Monthly Research

Heap Exploitのこれまでと現状

株式会社ＦＦＲＩ
http://www.ffri.jp

Ver 2.00.01

Heap Exploitとは

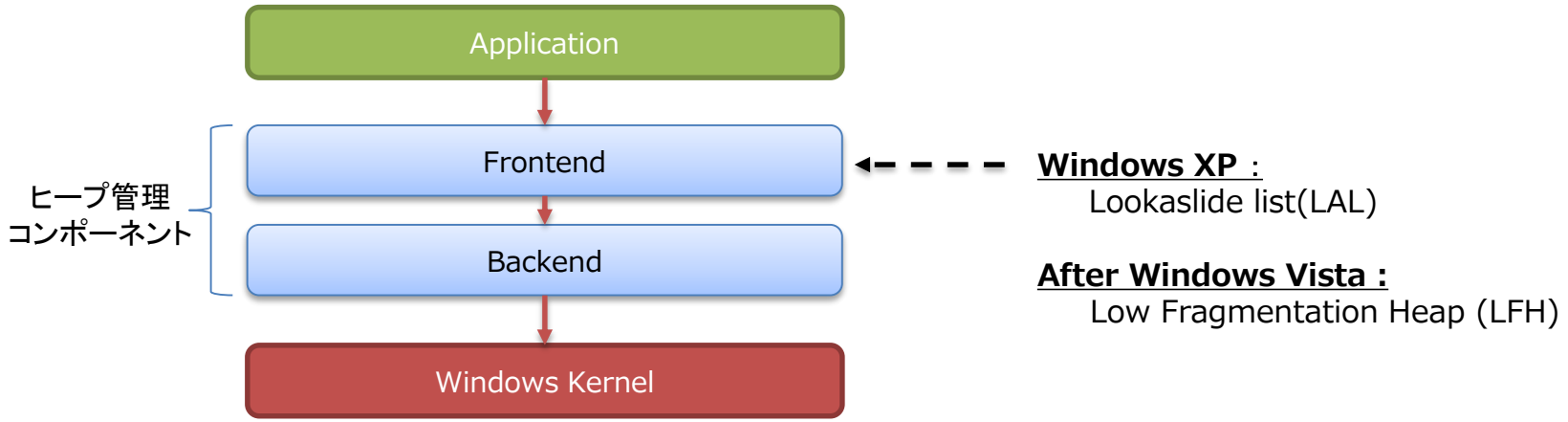
- Heap Exploitとはヒープ領域のメモリ管理の問題点を利用して攻撃を行うこと
- メモリ破壊関連の脆弱性を大きくカテゴリ分けすると以下の二つ
  - スタック上のメモリ破壊
  - ヒープ上のメモリ破壊
- どちらのメモリ破壊も主たる原因として「バッファオーバーフロー」がある
- 今回はヒープオーバーフローによる攻撃手法と防御手法のこれまでと現状について調査
- 対象はWindows XP以降のWindows OSのユーザーランドのヒープ
- 攻撃手法はヒープの数多くの構造や処理に対応するように多数存在するため、基本的な考え方とこれまでに導入されている防御策についてまとめた
- 各説明はどれも全体の理解を目的とするためにかなり簡略化されていることに注意

ヒープとは

- ヒープはプログラムの実行中にメモリを動的に確保するために利用されるメモリ領域のこと
- C/C++であれば、mallocやnewで確保される領域
- mallocやnewの実装は各プラットフォームで異なる
- Windowsでは主に以下のAPIを利用してHeapを操作する
  - HeapCreate　　ヒープ領域の作成
  - HeapAlloc　　ヒープ領域から特定のサイズのメモリ領域を取得
  - HeapFree　　取得したメモリ領域を開放

