EMET による SEHOP 実装の改良の提案

Fourteenforty Research Institute, Inc.

もくじ

もくじ 2

著作権 3

免責事項 4

更新履歴 6

文書情報 7

1. 概要 8

2. EMET 概要 9
   2.1. 機能概要 9
   2.2. 動作概要 10

3. SEH オーバーライトと SEHOP 11
   3.1. SEH オーバーライト 11
   3.2. SEHOP による防御 12

4. EMET による SEHOP 実装 15

5. EMET の SEHOP 回避 18
   5.1. EmetFinalExceptionHandler を持つ ERR の作成 18
   5.2. EMET の持つアドレス情報を用いた SEH チェーン再構築 20

6. SEHOP 実装の改良方法の提案 26

7. まとめ 27

8. 参考文献 28

付録 A. サンプルコード 29

著作権

当文書内の文章・画像等の記載事項は、別段の定めがない限り全て株式会社フォティーンフォティ技術研究所（以下、フォティーンフォティ）に帰属もしくはフォティーンフォティが権利者の許諾を受けて利用しているものです。これらの情報は、著作権の対象となり世界各国の著作権法によって保護されています。「私的使用のための複製」や「引用」など著作権法上認められた場合を除き、無断で複製・転用することはできません。

免責事項

当文書は AS-IS (現状有姿)にて提供され、フォティーンフォティは明示的かつ暗示的にも、いかなる種類の保証をも行わないものとします。この無保証の内容は、商業的利用の可能性・特定用途への適応性・他の権利への無侵害性などを保証しないことを含みます。たとえフォティーンフォティがそうした損害の可能性について通知していたとしても同様です。また「この文書の内容があらゆる用途に適している」あるいは「この文書の内容に基づいた実装を行うことが、サードパーティー製品の特許および著作権、商標等の権利を侵害しない」といった主張をも保証するものではありません。そして無保証の範囲は、ここに例示したもののみに留まるものではありません。

また、フォティーンフォティはこの文書およびその内容・リンク先についての正確性や完全性についても一切の保証をいたしかねます。

当文書内の記載事項は予告なしに変更または中止されることがありますので、あらかじめご了承下さい。

更新履歴

2010-11-01	1.0	
		初版

文書情報

発行元：　株式会社フォティーンフォティ技術研究所

連絡先：　株式会社フォティーンフォティ技術研究所

sales@fourteenforty.jp

〒162-0808

東京都新宿区天神町 8 番地　神楽坂 U ビル 2 階

1. 概要

マイクロソフトは 2010 年 9 月、Enhanced Mitigation Experience Toolkit 2.0 (以降、EMET) をリリースした。EMET は Windows XP, Vista, 7 や Windows Server 2003, 2008 向けにいくつかのセキュリティ上の防御機能を提供している。EMET の特徴は既存のプログラムに対して、再コンパイルの必要なしに防御手機能を提供し、プロセス単位でその有効/無効を設定できるといった点である。

EMET はそのうちの 1 つの防御機能として、SEHOP を提供している。SEHOP は Windows Vista SP1 から取り入れられた防御機能であり、Windows XP には本来備わっていない機能であるが、EMET はこの SEHOP を Windows XP にも提供している。リサーチペーパー「SEH オーバーライトとその Exploit 可能性[1]」では、SEHOP の効果について ASLR を併用しない場合、ほとんど効果がなくなってしまうことを指摘したが、Windows XP では ASLR が提供されていないため、EMET が SEHOP を提供してもその効果が得られるか疑問がある。

本書では、この SEHOP の効果について調査し、その問題点と、改良法について整理した。まず、EMET, SEHOP に関する技術情報を示し、その後 EMET による SEHOP の内部動作を説明し、その効果の問題点と改良方法について提案する。

なお、特に断りがない限り、本書では Windows XP SP3 (x86)を適用したシステムを対象としている。また EMET.dll のバージョンは 2.0.0. 1 のものを用いた。

2. EMET 概要

2.1. 機能概要

EMET は現在、以下の 6 つの防御機能を提供しており、GUI からアプリケーションごとにその有効・無効を設定することができる。

SEHOP

バッファオーバーフローを利用した攻撃に対抗するための防御機能。Windows Vista SP1 から提供されているが、Windows XP には提供されていない。EMET によって Windows XP でも利用できるようになる。

DEP

データ領域の実行防止機能。Windows XP からその機能は提供されているが、EMET によってアプリケーションごとにその有効・無効を設定できるようになる。

HeapSpray Allocations

ASLR 回避などのために利用される HeapSpray 攻撃に対する防御機能。Windows にはもともと存在しない防御機能。

Null page allocation

Null デリファレンスに対する防御機能。Windows にはもともと存在しない防御機能。

Mandatory Address Space Layout Randomization

モジュールのロードアドレスを強制的にランダマイズし、メモリロケーションを予

測しにくくする防御機能。ASLR は Windows Vista から提供されているが、EMET は ASLR 非対応のモジュールも強制的にランダマイズする。この機能は Windows XP, Windows Server 2003 では EMET を適用しても有効にならない。

Export Address Table Access Filtering (EAF)

EAT(Export Address Table)の読み書きを制限することで、シェルコードが API のアドレスを取得するのを防止する防御機能。Windows にはもともと存在しない防御機能。

2.2. 動作概要

EMET の動作詳細については、特に詳しく説明されていないが、簡単な解析をすることで Application Compatibility Database[4]と呼ばれる Windows が持っている互換性維持のための仕組みの上に構築されていることが分かる。

Windows はプロセス開始時に、そのプロセスが互換性の問題で特別な処理が必要だと判断した場合、登録しておいた DLL をロードするという機能を持っている。EMET はこの機能を利用し、対象プログラムがロードされると、自身の DLL（EMET.dll）がプロセス内にロードされ動作するように登録している。

