Windows 10 Mitigation Improvements

David Weston, Windows Offensive Security Research (OSR)
Matt Miller, Microsoft Security Response Center (MSRC)

August, 2016

This presentation is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.
Agenda

- Microsoft’s approach to data-driven software defense
- How we’ve adapted red teams to accelerate learnings
- Latest & greatest mitigation features in Windows 10
Problem: Augmenting Preventative Security

Preventative Security

- Preventative Strategy – SDL “Find all the bugs” before shipping
- Static Security Boundaries
- Focus on component level security – customer assets, configuration, 3rd party software largely out-of-scope
- Investigation of exploit and other attack techniques out-of-scope
- Engineering driven - Focus on abstraction to support scale and process
- Mitigation design and offensive security research is ad-hoc and specialized

Attackers are agile, adaptive, and results focused – effective techniques often don’t map to security boundaries

Market value for exploits is 10x vulnerabilities – preventative security is focused on lowest value asset

Attackers invest in developing tool sets and libraries – no proactive disruption, reactive response only after attacks

The cost for attackers is unknown – The current approach to security is abstracted from attacks – security effectiveness against real attacks unknown

“Assume breach” mitigation strategy augments preventative security
**Strategy:** Data-Driven Software Defense

**Analyze**
Analyze comprehensive set of real world data
Identify opportunities for tactical attack disruption and future strategic hardening

**Build**
Security engineers explore mitigation concepts with product owners
Security engineers prototype or productize mitigation design

**Evaluate**
Windows Offensive Security Research Team (OSR) evaluates mitigations and attempts to identify bypasses
Mitigation flaws are addressed
Analysis: High-level vulnerability & exploit trends

- **# of Microsoft RCE/EOP CVEs by patch year**
- **% of Microsoft RCE & EOP CVEs exploited within 30 days of patch**

**Analysis:**
High-level vulnerability & exploit trends

Vulnerabilities are increasing while evidence of actual exploits is decreasing due to mitigation investments.
Analysis: Flash Exploit Technique Trends

Exploit technique trend analysis drives new or improved mitigations

Prevalence of linear heap corruption drove adoption of Heap randomization for Flash on Windows

Worked with Adobe to mitigation via Vector Length and ByteArray Mitigations

Exploit technique trend analysis drives new or improved mitigations
• A focus on mitigations for disruption of invariant techniques used in exploits (ROP, Heap Spraying, UAF)
• In 2015 only 6 days with a known zero day Internet Explorer RCE exploit in-the-wild (down from 45, 135)
• Vulnerability volume has increased but number of zero day exploits has decreased

Mitigations were a key factor in zero day reduction trends
Microsoft Services Security: **Assume Breach**

- Azure, O365 and other have evolved the “assume breach” security methodology
- Focus on what happens after a “security boundary” is assumed breached
- **Detection, Containment, Response, and Recover**
- Services “**Redteams**” used to model breaches, evaluate detection, and simulate response process

**Prevent Breach**
- Threat model
- Code review
- Security testing
- Security development lifecycle (SDL)

**Assume Breach**
- War game exercises
- Central security monitors
- Live site penetration test

**Hypothesis:** We can use the assume breach approach to model exploitation and drive mitigations
Challenges with Adapting “Assume Breach”

- Mitigations are primarily reactive if driven by ITW data
- Reactive mitigations take time to develop
- New products do not get appropriate mitigations – until they are attacked
- We do not get accurate metrics on exploit and attack development

How do we design effective mitigations proactively?

PWN2OWN & Data Influenced Mitigations

| CFG: suppress sensitive APIs |
| Flash: Eliminate RWX ATL thunks |
| Junctions: Prevent sandbox processes from creating NTFS junctions – TH1 |
| Fonts: Move font parsing to user mode sandbox |
| Edge: Prevent content processes from creating child processes |
| Edge: Enable win32k system call restrictions |
## REDTEAM: Windows and Devices

### Model real-world attacks
- Model attacks based on ecosystem analysis and threat intelligence
- Evaluate the customer-promises from an attack perspective
- Provide data sets of detection-and-response
- Attack the full stack in production configuration (software, configuration, hardware, OEMs)

### Identify security gaps
- Measure Time-to-Compromise (MTTC) / Pwnage (MTTP)
- Identify invariant techniques for mitigation
- Simulate a real-world incident response before it occurs (process, owners, messaging)
- Provide detection guidance for Defenders

### Demonstrate impact
- Break-it-you-bought-it work with teams to address issues
- Design mitigations to drive up MTTC/MTTP metrics
- Enumerate business and legal risk
- Show business value, priorities, and investments needs with demonstrable attacks

Assume Breach: An Inside Look at Cloud Service Provider Security - Russinovich
REDTEAM Offensive Modeling Outcomes

- The Offensive Security Research team (OSR) operating for over a year in the Windows and Devices group
- Focused on end-to-end exploitation of common software/hardware scenarios – most without a prior known public attack
- Proactively discovered many new exploit techniques – drove mitigations into new versions of Windows prior to public use of technique
- Exploit invariants shared with Defender/ATP (BLUETEAM) to drive detection in-the-wild
- Demonstrated fundamental new memory corruption techniques and designed new general RCE mitigation for future version of Windows

Offensive security modeling is now a core pillar of Microsoft security strategy
Mitigation Improvements in Windows 10
Layered, data-driven software defense in Windows 10

Our Strategy

Make it difficult & costly to find, exploit, and leverage software vulnerabilities