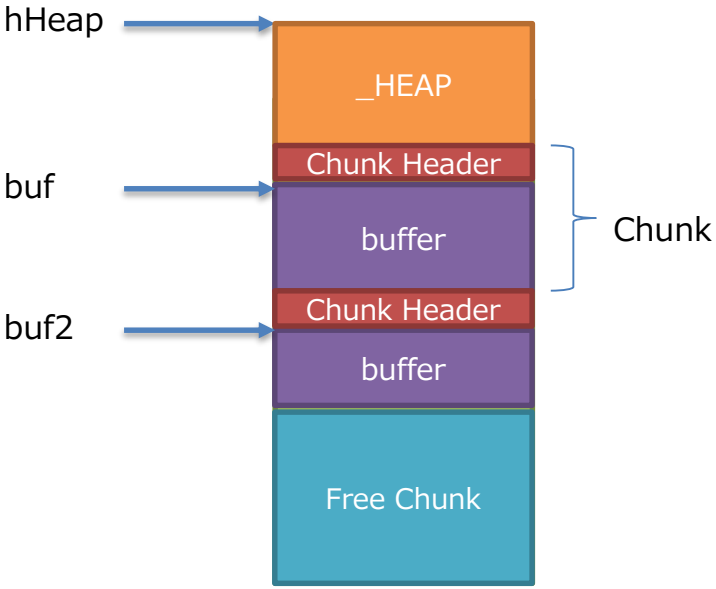
Windowsのヒープ管理コンポーネントの概観

- Windowsのヒープ管理（ユーザーランド）は大きく以下の二つのコンポーネントから構成される
  - Frontend
  - Backend
- Frontendはアプリケーションが直接やり取りするインタフェースとなる
  - 小さなメモリ確保のための最適化などを担当
  - 特定のメモリ確保要求に答えられる場合には、ここからメモリブロックが返される
  - 要求にこたえられない場合にはBackendに処理を受け渡す
  - 機構が2つあり、Windows XPではLookalide List(LAL)、Vista以降ではLow Fragmentation Heap(LFH)が利用されている



Heap APIの基本動作

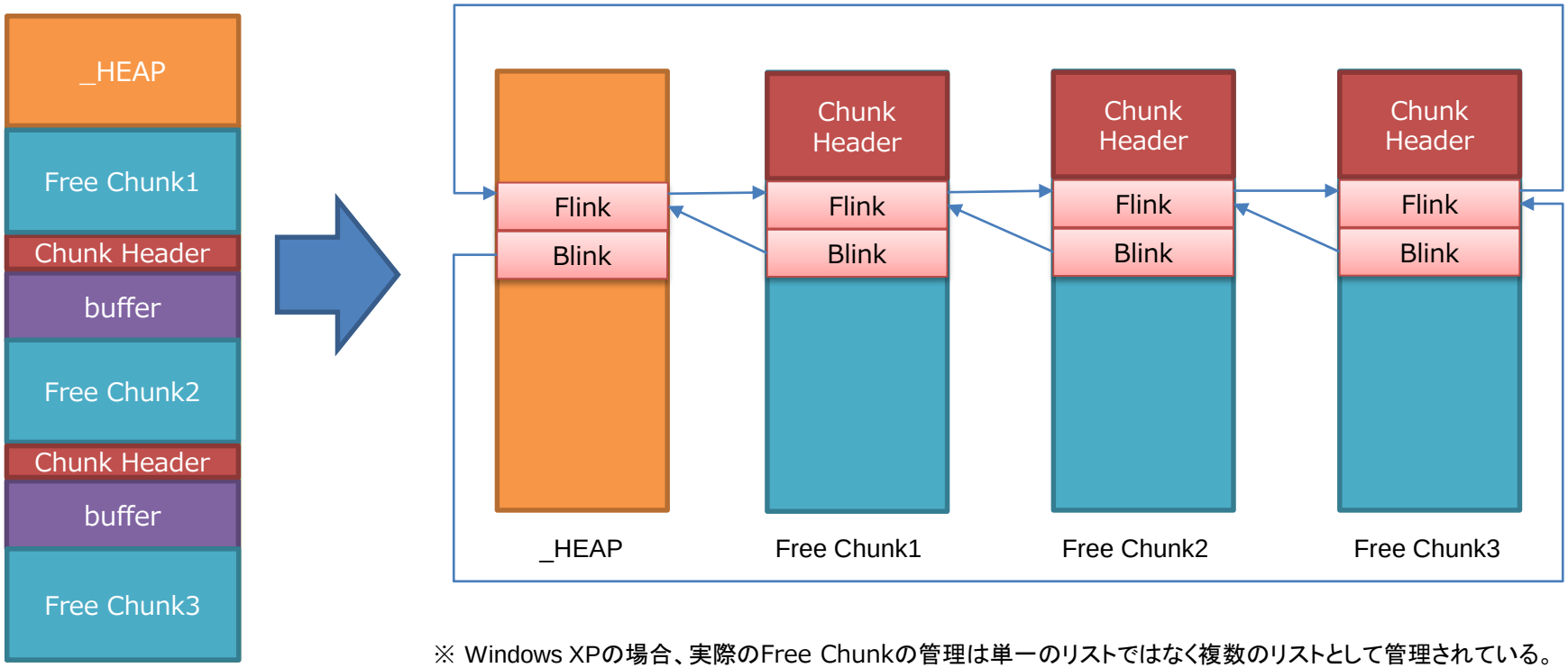
- HeapCreateによってヒープが作成される
- ヒープの先頭には_HEAP構造体が配置されており、そのアドレスがHANDLEとして返る
- _HEAP構造体にはヒープの管理情報が含まれている
- HeapAllocにはそのHANDLEとサイズを渡すことで要求したサイズの領域が返される。
- 返されたアドレスにアプリケーションはデータを格納することができる
- ヒープの管理のために、確保された領域のアドレスの2バイト前にその領域の管理情報(Chunk Header)が配置されている。
- まだ利用されていない領域はFree Chunkとして管理される




```
HANDLE hHeap = HeapCreate(...);
LPVOID buf   = HeapAlloc(hHeap, ... );
LPVOID buf2  = HeapAlloc(hHeap, ... );
```

