The Evolution of Microsoft's Exploit Mitigations

Past, Present, and Future

Tim Burrell
tim.burrell@microsoft.com
Microsoft Security Engineering Center (MSEC)
Security Science

Credits

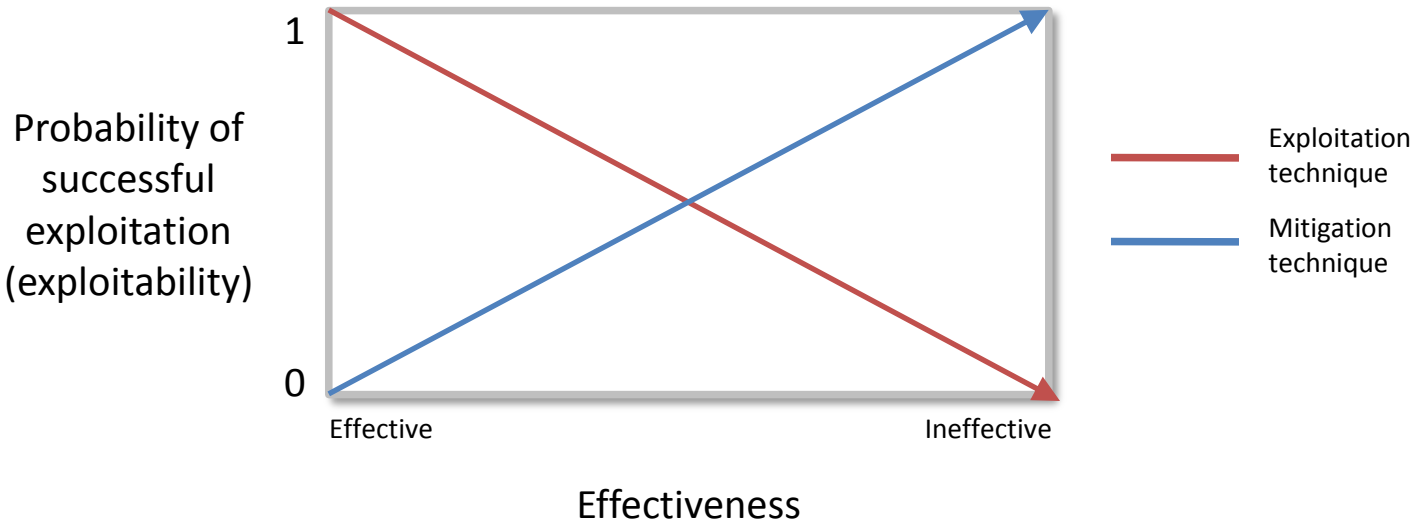
- Peter Beck, Matt Miller (MSEC)
- Louis Lafreniere (Compiler team)

- Many others in these teams who helped along the way

Agenda

- Defining the purpose of exploit mitigations
- Microsoft's exploit mitigation evolution
  - The past
  - The present/future
    - Windows7
    - Visual Studio 2010

The purpose of exploit mitigations



- Goal: decrease the probability of successful exploitation
  - Prevent the use of specific exploitation techniques
  - Reduce the reliability of exploitation techniques
- Generic protection for known & unknown vulnerabilities in all products, not just Microsoft products!

ACT I

THE PAST

Exploitation

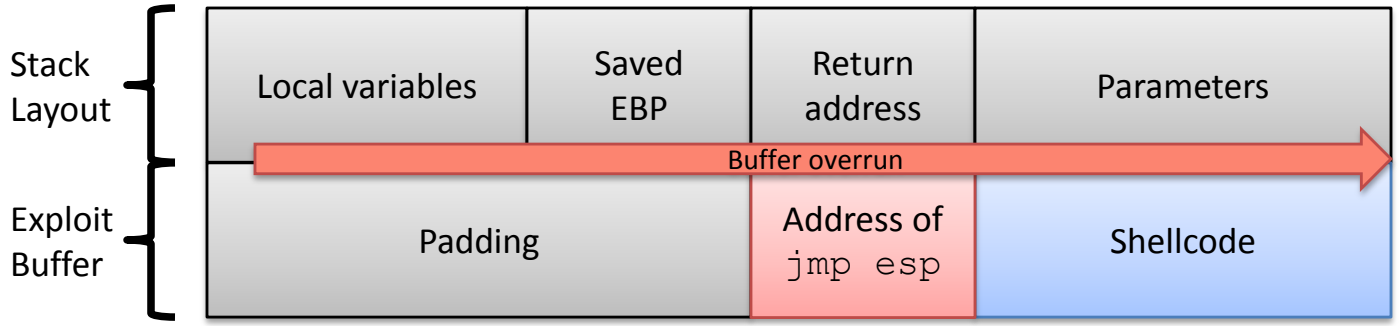
Pre-XP SP2: The era of uninhibited worms

- Reliable exploitation techniques already existed
  - And they affected Windows, too!
- Exploits were developed, worms raged
  - Jul, 2000: IIS Code Red (MS01-033)
  - Jan, 2003: SQL Slammer (MS02-039)
  - Aug, 2003: Blaster (MS03-026)
  - May, 2004: Sasser (MS04-011)
- No platform exploit mitigations existed
  - Attack surface was very big
  - Exploitation techniques were uninhibited

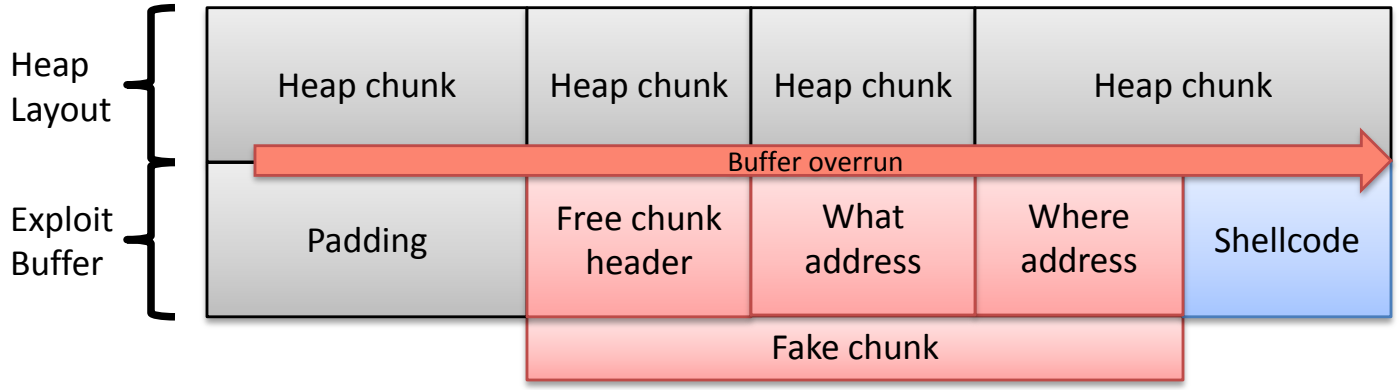
Exploitation

Same techniques, different OS

- Stack: return address overwrite [Aleph96]



