Vol.2011-CSEC-54 No.36
Vol.2011-SPT-1 No.36
2011/7/13

API フックによる DEP 回避の防止

栗原　寛昇，岡本　剛
神奈川工科大学大学院 工学研究科 情報工学専攻
〒243-0292 神奈川県厚木市下荻野 1030

概要：インターネットには，様々な脆弱なサービスが存在し，攻撃コードを自動生成するツールなどにより，脆弱性攻撃による脅威は深刻なものとなっている．脆弱性攻撃を防ぐ方法の 1 つに，Microsoft が実装した DEP(Data Execution Prevention)という機能がある．
しかし，DEP を回避する脆弱性攻撃が明らかになり，Windows のセキュリティ強度の低下が問題になっている． 本稿では，DEP を回避する脆弱性攻撃を検知して不正な命令の実行を防止するプログラムを提案し，実装した．さらに実装したプログラムにより，DEP 回避を防止できることを確認した．

Prevention using API hook against DEP Bypass

Hironori Kurihara, Takeshi Okamoto
Graduate School of Science Engineering,
Kanagawa Institute of Technology.
1030, Shimo-ogino, Atsugi, Kanagawa 243-0292, Japan

Abstract: Internet users are prone to various vulnerable services and tools that automatically generate attack codes posing serious problems for computer security. One security feature intended to prevent such problems is DEP (Data Execution Prevention) implemented by Microsoft.
DEP prevents executions of code from the data region of memory. Unfortunately, a DEP bypass method has recently been revealed, which means a decrease in the security level of Windows OS. In this paper we propose a method for prevention of DEP bypass, using API hooks. We tested the method and evaluated its effectiveness

1. はじめに

コンピュータのユーザーは，様々なソフトウェアを自由に選択して利用することができる．しかし，ソフトウェアには，脆弱性と呼ばれるソフトウェア開発段階で抱えた弱点を持つことが多く，悪意のあるユーザーは脆弱性を利用して攻撃を行うことがある．近年，インターネットの急速な普及に伴い，悪意のあるユーザーによる脆弱性攻撃の被害が深刻な問題となっている[1][2]． 脆弱性攻撃には様々な種類がある．この脆弱性攻撃を防ぐ方法として，Microsoft が提供するデータ実行防止機能（DEP : Data Execution Prevention) は，データ領域でのプログラム実行を強制的に遮断する機能がある．

しかし，近年 DEP を回避する脆弱性攻撃が，確認された[3]．DEP 回避を行う攻撃では，Windows 32 ビット API 関数が利用される[4]．この Windows 32 ビット API 関数を利用することを手がかりにして，本稿では， DEP 回避を行う API 関数呼び出しが行われるプロセスを API フックしてその実行を防止する手法を提案する．本稿は，DEP を回避する攻撃の原理を理解し，API フックを用いた DEP を回避する脆弱性攻撃を防止できるプログラムを作成した．API フックを行うことにより，作成したプログラムは，API 関数の引数の値を監視して， DEP 回避を検知した．また，DEP 回避をしようとしている API 関数の引数の値は DEP を有効にする引数の値に書き換えることにより，DEP 回避を行う攻撃を防止した．

2. DEP

データ実行防止 (DEP) は，Windows OS にメモリをチェックする機能を追加することにより，データ領域でコードの実行を防止する機能である．この機能は，ハードウェアとソフトウェアの両方に用意されている機能である．AMD64 や IA-32 シリーズ以降の実行禁止ビット（NX/XDbit)をサポートしたハードウェアおよび Microsoft Windows XP Service Pack 2 と Microsoft Windows Server 2003 Service Pack 1 以降の Windows OS によって DEP が適用される[5]．DEP の主な原理は，データページからのコード実行を防止することである．通常，コードはヒープやスタックからは実行されない．ハードウェア DEP では，ヒープやスタックから実行されようとしているコードを検出し，実行された場合は例外を生成する．ソフトウェア DEP は，Windows で起こる例外処理を行うコードを防止するのに役立つ機能である．DEP の有効範囲を示すものとして DEP ポリシーがある．これは，Windows がブート時に参照する boot.ini のカーネルパラメータ/NoExecute である．そのオプションを表 2.1 節に示す[6]．どの Windows OS でもオプションのデフォルト設定は，OptIn の状態である．

Vol.2011-CSEC-54 No.36
Vol.2011-SPT-1 No.36
2011/7/13

表 2.1 選択できるオプションとその意味

オプションの値	オプションの意味
OptIn	既定のプログラムに対してのみ適用する
OptOut	指定したプログラム以外すべて適用する
AlwaysOn	すべて適用する
AlwaysOff	すべて適用しない