Free Chunkの管理(Backend)

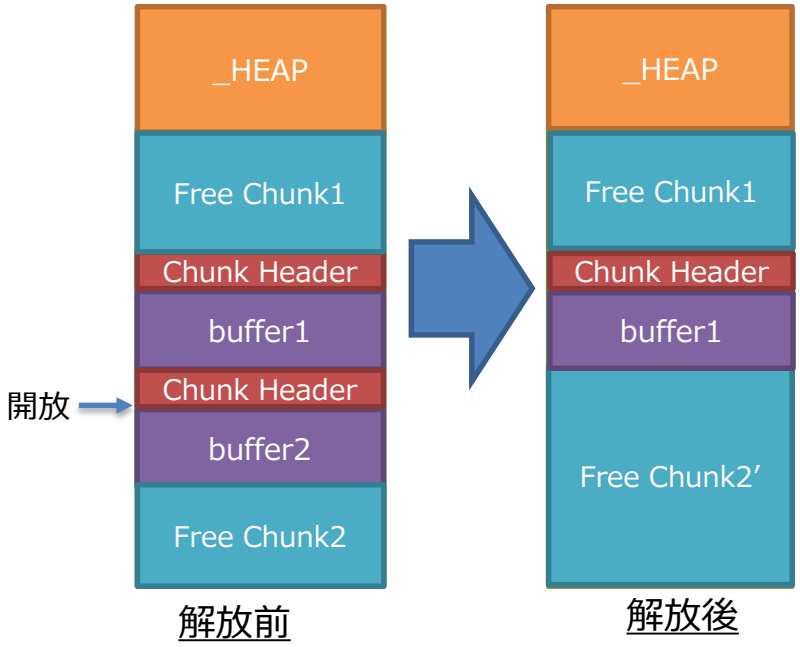
- アプリケーションがHeapAllocやHeapFreeをさまざまな順で呼び出すと確保されている領域と開放されている領域(Free Chunk)が交互に現れ、飛び飛びのメモリ領域が構成される
- Free Chunkの場所を管理するために、Windowsのヒープは双方向循環リンクリストを用いて管理している
- Free ChunkもChunk Headerを持つ



