保護機構において特別権限を持ち使用できる命令とアクセスできるデバイスに制限がない．

基本アーキテクチャとなっている．当時このコンピュータは，IBM の 80286 プロセッサを搭載しており，この時に BIOS として実装されていたファームウェアが今日の BIOS の仕様となっている．このため，現在の BIOS も Intel x86 アークテクチャプロセッサの 16bit 命令に，1MB メモリアドレッシングといった制限が残っている．

コンピュータは，その起動時に，ブートローダーを読み込むためにディスクを操作しファイルの読み書きをする状態になるまでのプログラムを ROM として実装しておく必要がある．このプログラムは IPL またはブートストラップと呼ばれる．また，デバイスの制御にあたって，レジスタやメモリの操作が必要となるが，コンピュータが実装するデバイスによってこの操作方法が異なる．このため，デバイスの操作をサブルーチンとして実装しておき，OS から API としてアクセスできるようにするとハードウェア間の違いを吸収することができ便利となる．そこで，この IPL やデバイス操作の API をまとめた ROM が BIOS である．

しかし，近年の OS はデバイスの操作をデバイスドライバの形で実装・管理し，OS が起動してから BIOS を通してデバイスにアクセスすることはない．ただし，BIOS のアクセスメソッドについての公開インターフェイスである SMBIOS や，電源管理の統一規格である ACPI については，現在も OS から BIOS へのコールを通じてアクセスされる．

2.3 UEFI

UEFI は，2000 年に Intel により提唱・開発が開始された BIOS の代替を目指した規格である．当初は EFI という名称であったが，2005 年の標準化団体発足に合わせ UEFI という呼称に改められた．

BIOS はその作られた当時の仕様が今なお引き継がれている．このため，Intel x86 アーキテクチャのリアルモードでのみ動作するため命令幅は 16bit に制限され，メモリアドレッシングは 1MB までである．また，BIOS で用いられるパーティションは MBR という形式でブートセクタ[*3]に格納され，容量が限られているため 4 個しかパーティションを作成できない．また，そのパーティションは 1 つにつき 2TiB までという制限が存在する．

一方，UEFI は，プロセッサアーキテクチャに依存せず命令幅にもメモリアドレッシングにも制限が存在しない．使用されるパーティションも GPT という形式で，128 個までパーティションが管理できる．また，1 つのパーティションのサイズも 8ZiB まで使用可能である．

UEFI では，ブートストラップについても，IPL ではなく簡易的なブートローダのようなものがブートマネージャーという名前で実装され，複数のデバイスやブートローダーをそこから選択可能である．

UEFI を使用するにあたっては，ディスクの最初のパーティションを FAT ファイルシステムによりフォーマットし，システムパーティションとして使用する．ブートローダーはディスクの先頭セクタではなく，このシステムパーティションにファイルとして配置されるため，一般にアクセス可能となっている．また，UEFI では MS-DOS 風のシェルが内蔵され，UEFI 上に実装されたアプリケーションを実行することも容易である [1].

BIOS と UEFI の比較を表 1 に示す．

3. UEFI における開発フレームワーク

本章では UEFI で使用できる開発フレームワークについて述べる．

3.1 gnu-efi

gun-efi は UEFI の開発用ラッパーライブラリで，オープンソースであるため Linux や FreeBSD といったプロジェクトと親和性が高く，多くのブートローダーのオープンソース実装がこれを用いて開発されている．

gnu-efi を用いる利点は， UEFI ターゲットの GNU Toolchain を用いることで gdb を用いたデバッグ開発が可能となることである．これは UNIX 文化に慣れた人間にとって開発がしやすい．なお，gnu という名前ではあるが，ライセンスは BSD ライセンスである．

3.2 EDK とそのプログラミング例

EDK は Intel が公式に開発，BSD ライセンスで公開しているフレームワークであり，実質上リファレンス実装である．パッケージという形で多くのライブラリが存在

*3 ディスクの先頭 512 バイト (先頭セクタ) に存在する，BIOS の IPL が読み出す部分である．通常，ブートローダーとパーティションテーブルが格納される．

表 1　BIOS と UEFI の比較

BIOS	UEFI
16bit 命令	命令幅制限なし
1MB メモリアドレッシング	メモリアクセス制限なし
Intel x86 アーキテクチャ	アーキテクチャ非依存
IBM によるデファクトスタンダード	標準化団体による仕様策定
	開発フレームワークが存在
	ネットワークスタック完備
	シェル内蔵

しており，標準 C ライブラリ・BSD ソケット・Python といったものが用意さているほかに，UEFI Shell や QEMU 用のイメージといったものも同梱している． また，Intel C Compiler，Visual Studio，GNU C Compiler でそれぞれ使用することができる．