3. SEH オーバーライトと SEHOP

3.1. SEH オーバーライト

SEH (Structured Exception Handling)オーバーライトはスタックオーバーフローを利用した主要な攻撃手法の 1 つであり、EXCEPTION_REGISTRATION_RECORD(以降、ERR)と呼ばれるスタック上の構造を利用しその実行の制御を奪うというものである。ERR は 8 バイトの構造体で、_next メンバと_handler メンバを持っている。SEH は、この構造体をスタック上に複数作成し、リスト構造（SEH チェーン）を構築することで、実現されている(図 3-1)。

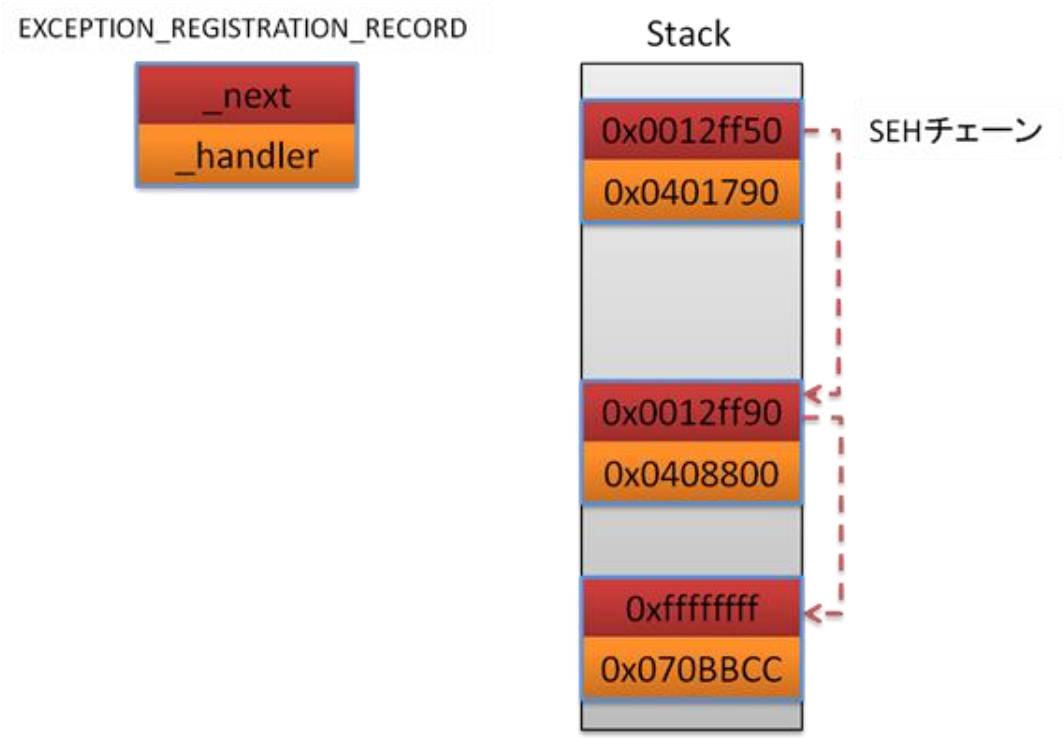

図 3-1 スタック上の SEH チェーン

例外が発生したとき、Windows はこのリスト構造を順番に確認しながら、_handler メンバ例外ハンドラとして呼び出していく。そのため、バッファオーバーフローにより ERR が書き換えられた場合、Windows は書き換えられたアドレスを例外ハンドラとして実行してしまう。

この攻撃手法に対していくつかの防御機能が提案され、Windows で実装されているが、そのうちの 1 つに SEHOP がある。

3.2. SEHOP による防御

SEHOP(Structured Exception Handling Overwrite Protection)は、Windows Vista SP1 から実装された SEH オーバーライトに対する防御機能である。

SEHOP は SEH チェーンの最後に、ntdll.dll 内の FinalExceptionHandler のアドレスを _handler メンバに持った ERR を挿入することで、攻撃者が SEH チェーンに何らかの変更を加えたかどうかを例外発生時に確認する方法をとっている。SEH オーバーライトでは、バッファオーバーフローを利用して、ERR の値を書き換えるため、SEH チェーンが途中で途切れてしまう（図 3-2）。そのため例外ハンドラ実行時に、SEH チェーンが FilanExceptionHandler を持たない ERR で終わっているなら、Windows は例外ハンドラの実行を行わないようにすることで、この攻撃から防御する。