※ Windows XPの場合、実際のFree Chunkの管理は単一のリストではなく複数のリストとして管理されている。
ここではExploitの説明を目的とするため、簡略化している

HeapFreeとCoalesce(連結)

- ある領域がHeapFreeで開放された場合、隣のChunkがFree Chunkであるときに、それらの連結が行われる。
- 後述のExploitではこの連結時に発生する双方向リンクリストのUnlink(リンクから要素の削除)動作の特徴を利用する
- 以下の図ではBuffer2が開放されるとCoalesceが発生する



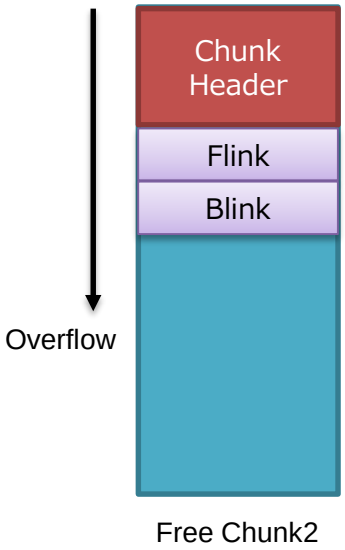
Buffer2の解放前、Free Chunk1とFree Chunk2はリンクリストで繋がっている。Buffer2が開放されるとBuffer2とFree Chunk2によってより大きなFree Chunk2'が作成され、Free Chunk1とFree Chunk2'のリンクが作成される。この過程で、Free Chunk2はリンクリストから削除される。そのため以下のようなコードが実行される

```
// Free Chunk2をリンクリストから削除
[Free Chunk2]->Blink->Flink = [Free Chunk2]->Flink
[Free Chunk2]->Flink->Blink = [Free Chunk2]->Blink
```

※ この処理はHeapFree時にFrontend(Lookaslide List)が利用されずにBackendが利用された場合にのみ発生する。
そのため、この処理を利用して攻撃する場合にはそのための条件を整える必要がある。

基本的なHeap Exploit

- Windows XP SP1まで可能であったヒープオーバーフローを利用した攻撃手法
- Free ChunkのFlink, Blinkを書き換えることで任意の4バイトを、任意のアドレスに書き込むことができる。
- 前スライドのFree Chunk2の先頭部分がオーバーフローによって書き換えられるとする
- CoalesceのタイミングでFlinkとBlinkの値が参照され、値が書き込まれる。



// [Free Chunk2]->Flinkおよび[Free Chunk2]->Blinkは書き換えられた値
[Free Chunk2]->Blink->Flink = [Free Chunk2]->Flink
[Free Chunk2]->Flink->Blink = [Free Chunk2]->Blink

任意のアドレスに任意の値を書き込める。

関数ポインタを書き換えることで、特定の位置にコード実行を移す
→PEB (Process Environement Block)内の関数ポインタ値を利用するなどの方法が知られている。
→上記2つの命令はセットで実行されるため、関数ポインタの値を書き換えようとすると、自動的に、
その関数ポインタが指し示す先に、関数ポインタ自体が存在するアドレス(たとえば、0x7FFDF01C)の値が書き込まれる。攻撃が成立するにはその値(0x7FFDF01C)がCPUで実行できなくてはいけない

