Exploit Mitigation Improvements in Windows 8

Ken Johnson, Matt Miller
Microsoft Security Engineering Center (MSEC)

Black Hat USA 2012
Acknowledgements

Many individuals have worked very hard to deliver the improvements we will discuss

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR/Silverlight</td>
<td>Reid Borsuk, Jesse Collins, Jeffrey Cooperstein, Nick Kramer</td>
</tr>
<tr>
<td>Visual Studio</td>
<td>Jonathan Caves, Tanveer Gani, Mark Hall, Lawrence Joel, Louis Lafreniere, Mark Levine, Steve Lucco, Mark Roberts, Andre Vachon, YongKang Zhu</td>
</tr>
<tr>
<td>Internet Explorer</td>
<td>David Fields, Forbes Higman, Eric Lawrence, Zach Murphy, Justin Rogers</td>
</tr>
<tr>
<td>Microsoft Research</td>
<td>Richard Black, Miguel Castro, Manuel Costa, Ben Livshits, Jay Stokes, Ben Zorn</td>
</tr>
</tbody>
</table>
Windows 8 Security Overview

**Secure Development**

The development of Windows 8 followed the Security Development Lifecycle (SDL) which defines best practices for secure software design, implementation, and testing.

**Securing the Boot**

Windows 8 includes new protections against root and boot kits. UEFI systems prevent malware from starting before Windows and protects the remainder of the boot process, including early launch antimalware software.

**Securing the Core**

Windows 8 includes numerous security features, such as mitigations, that make it difficult and costly for malware authors to develop reliable exploits for vulnerabilities.

**Securing After the Boot**

Windows Defender is shipped with every edition of Windows 8 and Internet Explorer’s Application reputation features have been moved into the platform such that users are protected regardless of the browser they use.
Framing the problem with exploit economics

We want to minimize ROI
Attacker Return = Opportunities to use X Gains per use
Attacking exploit economics

Gains per use = Cost to acquire vulnerability - Cost to weaponize

Today we will describe exploit mitigation improvements in Windows 8 that increase the cost of developing reliable exploits

- Use sandboxing to protect user privacy & data
- Find and eliminate vulnerability classes
- Use telemetry to minimize attack
- Break cookbook techniques & make exploits unreliable
- Attackers want to maximize ROI

We want to minimize ROI
VOID Function(LPCSTR Input) {
    CHAR Buffer[256];
    strcpy(Buffer, Input);
}

History of exploit mitigations on Windows

Stack-based vulnerability

- Linear return address overwrite
- Linear local variable overwrite
- Linear parameter overwrite
- Linear SEH record overwrite
- Non-linear/backward overwrites (e.g. a[n] = 0)

GS v1 (2002)
- GS v2 (2005)
- GS v3 (2010)

SafeSEH (2003)
SEHOP (2008)

Control of Instruction Pointer

Stack-based exploitation and mitigation techniques
History of exploit mitigations on Windows

VOID Function(LPCSTR Input) {
    PCHAR *Buffer = malloc(256);
    strcpy(Buffer, Input);
}

Heap-based vulnerability

Coalesce unlink overwrite

FreeList[] attacks

Lookaside list attacks

Heap cache attacks

LFH FreeEntryOffset attack

Application specific data overwrites

Use-after-free/Double free/Dangling pointer

Safe unlinking
(2004)

Vista heap hardening
(2006)

Overwrite or control a function pointer

Control of Instruction Pointer

Heap-based exploitation and mitigation techniques
History of exploit mitigations on Windows

**Control of Instruction Pointer**

- **Execute code from stack**
- **Execute code from heap (incl. heap spray)**
- **Execute JIT’d code (JIT spray)**
- **Execute code from a loaded image**

**DEP** (2004)

- **IE9 JS JIT mitigations** (2011)

