"FFR EXCALOC"

ー コンパイラのセキュリティ機能に基づいたExploitabilityの数値化 ー

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティ技術研究所
http://www.fourteenforty.jp

シニアソフトウェアエンジニア
石山智祥

はじめに

- 最近のコンパイラには、セキュリティを強化する機能が追加されている
- しかし、市場に流通しているソフトウェアには、コンパイラのセキュリティ機能が利用されていないケースが多い
- そのため、従来の手法でExploit可能となる脆弱性が多く発見されている
- 今回、実行ファイルを解析し適用されているセキュリティ機能を検出し、Exploitabilityを数値化することに挑戦

Agenda

1. コンパイラ, OSの提供するセキュリティ機能と実装
2. Exploitability計算のための特徴パラメータ検出
3. FFR EXCALOCを用いたExploitabilityの数値化
4. 今後の課題

1. コンパイラ, OSの提供するセキュリティ機能と実装

FFR

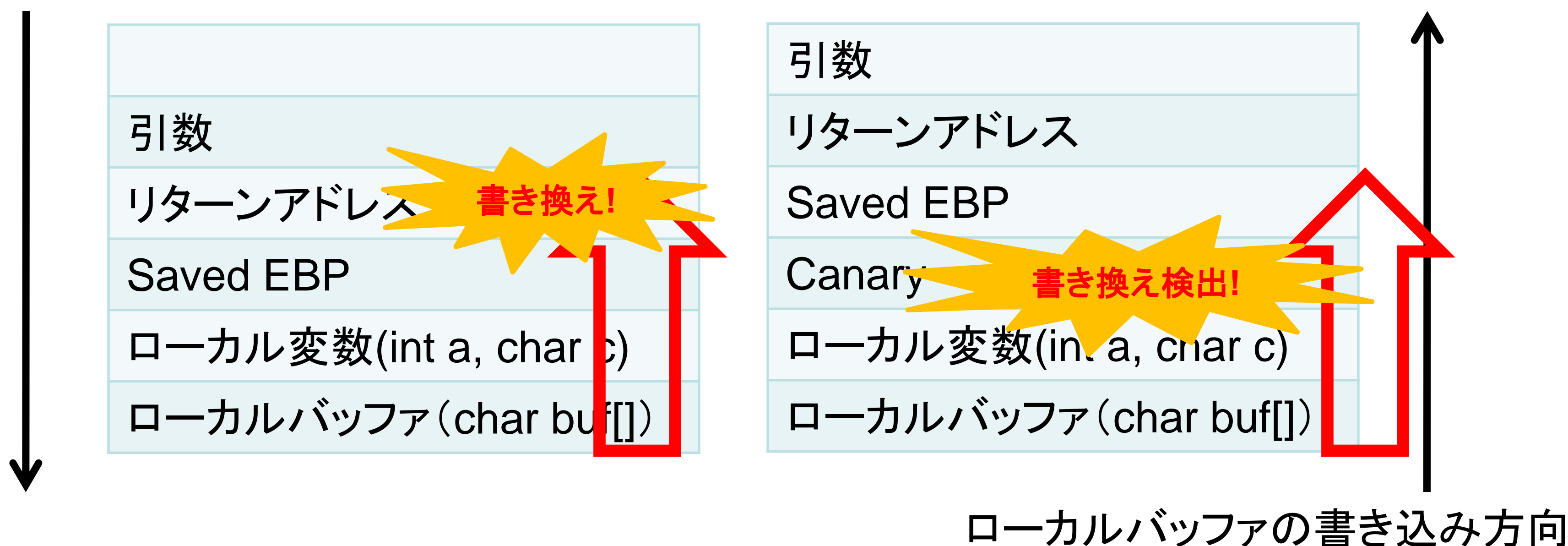
Visual C++（2005以降）

- /GSオプション
  - Canaryを使用したローカルバッファのオーバーフロー検出
  - Visual Studio .NETでは回避方法が存在したが、2005で改良された

- 最適化によるセキュリティを考慮したスタックレイアウト
  - ポインタ変数やポインタ引数をレジスタを使用するように最適化
  - これにより、ポインタを経由したメモリ上へのデータ書き込みを防ぐ

Visual C++ /GS

- 関数リターン時に、Canaryが書き換えられているか確認し、オーバーフローを検出



FFR

Visual C++ /GS

```
int vuln(char *arg)
{
    char buf[128];

    strcpy(buf, arg);

    printf("buf = %s¥n", buf);

    return 0;
}
```

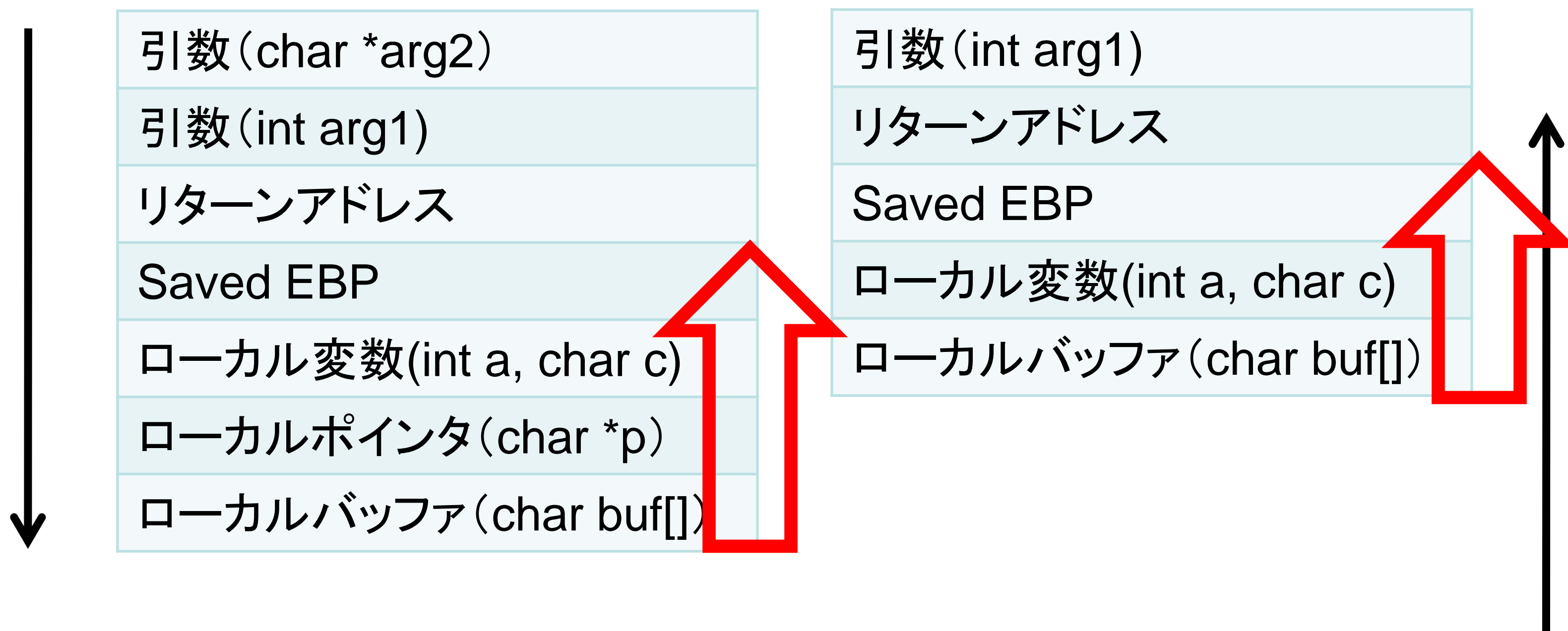
```
_vuln           proc near               ; CODE XREF:
                                        ; _main+29↓p

var_84          = byte ptr -84h
var_4           = dword ptr -4
arg_0           = dword ptr  4

                sub     esp, 84h
                mov     eax, __security_cookie
                xor     eax, esp
                mov     [esp+84h+var_4], eax
                mov     eax, [esp+84h+arg_0]
                lea     edx, [esp+84h+var_84]
                sub     edx, eax

loc_401020:                             ; CODE XREF:
                mov     cl, [eax]
                mov     [edx+eax], cl
                inc     eax
                test    cl, cl
                jnz     short loc_401020
                lea     eax, [esp+84h+var_84]
                push    eax
                push    offset aBufS
                call    ds:__imp__printf
                mov     ecx, [esp+8Ch+var_4]
                add     esp, 8
                xor     ecx, esp
                xor     eax, eax
                call    @__security_check_cookie@4 ;
                add     esp, 84h
                retn
_vuln           endp
```

Canary（cookie）を設定

Canary（cookie）を確認

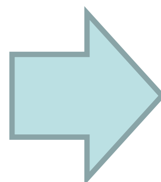
Visual C++　最適化によるStack Layout

- ローカルバッファの後ろに存在するローカルポインタをレジスタを使用し、スタック上に領域を設けない
- ポインタを渡す引数をレジスタでの引数として渡す

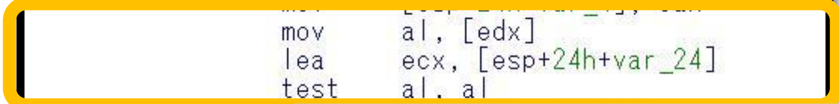


Visual C++ 最適化によるStack Layout

```c
void vuln(char *s, int l)
{
    char *a;
    char  buf[32];

    a = buf;
    while (*s != '¥0') {
        *a++ = *s++ + 1;
    }
    printf("buf = %s¥n", buf);

    return ;

}
```



```
_vuln           proc near               ; CODE XREF:

var_24          = byte ptr -24h
var_4           = dword ptr -4

                sub     esp, 24h
                mov     eax, ___security_cookie
                xor     eax, esp
                mov     [esp+24h+var_4], eax
                mov     al, [edx]
                lea     ecx, [esp+24h+var_24]
                test    al, al
                jz      short loc_401023

loc_401017:                             ; CODE XREF:
                inc     al
                inc     edx
                mov     [ecx], al
                mov     al, [edx]
                inc     ecx
                test    al, al
                jnz     short loc_401017

loc_401023:                             ; CODE XREF:
                lea     eax, [esp+24h+var_24]
                push    eax
                push    offset aBufS    ; "buf = %s¥
                call    ds:__imp__printf
                mov     ecx, [esp+2Ch+var_4]
                add     esp, 8
                xor     ecx, esp
                call    @__security_check_cookie@4 ;
                add     esp, 24h
                retn
_vuln           endp
```