Windows XP SP2での対策

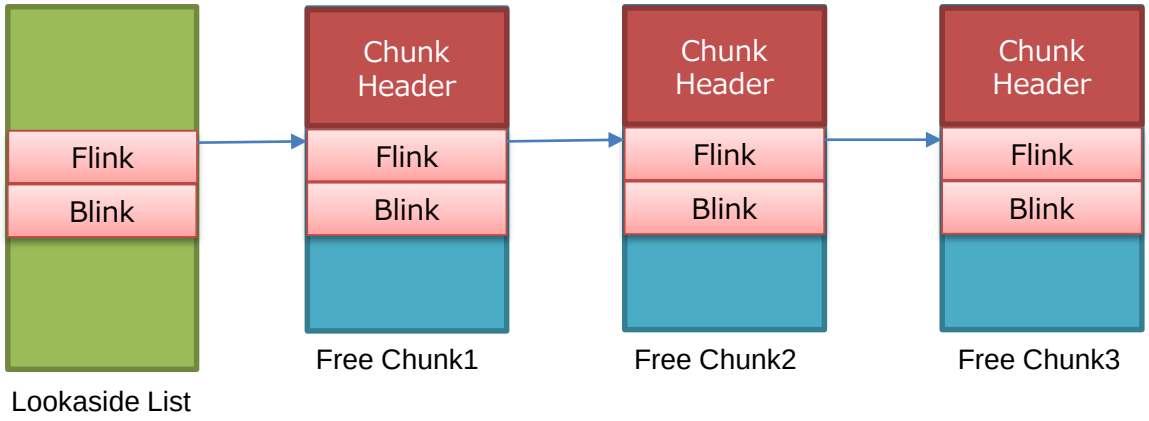
- Windows XP SP2からはHeap Exploit対策として主に以下の3つが導入されている
  - Chunk HeaderのCookieの導入
    - Chunk Header内に8bitのチェックサム(Cookie)を導入
    - その値をチェックすることでChunk Headerが書き換えられたかどうかを確認
    - Cookieの値はそのChunkのアドレスから計算される
  - Safe Unlinking
    - 双方向リンクリストからの要素の削除の前に、必ず以下の条件が整っていることを確認

      `[Chunk]->Flink->Blink == [Chunk]->Blink->Flink == [Chunk]`

      (前後の要素が自分を指していることを確認)
  - PEB Randomization
    - PEB(Process Environment Block)のアドレスをランダマイズ。
    - PEBはHeap Exploitに限らず攻撃に利用される値やアドレスを格納している
    - ランダマイズすることで攻撃の成功率を下げることができる

Windows XP SP2での対策の回避

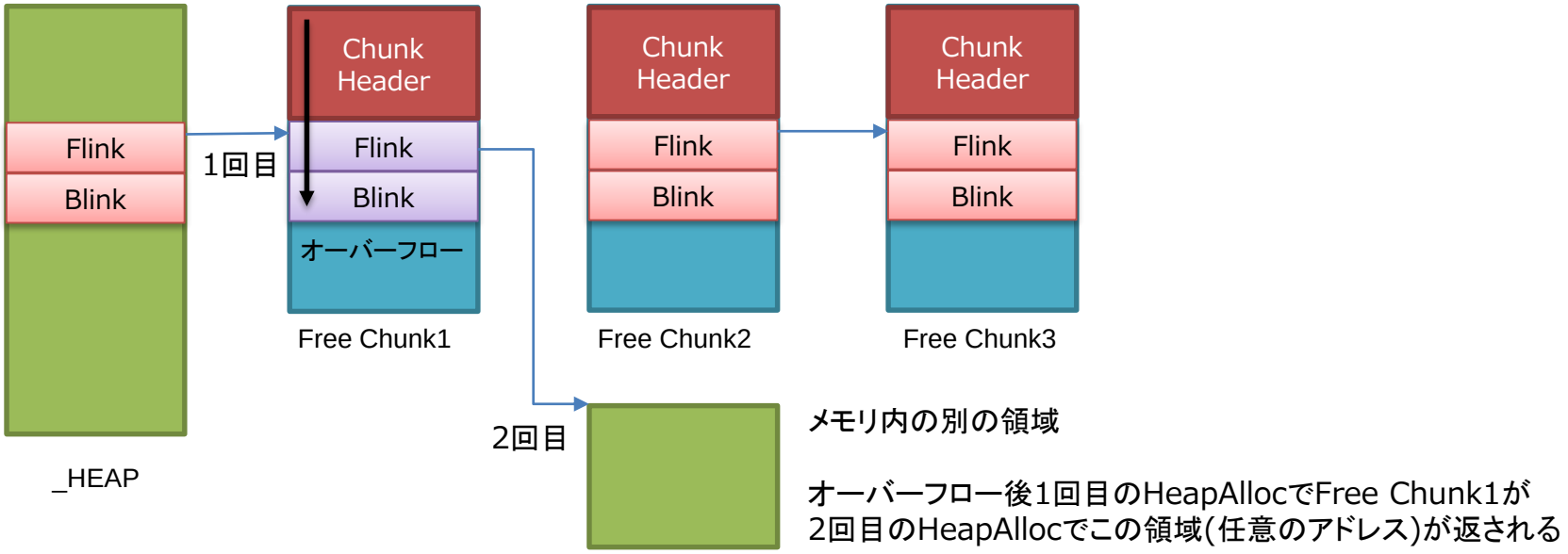
- Lookaside Listを用いた攻撃
  - HeapFreeによるメモリの開放は、最初にLookaslide Listによって管理される
  - Lookaslide Listはサイズごとに用意されたFree Chunkの単方向リスト
  - HeapAllocによってあるサイズのメモリ確保された場合、最初にこのLookalide ListにFree Chunkが存在するかどうかが調べられ、存在する場合にはここから返される。