3. DEP の回避

3.1 DEP 回避の手法

DEP 回避を行う攻撃は，Windows 32 ビット API 関数が利用される．DEP 回避を行うプロセスは，最初に API 関数を呼び出し，その API 関数の引数の値を DEP 回避できる値に設定する．そして，データ領域に働いている DEP を無効にする．API 関数が Windows で動作するモードは，ユーザーモードとカーネルモードに分けられる．ユーザーモードで動作する API 関数は，優先度が低いものである．次にカーネルモードで動作する API 関数は，優先度が高いものであり，OS の中心部分としてコンピュータ内のどの部分にもアクセスし変更できる．

3.2 DEP 回避の種類

本節では，2011 年 5 月の執筆時点で報告されている 6 つの API 関数を用いた DEP 回避について説明する．これらの API 関数を提供している DLL と Windows で動作しているモードを表 3.1 に示す．また，それぞれの DEP 回避を行う API 関数について，Windows のバージョンによって，API 関数を用いた DEP 回避が有効かどうか異なる．その結果を表 3.2 に示す[7]．表 3.2 では，OS に対して DEP 回避が有効な場合は×であり，OS に対して DEP 回避が無効である場合は○と表記する．

表 3.1DEP 回避に使用される API 関数

API 関数名	DLL	モード
VirtualAlloc 関数	kernel32.dll	ユーザーモード
VirtualProtect 関数		
HeapCreate 関数		
WriteProcessMemory 関数		
SetProcessDEPPolicy 関数		
NtSetInformationProcess 関数	ntdll.dll	ユーザーモード カーネルモード

表 3.2 API 関数を用いた DEP 回避の OS 対応表

API 関数名/ OS	XP SP2	XP SP3	Vista SP0	Vista SP1	Windows 7	Windows 2003	Windows 2008
VirtualProtect	×	×	×	×	×	×	×
VirtualAlloc	×	×	×	×	×	×	×
HeapCreate	×	×	×	×	×	×	×
SetProcessDEPPolicy	○	×	○	×	○	○	×
WriteProcessMemory	×	×	×	×	×	×	×
NtSetInformationProcess	×	×	×	○	○	×	○

3.2.1 VirtualProtect 関数を利用した DEP 回避

VirtualProtect 関数は，呼び出し側プロセスの仮想アドレス空間内のコミット済みページ領域に対するアクセス保護の状態を変更する機能を持つ関数である．関数のプロトタイプを図 3.1 に示す．

Vol.2011-CSEC-54 No.36
Vol.2011-SPT-1 No.36
2011/7/13

```
BOOL VirtualProtect(
   LPVOID lpAddress,       // コミット済みページ領域のアドレス
   DWORD dwSize,           // 領域のサイズ
   DWORD flNewProtect,     // 希望のアクセス保護
   PDWORD lpflOldProtect // 従来のアクセス保護を取得する変数のアドレス
);
```

図 3.1 VirtualProtect 関数

この関数を使用して DEP を回避するためには，まず，引数 lpAddress に DEP を無効にするコミット済みページ領域のベースアドレス(動的に作成された値)を指定して，引数 flNewProtect に PAGE_READWRITE(0x40)の値を指定し，コミット済みページ領域で読み取りと書き込みの両方のアクセス権を有効にする必要がある（コミット済みページ領域の DEP を無効にした状態にする）．DEP を無効にした領域に memcpy 関数などを使用してシェルコードをコピーすればシェルコードが実行できるようになる．この DEP 回避は，DEP ポリシーの全オプションの値で DEP 回避が可能である．

3.2.2 VirtualAlloc **関数を利用した** DEP **回避**

VirtualAlloc 関数は呼び出し側プロセスの仮想アドレス空間内のページ領域を，予約またはコミットする機能を持つ．関数のプロトタイプを図 3.2 に示す．

```
LPVOID VirtualAlloc(
   LPVOID lpAddress,          // 予約またはコミットしたい領域
   SIZE_T dwSize,             // 領域のサイズ
   DWORD flAllocationType,    // 割り当てのタイプ
   DWORD flProtect            // アクセス保護のタイプ
);
```