つぎに, EDK を用いたプログラミング例を示す．UEFI では以上の通り，EDK というフレームワークが使用できる．これは標準 C ライブラリを持っており，通常の C 言語のエントリポイントである

```
int main( int argc, char **argv )
```

が使用できるため pure C で書かれたアプリケーションの移植が非常にしやすい．

また，この形式のエントリポイントを使用しない場合は，定義ファイルにエントリポイント名を記述した上で，

```
EFI_STATUS
EFI_API
UefiMain(
    EFI_IMAGE_HANDLE ImageHandle,
    EFI_SYSTEM_TABLE SystemTable
);
```

を使用することになっている．

UEFI の API 特有のプログラミングでは，Protocol と呼ばれる API を用いる．エントリポイントの引数である SystemTable にはコンソールへのハンドルと，RuntimeServices · BootServices というものが格納された構造体で，ImageHandle は自身のイメージファイルへのハンドルである．

Protocol を使用するには BootServices のメンバ関数である OpenProtocl() 関数にプロトコル名，プロトコル構造体，プロトコルを適用したいファイルやデバイスのハンドルを渡して使用する．プロトコル構造体は実際に使用する API 関数へのポインタが詰まった構造体である．

4. UEFI における関連研究

4.1 Rootkit Detection Framework for UEFI

2012 年のセキュリティカンファレンス BlackHat にて，UEFI を用いて MacOSX 上で動作する Bootkit が ReversedLab により公開された．翌年 2013 年の BlackHat ではその対策法として，ReversedLab と DARPA の共同開発である UEFI 上で Rootkit の検査を行なう RDFU(Rootkit Detection Framework for UEFI) が公開された．これは，メモリ検査や UEFI ドライバ · サービスのチェック，ディスクを VMware の仮想ディスク形式にダンプといったさまざまな検査を行なえる強力なツールである [2].

4.2 Dreamboot

Dreamboot は Hack In the Box 2013 で QuarksLab により公開された Windows 8 x64 向けの Bootkit である．これは，ブートローダーをフックし OS のチェック機構をバイパスするものである．具体的には，ほぼ正規のブートローダーと同じ動作をするが，カーネルイメージを LoadImage() 関数によりロードし，StartImage() 関数で起動する．その間に，ロードしたイメージが配列の形でアクセスできることを利用し，メモリパッチを当て，チェック機構の部分を NOP 命令に変えてしまうという手法である [3].

4.3 Measured Boot

Measured Boot はハッキングカンファレンス DEFCON 20 にて発表された仕組みである．これは TPM(Trusted Platform Module) と呼ばれるセキュリティチップを利用し，OS のブート以前に読み込むドライバやブートローダーといったモジュールを読み込む前に，ハッシュを計算しログを作成する．この上でサーバーでそのハッシュ値について署名を確認するシステムである．この機能により読み込むモジュールが信頼されているものかどうかを確認することができる．

しかしこのシステムはまだ脆弱性を残しており，(1) 日々増加する TPM についてホワイトリストやパッチを追従させなければならないこと，(2) 増え続ける攻撃のトレンドがハードウェアでなくソフトウェアに移りつつあること，(3) ハイバネーションファイルが保護されていないこと，などが問題点としてあげられている [4].

5. 未知の脆弱性の発見と検証

今回，UEFI 環境での Linux について関連研究を参考にした上で，ハイバネーションイメージを標的にした攻撃手法を着想した．そこで，発見した未知の脆弱性とその対策について実証と研究を行なった．ハイバネーションとは，コンピュータを待機状態にする際，レジスタコンテキストとメモリイメージをディスクに退避してほぼ全てのデバイスへの電源供給をシャットダウンすることで消費電力を限りなくゼロに近くすることである．ハイバネーションイメージは Linux の場合スワップディスクに保存されるが，このイメージ改竄に対して保護されていない．

UEFI では，ディスクのハンドルを取得し，BLOCK IO PROTOCOL というセクターブロック毎の Read/Write をするための低レベル API を開くことができる．このことで，ファイルシステムを考慮しないディスクへの読み込みと書き込みを行なうことが容易に可能である．

そこで，ハイバネーションした Linux マシンが復帰する際，OS を起動する前にスワップディスクに対して BLOCK IO PROTOCOL を用いて直接読み書きを行なうプログラムを作成・実行した．このことで，ハイバネーションイメージをそのまま読み込むことが可能であることを確認した．図 1 にその様子を示す．