ローカル変数をレジスタで使用

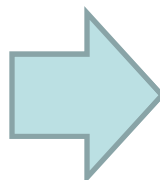
SSP（Stack Smashing Protector）

- スタックオーバーフローを検出し、任意コードの実行を抑制
  - IBM 江藤氏が考案したProPolice（*1）の再実装
  - GCC 4.1より正式に組み込まれている
  - Canaryを使用したオーバーフローチェック
  - Ideal Stack Layoutによるポインタ変数、ポインタ引数の保護

*1 http://www.trl.ibm.com/projects/security/ssp/main.html

SSP　－ Canary －

```c
void vuln(char *s, int l)
{
    int len;
    char buf[32];

    len = strlen(s);

    printf("length = %d, %d¥n",
            len, l);

    strcpy(buf, s);

    return ;

}
```



```
                public vuln
vuln            proc near               ; CODE XREF: main+71↓p

var_3C          = dword ptr -3Ch
src             = dword ptr -38h
var_2C          = dword ptr -2Ch
dest            = byte ptr -28h
var_8           = dword ptr -8
var_4           = dword ptr -4
arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch

                push    ebp
                mov     ebp, esp
                push    edi
                sub     esp, 44h
                mov     eax, [ebp+arg_0]
                mov     eax, large gs:14h
                mov     [ebp+var_8], eax
                mov     eax, [ebp+var_8]
                xor     eax, large gs:14h
                jz      short loc_8048470
                call    ___stack_chk_fail

loc_8048470:
                mov     edi, [ebp+var_4]
                leave
                retn
vuln            endp
```

Canaryを設定

Canaryを確認

SSP　－ Ideal Stack Layout －

- ローカルバッファの後ろにローカルポインタが存在しないようにレイアウト
- ポインタを渡す引数は、一度ローカル変数にコピーしてから使用する

スタックの成長方向

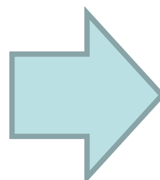
引数（char *arg2）
引数（int arg1)
リターンアドレス
Saved EBP
ローカル変数(int a, char c)
ローカルポインタ（char *p）
ローカルバッファ（char buf[]）

引数（char* arg2) Not use
引数（int arg1)
リターンアドレス
Saved EBP
ローカル変数(int a, char c)
ローカルバッファ（char buf[]）
ローカルポインタ（char *p）
コピーした引数（char *arg2）

ローカルバッファの書き込み方向

SSP　－ Ideal Stack Layout －

```
void vuln(char *s, int l)
{
    int len;
    char buf[32];

    len = strlen(s);

    printf("length = %d, %d¥n",
           len, l);

    strcpy(buf, s);

    return ;

}
```



```
                public vuln
vuln            proc near               ; CODE XREF: main+71↓p

var_3C          = dword ptr -3Ch
src             = dword ptr -38h
var_2C          = dword ptr -2Ch
dest            = byte ptr -28h
var_8           = dword ptr -8
var_4           = dword ptr -4
arg_0           = dword ptr  8
arg_4           = dword ptr  0Ch

                push    ebp
                mov     ebp, esp
                push    edi
                sub     esp, 44h
                mov     eax, [ebp+arg_0]
                mov     [ebp+src], eax
                mov     eax, large gs:14h
                mov     [ebp+var_8], eax
                xor     eax, eax
```

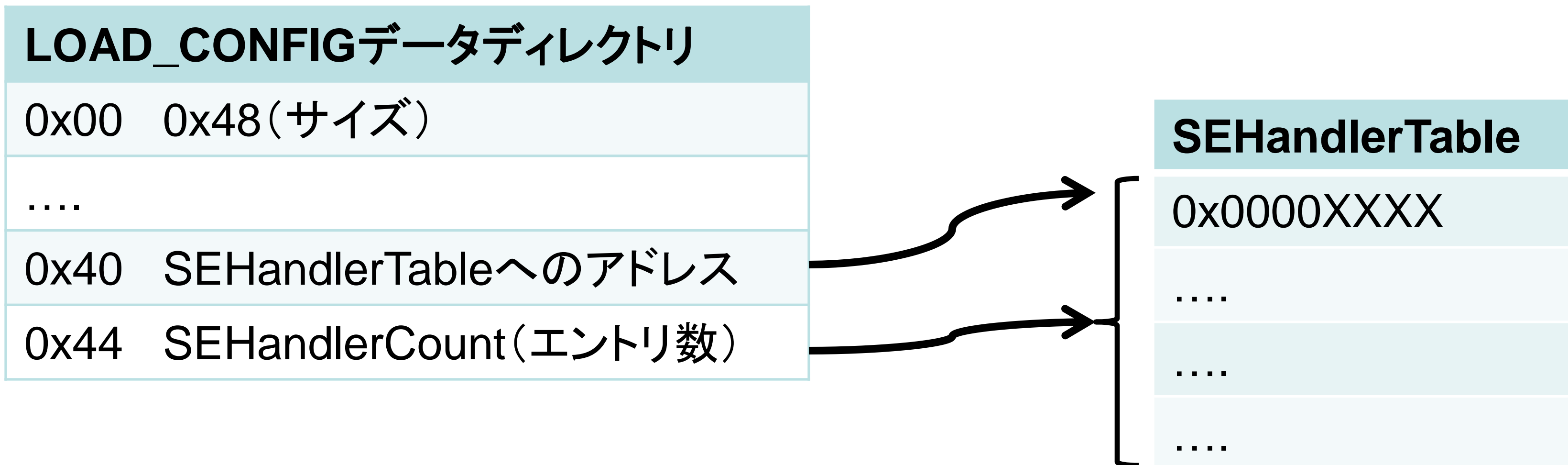
引数arg_0をsrcに代入

SafeSEH

- 実行可能な例外ハンドラをあらかじめテーブルに設定しておくことで、例外ハンドラを経由した任意コードの実行を防ぐ
  - Windows XP SP2から実装
  - 例外が発生した際に、KiUserExceptionDispatcher(ntdll.dll)内で、例外ハンドラテーブルをチェック
  - 例外ハンドラテーブルに含まれているハンドラであれば実行