図 3.2 VirtualAlloc 関数

この関数を使用して DEP を回避するためには．引数 lpAddress に DEP を無効にしたいメモリページ領域のアドレスを指定して，引数 flAllocationType に MEM_COMMIT(0x1000)の値を指定し，指定されたメモリページ領域を，メモリ内の物理格納域に確保する（DEP を無効にする領域を確保する）．次に，引数 flProtect に PAGE_READWRITE(0x40)の値を指定し，コミット済みページ領域で読み取りと書き込みのアクセス権を有効にする（コミット済みページ領域の DEP を無効にした状態にする）． DEP を無効にした領域に memcpy 関数などを使用してシェルコードをコピーすればシェルコードが実行できるようになる．この DEP 回避は，DEP ポリシーのすべてのオプションで可能である．

3.2.3 HeapCreate **関数を利用した** DEP **回避**

HeapCreate 関数は，HeapAlloc 関数を使って呼び出し側プロセスが使用できるヒープオブジェクトを作成する．プロセスの仮想アドレス空間内の領域を予約し，ブロック内の指定された初期のパートに物理格納域を割り当てる機能を持つ関数である．関数のプロトタイプを図 3.3 に示す．

```
HANDLE HeapCreate(
  DWORD    flOptions,          // ヒープ割り当て方法の属性
  SIZE_T   dwInitialSize,      // 初期のヒープサイズ
  SIZE_T   dwMaximumSize       // 最大ヒープサイズ
);
```

図 3.3 HeapCreate 関数

HeapAlloc 関数は，メモリブロックをヒープから割り当てる機能を有する．関数のプロトタイプを図 3.4 に示す．

```
LPVOID HeapAlloc(
   HANDLE hHeap,     // プライベートヒープブロックのハンドル
   DWORD dwFlags,    // ヒープの割り当て方法の制御
   SIZE_T dwBytes    // 割り当てたいバイト数
);
```

図 3.4 HeapAlloc 関数

この関数を使用して DEP を回避するためには，引数 flOptions に HEAP_CREATE_ENABLE_EXECUTE (0x00040000)という値を設定し，ヒープから割り当てられるすべてのメモリブロックが，コード実行を可能にする（割り当てられたすべてのメモリブロックが DEP を無効にした状態になる）．割り当てられたメモリブロックに memcpy 関数などを使用してシェルコードをコピーする．それにより，DEP が無効になった状態でシェルコードが実行できるようになる．この DEP 回避は，DEP ポリシーのすべてのオプションで可能である．

3.2.4 SetProcessDEPPolicy **関数を利用した** DEP **回避**

SetProcessDEPPolicy 関数は，プロセスに対して，DEP を有効にするか無効にするかどうかを制御することができる機能を持つ関数である．関数のプロトタイプを図 3.5 に示す．

Vol.2011-CSEC-54 No.36
Vol.2011-SPT-1 No.36
2011/7/13

```
BOOL WINAPI SetProcessDEPPolicy(
    DWORD dwFlags                  // 処理方法のオプション
);
```

図 3.5 SetProcessDEPPolicy 関数

この関数を使用して DEP を回避するためには，DEP ポリシーのオプションの値が OptIn か OptOut に設定されている必要がある．あるプロセスに対して DEP ポリシーのオプションの値を強制的に，Always Off にする．それにより，DEP を回避した状態で，悪意のある攻撃を行うことができる[8].

3.2.5 WriteProcessMemory **関数を利用した** DEP **回避**

WriteProcessMemory 関数は，指定されたプロセスのメモリ領域にデータを書き込む機能を持つ．書き込みたい領域全体がアクセス可能でなければならない．アクセス可能でない場合，関数は失敗する．この関数のプロトタイプを図 3.6 に示す.

```
BOOL WriteProcessMemory(
    HANDLE hProcess,                    // プロセスのハンドル
    LPVOID lpBaseAddress,               // 書き込み開始アドレス
    LPVOID lpBuffer,                    // データバッファ
    DWORD nSize,                        // 書き込みたいバイト数
    LPDWORD lpNumberOfBytesWritten //  実際に書き込まれたバイト数
);
```

図 3.6 WriteProcessMemory 関数

この関数を使用して DEP を回避するためには，目的の領域を決めて，書き込み可能な状態にする（DEP を無効にした領域を用意する）．その用意した領域に memcpy 関数などを使用してシェルコードをコピーすることで，DEP を回避して攻撃を行うことができる．この DEP 回避は，DEP ポリシーのすべてのオプションで可能である.