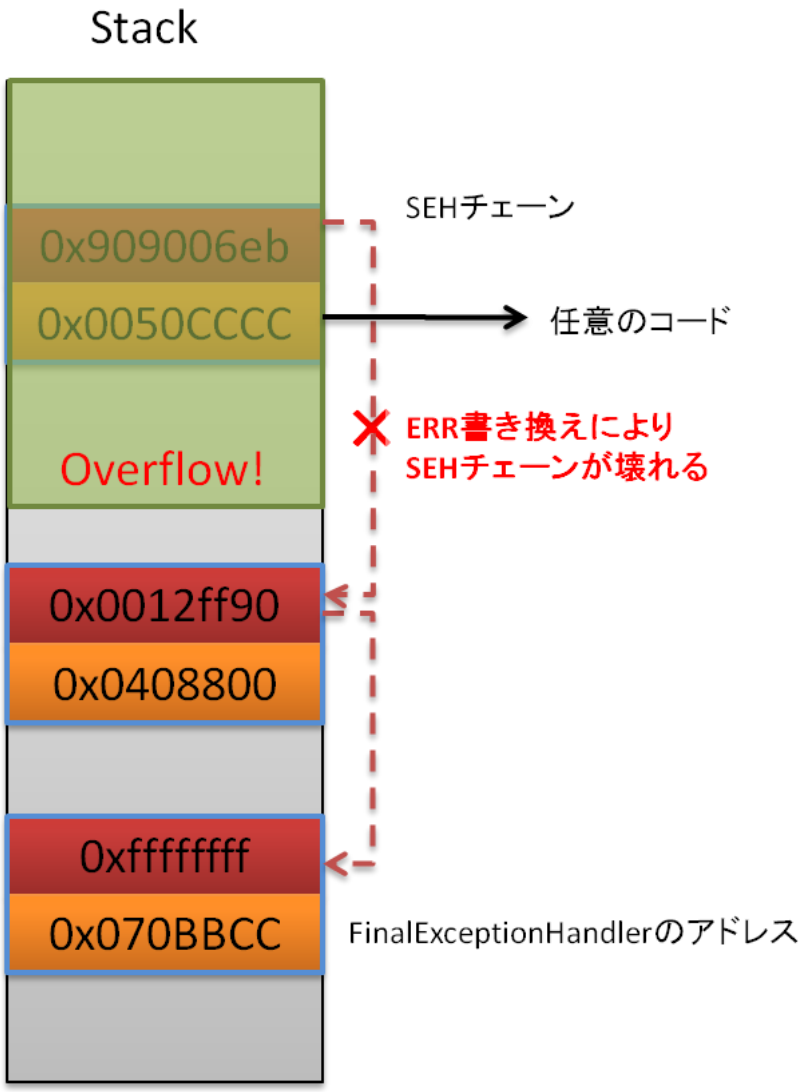

図 3-2 SEHOP の仕組み

しかし、「SEH オーバーライトとその Exploit 可能性[1]」で指摘したように、ASLR が有効でないシステムの場合、SEH チェーンの再構築という手法を用いて、この SEHOP を回避して攻撃することが可能である。SEHOP は Windows Vista SP1 以降ではもともと提供されているものなので、EMET がこれを提供する理由としては Windows Vista(SP なし)や Windows XP, Windows Server 2003 で利用できるという点が大きいが、Windows XP,

Windows Server 2003 では ASLR は実装されておらずその効果について疑問がある。

4. EMET による SEHOP 実装

前章で Windows にもともと実装されている SEHOP について説明したが、ここで EMET による SEHOP がどのように動作するか説明する。

EMET によって SEHOP が有効になったプロセスでは各スレッドスタックに対して EMET.dll によって、いくつかの処理が行われる。

EMET.dll はスレッドスタック上の SEH チェーンの最後に新しく作成した ERR をリンクする。通常 ERR はスタック上に作成されるが、EMET は自らスタック外のメモリ上に確保した領域にこの新しい ERR を作成し、それをリンクする。（図 4-1）