**Return into libc** (many variants)

**Return oriented programming (ROP)**

**ASLR** (2006)

- **Predictable mappings/info leaks**

Executing arbitrary code after gaining control of EIP

**eip=41414141 esp=0023f7dc ebp=0023f808... cs=0023 ss=002b ds=002b es=002b fs=... 41414141 ?? ?? ??**

Arbitrary code execution
## The state of memory safety exploits

| Most systems are not compromised by exploits | • About 6% of MSRT detections were likely caused by exploits [29]  
• Updates were available for more than a year for most of the exploited issues [29] |
| Most exploits target third party applications | • 11 of 13 CVEs targeted by popular exploit kits in 2011 were for issues in non-Microsoft applications [27] |
| Most exploits target older versions of Windows (e.g. XP) | • Only 5% of 184 sampled exploits succeeded on Windows 7 [28]  
• ASLR and other mitigations in Windows 7 make exploitation costly [30] |
| Most exploits fail when mitigations are enabled | • 14 of 19 exploits from popular exploit kits fail with DEP enabled [27]  
• 89% of 184 sampled exploits failed with EMET enabled on XP [28] |
| Exploits that bypass mitigations & target the latest products do exist | • Zero-day issues were exploited in sophisticated attacks (Stuxnet, Duqu)  
• Exploits were written for Chrome and IE9 for Pwn2Own 2012 |

**Bottom line: we must continue to increase the cost of exploitation for attackers**
Objectives & focus areas in Windows 8

**Objectives**
- Mitigate entire classes of vulnerabilities
- Break known exploitation techniques
- Increase the cost of exploitation in key domains

**Code Generation Security**

**Address Space Layout Randomization**

**Windows Heap**

**Windows Kernel**

**Default Settings**
Enhanced /GS, range checks, sealed optimization, and virtual table guard

CODE GENERATION
Enhanced /GS

- Enhanced GS stack buffer overrun protection
  - Released with Visual Studio 2010 [1]
  - Windows 8 is built with this enabled

- GS heuristics now protect more functions
  - Non-pointer arrays and POD structures

- GS optimization removes unnecessary checks
  - Safety proof means no check is needed

- Closes gaps in protection
  - MS04-035, MS06-054, MS07-017 (ANI)
Range Checks

• Compiler-inserted array bounds check (via /GS)

```c
CHAR Buffer[256];
UINT i; // possibly attacker controlled

if (i >= ARRAYSIZE(Buffer)) {
    __report_rangecheckfailure();
}
Buffer[i] = 0;
```

• Completely mitigates certain vulnerabilities
  – CVE-2009-2512, CVE-2010-2555

• Bounds check insertion is limited to specific scenarios
  – Assignment of NUL to a fixed-size stack/global array
Sealed optimization

• Optimization for “sealed” C++ types & methods

```cpp
class COptionElement sealed : public CElement
{
    DECLARE_CLASS_TYPES(COptionElement, CElement)
}
```

• Virtual method calls become direct calls

```cpp
COptionElement *optionElement;
optionElement->IsEnabled();
```

• Eliminating indirect calls reduces exploitation attack surface
  – Helps mitigate vulnerabilities like CVE-2011-1996
  – Devirtualized ~4,500 calls in mshtml.dll and ~13,000 in mso.dll
Virtual Table Guard

Probabilistic mitigation for vulnerabilities that enable vtable ptr corruption

IE10 has enabled this for a handful of key classes in mshtml.dll

<table>
<thead>
<tr>
<th>CElement::`vftable'</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualMethod1</td>
</tr>
<tr>
<td>VirtualMethod2</td>
</tr>
<tr>
<td>VirtualMethod3</td>
</tr>
<tr>
<td>VirtualMethod4</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

Bonus entry added to vtable. ASLR makes this entry’s value unknown to the attacker.