3.2.6 NtSetInformationProcess **関数を利用した** DEP **回避**

NtSetInformationProcess 関数は，プロセスの情報を設定する機能を持つ関数である．NtSetInformationProcess 関数のプロトタイプを図 3.7 に示す.

```
NTSYSAPI NTSTATUS NTAPI NtSetInformationProcess(
    IN HANDLE                           ProcessHandle,
    IN PROCESS_INFORMATION_CLASS ProcessInformationClass,
    IN PVOID                            ProcessInformation,
    IN ULONG                            ProcessInformationLength
);
```

図 3.7 NtSetInformationProcess 関数

この関数は，ZwSetInformationProcess 関数と呼ばれるネイティブ AP 関数 に関連している．ネイティブ API 関数は，システムを直接制御する Windows NT 系 の API 関数群である．多くはカーネルモードで動作する API 関数として使用される[9]．ネイティブ API 関数のほとんどはドキュメント化されておらず，通常のプログラムで使用することは推奨されていない API 関数である．そのため，Microsoft 社では，ネイティブ API 関数の情報を公開していない．そのため，関数のプロトタイプで，引数に対するコメントは記載しない．また，ネイティブ API 関数を使用した場合，そのプログラムは Microsoft 社からの保障もなく, OS のバージョンアップに伴う仕様変更に耐えられる保障がないとされている．ユーザーモード側の NtSetInformationProcess 関数からカーネルモード側の ZwSetInformationProcess 関数へ移り変わる流れを図 3.8 に示す.

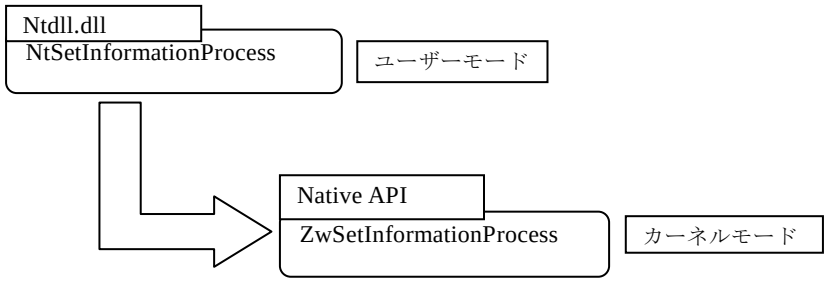

図 3.8 カーネルモード内の流れ

この関数を使用して DEP を回避するためには， DEP ポリシーのオプションの値が OptIn に設定されている必要がある． ZwSetInformationProces 関数の引数 ProcessInformationclass に 0x00000022（プロセスの DEP を有効にするか，無効にするかの設定を変更可能な状態にする）の値を設定し，引数 ProcessInformation に 0x00020410（現行プロセスにおいて DEP を無効にする）の値を設定することにより DEP を回避する.

Vol.2011-CSEC-54 No.36
Vol.2011-SPT-1 No.36
2011/7/13

4. DEP 回避の防止手法

本章では，DEP 回避を防止する手法を示す．DEP 回避を防止する手法として，API フックを用いる．API フックとは，プロセスが API 関数を呼び出した際に，その呼び出しに割り込むことである．API フックにより，API 関数の動作の監視や，指定された API 関数の代わりに任意の処理をさせることが可能となる．

API フックには，2 つの手法を用いた．1 つは，ユーザーモードで動作する API 関数をフックするために，関数のアドレスを管理しているモジュールのインポートセクションを利用した API フックである[10]. もう 1 つは，カーネルモードで動作する API 関数をフックするために，デバイスドライバによってフックする，カーネルモードでの API フックである[11]. 本研究では，それらの API フックを用いた DEP 回避の防止を行うプログラムを提案し開発した．

4.1 インポートセクションを利用した API フックを行うプログラム

本節では，開発したプログラムについて説明する．このプログラムは，ユーザーモードで動作する 5 つの API 関数を用いた DEP 回避を防止するプログラムである. 本研究で作成したものは，API フックの動作を行う hook.dll と，その DLL をリンクさせて実行し，フックした結果を表示する hook_view.exe を開発した．

4.1.1 開発環境

本プログラムの実装時の開発環境を以下に示す．

- Microsoft Windows XP Service Pack 3
- Microsoft Visual Studio 2005
- C 言語
- C++言語

4.1.2 インポートセクションを利用した API フック

ユーザーモードにおいて動作する API 関数をフックする方法で有効なのは，インポートセクションを利用した API フックである．関数アドレスを管理しているモジュールのインポートセクションには実行に必要な DLL や，DLL からインポートしている関数のアドレスが保存されている. このアドレスを書き換えることにより, 特定の API 関数の処理を置き換える方法がインポートセクションを利用したフックである．API フックを行うには，ターゲットプロセスの API 関数を置き換える API 関数を用意し，参照された直後にインポートアドレスを書き換える処理を行う DLL を作成する. ターゲットプロセスの API 関数が呼び出される場合，インポートアドレスが書き換えられているため，作成した DLL 内の API 関数が呼び出され，API 関数をフックすることができる．