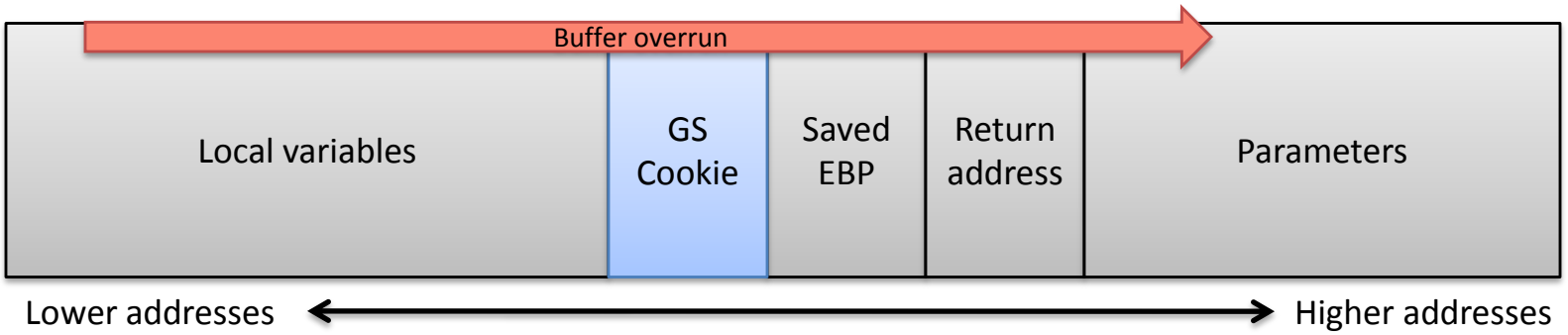
- Heap: free chunk unlink [Solar00,Maxx01,Anon01]



Mitigation

Visual Studio 2002

- GS v1 released



- Behavior
  - Compiler heuristics identify at-risk functions
  - Prologue inserts cookie into stack frame
  - Epilogue checks cookie & terminates on mismatch

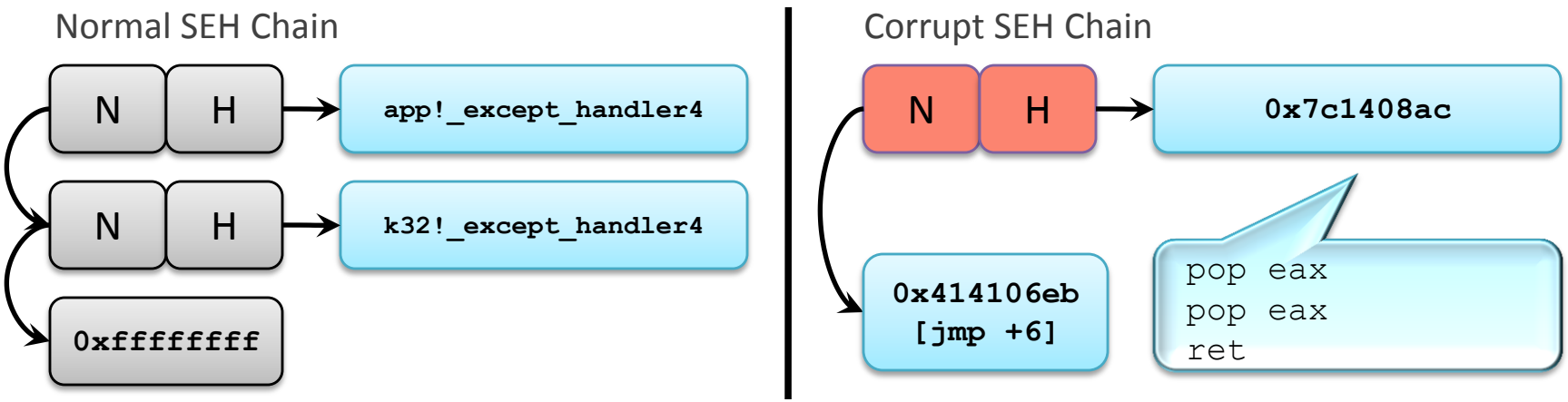
Exploitation

GS v1 weaknesses

- Adjacent local/parameter overwrite [Ren02]

```
void vulnerable(char *in) {
   int unsafe = 0; char buf[256];
   strcpy(buf, in);                          ← overflow!
   if (unsafe != 0) DoSomethingUnsafe();     ← unsafe is corrupt
   return;                                   ← GS cookie checked
}
```

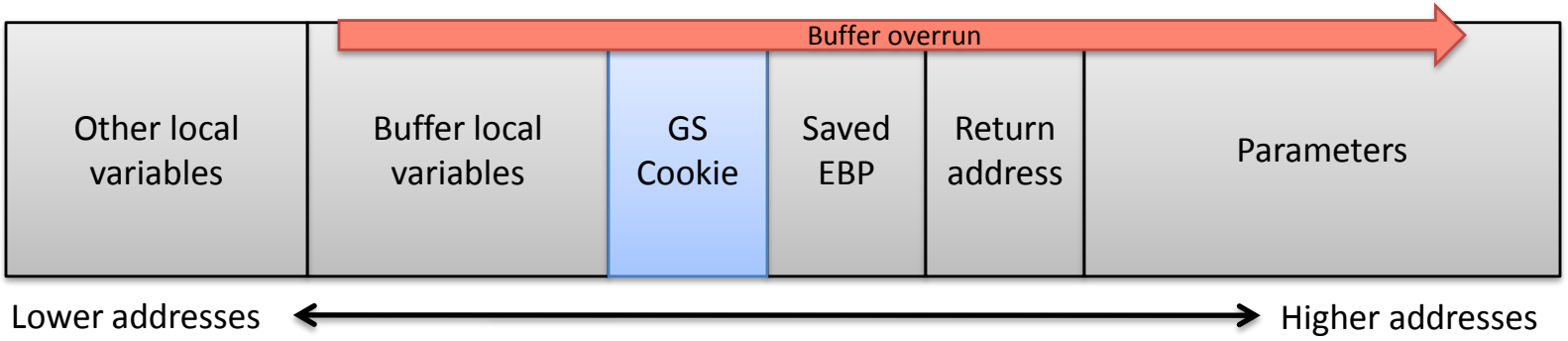
- SEH overwrite bypass [Litchfield03]