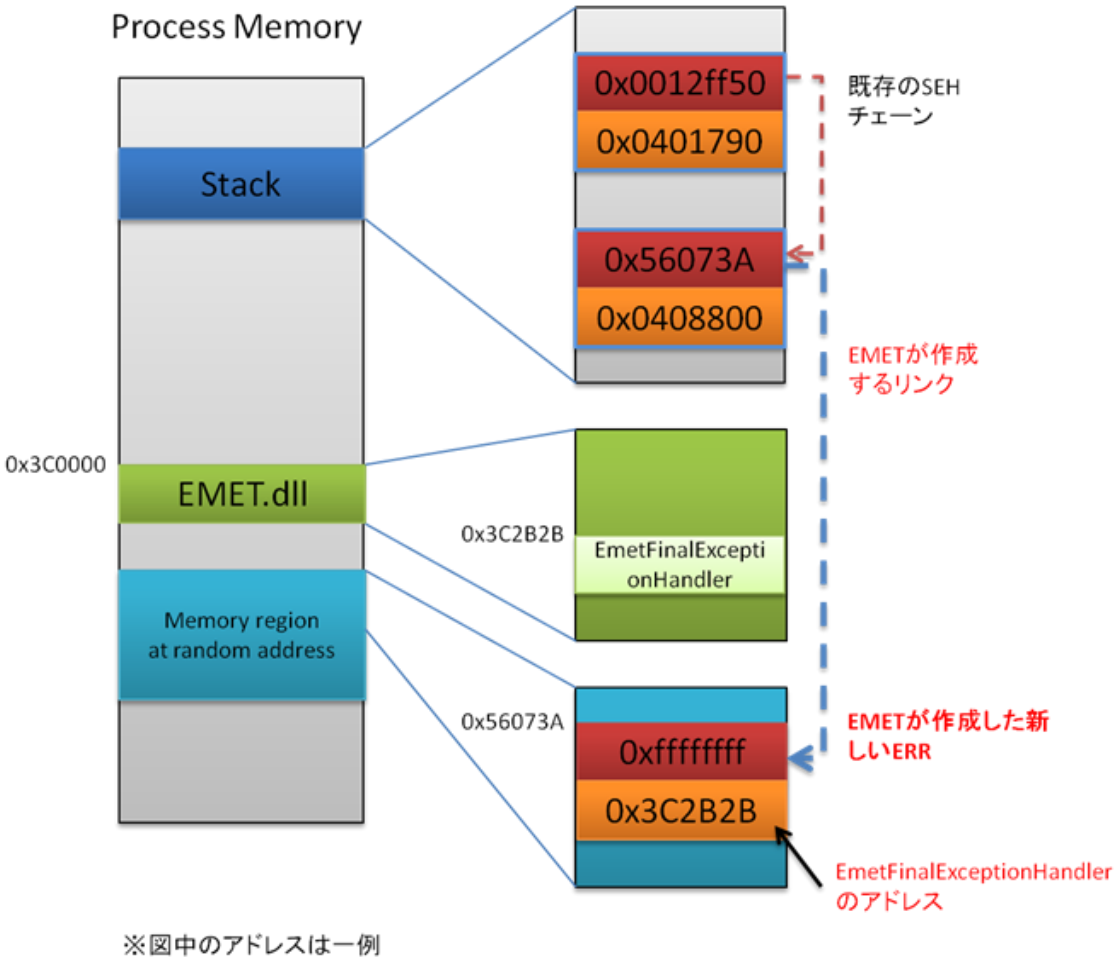

図 4-1 EMET の SEHOP 実装

この図では EMET が作成した新しい ERR の存在しているアドレスは 0x56073A としているが、この値はランダマイズされており、プロセス起動のたびに異なる。この ERR の_handler メンバは EMET.dll 内の特定のアドレス(オフセット 0x2B2B)を指している（以降、このアドレスを EmetFinalExceptionHandler と呼ぶ）。また、この_next メンバは SEH チェーンの最後を表す 0xFFFFFFFF となっている。

EMET は例外が発生すると、SEH による例外処理の前にこの SEH チェーンをチェックし、

SEH チェーンの最後の ERR の_handler メンバが EmetFinalExceptionHandler のアドレスを持っているかを調べることで SEHOP を実装している。

5. EMET の SEHOP 回避

EMET による SEHOP はこれまで説明したように動作するが、SEHOP によるチェックを回避できるように SEH チェーンを再構築できれば、攻撃を成功させることができる。

5.1. EmetFinalExceptionHandler を持つ ERR の作成

Windows XP や Windows Server 2003 における EMET の SEHOP 実装の 1 つの問題点は ASLR によって EMET.dll のロードアドレスがランダマイズされない点にある。そのため EmetFinalExceptionHandler の値が予測し易くなっている。実際のロードアドレスはプロセスごとに異なるが、プロセスが同じであれば毎回同じアドレスにロードされるため、攻撃はしやすくなる。

表 1 は Windows XP SP3 において、calc.exe と Adobe Reader 9.4.0 を起動したときの EMET 関連のアドレスを示したものである。これらのアドレスはプロセス起動毎に同じアドレスである。

表 1 プロセスごとの EMET 関連アドレス

	calc.exe	Adobe Reader 9.4.0
EMET.dll base address	0x430000	0x3C0000
Address of EmetFinalExceptionHandler	0x432B2B	0x3C2B2B

これらのアドレスが分かっていることで、バッファオーバーフローを利用して最後の ERR が EmetFinalExceptionHandler を持つような SEH チェーンを再構築することができる。具

体的には以下のように SEH チェーンを再構築すればよい。

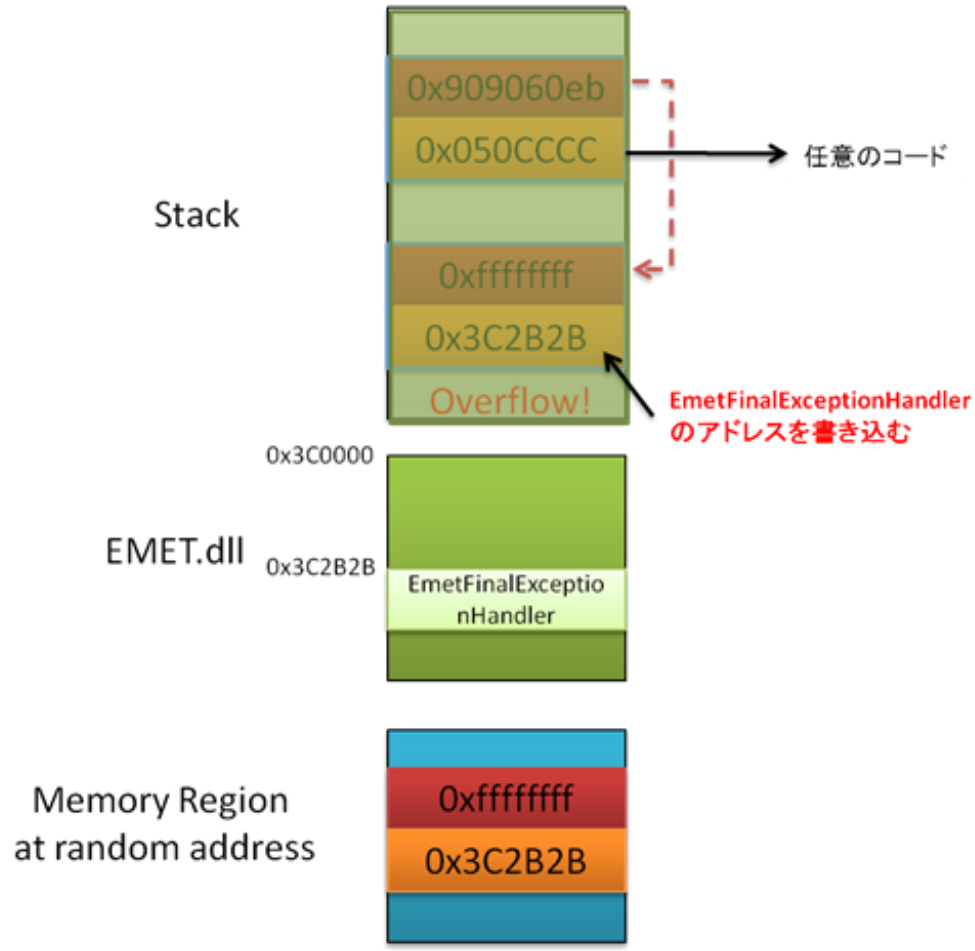

図 5-1 EmetFinalExceptionHandler を直接持つ SEH チェーンの再構築

この方法は ASLR が有効になっていない Windows Vista SP1 や Windows 7 の SEHOP 回避の方法と同じである。

しかし EMET の SEHOP 実装はこれだけでは回避できない。このような SEH チェーンを構築して、例外をディスパッチしようとすると例外ハンドラは実行されない。これは、EMET が SEH チェーンの中に自らが作成した ERR を持たない場合に、実行させないようにしているた