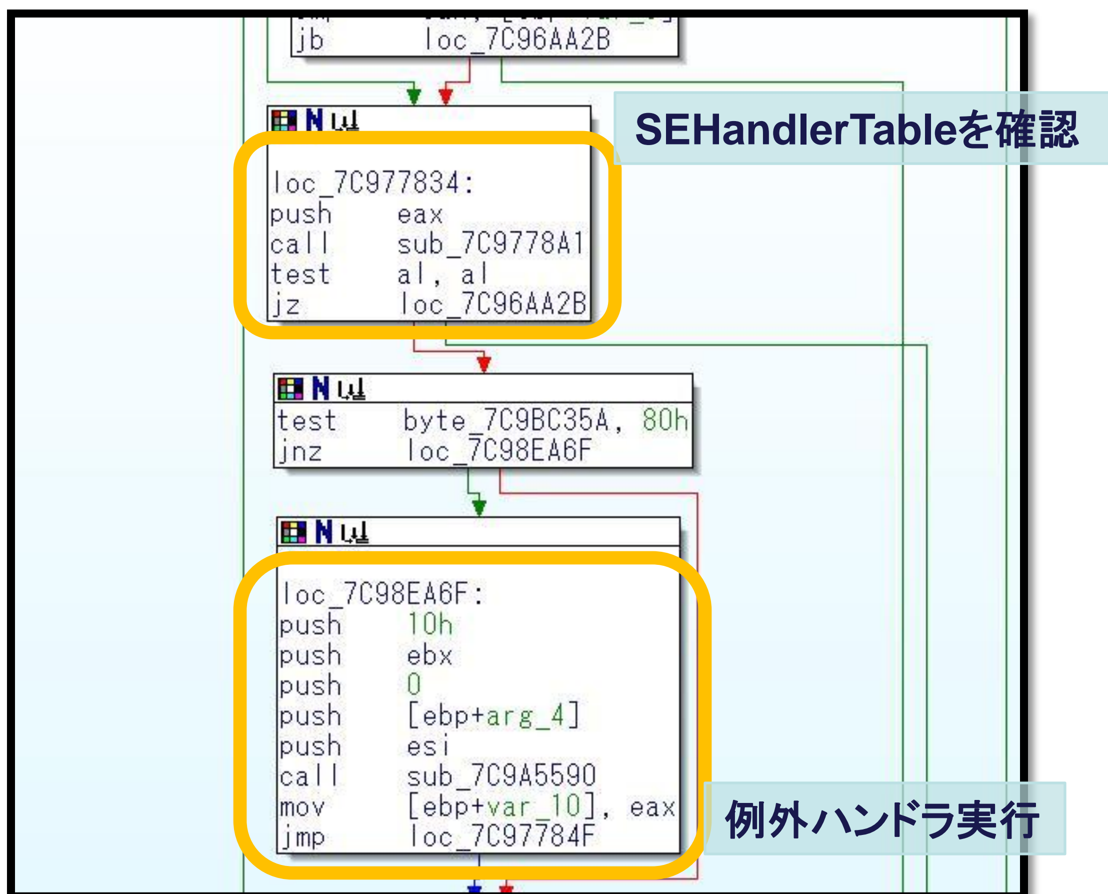
例外ハンドラテーブル（SEHandlerTable）

- 実行ファイルのLOAD_CONFIGデータディレクトリに存在



例外ハンドラテーブル（SEHandlerTable）

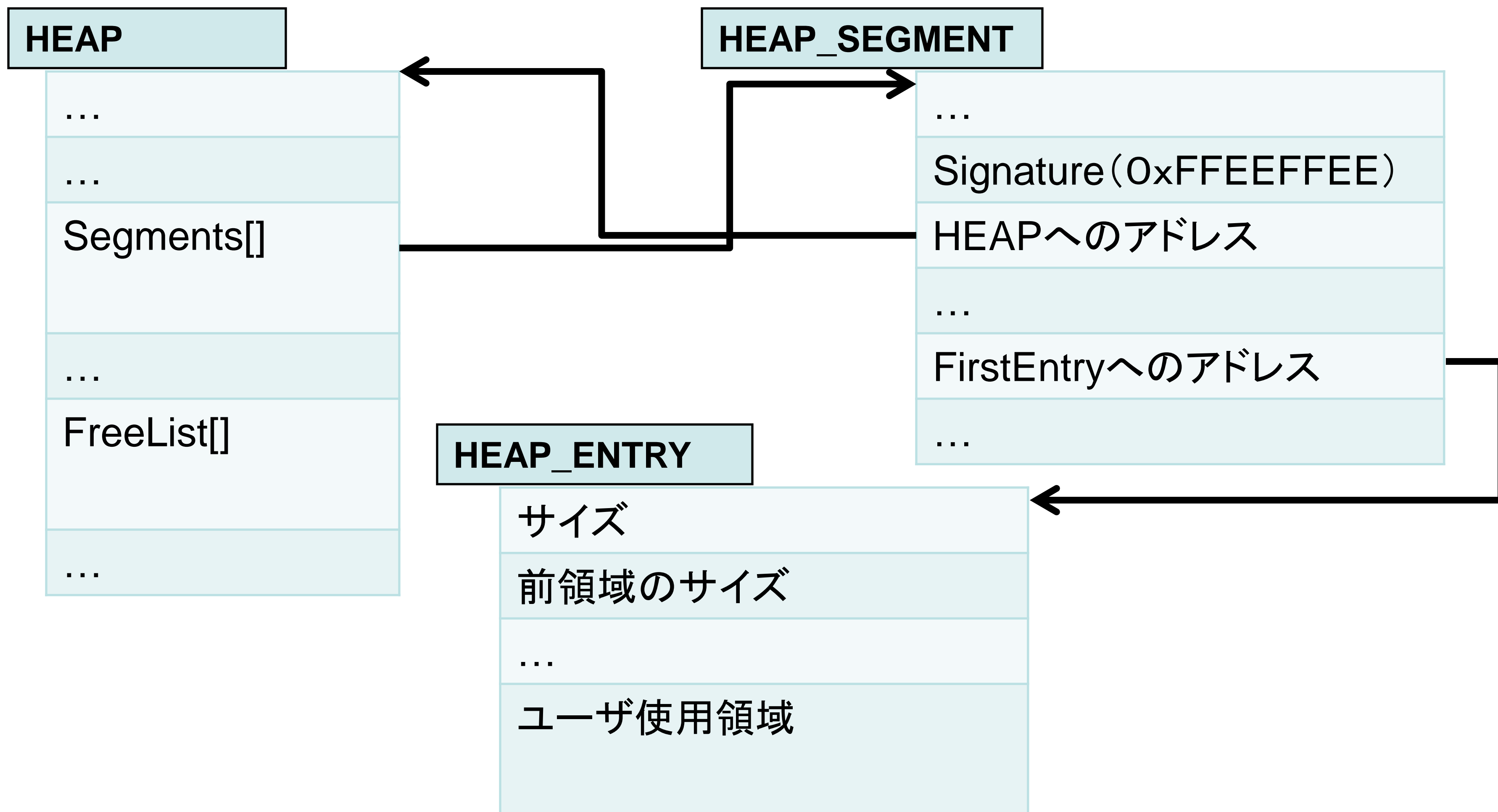
- SEHandlerTableのチェック



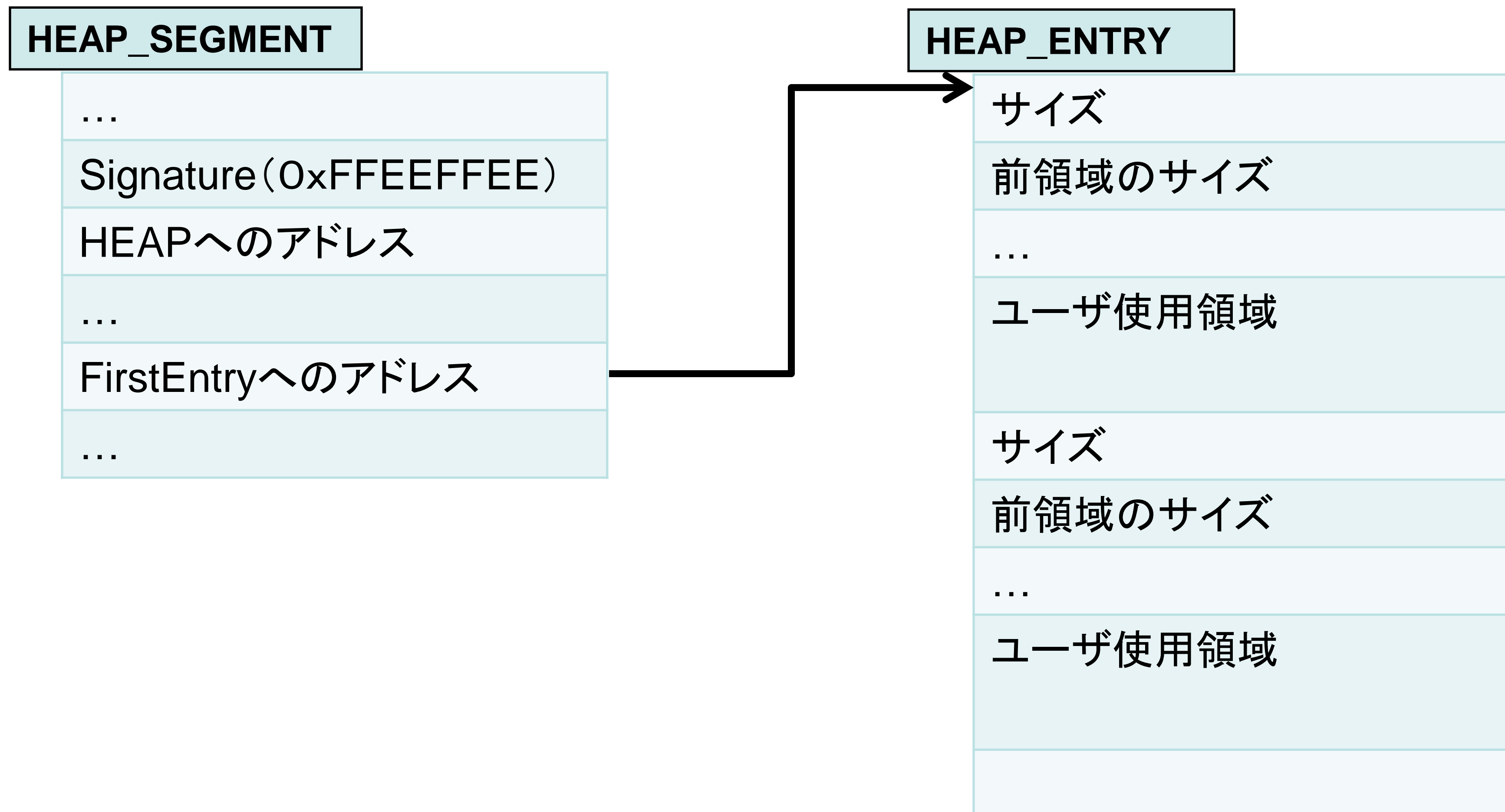
Heap Management （Windows（HeapAlloc/HeapFree））

- Windowsの提供するHeap Manager
  - HeapCreateを使用することで、Heap Managerを分けることが可能
  - 実装はntdll.dll内のRtlAllocateHeap/RtlFreeHeap
  - FreeListでDoubleLinkedListを使用
  - Windows XP SP2以降では、バウンダリチェックやSafe Unlinkといったセキュリティ機能が実装されている

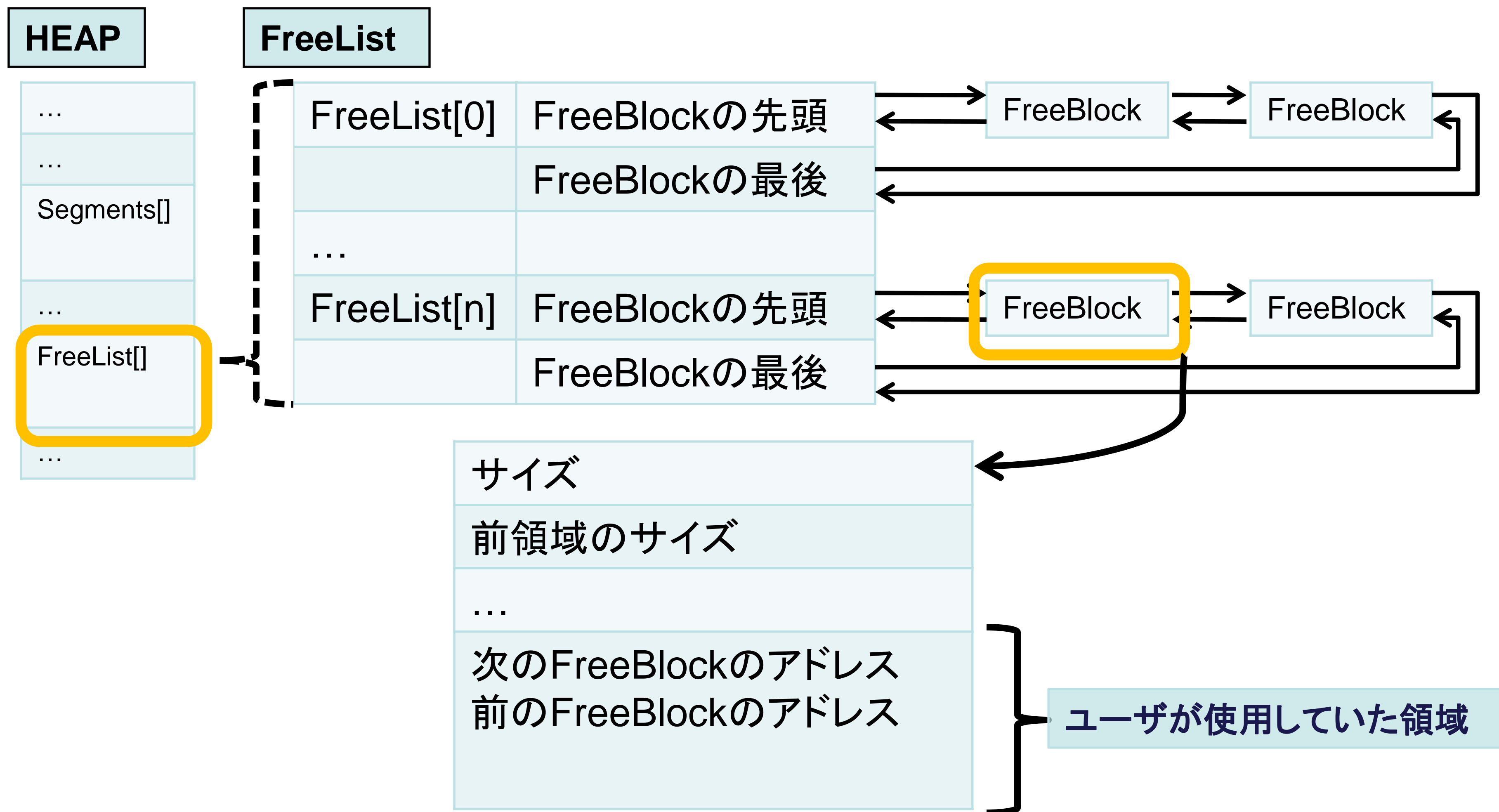
Heap Management（Windows（HeapAlloc/HeapFree））