- Lookaslide Listはそれぞれのサイズごとの複数の単方向リンクリストからなる
- 一つのリストは単一のサイズのChunkが連結される。
  （上図ではそのうちの一つのサイズのリストのみ示している）
- 一つのリスト内に最大で4つのChunkを持つ。
  （それ以上にその同じサイズのChunkが開放された場合はBackendによって開放処理が行われる）
- Chunkの開放、取得時には共にリストの先頭にChunkの追加、削除が行われる。

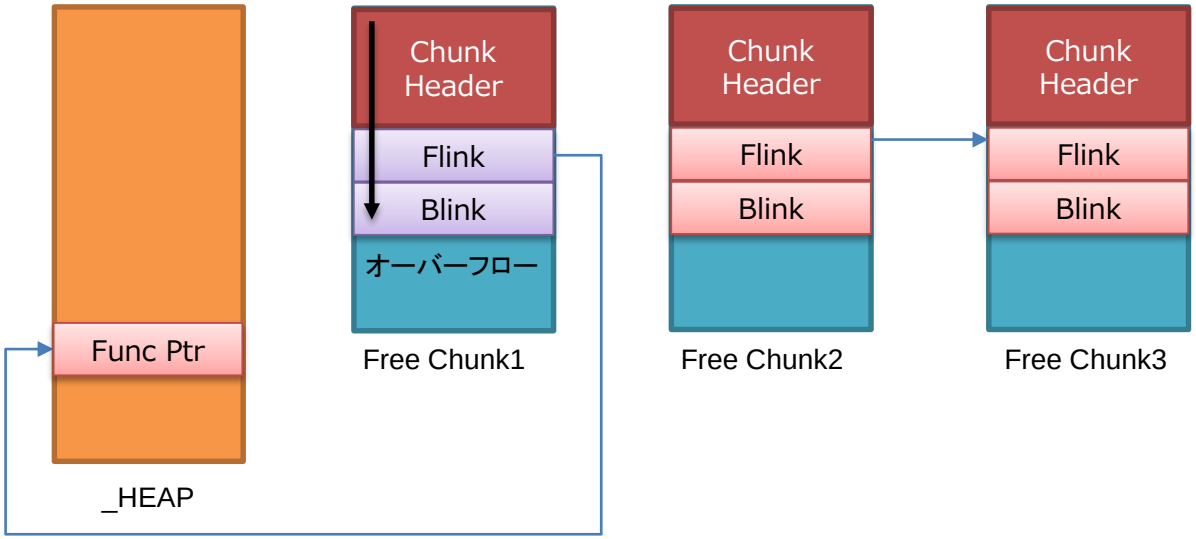
Windows XP SP2での対策の回避(Cont.)

- Lookaside Listを用いた攻撃
  - Lookaside Listでは重要な二つの防御策が働かない
    - Lookaside ListのChunkの確保にはCookieのチェックが行われない
    - Safe Unlinkingが行われない(双方向リストではないため、チェックが不可能)
  - 下図、Free Chunk1のヘッダ領域がオーバーフローによって書き換えられるとする
  - その後、2回のHeapAllocが呼び出されると、任意の領域をアロケートすることができる



Windows XP SP2での対策の回避(Cont.)