Enabled by adding an annotation to a C++ class

mshtml!CElement::Doc:

```
63700e70  mov     eax, dword ptr [ecx]
63700e72  cmp     [eax+1B8h], offset mshtml!_vtguard
63700e7c  jne     mshtml!CElement::Doc+0x18 (63700e88)
63700e7e  call    dword ptr [eax+0ACh]
63700e84  mov     eax, dword ptr [eax+0Ch]
63700e87  ret
63700e88  call    mshtml!__report_gsfailure
```

Check added at virtual method call sites. If vtable[vtable_vte] != vtguard then terminate the process.
Force ASLR, bottom-up/top-down randomization, and high entropy

ADDRESS SPACE LAYOUT RANDOMIZATION
Retrospective: ASLR

- ASLR was first introduced in Windows Vista
  - Led to a big shift in attacker mentality

- Attackers now depend on gaps in ASLR
  - EXEs/DLLs not linked with /DYNAMICBASE [2]
  - Address space spraying (heap/JIT) [3]
  - Predictable memory regions [4]
  - Information disclosures [5]

- ASLR has been substantially improved in Windows 8
Force ASLR

- Many exploits depend on non-ASLR DLLs
  - Behaves as if an image’s preferred base is not available
  - Bottom-up randomization provides entropy for these images
- Processes can now force non-ASLR images to be randomized
  - Also supported on Windows 7 with KB2639308 installed

Outcome: attackers can no longer rely on non-ASLR images
Bottom-up & top-down randomization

Windows 7
- Heaps and stacks are randomized
- PEBs/TEBs are randomized, but with limited entropy
- VirtualAlloc and MapViewOfFile are not randomized
- Predictable memory regions can exist as a result

Windows 8
- All bottom-up/top-down allocations are randomized
- Accomplished by biasing start address of allocations
- PEBs/TEBs now receive much more entropy
- Both are opt-in (EXE must be dynamicbase)

Outcome: predictable memory regions have been eliminated
High entropy ASLR for 64-bit processes

ASLR in Windows 8 takes advantage of the large address space (8TB) of 64-bit processes

- **High entropy bottom-up randomization**
  - 1 TB of variance in bottom-up start address
  - Breaks traditional address space spraying (heap/JIT)
  - Processes must opt-in to receive this behavior

- **High entropy top-down randomization**
  - 8 GB of variance in top-down start address
  - Automatically enabled if top-down randomization is on

- **High entropy image randomization**
  - Images based above 4 GB receive more entropy
  - All system images have been moved above 4 GB

Outcome: probability of guessing an address is decreased and disclosures of memory addresses must include more than the low 32 bits
### ASLR entropy improvements

<table>
<thead>
<tr>
<th>Entropy (in bits) by region</th>
<th>Windows 7</th>
<th></th>
<th>Windows 8</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>32-bit</td>
<td>64-bit</td>
<td>32-bit</td>
<td>64-bit</td>
<td>64-bit (HE)</td>
</tr>
<tr>
<td>Bottom-up allocations (opt-in)</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>Stacks</td>
<td>14</td>
<td>14</td>
<td>17</td>
<td>17</td>
<td>33</td>
</tr>
<tr>
<td>Heaps</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>24</td>
</tr>
<tr>
<td>Top-down allocations (opt-in)</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>PEBs/TEBs</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>EXE images</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>17*</td>
<td>17*</td>
</tr>
<tr>
<td>DLL images</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>19*</td>
<td>19*</td>
</tr>
<tr>
<td>Non-ASLR DLL images (opt-in)</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>24</td>
</tr>
</tbody>
</table>

* 64-bit DLLs based below 4GB receive 14 bits, EXEs below 4GB receive 8 bits

ASLR entropy is the same for both 32-bit and 64-bit processes on Windows 7

64-bit processes receive much more entropy on Windows 8, especially with high entropy (HE) enabled
Removal of information disclosure vectors