Heap Management（Windows（HeapAlloc/HeapFree））



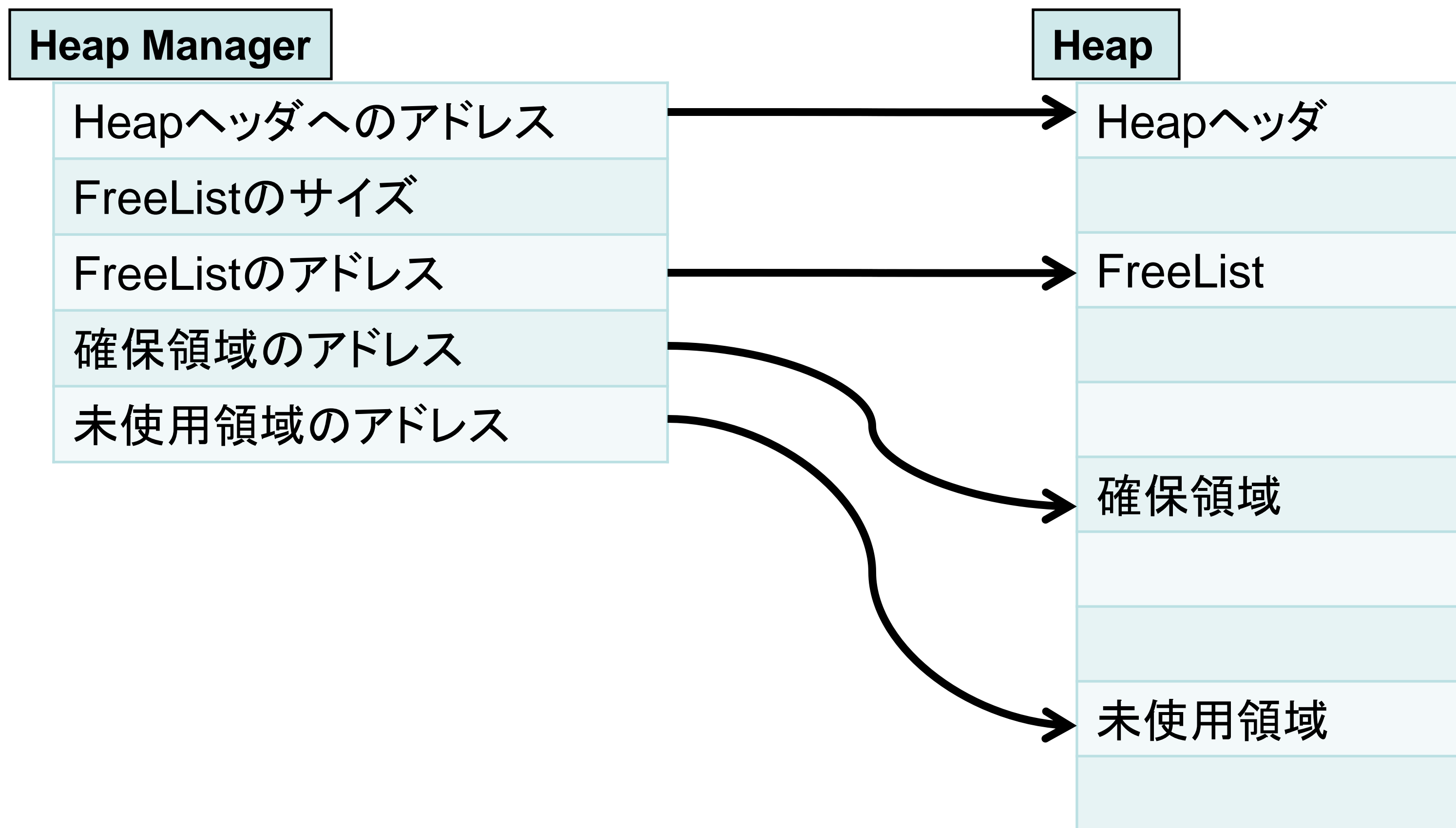
Heap Management（Windows（HeapAlloc/HeapFree））



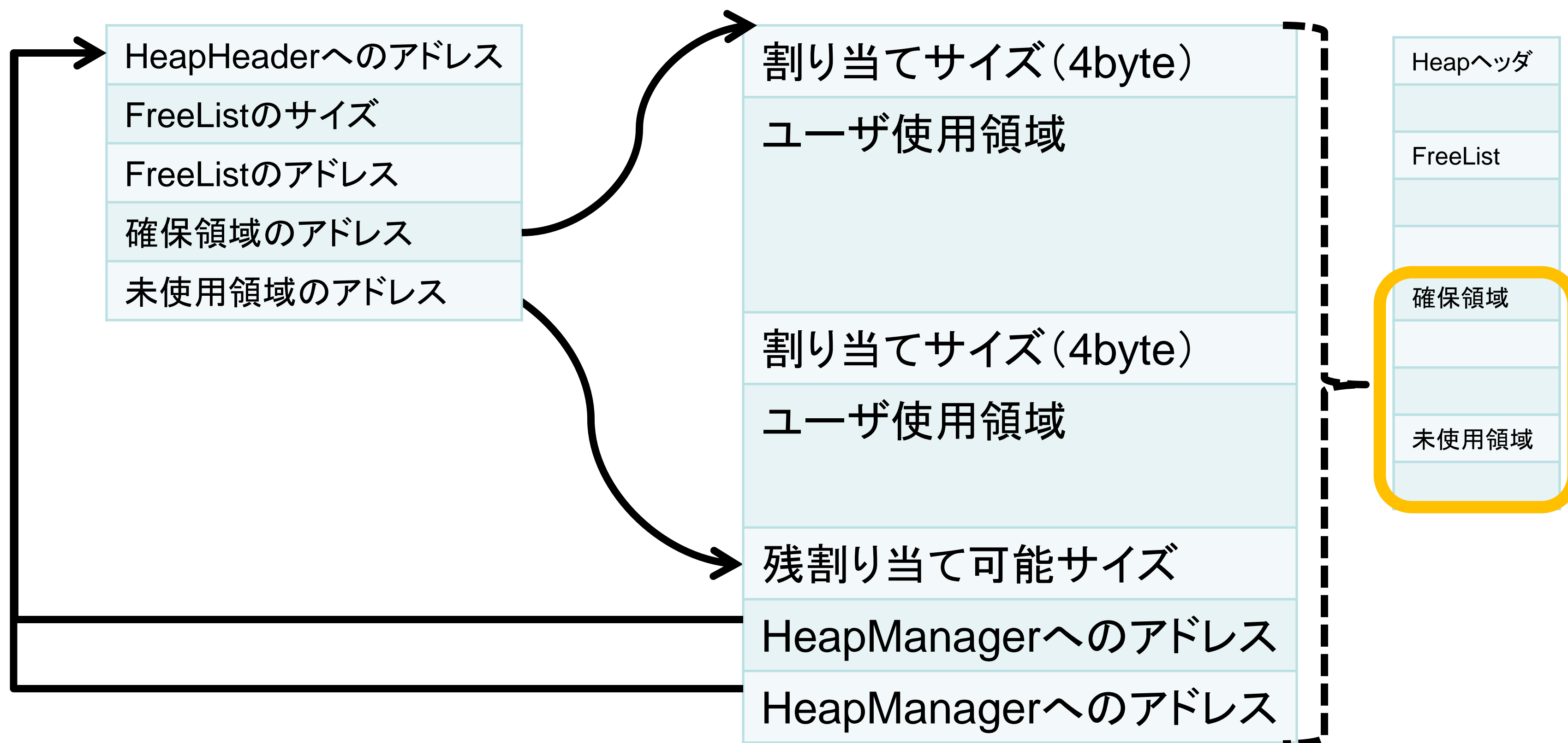
Heap Management (Borland Compiler (malloc/free))

- Borland C++ Compiler(BCC32)でのHeap Manager
  - VirtualAllocで確保した領域を独自のHeap Managerで管理
  - 要求サイズが0x100000以上の場合はVirtualAllocでメモリ確保
  - FreeListでDoubleLinkedListを使用
  - バウンダリチェック等のセキュリティ機能は実装されていない