4.1.3 アルゴリズム

インポートセクションを利用した API フックを行うプログラムの処理の流れをフローチャートとして図 4.1 に示す．

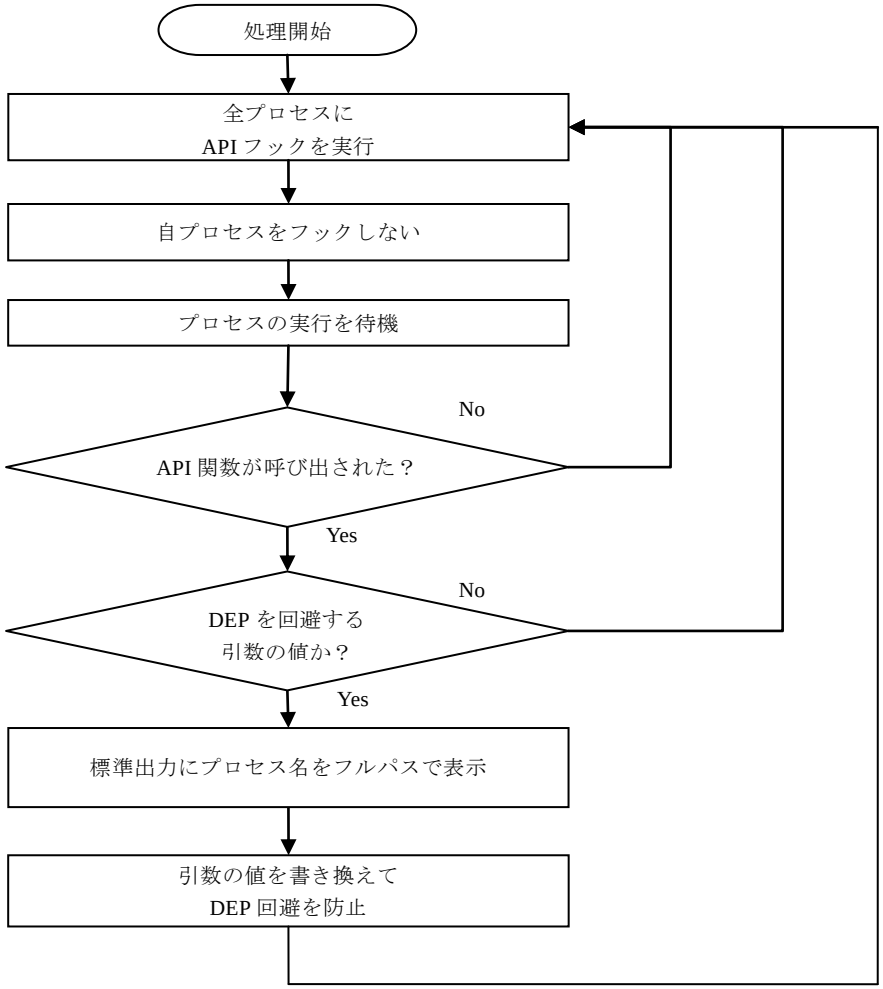

図 4.1 フローチャート

4.2 カーネルモードで API フックを行うデバイスドライバ

本節では，作成したデバイスドライバについて説明する．このデバイスドライバは，カーネルモードで動作するネイティブ API 関数を用いた DEP 回避を防止するプログラムである．開発したプログラムは，API フックを行うシステムファイル hook.sys と，ドライバをインストールするプログラム DrvInstall.exe を用意した．

4.2.1 開発環境

本プログラムの実装時の開発環境を以下に示す．

- Microsoft Windows XP Service Pack 3
- Microsoft Visual Studio 2005
- WDK 7.1.0
- C 言語
- C++言語

4.2.2 カーネルモードでの API フック

カーネルモードで動作する API 関数をフックするためには，フックを行うプログラムもカーネルモードで動作していなければならない．そのため，デバイスドライバを作成して，Windows のカーネルモードで動作しているネイティブ API 関数を調べ，プロセスを検知する．カーネルモードで動作するネイティブ API 関数の呼び出しは，システムコールテーブルが制御している．

通常，システムコールテーブルから参照して使用するネイティブ API 関数のアドレスを調べて，ネイティブ API の呼び出しを行う．システムコールテーブルを書き換え，フック対象である関数のアドレスを作成した API 関数のアドレスに置き換えることにより，作成した hook.sys の API 関数が呼び出されるようになる．