- Lookaside Listを用いた攻撃
  - Flinkの値が関数ポインタを含むアドレスを指すようにオーバーフローさせる
  - その後2回目のアロケーションで取得される領域に格納されるデータをコントロールできれば、関数ポインタの値を任意の値で上書き可能
  - 以下の図の場合_HEAP構造体に存在する関数ポインタのアドレスを確保するようにしている(この関数ポインタはヒープ関連の処理のなかで利用され、呼び出される)



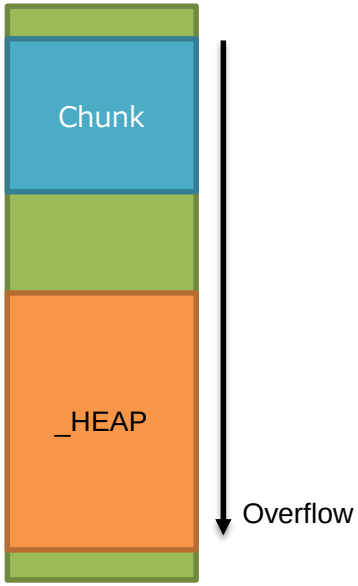
- そのほかにも多数、条件によって実行可能な攻撃方法が公開されている

Windows Vistaから導入された対策

- Low Fragmentation Heapの導入
  - FrontendがLookaslide Listの代わりにLow Fragmentation Heapに置き換わっている
  - Lookaslide Listを利用した攻撃が不可能に
- Block Metadataのランダマイズ
  - Chunk Headerの値が_HEAP->Encodingの値でXORされるようになった。
  - Cookieを予測してChunk Headerを上書きしても、攻撃者の意図した通りの値にはならなくなった
- Enhanced entry header cookie
  - Cookieの値がChunk Headerのチェックも含めるようになった。
  - これまでは、そのChunkのアドレスを元に計算された値だったが、Chunk Headerの値も含めて計算されるようにすることで、Chunk Headerの書き換えの検知が強化された
- Heap base randomization
  - _HEAPのアドレスをランダマイズ
  - _HEAP内のデータを書き換えられる可能性を低減
- Heap function pointer encoding
  - _HEAP構造体の中に存在する関数ポインタのエンコーディング
  - 関数ポインタの値を書き換え攻撃に対する対策

Windows Vistaから導入された対策の回避

- _HEAP構造体の上書き
  - ヒープオーバーフローを用いて_HEAP構造体を書き換える
  - 関数ポインタを書き換えることで、任意のアドレスの実行が可能に
  - ただし、いくつかの条件が必要
    - _HEAP構造体より下位のアドレスにオーバーフローする領域が配置されていること
    - オーバーフローするChunkのアドレスが分かっていること
    - オーバーフローするサイズが十分大きいこと



_HEAP構造体内の関数ポインタはXORエンコードされているが、XORの対象となる値も
_HEAP構造体内に存在するため、その値と共に書き換えることで、
関数ポインタを任意のアドレスに書き換えることができる。

Windows Vistaから導入された対策の回避(Cont.)

- その他、以下のような攻撃手法が公開されている
  - LFHのオーバーフローを利用した_HEAP構造体の書き換え、
  - LFHのオーバーフローを利用した、アプリケーションのオブジェクトの書き換え
  - _HEAP構造体のFreeとReallocationによる関数ポインタの書き換え