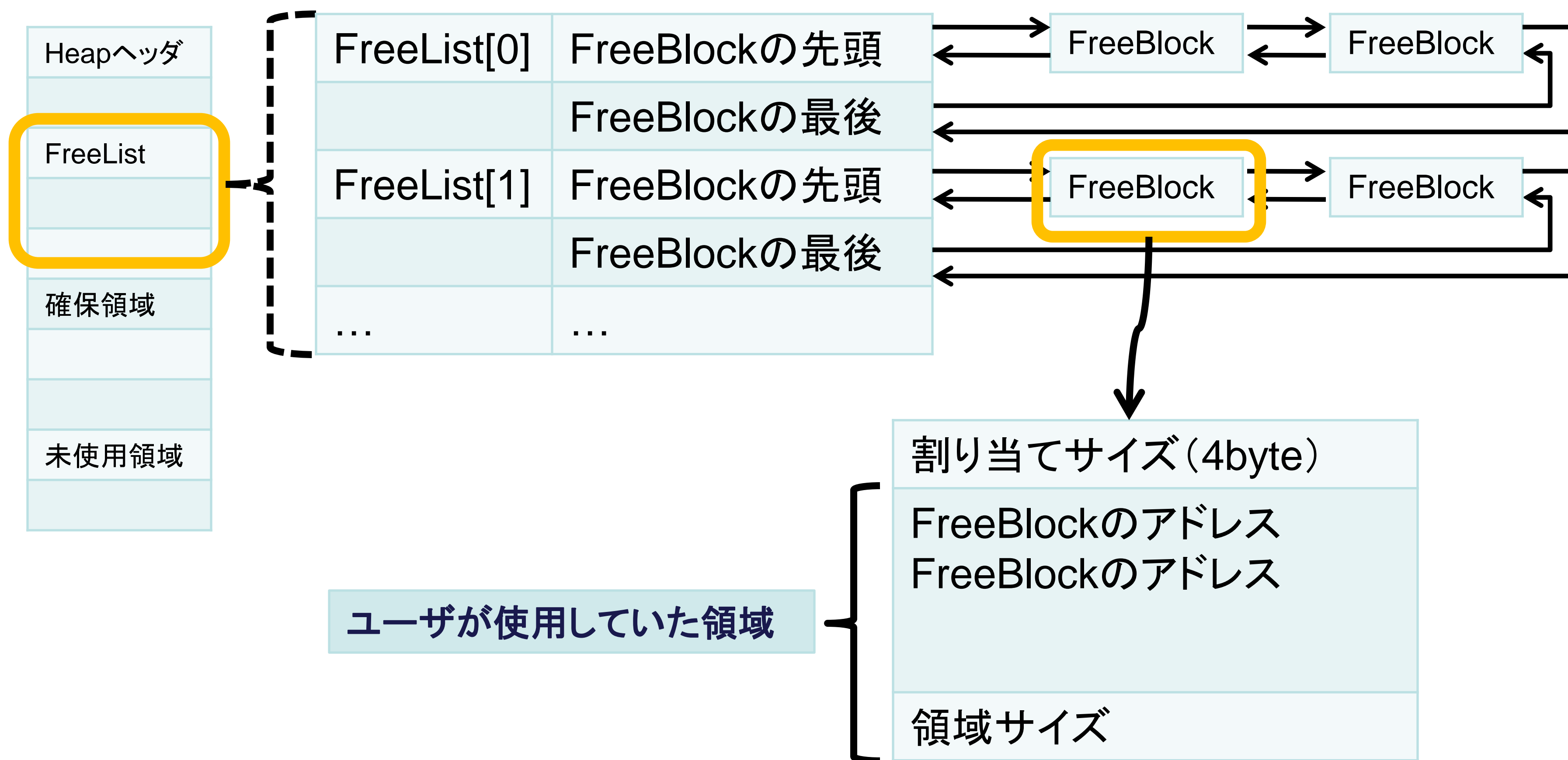
Heap Management（Borland Compiler（malloc/free））



Heap Management (Borland Compiler (malloc/free))



Heap Management（Borland Compiler（malloc/free））



Heap Management（Borland Compiler（malloc/free））

- malloc実行時（FreeListから確保）

```
loc_401D0E:                             ; CODE XREF: sub_40
                cmp     ebx, dword_40A264
                jnb     short loc_401D8C
                mov     ecx, ebx
                add     ecx, ecx
                mov     eax, ecx
                add     eax, dword_40A278
                add     eax, 0FFFFFFF4h
                mov     edx, [eax+4]
                cmp     eax, edx
                jz      short loc_401D52
                mov     eax, edx
                and     dword ptr [eax], 0FFFFFFFEh
                mov     edx, [eax]
                and     edx, 0FFFFFFFCh
                and     dword ptr [eax+edx+4], 0FFFFFFFDh
                mov     edx, [eax+4]
                mov     ecx, [eax+8]
                mov     [edx+8], ecx
                mov     ecx, [eax+8]
                add     eax, 4
                mov     [ecx+4], edx
                jmp     loc_401E..
```

FreeListのつなぎ換え

FFR

Heap Management (Borland Compiler (malloc/free))

- malloc実行時(未使用領域から確保)

```
                cmp     eax, offset unk_40A27C
                mov     ecx, [eax]
                and     ecx, 0FFFFFFFCh
                mov     esi, ecx
                sub     esi, ebx
                cmp     esi, 10h
                jnb     short loc_401E0F
                and     dword ptr [eax], 0FFFFFFFEh
                mov     edx, [eax]
                and     edx, 0FFFFFFFCh
                and     dword ptr [eax+edx+4], 0FFFFFFFDh
                cmp     ecx, dword_40A264
                jb      short loc_401DF8
                mov     ecx, [eax+4]
                mov     off_40A288, ecx

loc_401DF8:                             ; CODE XREF: sub_4
                mov     edx, [eax+4]
                mov     ecx, [eax+8]
                mov     [edx+8], ecx
                mov     ecx, [eax+8]
                add     eax, 4
                mov     [ecx+4], edx
                jmp     loc_401EAA
```

未使用領域から確保

FFR

Heap Management (Borland Compiler (malloc/free))

- free実行時

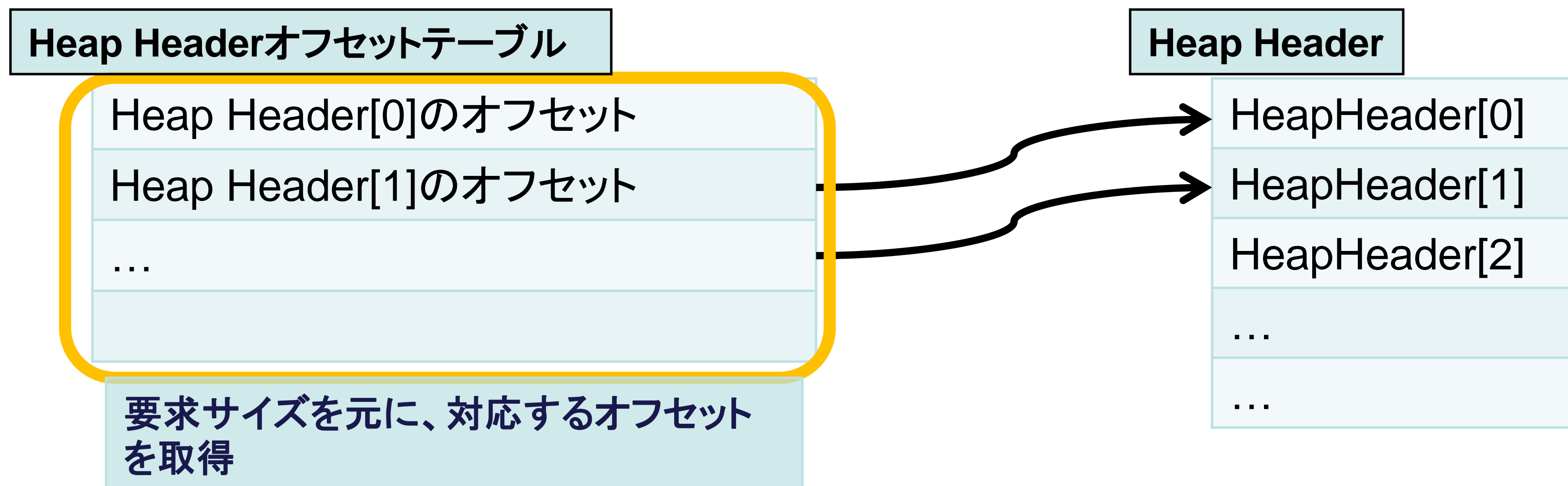
```
loc_401C23:                                    ; CODE XREF:
                mov     ebx, [edx+4]
                mov     [eax+4], ebx
                mov     [eax+8], edx
                mov     ebx, [eax+4]
                mov     [ebx+8], eax
                mov     [edx+4], eax
                lea     edx, [ecx+4]
                mov     [eax+ecx], edx
                mov     edx, [eax]
                mov     ecx, edx
                and     ecx, 0FFFFFFFCh
                cmp     dword ptr [eax+ecx+4], 2
                jnz     short loc_401C70
                mov     ecx, dword_40A28C
                cmp     ecx, dword_40A290
                jbe     short loc_401C5F
                mov     ecx, dword_40A268
                jmp     short loc_401C65
```