Mitigation

Visual Studio 2003

- GS v1.1 released with VS2003

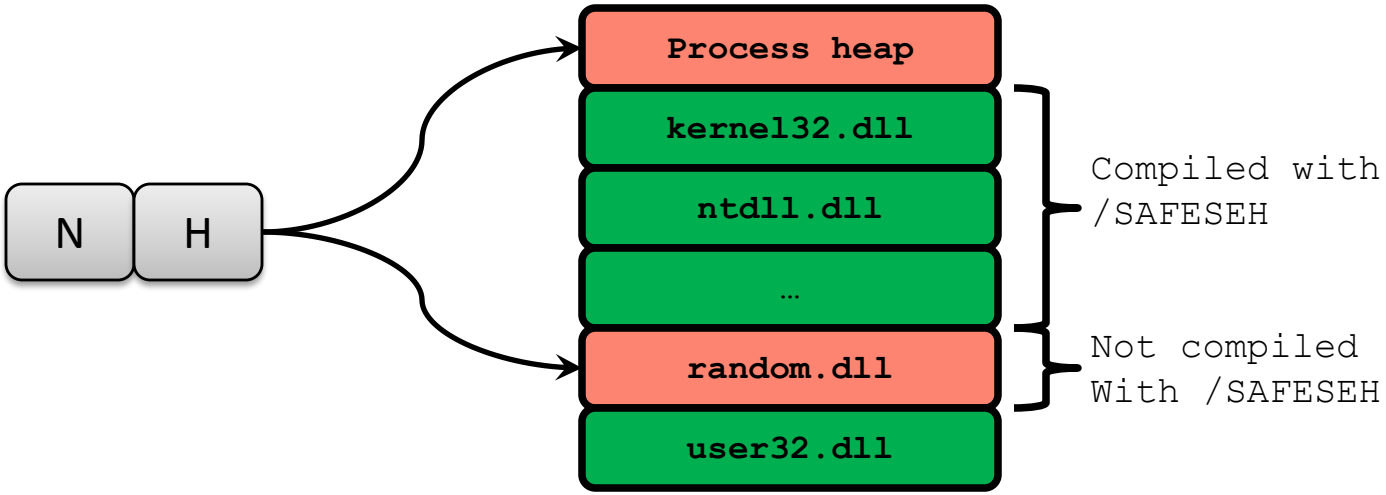


- SafeSEH added, reliant on XP+ & recompile

Exploitation

SafeSEH evasions

- Limitations of SafeSEH
  - Handler can be in an executable non-image region
  - Handler can be inside a binary lacking SafeSEH

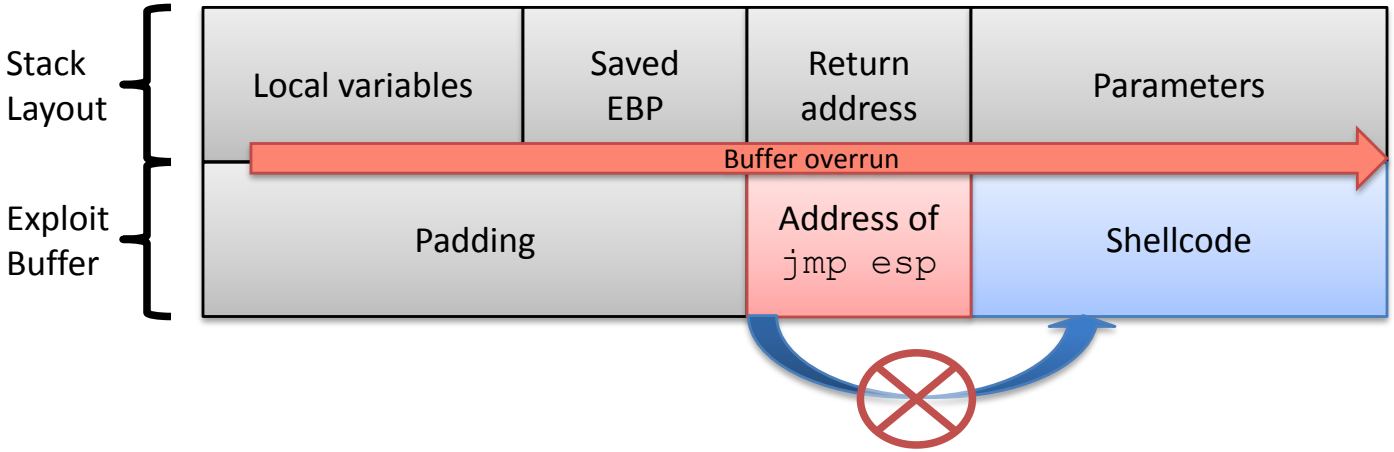


Mitigation

Windows XP SP2 arrives

- System binaries built with GS v1.1 & SafeSEH

- Data Execution Prevention (DEP)
  - Hardware-enforced non-executable pages
  - Software-enforced SEH handler validation



Stack, heap, and other regions are now non-executable

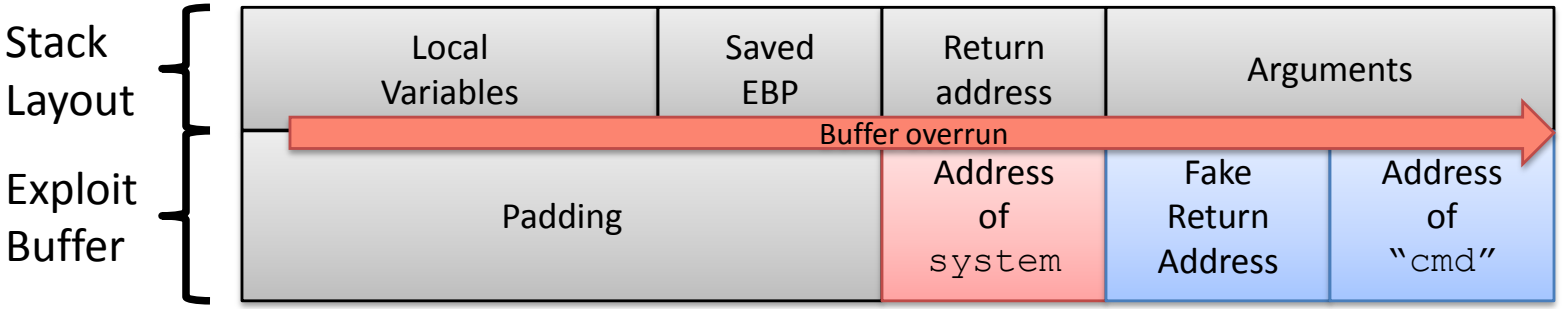
Windows XP SP2 arrives

- First round of heap mitigations
  - Safe unlinking `(E->B->F == E->F->B == E)`
  - Heap header cookie validation
- Limited randomization of PEB/TEB
  - Reduces the reliability of certain techniques
- Pointer encoding
  - Protect UEF, VEH, and others via EncodeSystemPointer

Exploitation

Same NX bypass, new OS

- Return to libc [Solar97,Nergal01]