めだと考えられる。[1]

5.2. EMET の持つアドレス情報を用いた SEH チェーン再構築

すでに説明したように、EMET による SEHOP 有効時の SEH チェーンの最後の ERR はスタック上には存在せず、EMET が確保した領域に存在している。この ERR を直接上書きした SEH チェーンにリンクすることができれば、EMET による SEHOP を回避することができる。ただし、「4 EMET による SEHOP 実装」でも説明したように、この最後の ERR はランダムなページのランダムなアドレスに確保されるため、アドレスの予測が困難になっている。そのため直接この ERR を SEH チェーンの最後にリンクさせることは難しい(図 5-2)。

[1] これは動作の観察から得られた結論で、リバースエンジニアリングにより確認したものではない。この後の説明で分かるように、この条件を満たすことで例外ハンドラが呼び出されるようになるという事実から得られた結論である。

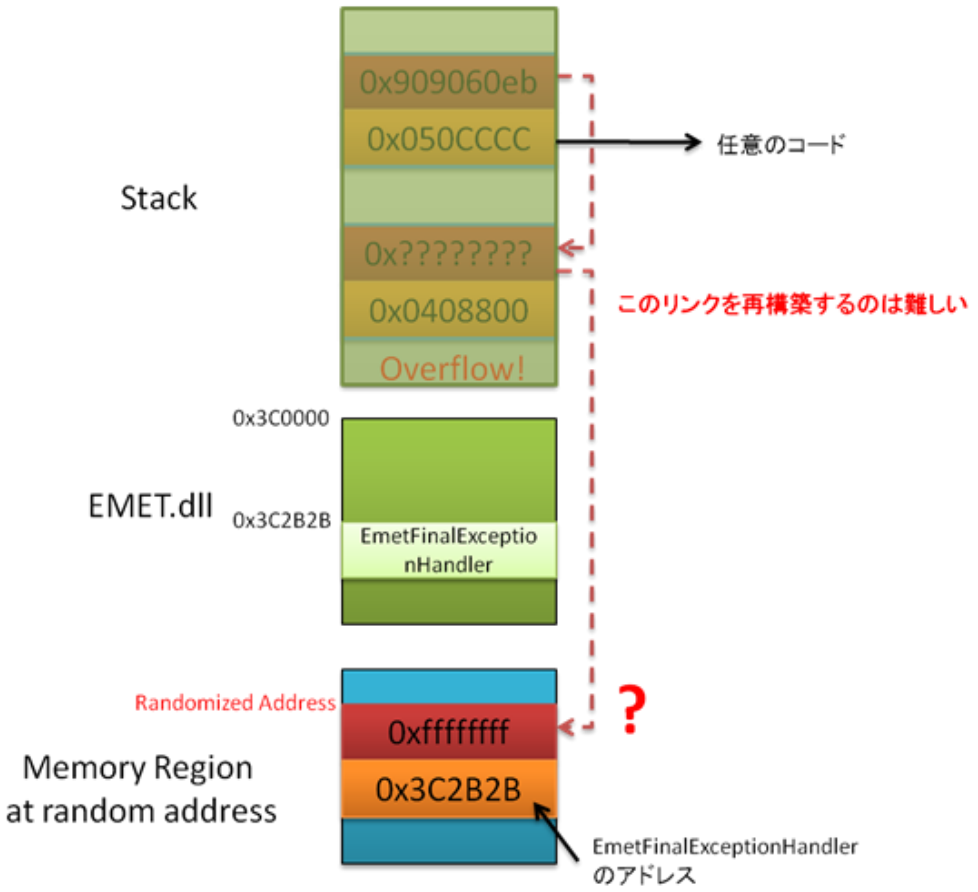

図 5-2 EMET が作成する ERR へのリンク再構築

しかし、ここで EMET の SEHOP 実装のもう一つの問題点によりこのリンクを作成することができるようになる。もう 1 つの問題点は、この最後のランダマイズした ERR のアドレスをある固定アドレスに持ってしまっていることである。EMET.dll は内部に"almostro"と呼ばれるセクションを持っている。このセクションは EMET.dll のベースアドレスからのオフセット 0xC000 から存在しているが、この先頭部分にこの ERR のアドレスが保存されている。EMET.dll 自体は固定アドレスにロードされるため、"almostro"の先頭アドレスも予測できる。"almostro"セクションの残りの部分は 0 で埋められている（図 5-3)。