4.2.3 アルゴリズム

カーネルモードで API フックを行うデバイスドライバの処理の流れをフローチャートとして図 4.2 に示す．

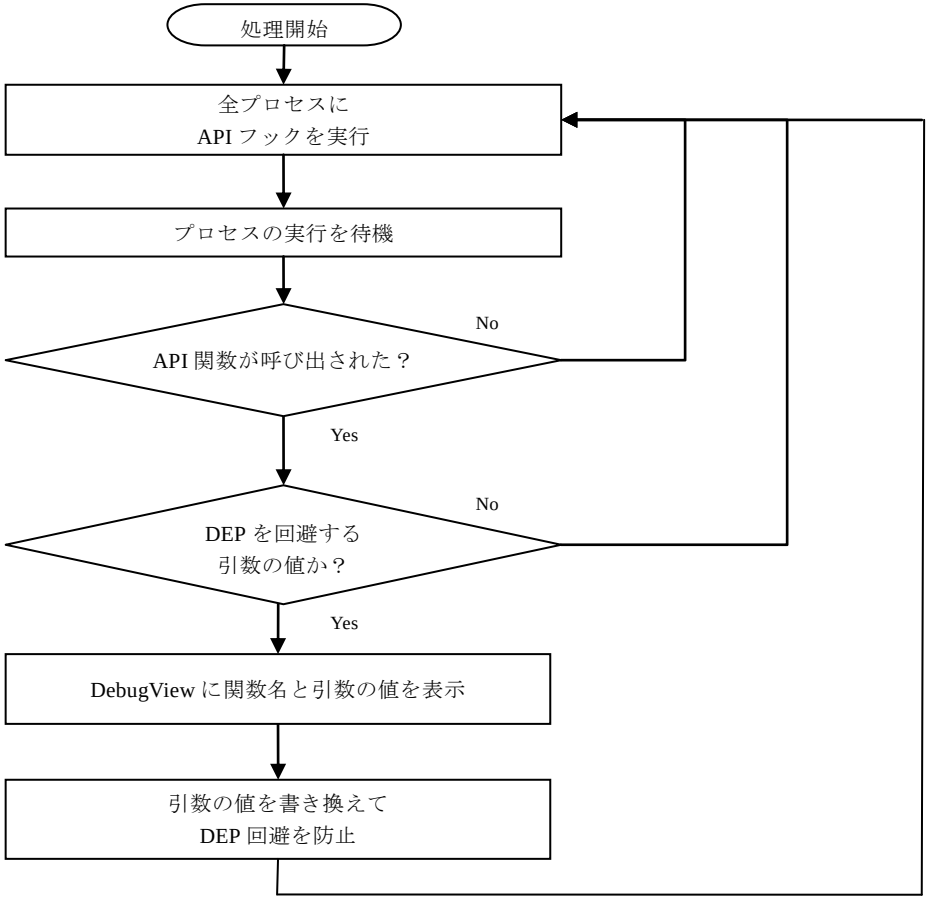

図 4.2 フローチャート

5. 動作検証

5.1 検証環境

本稿では，VMware Workstation7.1 が提供する仮想環境で本手法の有効性を評価した．仮想マシンの OS には，Windows XP SP2 と Windows XP SP3 そして Windows7 を用いた．また，Microsoft から提供されているセキュリティツール EMET を用いて，DEP 回避の防止を検証した[12]．EMET には，Dynamic Data Execution Prevention (動的デー

タ実行防止) 機能を含む 4 つの脆弱性緩和機能をサポートしている. Dynamic DEP は，コンピュータのセキュリティ侵害に利用されかねない実行可能なコードがメモリ空間で実行されるのを防ぐ機能である[13]．EMET で保護するプロセスを選択することによって，そのプロセスに対して，EMET の機能が有効になる．EMET は，Windows XP SP3 以降の環境で使用できる．DEP 回避を行う攻撃には，Metasploit Framework 3.7.1 を用いる．Metasploit framework とは，脆弱性検証ツールであり，システムの脆弱性を調査することができる．本稿では，以下の 2 つの脆弱性に対して DEP 回避を行う脆弱性攻撃を行う．

- マイクロソフト セキュリティ情報 MS11-003 - Internet Explorer 用の累積的なセキュリティ更新プログラム (2482017) [14]

- マイクロソフト セキュリティ情報 MS08-067 - Server サービスの脆弱性により，リモートでコードが実行される (958644) [15]

具体的な脆弱性攻撃名は，それぞれ windows/browser/ms11_003_ie_css_import と windows/smb/ms08_067_netapi である．攻撃に使用するシェルコードは，windows/exec を選択した. このシェルコードは, 侵入したシステムで任意のコマンドを実行できる．本稿では，任意のコマンドとして電卓を指定した．つまり，脆弱性攻撃が成功したとき，標的のコンピュータで電卓が起動する．