- Many variations
  - Return into VirtualProtect/VirtualAlloc
  - Disable DEP via ProcessExecuteFlags [Skape05]
  - Create executable heap & migrate to it
  - Return-oriented programming [Shacham08]

Exploitation

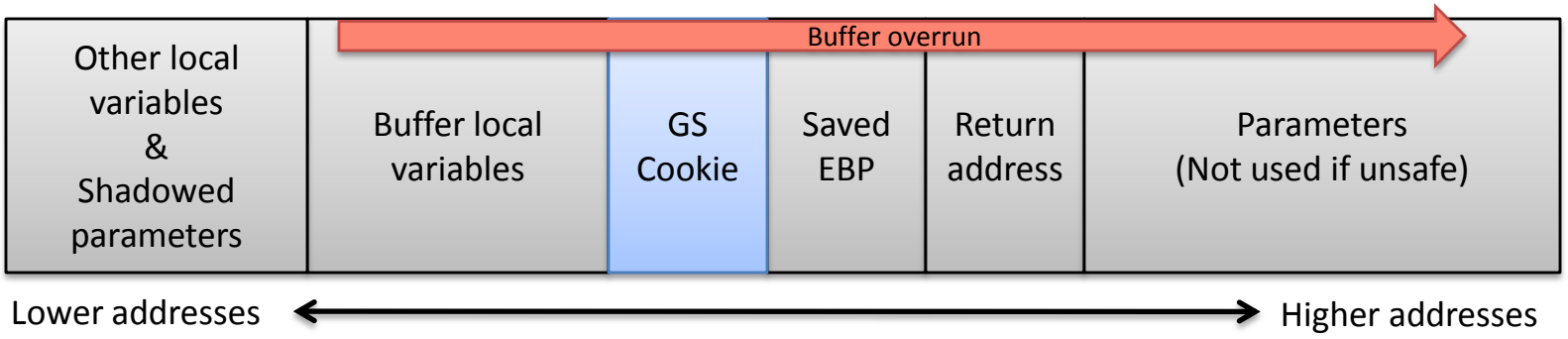
New heap techniques, less universal

- Unsafe lookaside list allocations [Anisimov04,Conover04-2]
  - Overwrite free chunk on lookaside list & then cause allocation
- Unsafe unlinking of free chunks [Conover04-2]
  - Overwrite free chunk with specific Flink and Blink values
- Unsafe unlink via `RtlDeleteCriticalSection` [Falliere05]
  - Overwrite critical section structure on heap & delete it
- Exploiting `FreeList[0]` [Moore05]
  - Overwrite free chunk stored at FreeList[0] with specific data

Mitigation

Visual Studio 2005

- GS v2 released with VS2005
  - Shadow copy of parameters is made
  - Strict GS pragma

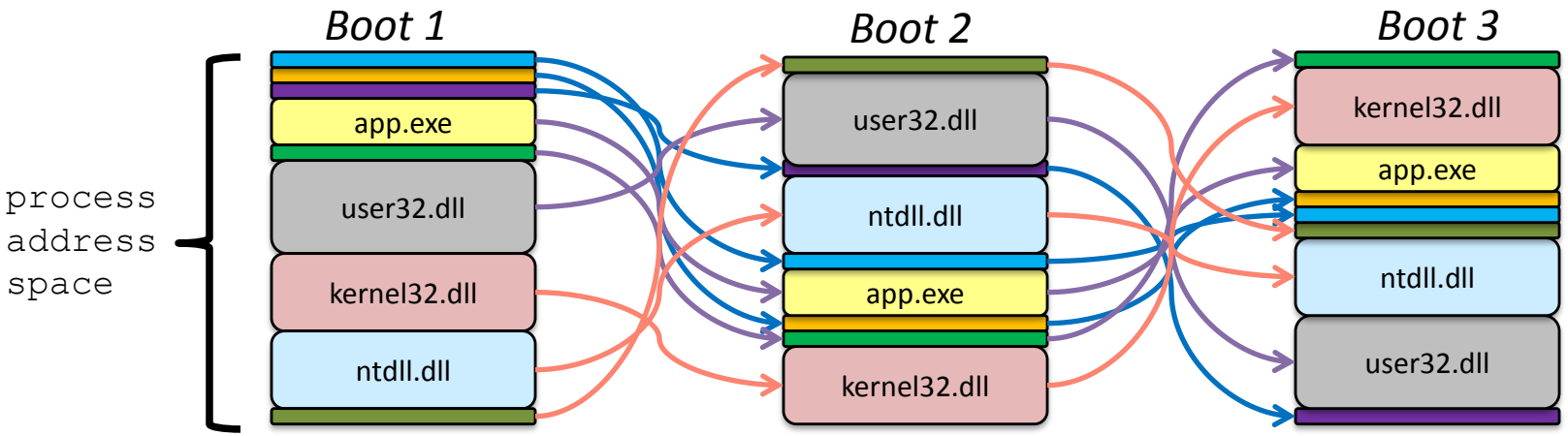


- C++ `operator::new` integer overflow detection [Howard07]

Mitigation

Windows Vista arrives

- Address Space Layout Randomization (ASLR) [PaX02]
  - Make the address space unpredictable



Region	Entropy
Image	8 bits
Heap	5 bits
Stack	14 bits

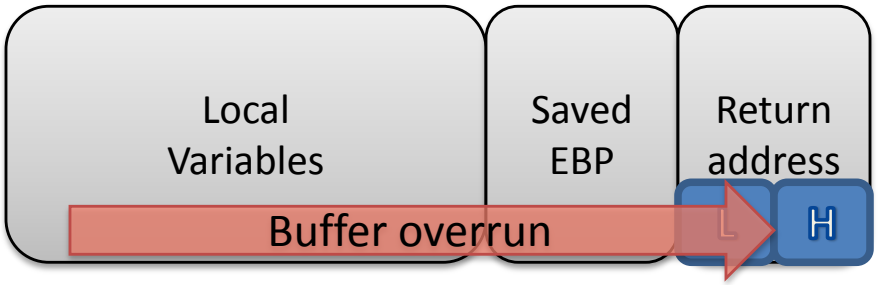
Windows Vista arrives

- Second round of heap mitigations [Marinescu06]
  - Removal of lookaside lists and array lists
  - Block metadata encryption
  - Header cookie scope extended, validated in more places
  - Dynamic change of heap allocation algorithms (LFH)
  - Terminate on heap corruption (default for system apps)
  - `RtlDeleteCriticalSection` technique mitigated by `RtlSafeRemoveEntryList`
  - `FreeList[0]` technique mitigated by `RtlpFastRemoveFreeBlock`

Exploitation

Same ASLR evasions, new OS

- Partial address overwrite [Durden02]




```
memcpy(
   dest,      ← Stack buf
   src,       ← Controlled
   length);   ← Controlled
```