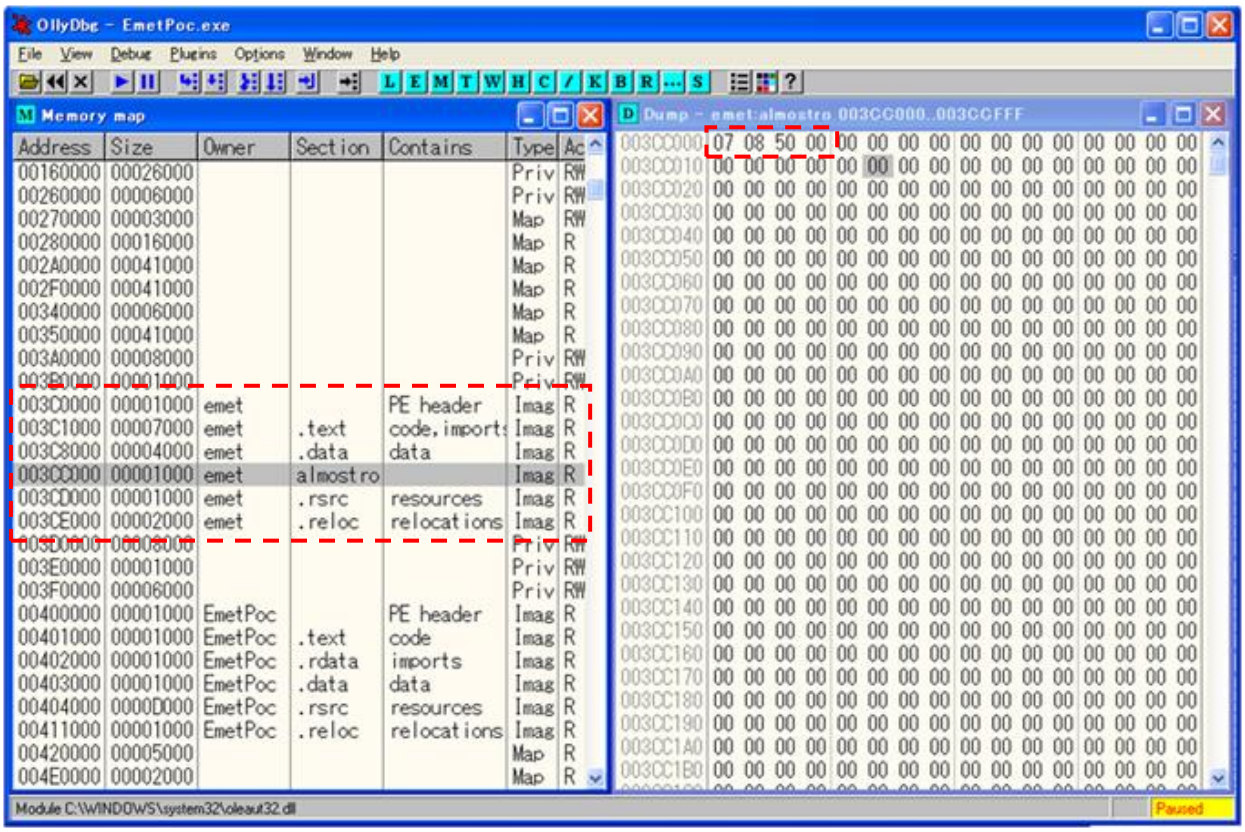

図 5-3 EMET.dll の"almostro" セクション

このことを利用し、バッファオーバーフローによる SEH チェーン再構築時に、"almostro" の先頭アドレスを ERR とみなし、SEH チェーンの最後にリンクした場合、図 5-4 のような SEH チェーンが構築できる。

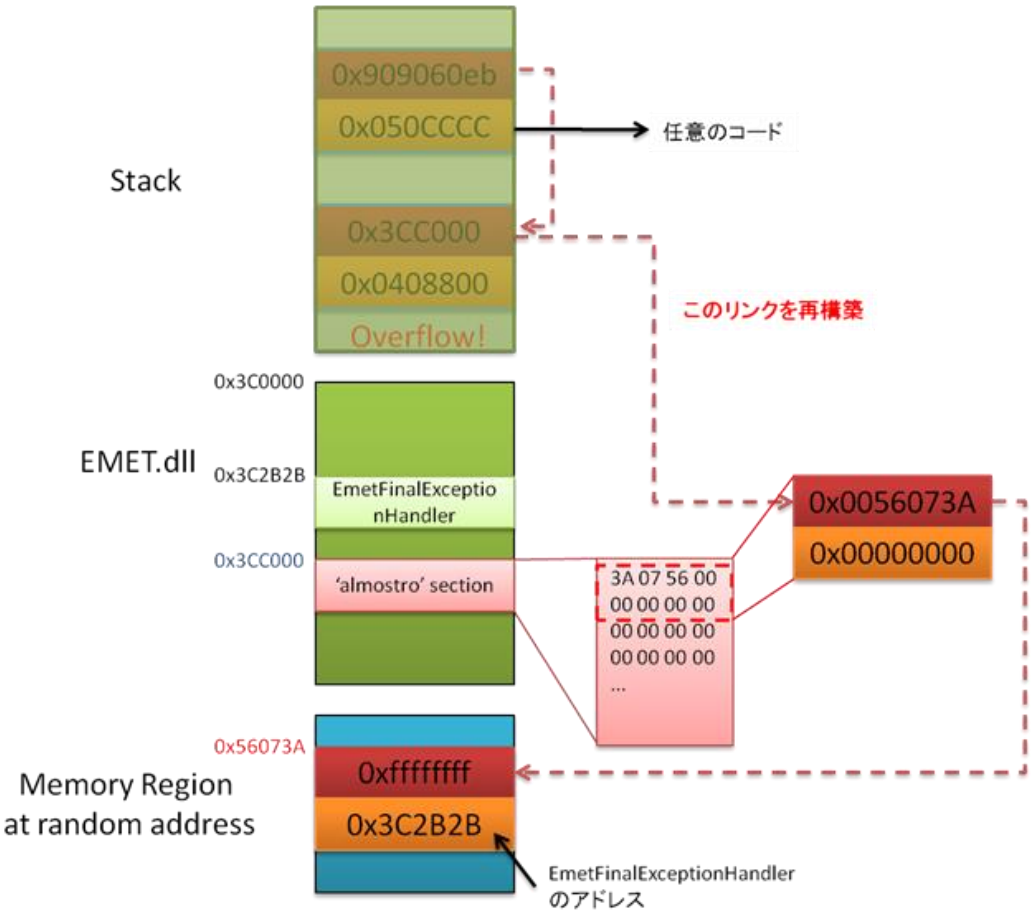

図 5-4 almostro セクションの先頭を ERR とみなしてリンクした SEH チェーン

このようにすると、ランダマイズされている EMET が作成する ERR を SEH チェーンにリンクすることができる。そのため、EMET は例外処理の際、問題なく最後の自ら作成した ERR を SEH チェーンの中に見つけることができ、この SEH チェーンを問題ないと判定してしまうため、SEHOP を回避することができる。