FreeListへの追加

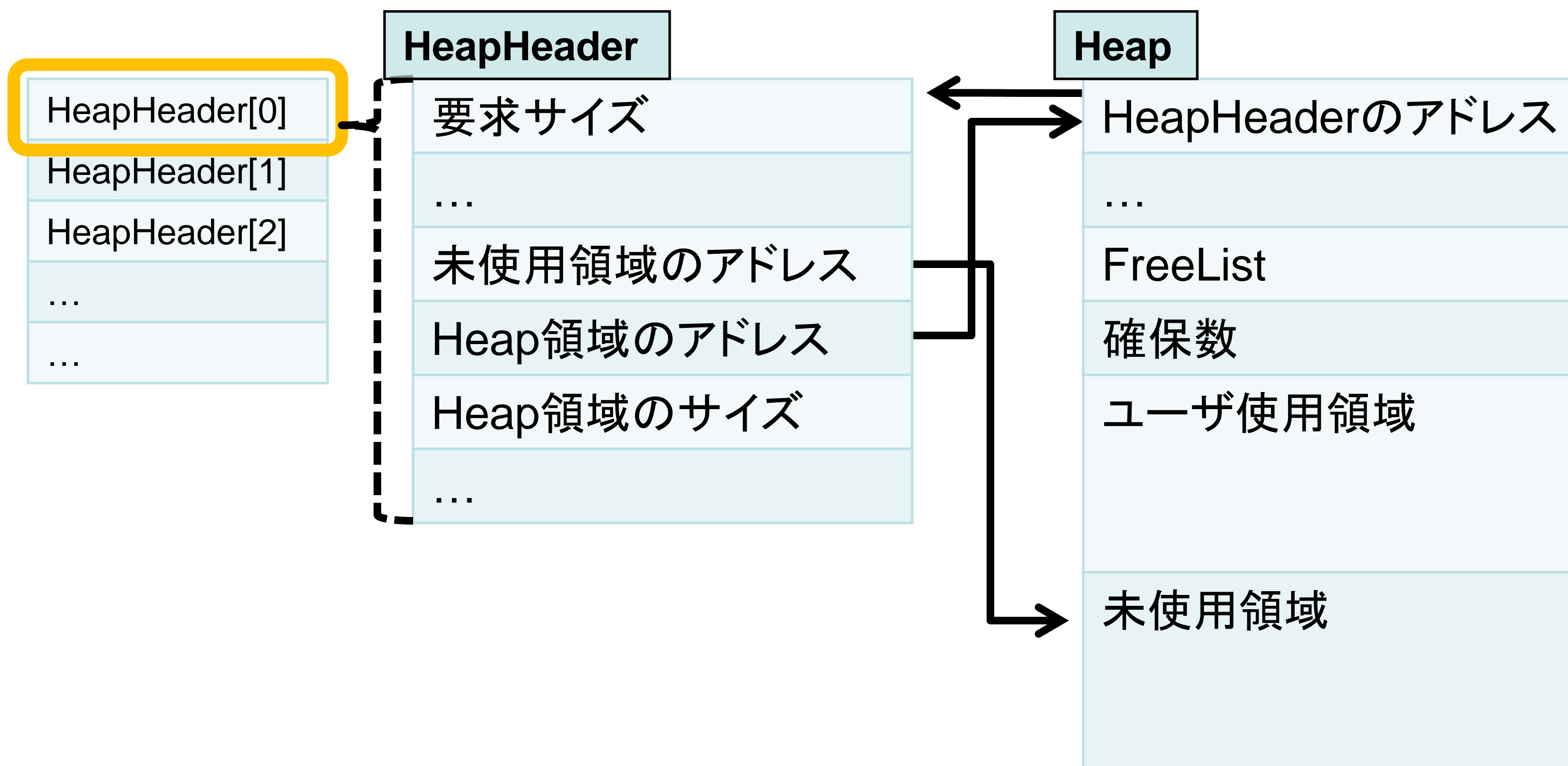
Heap Management（Delphi（GetMem/FreeMem））

- Borland DelphiのGetMem, FreeMemにて使用するHeap Manager
  - 要求サイズごとに確保されたメモリブロックから割り当てる
  - 要求サイズごとに、メモリブロックの管理を行うHeap Headerが存在
  - FreeListではSingle Linked Listを使用
  - セキュリティ機能は実装されていない