- Address information disclosure [Soeder06]
- Reduced entropy on some platforms [Whitehouse07]
- Brute forcing [Nergal01, Durden02, Shacham04]
- Non-relocateable/predictable addresses [Sotirov08]

Exploitation

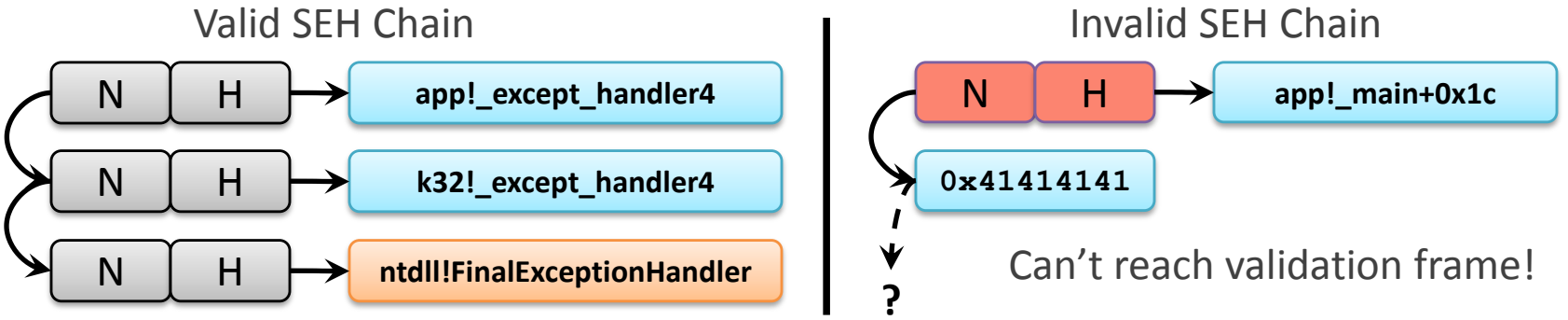
Newer heap techniques, partial & still less universal

- HEAP structure overwrite [Hawkes08]
  - Overwrite pointer in alloc'd chunk with heap base
  - Cause pointer to be freed & then re-allocated
  - Overwrite with specially crafted HEAP structure
- LFH bucket/header overflow [Hawkes08]
- Still need to evade DEP and ASLR if enabled

Mitigation

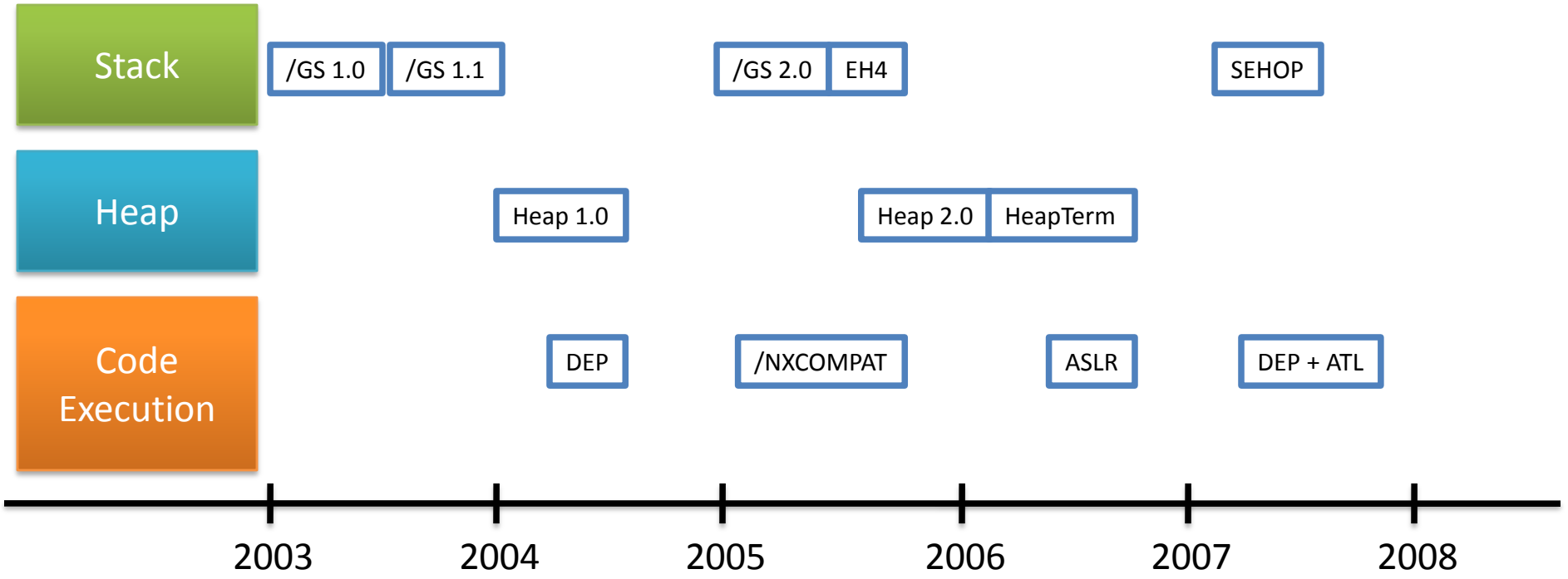
Windows Vista SP1 and Windows Server 2008 RTM

- SEH Overwrite Protection (SEHOP)
  - Dynamic SEH chain validation
  - GS+SEHOP = robust mitigation for most stack buffer overruns!



- Kernel mode ASLR
  - NT/HAL (5 bits of entropy)
  - Drivers (4 bits of entropy)

Exploit Mitigations Timeline



ACT II

THE PRESENT

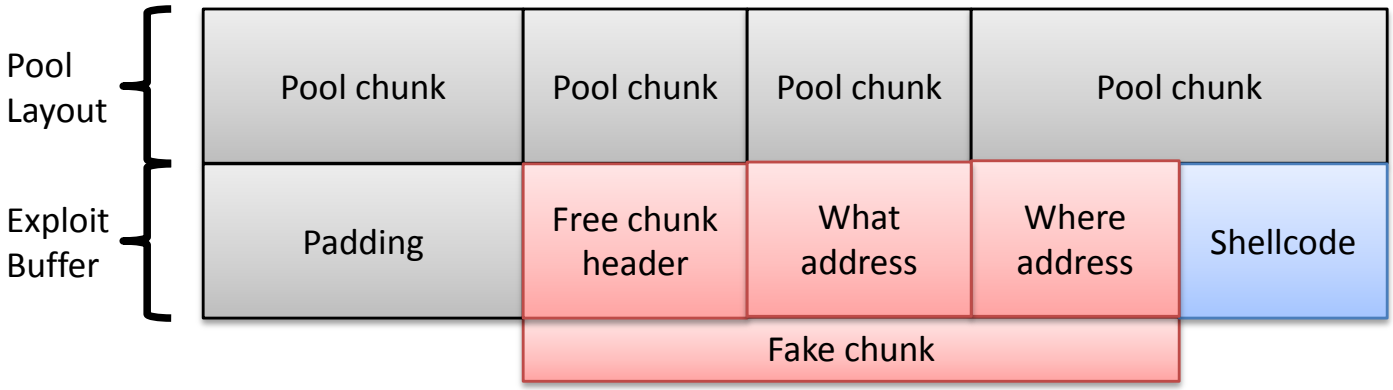
Evolution of OS mitigations

- XPSP2
  - GS applies to both kernel and user mode
  - Heap mitigations are user mode only
- Vista: DEP + ASLR
  - Significantly increase difficulty of user mode exploitation
- Windows 7
  - Further improve kernel mode mitigations