EMET は全てのスレッドで 1 つの自ら作成した ERR と EmetFinalExceptionHandler を共有しているため、スレッドによってこれらの SEH チェーンの再構築に用いるアドレスが変わることはない。

これを実際に行うサンプルコードを「付録 A サンプルコード」に示した。このコードは意図的にバッファオーバーフローを発生させ、スタック上の SEH チェーンの最初の ERR を上書きし、その_next メンバを EMET.dll の'almostro'セクションの先頭に、_handler メンバをこのコード内の msg()関数を指すように書き換える。このようにすることで EMET の SEHOP が有効でも msg()関数が呼ばれることが分かる。

サンプルコードは Visual Studio 2008 SP1 で/Od (最適化 Off)と/SAFESEH:NO のオプションを指定してコンパイルした。Windows XP SP3 上で、全ての EMET の防御機能を有効にしてテストした。

このコードを実行して、バッファオーバーフローが起き、例外が発生した直後の状態を図 5-5 に示した。画面左上に、SEH チェーン内の ERR のリストが表示されており、最初の ERR の例外ハンドラが msg()関数を指ており、リストは EMET.dll の用意した ERR へとつながっていることが分かる。画面右下のリストはスタックの状態であり、赤線で囲まれた部分が SEH チェーンの最初の ERR で、その_next メンバが'almostro'セクションの先頭アドレス（0x3CC000）になっている。

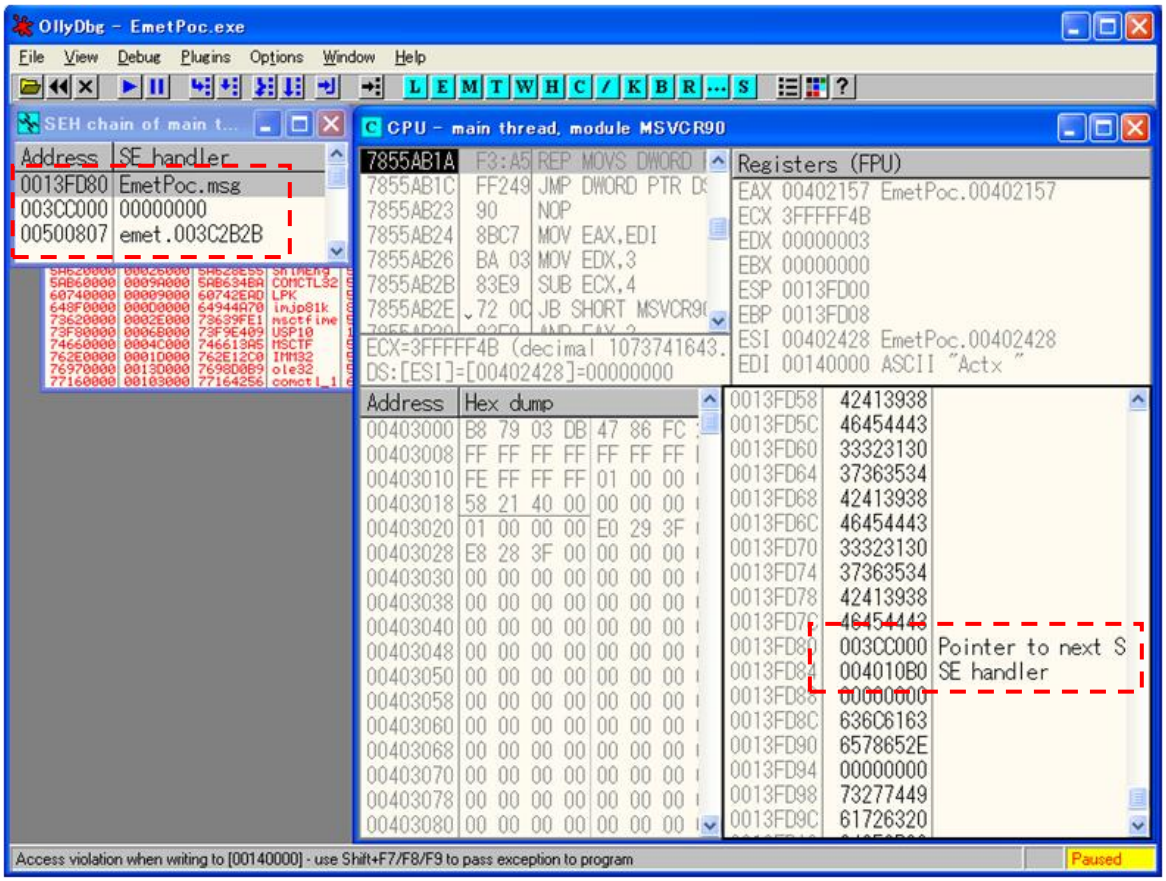

図 5-5 バッファオーバーフローにより SEH チェーンが再構築される様子

このサンプルコードでは EMET の SEHOP のバイパスの様子を分かりやすく見せるため、単純に msg()関数を呼び出すだけになっているが、「SEH オーバーライトとその Exploit 可能性[1]」で紹介したようなさまざまな手法を用いると、DEP や SafeSEH なども同時に回避できる可能性がある。