Windows 8から導入された対策

- _HEAP構造体のFreeの禁止
  - 前スライド「_HEAP構造体のFreeとReallocationによる関数ポインタの書き換え」の対策
- _HEAP構造体内の関数ポインタのエンコードの強化
  - XORエンコードされる対象がこれまで_HEAP構造体内にあったため、両方の値を同時に書き換えられる場合、関数ポインタに任意のアドレスで上書き可能であった
  - XORエンコードの対象をグローバルに持つようにしたため、同時に書き換えられることがなくなった
- ガードページの導入
  - ヒープで利用される様々領域の間にGuard Page(アクセス不可領域)を導入
  - オーバーフローによって書き換え可能な領域に制限がかかる
- LFHのアロケーションのランダマイズ
  - Chunkの配置を攻撃者が予め予測できる場合、オーバーフロー発生時に、アプリケーションの重要なメモリ領域の書き換えの可能性があった。ランダマイズにより、これも非常に困難となった。
- ヒープエラー時のプロセスの終了
  - ヒープ処理で例外が発生した場合にプロセスを終了するように変更
  - これまでヒープ処理で例外が発生しても、致命的なエラーとはならずに処理が続いた。このため攻撃者が何度もヒープオーバーフローを試すことが可能であったが、それができなくなった。
- その他にもここで紹介されていない攻撃に対する対策が多数追加されている
  - http://blogs.technet.com/b/srd/archive/2013/10/29/software-defense-mitigation-heap-corruption-vulnerabilities.aspx

まとめ

- Heap ExploitはWindows Vista以降Chunk Headerのランダマイズやエンコーディングによって困難になっている
- Windows 8からアロケートされるメモリ領域のランダマイズやGuard Pageの導入によりヒープオーバーフローの攻撃はさらに困難に
- ただし、ヒープ自体の構造や処理が複雑であるため、新たな処理や機構を利用した攻撃手法が開発される可能性は残る
- 今回、紹介した攻撃方法以外にも、数多くのヒープオーバーフローを用いた攻撃手法が公開されている。詳細は参考文献を参照。

参考資料

- Third Generation Exploitation
  - http://www.blackhat.com/presentations/win-usa-02/halvarflake-winsec02.ppt
- Windows Heap Overflows
  - http://www.blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt
- Reliable Windows Heap Exploits
  - http://www.cybertech.net/~sh0ksh0k/heap/CSW04%20-%20Reliable%20Heap%20Exploitation.ppt
- Windows Heap Exploitation(Win2KSP0 through WinXPSP2)
  - http://www.cybertech.net/~sh0ksh0k/projects/winheap/XPSP2%20Heap%20Exploitation.ppt
- Exploition Freelist[0] On XP Service Pack 2
  - http://www.orkspace.net/secdocs/Windows/Protection/Bypass/Exploiting%20Freelist%5B0%5D%20On%20XP%20Service%20Pack%202.pdf
- Windows Vista Heap Management Enhancements
  - https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Marinescu.pdf
- Understanding and bypassing Windows Heap Protection
  - http://mirror7.meh.or.id/Windows/heap/Heap_Singapore_Jun_2007.pdf
- Heaps About Heaps
  - https://www.insomniasec.com/downloads/publications/Heaps_About_Heaps.ppt
- Attacking the Vista Heap
  - https://www.lateralsecurity.com/downloads/hawkes_ruxcon-nov-2008.pdf

参考資料

- Practical Windows XP/2003 Heap Exploitation
  - http://www.blackhat.com/presentations/bh-usa-09/MCDONALD/BHUSA09-McDonald-WindowsHeap-PAPER.pdf
- Preventing the exploitation of user mode heap corruption vulnerabilities
  - http://blogs.technet.com/b/srd/archive/2009/08/04/preventing-the-exploitation-of-user-mode-heap-corruption-vulnerabilities.aspx
- Understanding the Low Fragmentation Heap
  - http://illmatics.com/Understanding_the_LFH.pdf
  - http://illmatics.com/Understanding_the_LFH_Slides.pdf
- Windows 8 Heap Internals
  - http://media.blackhat.com/bh-us-12/Briefings/Valasek/BH_US_12_Valasek_Windows_8_Heap_Internals_Slides.pdf
- Advanced Heap Manipulation in Windows 8
  - https://media.blackhat.com/eu-13/briefings/Liu/bh-eu-13-liu-advanced-heap-WP.pdf
- Software Defense: mitigating heap corruption vulnerabilities
  - http://blogs.technet.com/b/srd/archive/2013/10/29/software-defense-mitigation-heap-corruption-vulnerabilities.aspx

Contact Information

E-Mail ：research—feedback@ffri.jp
Twitter：@FFRI_Research