Exploitation

Pool Overruns

- Very similar to Heap Overruns
- Allow arbitrary write what/where via unlink
- Occurs when
  - Merging adjacent free chunks
  - Removing chunk from *ListHead*



Mitigation

Safe unlinking

- Checks integrity of `LIST_ENTRY` structure

  `(E->B->F == E->F->B == E)`

- XPSP2 added this check in Heap

- Windows 7 RC has check in Kernel Pool
  - **Free** as well as Checked builds

Mitigation

Safe Unlinking - benefits

- Security
  - Mitigates arbitrary writes via unlink
  - Other exploit vectors far less generic
- Reliability
  - Detects corruption as early as possible
  - Bugchecks with unique code (0x19, 3, ...)

Mitigation

Safe Unlinking - costs

- Performance
  - Doesn’t hit performance measurably
  - A few extra instructions
  - No additional paging
- Compatibility
  - Pool corruption always bad, no exceptions

Exploitation

MS08-001 IGMP Pool Overrun

- Pool overrun in *tcpip.sys* [Kortchinsky08]
- Root cause is arithmetic overflow in buffer size calculation
  - One loop counts entries using a 16-bit counter
  - Counter wraps around past 65535
  - Memory is allocated based on counter
  - A different loop copies entries into buffer

Exploitation

MS08-001 IGMP Pool Overrun

```
UINT16 SourceCount = 0;
for (...)
{
    if (...) SourceCount++;
}

RecordEntry = ExAllocatePoolWithTag(
  NonPagedPool,
  HeaderSize + (AddressBytes * SourceCount),
  IpGenericPoolTag);
```

Exploitation

MS08-001 IGMP Pool Overrun

```
SourceList = RecordEntry + HeaderSize;
for (...)
{
  if (...){
     RtlCopyMemory(
          SourceList,
          ...,
          AddressBytes);
     SourceList += AddressBytes;
  }
}
```

Pool Mitigations

- Safe unlinking prevents all current variants of documented pool overrun exploits
- "Makes it immeasurably harder to exploit"
  - We're not saying impossible
  - Also mitigates MS07-017, MS08-001, MS08-007
- Only safe unlinking right now
  - No pointer encoding, cookies etc
  - No protection of *LookAside* lists

Other enhancements

- Increased entropy for kernel mode ASLR
  - Drivers: 6 bits on x86, 8 bits on x64

ACT III

THE FUTURE

Mitigation

GS – effective or not?

- Vista
  - GS fundamentally the same
  - Many bypasses closed off via OS improvements
    - EH abuse
    - NX/DEP
    - ASLR
- Vista released worldwide 30th January 2007
- MS07-017 security bulletin 10th April 2007
  - Trivially exploitable stack overflow in ANI file parsing

Mitigation

The GS heuristic

- Not all functions GS-protected
  - Obvious and less obvious performance cost
- Insert cookie for
  - arrays of size>4 with element size <= 2 (char/wchar)
  - Structures containing arrays with element size <=2
- Originally designed to mitigate overflows arising from untrusted **string** data

Mitigation

MS07-017 – ANI stack overflow

- The target of the overflow was a ANIHEADER structure on the stack:

```
typedef struct _ANIHEADER {
   DWORD cbSizeof;
   DWORD cFrames;
   DWORD cSteps;
   DWORD cx, cy;
   DWORD cBitCount, cPlanes;
   DWORD jifRate;
   DWORD fl; } ANIHEADER, *PANIHEADER;
```

Mitigation

MS07-017 – ANI stack overflow

- The ANIHEADER overflow equivalent to:

```
ANIHEADER   myANIheader;
memcpy(&myANIheader,
    untrustedFileData->headerdata,
    untrustedFileData->headerlength);
```

- No character buffers on the stack
  - ⇒No GS protection
  - ⇒myANIheader is being *treated* like a character buffer

Mitigation

Target buffer mitigated by GS?

Security bulletin	GS?	
MS03-026 (Blaster)	**Yes**	
MS06-040	**Yes**	
MS07-029	**Yes**	
MS04-035 (Exchange)	**No**	DWORD array
MS06-054 (.PUB)	**No**	structure populated from file
MS07-017 (.ANI)	**No**	structure populated from file

Mitigation

Vista SP1

- In development at time of ANI vulnerability
- #pragma strict_gs_check?
- More aggressive GS heuristic
- Much more aggressive GS heuristic
- Any address-taken local variable is considered a potential target!

Mitigation

strict GS

Target buffer mitigated by GS?

Security bulletin	Legacy GS		Strict GS
MS03-026 (Blaster)	Yes		Yes
MS06-040	Yes		Yes
MS07-029	Yes		Yes
MS04-035 (Exchange)	No	DWORD array	Yes
MS06-054 (.PUB)	No	Data structure	Yes
MS07-017 (.ANI)	No	Data structure	Yes

Mitigation

strict GS

```
#pragma strict_gs_check(on)
void main()
{
    int i;

    printf(“%d”, (int) &i); // address-taken
}
```

Mitigation

strict GS

- Applied in a very targeted way for Vista SP1

Binary	Functions in DLL	OS	Number of cookies	% protected functions	Factor increase
qasf.dll	1526	Vista RTM (GS)	58	3.80%	3.5
		Vista SP1 (strict GS)	202	13%	
avifil32.dll	494	Vista RTM (GS)	40	8.10%	3.4
		Vista SP1 (strict GS)	134	27%	
WMASF.dll	1484	Vista RTM (GS)	40	2.70%	13.1
		Vista SP1 (strict GS)	524	35%	

- But not suitable for system-wide deployment

  $\Rightarrow$GS++

Issues of scale

Vista SP1 approach was targeted

Can we make the default /GS better?

Enhancing GS

- Increased coverage
  - Protect more stuff

- Smarter coverage
  - Don’t protect where it’s unnecessary

- Different models for how this might work

Mitigation

GS++ heuristic ?

- All arrays?
- All structures?

Performance concerns!

What subset is most likely to contain untrusted data?