脆弱性攻撃 windows/browser/ms11_003_ie_css_import は， VirtualAlloc 関数を使用して DEP 回避を試みる．この攻撃は，Internet Explorer の脆弱性を利用した攻撃で，この攻撃を行うと OS のバージョンに関わらず，Internet Explorer 7 では，電卓が 1 回起動し，Internet Explorer 8 では，電卓が 2 回起動した結果が得られた．この攻撃は，DEP を回避するために，VirtualAlloc 関数以外の API 関数を使用しておらず，複数の API 関数を使用して DEP 回避を行う処理はコード上にない[16]．Internet Explorer 8 で電卓が，2 回起動した原因としては，問題が特定されたら、タブが自動的に復元する機能により，攻撃も同時に復元されて電卓が起動したと考えられる．

そして，脆弱性攻撃 windows/smb/ms08_067_netapi は， NtSetInformationProcess 関数を使用した DEP 回避を試みる．カーネルモードでプロセスが起動するため，電卓は，Windows の画面には表示されなかった．しかし，Windows タスクマネージャで，電卓のプロセスが起動したのを確認した．

本稿で OS 毎に試した攻撃結果を表 5.1 に示す．表 5.1 では，OS に対して DEP 回避が有効な場合は×であり, OS に対して DEP 回避が無効である場合は○と表記する．

表 5.1 脆弱性攻撃の OS 対応表

脆弱性攻撃名/OS	Windows XP SP2	Windows XP SP3	Windows7
windows/browser/ms11_003_ie_css_import	×	×	×
windows/smb/ms08_067_netapi	×	×	○

5.2 検証結果

脆弱性攻撃 windows/browser/ms11_003_ie_css_import を利用した DEP 回避は，インポートセクションを利用した API フックを行うプログラムの結果を，脆弱性攻撃 windows/smb/ms08_067_netapi を利用した DEP 回避は，カーネルモードでの API フックを行うデバイスドライバの結果をこの節で述べる．

5.2.1 windows/browser/ms11_003_ie_css_import による DEP 回避の防止結果

windows/browser/ms11_003_ie_css_import による DEP 回避は，VirtualAlloc 関数の引数 flAllocationType に MEM_COMMIT(0x1000) の値が，引数 flProtect に PAGE_READWRITE(0x40)の値が指定されている． Internet Explorer は，正常な動作を行っている場合でも，DEP 回避に用いられる API 関数を使用しているが，提案手法が Internet Explorer に対して，正常に動作することを確認した．さらに， Flash や Adobe Reader そして JAVA を使用したときも正常に動作することを確認した．

脆弱性攻撃 windows/browser/ms11_003_ie_css_import の攻撃は, DEP 回避を行っている．そのため，提案手法では，DEP 回避を行っているプロセス名を，フルパスで表示した．それにより DEP 回避を防止したことを確認できた．具体的には，DEP を回避するために API 関数の引数 flNewProtect に設定されている PAGE_READWRITE(0x40) の値を，PAGE_GUARD(0x100) の値を設定することにより，DEP 回避を防止できた．

しかし，Internet Explorer 8 に対する攻撃で，提案手法では，1 回目の攻撃は防止できたが，2 回目の攻撃は，防止できなかった．一方，EMET では，1 回目の攻撃は防止できなかったが，2 回目の攻撃は防止できた．これらの原因は，1 回目の攻撃は，DEP 回避が静的なものであり，2 回目の攻撃は DEP 回避が動的なものであることから，この結果が得られたと考えられる．

この結果から，提案手法と EMET の両方利用することによって，1 回目と 2 回目の両方で，Internet Explorer 8 に対するこの脆弱性攻撃を防止できることを確認した（表 5.2）．したがって，脆弱性攻撃 windows/browser/ms11_003_ie_css_import に対して，提案手法と EMET を用いることが有効な手段といえる．表 5.2 では，脆弱性攻撃に対して DEP 回避を防止できる場合は○であり，脆弱性攻撃に対して DEP 回避を防止できない場合は×と表記する．

Vol.2011-CSEC-54 No.36
Vol.2011-SPT-1 No.36
2011/7/13

表 5.2 windows/browser/ms11_003_ie_css_import の攻撃を防止した結果

windows/browser/ms11_003_ie_css_import / 防止手法	提案手法	EMET	提案手法+EMET
1 回目の攻撃	○	×	○
2 回目の攻撃	×	○	○