6. SEHOP 実装の改良方法の提案

現在の EMET の SEHOP は、その実装が分かってしまうと、バイパスできてしまう可能性がある。これは EMET が固定的なアドレスに自らが作成した ERR のアドレスを保存しているためである。

考えられる改良方法の 1 つは、そのアドレスをどこにも保存しないことである。しかし、この場合、新たなスレッドが作られた際に、既存の SEH チェーンの最後の ERR の_next メンバに設定する値を保持していないことになるので、スレッドごとに新たに ERR を作成する必要がある。

もう一つの方法は、EMET が作成する ERR のアドレスを 0xFFFFFFFF などとの XOR を取って保存しておくことである。このような形にしておくことで、SEH チェーンに直接リンクされることを防ぐことができる。そのアドレスが必要になったときには、再度 0xFFFFFFFF との XOR を取って計算すればよい。

7. まとめ

EMET が提供する SEHOP は、Windows XP や Windows Server 2003 でも利用できるという利点があるが、その効果が弱い実装となっている。既存のマルウェアなどからプロセスを保護するという点、また攻撃成功の確率を下げるという点では、現在の実装でも有効ではあるが、より攻撃を難しくする実装が可能であり、新たな攻撃の可能性を取り除くような改良が求められる。

8. 参考文献

[1] SEH オーバーライトとその Exploit 可能性

http://www.fourteenforty.jp/research/research_papers/SEH_Overwrite.pdf

http://www.fourteenforty.jp/research/research_papers/SEH%20Overwrite_CanSecWest2010.pps

[2] Announcing the upcoming release of EMET v2

http://blogs.technet.com/b/srd/archive/2010/07/28/announcing-the-upcoming-release-of-emet-v2.aspx

[3] How EMET works

http://0xdabbad00.com/2010/09/12/how-emet-works/

[4] Secrets of the Application Compatibility Database (SDB) – Part 1, 2, 3

http://www.alex-ionescu.com/?p=39

http://www.alex-ionescu.com/?p=40

http://www.alex-ionescu.com/?p=41

[5] Application Compatibility Database

http://msdn.microsoft.com/en-us/library/bb432182(v=VS.85).aspx

付録 A.　サンプルコード

```
// This code demonstrates how EMET's SEHOP is bypassed.
// This code assumes that EMET.dll is loaded at the address of 0x3C0000

// I compiled this code with Visual Studio 2008 SP1 with /Od (no optimization) and /SAFESEH:NO
options.
// I tested this code on Windows XP SP3.

// This program is really simple. It's based on "Win32 Windows Program" skelton generated by Visual
Studio 2008's project wizard.
// When you left click on the window vulnerable_func() will be called.

#include "stdafx.h"
#include "EmetPoc.h"

const char * user_input =
"0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF0123456789ABCDEF" //Padding 64byte
"0123456789ABCDEF" // Another padding 16byte
"¥x00¥xC0¥x3C¥x00" // Overwrite _next member of ERR on the stack. ( Points head of 'almostro'
section in EMET.dll)
"¥xB0¥x10¥x40¥x00" // Address to execute ( msg function ).
;

// In this function, an exception occurs by buffer overflow.
int vulnerable_func( const char *src , int size){
    __try{
        char buf[64];
              if( size < 64)
                      memcpy( buf ,src , size); // Exploitable!
    }
    __except( EXCEPTION_EXECUTE_HANDLER ){
        return 1;
    }
    return 0;
}

// Function to call as SEHandler. Just for demonstration.
void msg(){
    WinExec("calc.exe" , SW_SHOW );
      MessageBoxA( 0 , "It's cracked" , "It's cracked" , MB_OK );
    ExitProcess(0);
```

```
}

#define MAX_LOADSTRING 100
HINSTANCE hInst;
TCHAR szTitle[MAX_LOADSTRING];
TCHAR szWindowClass[MAX_LOADSTRING];

ATOM                    MyRegisterClass(HINSTANCE hInstance);
BOOL                    InitInstance(HINSTANCE, int);
LRESULT CALLBACK        WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK        About(HWND, UINT, WPARAM, LPARAM);


int APIENTRY _tWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int
nCmdShow)
{
... // Here is the same code generated by VS2008
}

// This function is almost the same code generated by VS2008 except the code calling
vulnerable_func() on LBUTTONDOWN.
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
       int wmId, wmEvent;
       PAINTSTRUCT ps;
       HDC hdc;

       switch (message)
       {
       case WM_COMMAND:
                wmId    = LOWORD(wParam);
                wmEvent = HIWORD(wParam);
                switch (wmId)
                {
                case IDM_ABOUT:
                        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                        break;
                case IDM_EXIT:
                        DestroyWindow(hWnd);
                        break;
                default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
                break;
       case WM_PAINT:
                hdc = BeginPaint(hWnd, &ps);
```

```
                EndPaint(hWnd, &ps);
                break;
        case WM_DESTROY:
                PostQuitMessage(0);
                break;
        case WM_LBUTTONDOWN:
                vulnerable_func( user_input , -1);    // calling vulnerable_func to cause buffer
overflow intentionally.
          break;
        default:
                return DefWindowProc(hWnd, message, wParam, lParam);
                msg(); // This is never called. Just for msg() not being disapeared by a compiler.
        }
        return 0;
}


ATOM MyRegisterClass(HINSTANCE hInstance)
{
... // Here is the same code generated by VS2008
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
... // Here is the same code generated by VS2008
}

INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
... // Here is the same code generated by VS2008
}
```

コード 8-1 EMET の SEHOP バイパス