Mitigation

GS++ heuristic

Arrays where element type not of pointer type:



```
char myBuf[]
DWORD myBuf[]
HANDLE myBuf[]
```

and size of array is >2 elements

Mitigation

GS++ heuristic

- Structures:
  - Containing an array where element type is not of pointer type.
  - Made up of pure data:
    - No members of pointer type
    - >8 bytes in size
    - Default constructor/destructor

```
struct _ANIHEADER{
DWORD cbSizeof;
DWORD cFrames;
DWORD cSteps;
DWORD cx, cy;
DWORD cBitCount
DWORD cPlanes;
DWORD jifRate;
DWORD fl; };
```

Mitigation

Impact on cookie count

GS-protected functions in sample code

	Original GS	VS2010 GS
User/client	9608	12846
Kernel	2361	4686
User/client (% total fns)	6.0%	8.0%
Kernel mode (% total fns)	5.2%	10.4%

⇒Cookie increase between 2% and 5%

Mitigation

GS optimization

- No GS cookies when usage is provably safe

```
STDAPI ConsumeData(BYTE *pbData)
{
   BYTE Temp[MAX];

   if (pbData)
   {
     ...
         memcpy (Temp,  pbData,  ARRAYSIZE(Temp));
     ...
   }
```

Mitigation

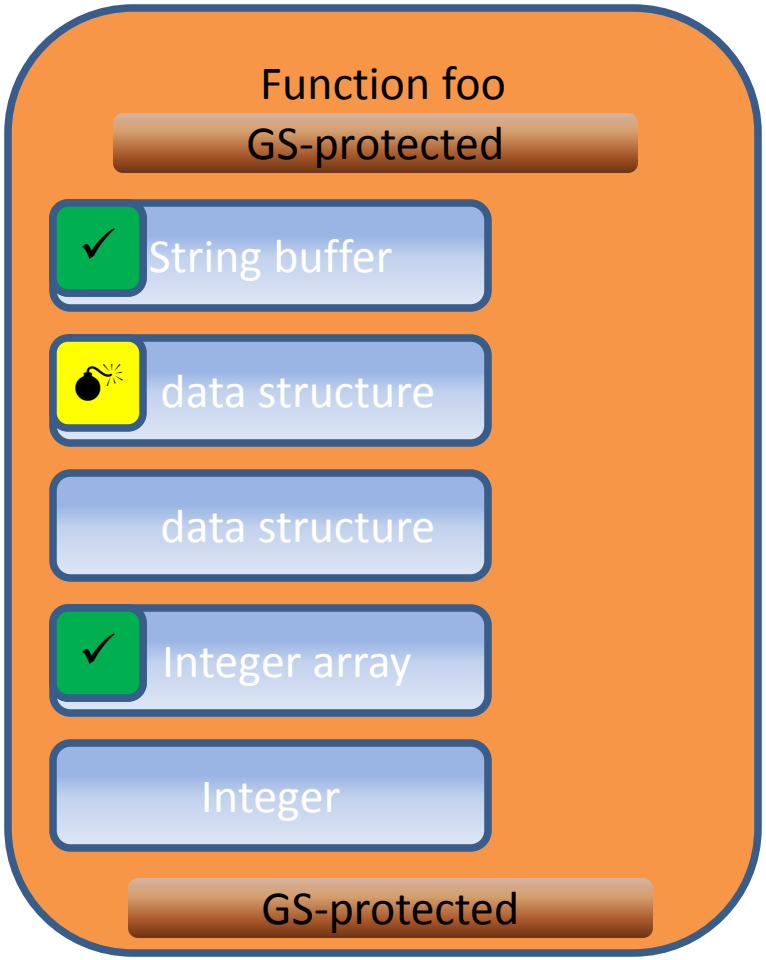
GS optimization

- No GS cookies when usage is provably safe

```
STDAPI FillBuffer(wchar_t *pBuf, int count)
{
    ...
    memcpy (pBuf, GetData(), count*sizeof(wchar_t));
    ...
}
STDAPI ParseData()
{
   wchar_t buffer[BUF_SIZE];
   FillBuffer(buffer, _countof(buffer));
   ...
}
```

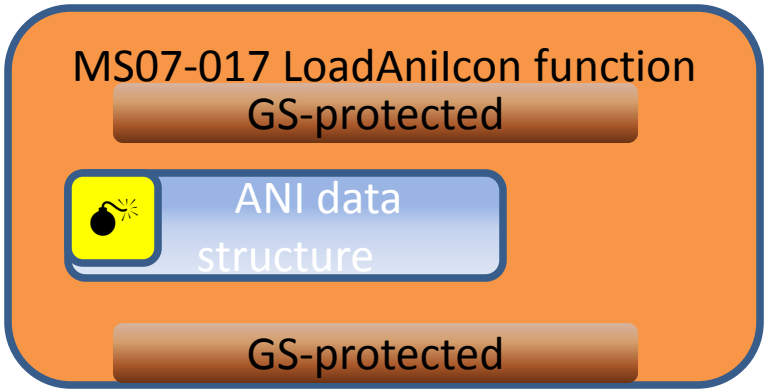
Mitigation

GS enhancements [VS2010]



- GS heuristic
  - Identify more *potential* hazards
- GS optimization
  - Some *potential* hazards turn out to be **safe**

Increased scope of heuristic:



Impact on cookie count

	Original GS	VS2010 GS	VS2010 GS [with GS opt]
User/client	9608	12846	11654
Kernel	2361	4686	3909
User/client (% total fns)	6.0%	8.0%	7.3%
Kernel mode (% total fns)	5.2%	10.4%	8.7%

Mitigation

Impact on stack overflow security bulletins

Security bulletin	Original GS	VS2010 GS	Strict GS
MS03-026 (Blaster)	Yes	Yes	Yes
MS06-040	Yes	Yes	Yes
MS07-029	Yes	Yes	Yes
MS04-035 (Exchange)	No	Yes	Yes
MS06-054 (.PUB)	No	Yes	Yes
MS07-017 (.ANI)	No	Yes	Yes