5.2.2 windows/smb/ms08_067_netapi による DEP 回避の防止結果

脆弱性攻撃 windows/smb/ms08_067_netapi による DEP 回避では，ZwSetInformationProces 関数の引数 ProcessInformationclass に 0x00000022（プロセスの DEP を有効にするか, 無効にするかの設定を変更可能な状態にする）の値が設定され，引数 ProcessInformation に 0x00020410（現行プロセスにおいて DEP を無効にする）の値が設定されていた．

提案手法では，DEP 回避を行っているプロセスが使用している関数名と引数の値を表示した．それにより DEP 回避を検知したことを確認した．具体的には，DEP を回避する引数 ProcessInformation に設定されている 0x00020410 の値を 0x0013E574 に書き換えることにより，DEP を有効にして DEP 回避を防止できた．そして，この脆弱性攻撃は，EMET でも攻撃を防止した（表 5.3）EMET では，Dynamic DEP の機能により，DEP ポリシーを OptIn の状態から動的に AlwaysOn の状態に変更して，DEP 回避を防止していると考えられる．表 5.3 では，脆弱性攻撃に対して DEP 回避を防止できる場合を○と示している．

表 5.3 windows/smb/ms08_067_netapi の攻撃を防止した結果

脆弱性攻撃名 / 防止手法	提案手法	EMET
windows/smb/ms08_067_netapi	○	○

6. おわりに

DEP を回避する脆弱性攻撃が作成され，Windows のセキュリティ強度が低下している．本稿では，API フックを用いて DEP を回避する脆弱性攻撃を防止できるプログラムを開発した．

Metasploit Framework を用いて DEP 回避の攻撃を行い，それを防止できることを確認した．1 つの検証結果では，提案手法で攻撃を防止できないことが明らかになったが，EMET と併用することにより，DEP を回避する脆弱性攻撃の対策が可能であることを確認した．今後の課題として， 提案手法の改良と様々な DEP 回避を行う脆弱性攻撃に対する有効性評価が挙げられる．

参考文献

1) Cyrus Peikari, Anton Chuvakin：セキュリティウォリア,O'REILLY,pp163(2004)
2) Jon Eerickson：HACKING：美しき策謀,O'REILLY,pp12-15(2005)
3) Windows のセキュリティ機能「DEP」を回避する新手法が公開される：
http://www.computerworld.jp/topics/vs/175929.html
4) Microsoft：MSDN ライブラリ,http://msdn.microsoft.com/ja-jp/library/
5) 斎藤和典,シェルコードに起因する脅威と対策の現状,RSACONFERENCE,(2009)
6) Windows XP Service Pack 2,Windows XP Tablet PC Edition 2005,および Windows Server 2003 のデータ実行防止 (DEP) 機能の詳細,http://support.microsoft.com/kb/875352/ja
7) CorelanTeam：http://www.corelan.be:8800/index.php/2010/06/16/exploit-writing-tutorial-part-10
-chaining-dep-with-rop-the-rubikstm-cube
8) Bernardo Damele A. G.：DEP bypass with SetProcessDEPPolicy(),
http://bernardodamele.blogspot.com/2009/12/dep-bypass-with-setprocessdeppolicy.html
9) 安藤類央,Nguyen Anh Quynh ,須崎有康：Windows OS のメモリ挙動モニタと Libvirt によるゼロデイ攻撃の検出システムの構築,SCIS 2009 (2009)
10) Jeffrey Richter: Advanced Windows 第 5 版 下,日経 BP ソフトプレス,pp316(2008)
11) 愛甲健二：ハッカープログラミング大全攻撃編,データハウス,pp400-pp411 (2006)
12) Microsoft：Enhanced Mitigation Experience Toolkit
v2.0,http://www.microsoft.com/downloads/en/details.aspx?FamilyID=c6f0a6ee-05ac-4eb6-acd0-362559fd2f04
13) japan.internet.com：Microsof,脆弱性緩和ツールの新版『EMET 2.0』を公開,http://japan.internet.com/webtech/20100907/11.html
14) Microsoft：Microsoft Security Bulletin MS08-067 ,
http://www.microsoft.com/technet/security/bulletin/MS08-067.mspx
15) Microsoft：Microsoft Security Bulletin MS11-003,
http://www.microsoft.com/technet/security/bulletin/MS11-003.mspx
16) Metasploit Framework ：ms11_003_ie_css_import.rb,
http://dev.metasploit.com/redmine/projects/framework/repository/entry/modules/exploits/windows/browser/ms11_003_ie_css_import.rb