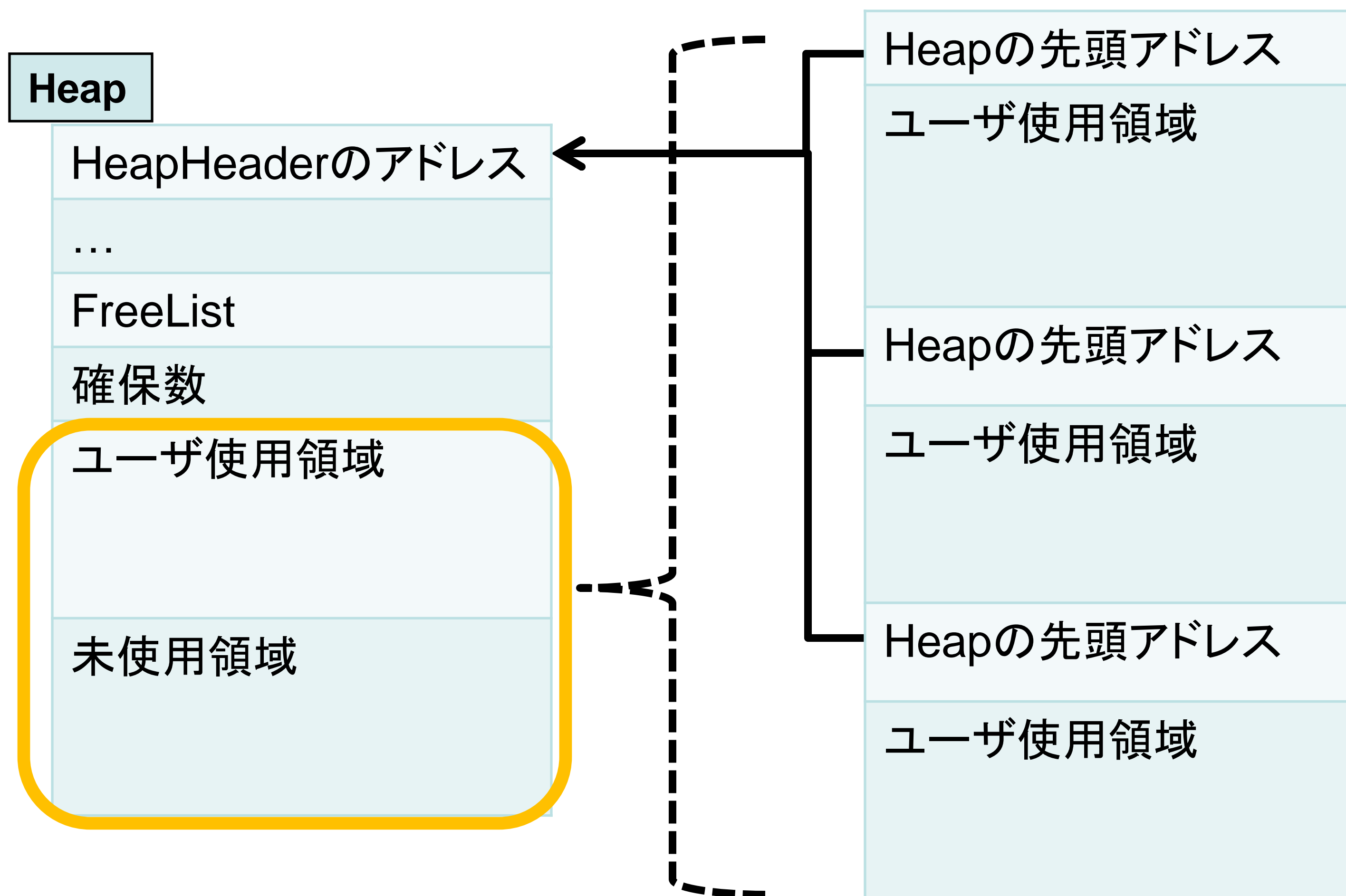
Heap Management（Delphi（GetMem/FreeMem））



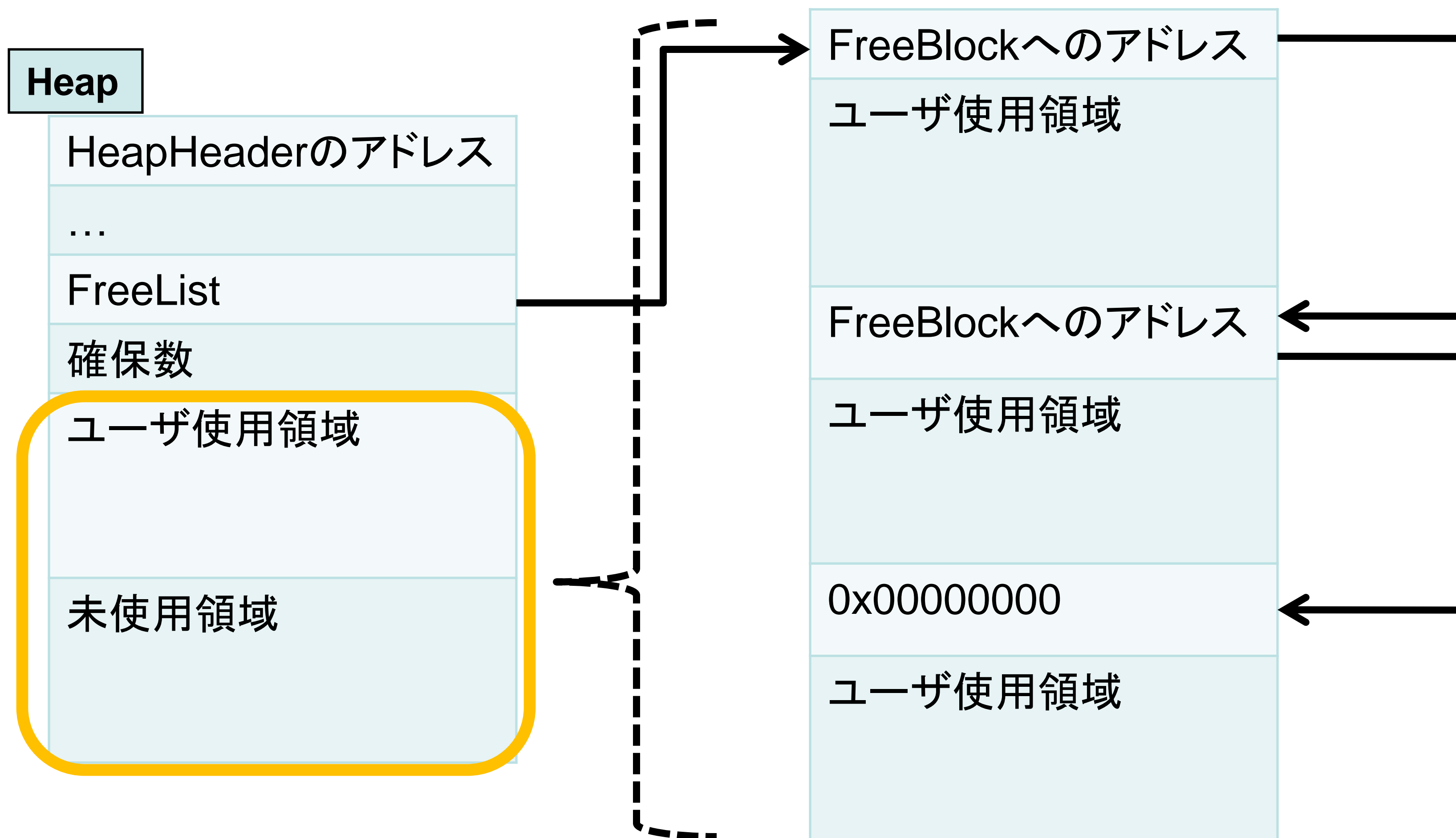
Heap Management（Delphi（GetMem/FreeMem））



Heap Management（Delphi（GetMem/FreeMem））



Heap Management（Delphi（GetMem/FreeMem））



FFR

Heap Management（Delphi（GetMem/FreeMem））

- GetMem実行時（FreeListから確保）

```
; __fastcall System::SysGetMem(int)
@System@SysGetMem$qqri proc near           ; CODE XREF: Sy
                                           ; System::SysRe
                lea     edx, [eax+3]
                shr     edx, 3
                cmp     eax, 0A2Ch
                push    ebx
                mov     cl, ds:byte_40
                ja      loc_401860
                test    cl, cl
                movzx   eax, ds:byte_40B5B8[edx]
                lea     ebx, unk_40A034[eax*8]
                jnz     short loc_40169C

loc_401646:                                ; CODE XREF: Sy
                                           ; System::SysGe
                mov     edx, [ebx+4]
                mov     eax, [edx+8]
                mov     ecx, 0FFFFFF
                cmp     edx, ebx
                jz      short loc_401
                add     dword ptr [edx+0Ch], 1
                and     ecx, [eax-4]
                mov     [edx+8], ecx
                mov     [eax-4], edx
                jz      short loc_40168C
                mov     byte ptr [ebx], 0
                pop     ebx
                retn
```

オフセットテーブルからHeapHeaderを取得

FreeListから確保メモリを取得

FFR

Heap Management（Delphi（GetMem/FreeMem））

- GetMem実行時（未使用領域から確保）

```
loc_40166C:                                             ; CODE XREF
                mov     edx, [ebx+10h]
                movzx   ecx, word ptr [ebx+2]
                add     ecx, eax
                cmp     eax, [ebx+0Ch]
                ja      short loc_4016E8
                add     dword ptr [edx+0Ch], 1
                mov     [ebx+8], ecx
                mov     byte ptr [ebx], 0
                mov     [eax-4], edx
                pop     ebx
                retn
; ---------------------------------------------------------------------------
```

未使用領域からメモリ確保

FFR

Heap Management（Delphi（GetMem/FreeMem））

- FreeMem実行時

```
                test    bl, bl
                mov     ebx, [edx]
                jnz     short loc_401A00

loc_40199E:                             ; CODE XREF: Syste
                                        ; System::SysFreeM
                sub     dword ptr [edx+0Ch], 1
                mov     eax, [edx+8]
                jz      short loc_4019D4
                test    eax, eax
                mov     [edx+8], ecx
                lea     eax, [eax+1]
                mov     [ecx-4], eax
                jz      short loc_4019BC
                xor     eax, eax
                mov     [ebx], al
                pop     ebx
                retn
; ---------------------------------------------------------------
                align 4

loc_4019BC:                             ; CODE XREF: Syste
                mov     ecx, [ebx+4]
                mov     [edx+14h], ebx
                mov     [edx+4], ecx
                mov     [ecx+14h], edx
                mov     [ebx+4], edx
                mov     byte ptr [ebx], 0
                xor     eax, eax
                pop     ebx
                retn
; ---------------------------------------------------------------
```

FreeListの先端に開放アドレスを追加

2. Exploitability計算のための特徴パラメータ検出

任意コード実行が可能となりえる要素

- ローカルバッファ（スタックオーバーフロー）
  - スタック上に確保されたバッファ領域がオーバーフロー
  - オーバーフローにより、リターンアドレス等が書き換えられ任意コード実行が行われる
- ヒープバッファ（ヒープオーバーフロー）
  - ヒープに確保されたバッファがオーバーフロー
  - オーバーフローにより、ヒープを管理するリスト構造が破壊され任意のアドレス上のコードが実行される

スタック上のローカル変数

- ローカル変数の特徴
  - EBP経由で参照されるローカル変数は[EBP-X]というオフセットでアクセスされる(EBPより下位の位置に配置される)
  - ローカル変数を最初に使用する箇所では、ローカル変数に値を設定する処理が多い(Dst operandに指定される)

FFR

スタック上のローカル変数

[EBP-X]の形式でアクセス

```
hHeap           = dword ptr -14h
lpMem           = dword ptr -10h
var_C           = dword ptr -0Ch
Dst             = dword ptr -8
var_4           = dword ptr -4
Source          = dword ptr  8

                push    ebp
                mov     ebp, esp
                sub     esp, 14h
                mov     [ebp+Dst], 0
                mov     [ebp+var_4], 0
                mov     [ebp+lpMem], 0
                mov     [ebp+var_C], 0
                push    10000h          ;
                push    1000h           ;
                push    0               ;
                call    ds:HeapCreate
                mov     [ebp+hHeap], eax
```

はじめに値の設定が行われる

FFR

ローカルバッファの検出

- アセンブラコード上でのローカルバッファの特徴
  - 通常の変数とは異なり、最初のアクセス時にSrc operandになっている場合がある
  - LEA命令にてスタック上のアドレスをレジスタに取得、アドレッシングアクセスを行う
  - ベースレジスタ＋インデックスレジスタの組み合わせで、アドレッシングアクセスを行う

ローカルバッファの検出

```
add     n, ecx
inc     nelem
jmp     loc_409DA6
```

[EBP+インデックスレジスタ]の形式でアクセス

```
mov     byte ptr [ebp+ebx+var_8684], 0
add     eax, ebx
add     eax, 2
mov     [ebp+var_3C], eax
mov     edx, dword_4CD308
add     edx, [ebp+var_3C]
```

LEA命令でアドレスを取得

```
lea     ecx, [esp+24h+var_24]
jz      short loc_401023
```

アドレッシングアクセス

```
inc     edx
mov     [ecx], al
mov     al, [edx]
inc     ecx
jnz     short loc_401017
```

FFR

ポインタの検出

- アセンブラコード上でのポインタの特徴
  - 配列のアクセス方法と似ている
  - アドレッシングアクセスを行っている
  - 配列との違いは、レジスタにアドレスを設定する際にLEAではなくMOV命令を使用している
  - 特定のアドレス領域の値を設定しアクセスを行う場合がある

FFR

ポインタの検出

MOV命令でアドレスを設定

```
mov     eax, [ebp+ms_exc.exc_ptr]
mov     ecx, [eax]
mov     ecx, [ecx]
mov     [ebp+var_20], ecx
push    eax
push    ecx
```

アドレッシングアクセスを行う

FFR

例外ハンドラの検出

- アセンブラコード上での例外ハンドラを使用している関数の特徴
  - fs:0の読み込みと書き込みを行っている
  - fs:0の値をスタックにPUSHしている
  - fs:0をPUSHするひとつ前のPUSH命令のオペランドが例外ハンドラ
  - コンパイラの種類や、最適化のオプションによっては例外ハンドラ設定関数を使用している場合がある

FFR

例外ハンドラの検出

関数内で例外ハンドラの設定（VC）

```
push    ebp
mov     ebp, esp
push    0FFFFFFFEh
push    offset unk_402450
push    offset __except_handler4
mov     eax, large fs:0
push    eax
```

fs:0をEAXに格納してからPUSH

関数内で例外ハンドラの設定（Delphi）

```
xor     eax, eax
push    ebp
push    offset loc_41092C
push    dword ptr fs:[eax]
mov     fs:[eax], esp
```

fsのオフセット指定にレジスタを使用

例外ハンドラ設定関数（VC）

```
push    eax
mov     [ebp-18h], esp
push    dword ptr [ebp-8]
mov     eax, [ebp-4]
mov     dword ptr [ebp-4], 0FFFFFFFEh
lea     eax, [ebp-10h]
mov     large fs:0, eax
retn
```

fs:0の値を更新してリターン

例外ハンドラ設定関数（BCC）

```
mov     dword ptr [ebx+4], offset
mov     word ptr [ebx+10h], 0
mov     word ptr [ebx+12h], 0
mov     dword ptr [ebx+1Ch], 0
mov     eax, fs:0
mov     [ebx], eax
mov     fs:0, ebx
pop     ebx
retn
```

fs:0の値を更新してリターン

Canaryの検出

- アセンブラコード上でのCanaryの特徴
  - コンパイラによって実装が異なるが大きな流れは同じ
    - 関数のはじめで、特定の値を設定している
    - 関数の最後で、値のチェック処理を行っている
  - 関数の最後のチェック処理は別関数の可能性がある
  - コンパイラごとに検出処理を分ける必要もある

SafeSEHの検出

- SafeSEHが有効になっているバイナリファイルの特徴
  - PEヘッダのDataDirectoryにLOAD_CONFIGディレクトリが存在する
  - LOAD_CONFIGディレクトリのフォーマットが特定のフォーマットになっている