- Information disclosures can be used to bypass ASLR
- Disclosure via an arbitrary read is now less reliable
  - Predictable mappings have been eliminated
- SharedUserData is still predictable, but less useful
  - Image pointers have been removed, breaking known techniques [4,6]
Integrity checks, guard pages, and allocation order randomization

WINDOWS HEAP
Retrospective: Windows Heap

• Windows Vista heap hardening was very effective [11]
  – Only one documented exploit that corrupts metadata [9]

• New attacks have been proposed by researchers
  – Corrupting the HEAP data structure [7]
  – LFH bucket overwrite [7]
  – LFH FreeEntryOffset corruption and depth desync [8,12]

• Real-world exploits target app data on the heap [10]
  – No heap safeguards exist today for this
Windows 8 heap architecture

The general design of the Windows heap is unchanged in Windows 8

- **HeapAlloc(heap, flags, size)**
- **Frontend allocator (LFH)**
  - Used for sizes < 16KB
- **Backend allocator**
  - Used by frontend and for sizes less than 512K (x86) or 1MB (x64)
- **Virtual memory allocator**
  - Used by backend and for large allocation sizes
- **Kernel memory manager**
## LFH design changes & integrity checks

<table>
<thead>
<tr>
<th>Change in Windows 8</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFH is now a bitmap-based allocator</td>
<td>LinkOffset corruption no longer possible [8]</td>
</tr>
<tr>
<td>Multiple catch-all EH blocks removed</td>
<td>Exceptions are no longer swallowed</td>
</tr>
<tr>
<td>HEAP handle can no longer be freed</td>
<td>Prevents attacks that try to corrupt HEAP handle state [7]</td>
</tr>
<tr>
<td>HEAP CommitRoutine encoded with global key</td>
<td>Prevents attacks that enable reliable control of the CommitRoutine pointer [7]</td>
</tr>
<tr>
<td>Validation of extended block header</td>
<td>Prevents unintended free of in-use heap blocks [7]</td>
</tr>
<tr>
<td>Busy blocks cannot be allocated</td>
<td>Prevents various attacks that reallocate an in-use block [8,11]</td>
</tr>
<tr>
<td>Heap encoding is now enabled in kernel mode</td>
<td>Better protection of heap entry headers [19]</td>
</tr>
</tbody>
</table>

**Outcome:** attacking metadata used by the heap is now even more difficult
Guard pages

- Guard pages are now used to partition the heap
  - Designed to prevent & localize corruption in some cases
  - Touching a guard page results in an exception

- Insertion points for guard pages are constrained
  - Large allocations
  - Heap segments
  - Max-sized LFH subsegments (probabilistic on 32-bit)
Allocation order randomization

- Allocation order is now nondeterministic (LFH only)
  - Exploits often rely on surgical heap layout manipulation [10]
  - Randomization makes heap normalization unreliable

Windows 7 LFH block allocation behavior

Windows 8 LFH block allocation behavior

- Maximizing reliability is more challenging
  - Application-specific and vulnerability-specific
  - May require corrupting more data (increasing instability)
  - May require allocating more data (triggering guard pages)
DEP, ASLR, SMEP/PXN, NULL dereference protection, and pool integrity checks

WINDOWS KERNEL
Retrospective: Windows Kernel

- Kernel vulnerabilities have been less targeted
  - Relatively few remote kernel exploits exist
  - User mode exploitation is better researched

- Attack focus is shifting more toward the kernel
  - Interest in sandbox escapes is increasing
  - Local kernel exploitation techniques well-understood
  - New kernel pool attacks have been proposed [13]
  - Sophisticated remote kernel exploits exist [14,21]
DEP is broadly enabled in the Windows 8 kernel