Mitigation

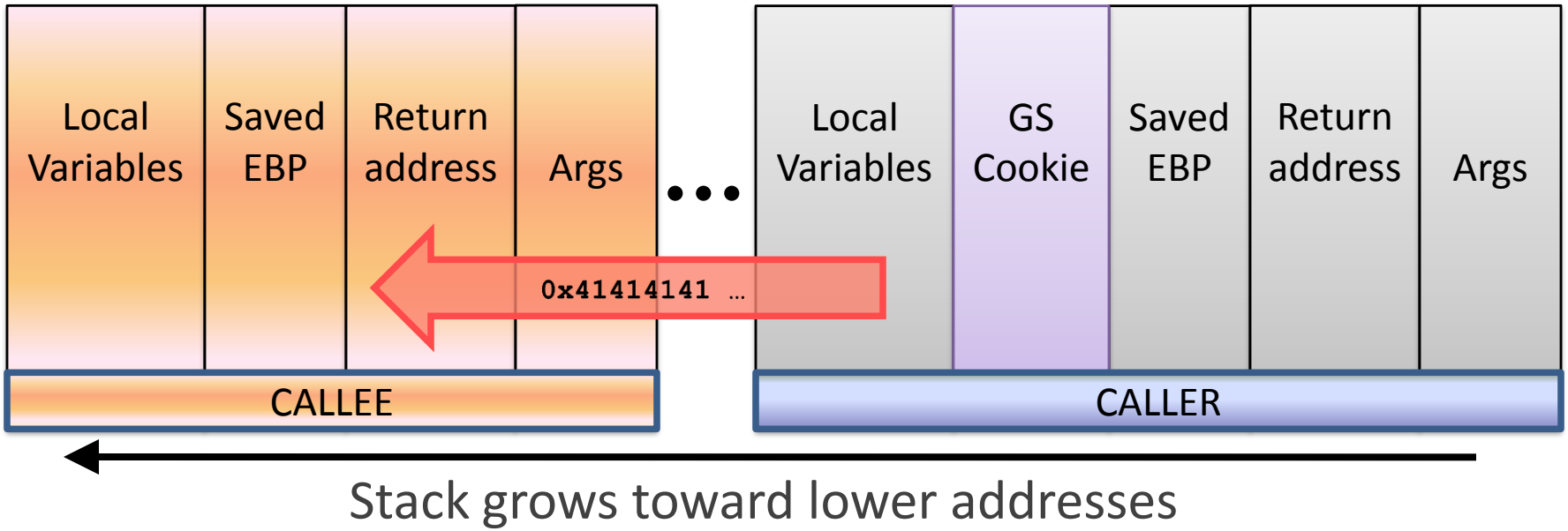
… but GS not a panacea

Security bulletin	Original GS	VS2010 GS	Strict GS
MS03-026 (Blaster)	Yes	Yes	Yes
MS06-040	Yes	Yes	Yes
MS07-029	Yes	Yes	Yes
MS04-035 (Exchange)	No	Yes	Yes
MS06-054 (.PUB)	No	Yes	Yes
MS07-017 (.ANI)	No	Yes	Yes
MS08-072	N/A	N/A	N/A
MS08-067	N/A	N/A	N/A

Mitigation

Still need to write secure code!

- Even the new heuristic will not cover all cases
- GS does not apply to some types of stack-based attacks (for example underflow).



Mitigation

Enhanced GS

- In Visual Studio 2010
  - Same /GS switch
  - Enhanced GS++ heuristic
  - GS optimization

Conclusion

- Modern exploitation is difficult & not universal
  - Techniques are tied to specific vulnerability scenarios
- Gaps do exist that can make exploitation easier
  - But these are the exception, not the rule
- We are committed to protecting our customers
  - Continued improvement of our mitigation technology
  - Providing actionable exploitability data with bulletins

Questions?

Thank you!

- Security Science at Microsoft
  - http://www.microsoft.com/security/msec/default.aspx
- Security Research & Defense blog
  - http://blogs.technet.com/swi/default.aspx

References

[Aelph96] Aleph1. Smashing the stack for fun and profit. Phrack 49. Nov, 1996.

[Solar97] Solar Designer. Getting around non-executable stack (and fix). Bugtraq. Aug, 1997.

[Solar00] Solar Designer. JPEG COM Marker Processing Vulnerability in Netscape Browsers. Bugtraq. Jul, 2000.

[Maxx01] MaXX. Vudo malloc tricks. Phrack 57. Aug, 2001.

[Anon01] Anonymous. Once upon a free(). Phrack 57. Aug, 2001.

[Nergal01] Nergal. Advanced return-into-libc exploits (PaX case study). Phrack 58. Dec, 2001.

[Ren02] Chris Ren, Michael Weber, and Gary McGraw. Microsoft Compiler Flaw Technical Note. Feb, 2002.

[Durden02] Tyler Durden. Bypassing PaX ASLR Protection. Phrack 59. Jul, 2002.

[PaX02] PaX Team. Address space layout randomization. 2002.

[Litchfield03] David Litchfield. Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server. Sep, 2003.

[Anisimov04] Alexander Anisimov. Defeating Microsoft Windows XP SP2 Heap protection. 2004.

[Conover04] Matt Conover, Oded Horovitz. Reliable Windows Heap Exploits. CanSecWest. 2004.

[Conover04-2] Matt Conover. Windows Heap Exploitation (Win2KSP0 through WinXPSP2). SyScan. 2004.

[Litchfield04] David Litchfield. Windows Heap Overflows. Black Hat USA. 2004.

[Falliere05] Nicolas Falliere. A new way to bypass Windows heap protections. Sep, 2005.

[Skape05] Skape, Skywing. Bypassing Windows Hardware-enforced DEP. Uninformed. Sep, 2005.

[Nagy05] Ben Nagy. Beyond NX: an attacker's guide to Windows anti-exploitation technology. PakCon. Oct, 2005.

[Moore05] Brett Moore. Exploiting FreeList[0] on Windows XP Service Pack 2. Dec, 2005.

[Marinescu06] Adrian Marinescu. Windows Vista Heap Management Enhancements. Black Hat USA. Aug, 2006.

References

[Skape06] Skape. Preventing the Exploitation of SEH Overwrites. Uninformed. Sep, 2006.

[Soeder06] Derek Soeder. Memory Retrieval Vulnerabilities. Oct, 2006.

[Howard07] Michael Howard. Why Windows Vista is unaffected by the VML Bug. Jan, 2007.

[Sotirov07] Alexander Sotirov. Windows ANI header buffer overflow. Mar, 2007.

[Waisman07] Nicolas Waisman. Understanding and Bypassing Windows Heap Protection. Jul, 2007.

[Whitehouse07] Ollie Whitehouse. GS and ASLR in Windows Vista. Black Hat USA. Aug, 2007.

[Dowd08] Mark Dowd. Application-specific attacks: Leveraging the ActionScript Virtual Machine. Apr, 2008.

[Lawrence08] Eric Lawrence. IE8 Security Part I: DEP/NX Memory Protection. Apr, 2008.

[Sotirov08] Alexander Sotirov and Mark Dowd. Bypassing Browser Memory Protections. Black Hat USA. Aug, 2008.

[Hawkes08] Ben Hawkes. Attacking the Vista Heap. Black Hat USA. Aug, 2008.

[Shacham08] Hovav Shacham. Return-Oriented Programming. Exploits Without Code Injection. Black Hat USA. Aug, 2008.

[Alberts09] Bas Alberts. A bounds check on the exploitability index. Feb, 2009.

[Korthcinsky08] Kostya Kortchinsky. Real World Kernel Pool Exploitation, SyScan 08 Hong Kong