Heap Managerの検出

- 使用しているHeap Managerを検出
  - 独自実装のHeap Managerを検出するのは困難
  - Import Tableから使用しているAPIを列挙し、メモリ確保系のAPIの使用頻度から利用しているHeap Managerを判断

FFR

使用しているコンパイラの検出

- DOSヘッダからPEヘッダへのオフセットの値
  - DOSヘッダからPEヘッダのオフセットの値は、コンパイラによって異なる。
  - DOSヘッダからPEヘッダのオフセット値は、コンパイラが同じであれば同じになる可能性が高い
- DOSプログラムとして起動した際に実行されるコード
  - DOSプログラムとして起動した際に実行されるコードでは、コンパイラによって実行コードに違いがある
- これ以外にもいくつか検出方法が存在…

使用しているコンパイラの検出

Visual C++

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ｸ.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 E0 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..ｺ..ｴ.ﾍ!ｸ.Lﾍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  53 1E DB 5B 17 7F B5 08 17 7F B5 08 17 7F B5 08  S.ﾛ[..ｵ...ｵ...ｵ.
```

BCC32

```
00000000  4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00  MZP.............
00000010  B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00  ｸ.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 00  ................
00000040  BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90  ｺ....ｴ.ﾍ!ｸ.Lﾍ!瑞
00000050  54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73  This program mus
00000060  74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57  t be run under W
00000070  69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00  in32..$7........
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

GCC

```
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ｸ.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..ｺ..ｴ.ﾍ!ｸ.Lﾍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  50 45 00 00 4C 01 05 00 C7 3A CB 48 00 14 00 00  PE..L...ﾇ:ﾋH....
```

Delphi

```
00000000  4D 5A 50 00 02 00 00 00 04 00 0F 00 FF FF 00 00  MZP.............
00000010  B8 00 00 00 00 00 00 00 40 00 1A 00 00 00 00 00  ｸ.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ................
00000040  BA 10 00 0E 1F B4 09 CD 21 B8 01 4C CD 21 90 90  ｺ....ｴ.ﾍ!ｸ.Lﾍ!瑞
00000050  54 68 69 73 20 70 72 6F 67 72 61 6D 20 6D 75 73  This program mus
00000060  74 20 62 65 20 72 75 6E 20 75 6E 64 65 72 20 57  t be run under W
00000070  69 6E 33 32 0D 0A 24 37 00 00 00 00 00 00 00 00  in32..$7........
00000080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

3. FFR EXCALOCを用いたExploitabilityの数値化

FFR EXCALOC

- 動作環境
  - IDA Pro Version 5.2 のプラグイン
  - 対象フォーマットはPEファイル
  - 現時点では、静的解析のみで特徴パラメータの抽出を行う
- 開発環境
  - Windows XP SP2
  - Visual Studio 2008

特徴パラメータの検出

- PEファイル全体に影響するパラメータ
  - 使用しているコンパイラ
  - Heap Manager
  - SafeSEHの有効/無効
  - 適用されているセキュリティ機能

- 関数単位で影響するパラメータ
  - 関数内のローカルバッファの検出
  - 関数内のポインタの検出
  - 関数内の例外ハンドラの検出

FFR

特徴パラメータの抽出例（1）

- ブラウザソフトI

Compiler Type	Visual C++
Stack Protection	/GS
Safe SEH	あり
Heap Manager	Windows Heap Manager

アドレス	サイズ	参照数	配列	ポインタ	例外ハンドラ
0x00401609	0x88	0x02	なし	あり	なし
0x00401887	0x14D	0x01	あり	あり	あり
0x00401DE4	0x2AB	0x01	あり	あり	なし
…					

FFR

特徴パラメータの抽出例（2）

- グラフィックソフトH

Compiler Type	Visual C++
Stack Protection	なし
Safe SEH	なし
Heap Manager	なし

アドレス	サイズ	参照数	配列	ポインタ	例外ハンドラ
0x00401000	0xDC	0x01	あり	なし	あり
0x004010DC	0x08	0x01	なし	なし	なし
0x004010F0	0x59	0x01	なし	なし	あり
…					

FFR

特徴パラメータの抽出例（3）

- サーバソフトT

Compiler Type	Borland Delphi
Stack Protection	なし
Safe SEH	なし
Heap Manager	Delphi Heap Manager

アドレス	サイズ	参照数	配列	ポインタ	例外ハンドラ
0x0040114D	0x1CF	0x01	あり	なし	なし
0x00403AEA	0x105	0x06	あり	なし	なし
0x00403EEC	0x62	0x03	なし	なし	なし
…					

FFR

Exploitability計算

- 基準値（関数単位で計算）

基準値 = 関数のサイズ × 関数の参照数

- 抽出したパラメータを考慮した値

基準値 × A × B × C

A	配列の有無によるパラメータ。Canaryの有無により重み付けを変化。 （配列なし = 1, 配列ありかつCanaryあり = 1, その他 = 2）
B	ポインタの有無によるパラメータ。Heap Managerの種類により重み付けを変化 （Windows = 1, Borland C++ = 2, Delphi = 2）
C	例外ハンドラの有無によるパラメータ。SafeSEHの有無により変化 （例外なし = 1, 例外ありSafeSEHあり = 1, 例外ありSafeSEHなし = 2）

FFR

Exploitability計算

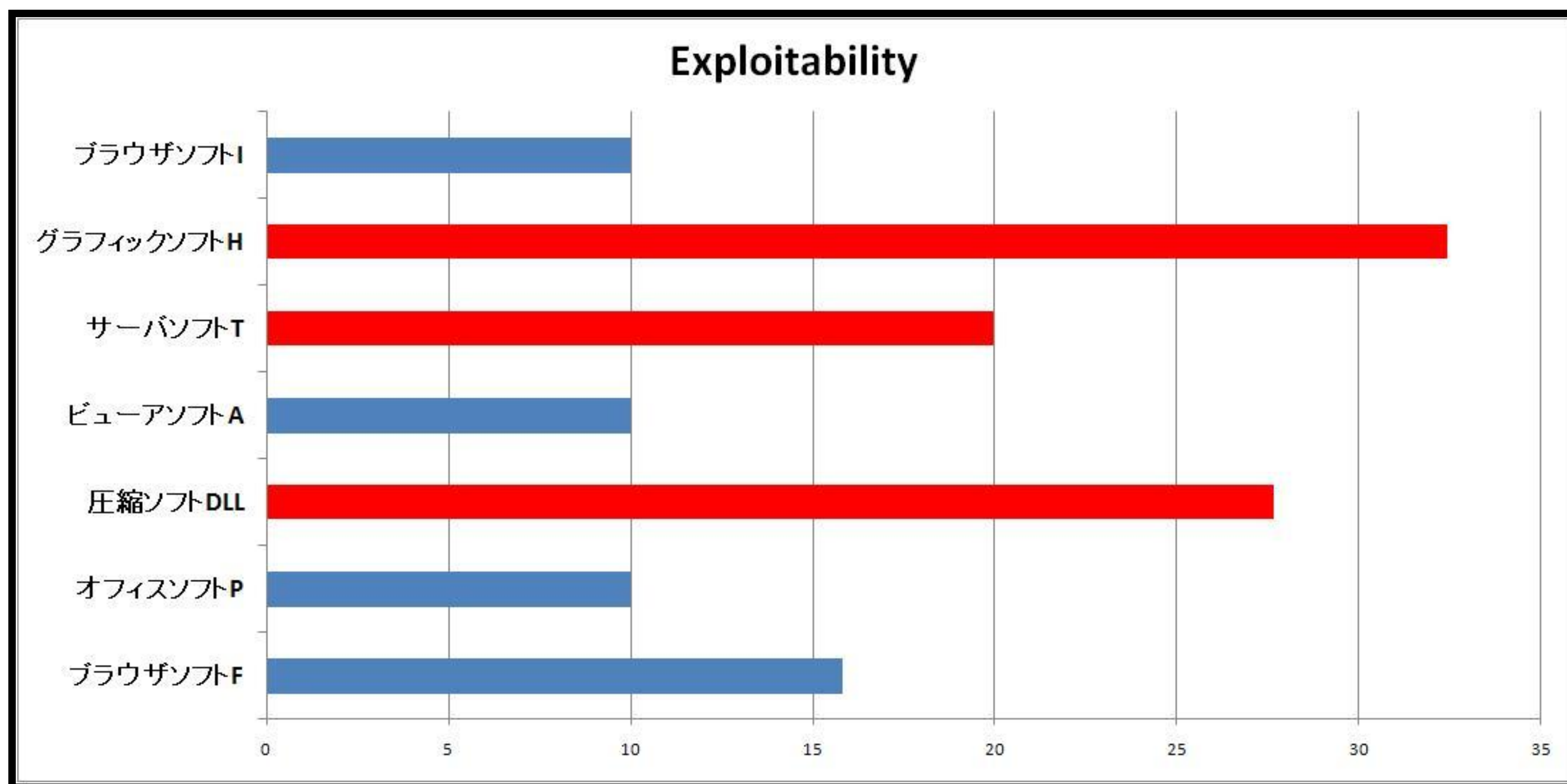
- 実行ファイルのExploitability計算

Exploitability = パラメータ考慮値の和 / 基準値の和

- 関数単位に計算した値をすべて合計
- それぞれの合計値を割り算
- 数値が大きいほど、Exploitabilityが高い

Exploitability計算結果

- 先ほどの特徴パラメータを抽出したアプリケーション（＋α ）の計算結果



4. 今後の課題

FFR

より正確なExploitabilityを計算するために…

- より正確な要素の検出
  - 配列と構造体の区別
  - 関数ポインタの検出
  - …
- 動的解析を併用したExploitabilityの計算
  - 対象アプリケーションをデバッガで起動させ、実行コードの流れからExploitability計算に必要な要素を検出する
    - 関数のパラメータ
    - ヒープバッファ
    - …

ありがとうございました

Fourteenforty Research Institute, Inc.
株式会社 フォティーンフォティ技術研究所
http://www.fourteenforty.jp

シニアソフトウェアエンジニア
石山智祥 <ishiyama＠fourteenforty.jp>