Many kernel memory regions were unnecessarily executable in Windows 7 and prior

<table>
<thead>
<tr>
<th></th>
<th>x86 (PAE)</th>
<th>x64</th>
<th>ARM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Win7</td>
<td>Win8</td>
<td>Win7</td>
</tr>
<tr>
<td>Paged pool</td>
<td>X</td>
<td>X</td>
<td>NX</td>
</tr>
<tr>
<td>Non-paged pool</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Non-paged pool (NX)</td>
<td>N/A</td>
<td>NX</td>
<td>N/A</td>
</tr>
<tr>
<td>Session pool</td>
<td>X</td>
<td>X</td>
<td>NX</td>
</tr>
<tr>
<td>Image data sections</td>
<td>X</td>
<td>X</td>
<td>NX</td>
</tr>
<tr>
<td>Kernel stacks</td>
<td>NX</td>
<td>NX</td>
<td>NX</td>
</tr>
<tr>
<td>Idle/DPC/Initial stacks</td>
<td>X</td>
<td>NX</td>
<td>X</td>
</tr>
<tr>
<td>Page table pages</td>
<td>X</td>
<td>NX</td>
<td>X</td>
</tr>
<tr>
<td>PFN database</td>
<td>X</td>
<td>NX</td>
<td>X</td>
</tr>
<tr>
<td>System cache</td>
<td>X</td>
<td>NX</td>
<td>X</td>
</tr>
<tr>
<td>Shared user data</td>
<td>X</td>
<td>NX</td>
<td>X</td>
</tr>
<tr>
<td>HAL heap</td>
<td>X</td>
<td>NX</td>
<td>X</td>
</tr>
</tbody>
</table>

X = executable  NX = non-executable

Windows 8 introduces NonPagedPoolNx for non-executable non-paged pool allocations (default on ARM)

NX HAL heap and NonPagedPoolNx break the assumptions of exploits for MS09-050
Kernel ASLR improvements

• Kernel ASLR was first added in Server 2008 RTM
  – 4 bits of entropy for drivers, 5 bits for NTOS/HAL
  – Driver entropy was improved in Windows 7

• Entropy has been further improved in Windows 8
  – Biasing of kernel segment base
  – NTOS/HAL receive 22 bits (64-bit) and 12 bits (32-bit)
  – Various boot regions also randomized (P0 idle stack)
Support for SMEP/PXN

• New processor security feature
  – Prevents supervisor from executing code in user pages
  – Most exploits for local kernel EOPs rely on this today
  – Requires Intel Ivy Bridge or ARM with PXN support

• SMEP/PXN + DEP make exploitation more difficult
  – Strong mitigation for some issues (CVE-2010-2743 from Stuxnet)
  – Attackers need to leverage code in kernel images [15]
NULL dereference protection

• Kernel NULL dereferences are a common issue
  – Examples include MS08-025, MS08-061, MS09-001

• Local exploitation is generally straightforward
  – NULL is part of the user mode address space
  – Kernel currently allows user processes to map NULL page

• Windows 8 prohibits mapping of the first 64K
  – All kernel NULL dereferences become a DoS (not EoP)
  – NTVDM has been disabled by default as well
  – Enabling NTVDM will disable NULL dereference protection
Kernel pool integrity checks

• The kernel pool allocator is similar to the heap
  – Implementation is very different, though

• New integrity checks block various attacks [13]
  – Process quota pointer encoding
  – Lookaside, delay free, and pool page cookies
  – PoolIndex bounds check
  – Additional safe unlinking checks
Other improvements

• Safe unlinking has been enabled globally
  – Previously only used in the heap and kernel pool
  – Now applies to all usage of LIST_ENTRY, closing known gaps [16]
  – New “FastFail” mechanism enables rapid & safe process termination

• Improved entropy for GS and ASLR
  – Use of PRNG seeded by TPM/RDRAND/other sources
  – Hardcoded GS initialization is overridden by the OS
  – Addresses weaknesses described in attack research [17, 18]

• Object manager hardened against reference count overflows

• Resolved kernel information disclosure via certain system calls [22]
ARM, Inbox, Windows 8 style apps, IE 10, the new Office, and other applications