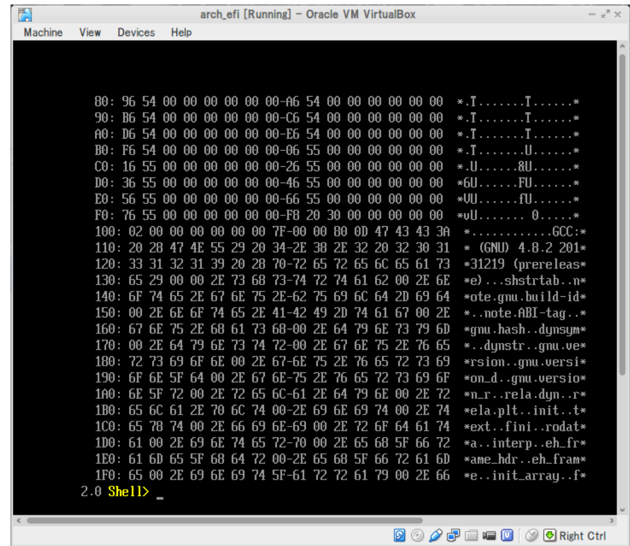

図 1　スワップディスクを読み出す様子

また，読み込んだハイバネーションイメージには配列としてバイト単位でアクセス可能である．このため，メモリパッチ同様の手法により命令の書き換えを行ない，任意のコードを書き込み，スワップディスクに書き戻すことが可能であることを確認した．これによりハイバネーションから復帰する際任意のコードを実行することが可能である．

ハイバネーションイメージは LZO 形式で圧縮され CRC32 によりチェックサムの確認が Linux カーネルで行なわれるものの，LZO 形式の圧縮・解凍方法は広く知られており，また，CRC32 についても完全性を担保するためのチェックサムであるためにデータ改竄に対しては脆弱であると言える．

6. 対策の提案

前章で確認した事象について，いくつかの対策手法を考えた．

(1) ボリュームの暗号化

Linux は標準でボリュームを LVM などを用いて暗号化することが可能である．しかし，boot 時にランダムキーを生成しこのキーを用いて暗号化するために，ハイバネーションからの復帰時に暗号化されたディスクを復号化するのは不可能である．既に，暗号化の際使用するキーをランダムキーから任意のパスワードに置き換

える work around も Ubuntu community で紹介されている．しかし，全てのユーザーが swap のパスワードを記憶しなければならない，ディストリビューションの配布元などからの公式サポートが得られない，といったデメリットがあるため現実的ではない．

(2) ハッシュや署名による認証

前章での提案手法同様、ハッシュや署名自体が改竄される可能性は十分にある．そこで、フラッシュメモリのような別のデバイスに，ハッシュを保存する仕組みを実装するブートローダーをインストールしておき，ブートは常にその外部デバイスから行なう．ハイバネーション時にはそのデバイスをユーザーが手元に置くようにする．このときは，ブートローダがなく，もし別にブートローダーが用意できてもハッシュの整合が取れずハイバネーションから復帰できない．このため，第三者の悪用からコンピュータを守ることができる．また，Linux カーネルそのものを改変し，改竄に対して有効なハッシュの検証の仕組みをハイバネーション時に実行するのも有効だと思われる．しかし，カーネルの改変を最新版のカーネルに追従するか，本家に取り込んでもらう必要が発生するため，ハードルが高く，困難である．

7. まとめ

本研究では，UEFI において未知の脆弱性について実証実験を行なった．また，その脆弱性について対策の提案を行なった．これにより，UEFI ではまだ多くの脆弱性があり，これからも対策を講じる必要性が判明した．そして，UEFI についての動向は変化が激しいため，その動向に追随する必要もあると考える．

今後の課題としては，前章であげた対策のうち，ブートローダについて実際に実装し検証を進める必要がある．また，他にどのような対策がとられており，同様の脆弱性について議論されているか，などといったことについてカンファレンスやユーザーコミュニティの動向をリサーチしなければならないだろう．

参考文献

[1] Michael Rothman et al. *Harnessing the UEFI Shell: Moving the Platform Beyond DOS*. Intel Press, 2010.

[2] Mario Vuksan and Tomislav Pericin. *Rootkit Detection Framework for UEFI*. URL: http://www.reversinglabs.com/resources/open-source/rdfu.html.

[3] Sébastien Kaczmarek. *UEFI and Dreamboot*. URL: http://www.quarkslab.com/dl/13-04-hitb-uefi-dreamboot.pdf.

[4] Dan Griffin. *Hacking Windows Vista Security*. URL: http://www.defcon.org/images/defcon-20/dc-20-presentations/Griffin/DEFCON-20-Griffin-Hacking-Measured-Boot-and-UEFI.pdf.