DEFAULT SETTINGS
ARM default settings

All applicable mitigations are enabled on ARM

<table>
<thead>
<tr>
<th>Feature</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEP</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (images)</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (force relocate)</td>
<td>N/A (all images are ASLR)</td>
</tr>
<tr>
<td>ASLR (bottom-up)</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (top-down)</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (high entropy)</td>
<td>N/A (not 64-bit)</td>
</tr>
<tr>
<td>SEHOP</td>
<td>N/A (not needed)</td>
</tr>
<tr>
<td>Heap termination</td>
<td>On</td>
</tr>
<tr>
<td>Kernel NULL dereference</td>
<td>On</td>
</tr>
<tr>
<td>Kernel SMEP</td>
<td>On</td>
</tr>
</tbody>
</table>

Lack of application compatibility concerns enables us to be more aggressive

ARM PE images must opt-in to DEP and ASLR

Kernel will fail to load images that do not
Application default settings

All applicable mitigations are enabled for Windows 8 UI style apps

<table>
<thead>
<tr>
<th>Default settings for Windows 8 client</th>
<th>32 bit (x86)</th>
<th>64 bit (x64)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Win8 UI apps</td>
<td>Inbox</td>
</tr>
<tr>
<td>DEP</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (images)</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (force relocate)</td>
<td>On</td>
<td>OptIn</td>
</tr>
<tr>
<td>ASLR (bottom-up)</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (top-down)</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>ASLR (high entropy)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>SEHOP</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>Heap termination</td>
<td>On</td>
<td>On</td>
</tr>
</tbody>
</table>

Internet Explorer 10 and the new Office also enable all applicable mitigations.
Enabling opt-in mitigations

**Opt-in methods**
- “MitigationOptions” Image File Execution Option (IFEO)
- Process creation attribute (via UpdateProcThreadAttribute)
- SetProcessMitigationPolicy API
- Linker flag

<table>
<thead>
<tr>
<th>Opt-in mitigation</th>
<th>IFEO</th>
<th>Proc Attr</th>
<th>API</th>
<th>Linker flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom-up randomization</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>/DYNAMICBASE (on EXE)</td>
</tr>
<tr>
<td>Top-down randomization</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>/DYNAMICBASE (on EXE)</td>
</tr>
<tr>
<td>Bottom-up randomization (high entropy)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>/HIGHENTROPYVA (on EXE)</td>
</tr>
<tr>
<td>ASLR</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>/DYNAMICBASE</td>
</tr>
<tr>
<td>Force ASLR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>None</td>
</tr>
<tr>
<td>DEP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>/NXCOMPAT (on EXE)</td>
</tr>
<tr>
<td>SEHOP</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>None*</td>
</tr>
<tr>
<td>Heap termination</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>None*</td>
</tr>
</tbody>
</table>

* EXEs with a subsystem version >= 6.2 will automatically enable these mitigations
Expectations for exploits on Windows 8

• Writing exploits for Windows 8 will be very costly
  – Some vulnerability classes are now entirely mitigated
  – Many attack techniques are now broken or unreliable

• Attackers will likely focus their attention on
  – Desktop apps that do not enable all applicable mitigations
  – Desktop apps running on previous versions of Windows
  – Refining methods of disclosing address space information
  – Researching new exploitation techniques [20]

• We will continue to evolve our mitigation technologies
Call to action

- Upgrade to Windows 8 and IE 10 😊
  - 64-bit is best from a mitigations perspective
  - Enable “Enhanced Protected Mode” for IE 10

- Software vendors
  - Build your applications with Visual Studio 2012 [31]
  - Enable new opt-in mitigations

- Driver writers
  - Port your drivers to use NonPagedPoolNx
Questions?

Contact us at switech@microsoft.com

Were you fascinated by the topics discussed in this presentation?

We are hiring.

References

   http://illmatics.com/Understanding_the_LFH_Slides.pdf
References


References