Our Tactics

Eliminate entire classes of vulnerabilities
Break exploitation techniques
Contain damage & prevent persistence
Limit the window of opportunity to exploit
Acknowledgements

Many teams and individuals worked very hard on what we are about to talk about

| Internet Explorer, Edge, & Chakra | Dave Buchthal, Shubham Chopra, Crispin Cowan, Bo Cupp, Mike Decker, Jim Fox, Matt Gradwohl, John Hazen, C.J. Hebert, Forbes Higman, Michael Howell, Sermet Iskin, Rick James, Riff Jiang, Venkat Kudallur, Louis Lafreniere, Curtis Man, Ed Maurer, Bruce Morgan, Kamen Moutafov, Zach Murphy, Vidya Nallathimmayyagari, Justin Rogers, Todd Sahl, Saranya Kalpathy Seshadri, Bob Schroder, Kirk Sykora, Jason Weber |
| SmartScreen | Costas Boulis, Ryan Colvin, Jeb Haber, Jeff McKune, Anthony Penta |
| Visual Studio | Natalia Glagoleva, Shayne Hiet-Block, Jim Hogg, Jim Radigan, Asmaa Taha, YongKang Zhu |
| Microsoft Research (MSR) | Richard Black, Miguel Castro, Manuel Costa, Austin Donnelly |

We would also like to thank our Mitigation Bypass & Defense Bounty participants for helping us improve our defenses!

Our sincere apologies to anyone who we have unintentionally failed to list – so many people have contributed!
Eliminating classes of vulnerabilities

We move beyond the “hand-to-hand combat” of finding and fixing individual issues by identifying ways to eliminate entire classes of vulnerabilities.

**Goal:** Increase attacker cost of finding exploitable vulnerabilities
We closely study vulnerability root cause trends

Microsoft security engineers categorize the root cause of every vulnerability and look for patterns

**Stack corruption** issues have been essentially eliminated

**Use after free** issues rose dramatically in 2013 & 2014 but have since decreased

**Heap out-of-bounds read, type confusion**, and **DLL planting** have increased

Root causes of Windows, Internet Explorer, and Edge Remote Code Execution (RCE) CVEs by patch year
Memory Garbage Collection (MemGC)

The vast majority of the use after free issues we observed were in our DOM engine in Internet Explorer.

Without MemGC

// 1. Allocate object
p = new COptionElement();

// 2. Free object
delete p;

// 3. Use freed object
p->Foo();

Attacker reallocates p as a new type

// 2. Zero object, but don’t free
ZeroMemory(p, sizeof(T));

// 3. Garbage collection phase
frees all objects with no references (stack, registers, heap)

With MemGC

MemGC is a conservative garbage collector (GC) for our DOM engine that makes DOM use after free issues non-exploitable.

Tactic | Applies to | First shipped
--- | --- | ---
Eliminate entire classes of vulnerabilities | Edge on Windows 10 and backported to IE9+ on Windows Vista+ | July, 2015 (Windows 10 RTM)
Type confusion protection

In late 2014, we began investigating ways to eliminate type confusion issues as these were being increasingly reported.

We found that ~50% were due to bad casts

### IE Type Confusion Cases
- 6 cases (21%)
- 3 cases (10%)
- 14 cases (48%)

Categories of type confusion issues observed from 2012 through 2014

Of which ~50% appeared to be recurring patterns

### Bad Casts
- 1 case (1 case)
- 1 case (2 cases)
- 1 case (2 cases)
- 1 case (36%)

Classes that were involved in type confusions resulting from bad casts

We introduced additional checks to eliminate ~50% of type confusions (recurring bad casts and branching on wrong CTreePos)

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Applies to</th>
<th>First shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminate entire classes of vulnerabilities</td>
<td>Edge and IE11 on Windows 10 and backported to IE10+ on Windows 7+</td>
<td>July, 2015 (Windows 10 RTM)</td>
</tr>
</tbody>
</table>
Edge attack surface reduction

With the Edge browser, we also seized the opportunity to drastically reduce the attack surface exposed to the web.

✓ No legacy document modes
✓ No legacy script engines (VBScript, JScript)
✓ No Vector Markup Language (VML)
✓ No Toolbars
✓ No Browser Helper Objects (BHOs)
✓ No ActiveX controls

Tons of code was removed as a result!

In the past year
Edge had 56% fewer RCE CVEs compared to Internet Explorer
Internet Explorer RCE CVEs decreased 40% H/H

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Applies to</th>
<th>First shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eliminate entire classes of vulnerabilities</td>
<td>Edge on Windows 10</td>
<td>July, 2015 (Windows 10 RTM)</td>
</tr>
</tbody>
</table>
Breaking exploitation techniques

We assume that we won’t be able to eliminate all vulnerabilities, so we look for ways to break the techniques that attackers can use to exploit them.

Goal: Increase attacker cost of developing a reliable exploit for a vulnerability
Exploiting vulnerabilities has become increasingly difficult

**Exploitation used to be simple**

*Circa 2003; exploit steps for CVE-2003-0344*

- Trigger stack buffer overrun
- Overwrite return address with predictable address of a "JMP ESP"
- Execute shellcode from the stack
- Arbitrary native code execution

**Now, it is much more involved**

- Place array length at a predictable location (via heap spray/massage)
- Modify array length via memory corruption, enabling arbitrary read/write
- Use arbitrary read/write to discover DLL base address
- Construct ROP payload by searching for code sequences in the DLL
- Corrupt C++ virtual table pointer and trigger virtual method call to first gadget
- Execute ROP payload (typically to make shellcode executable)
- Execute arbitrary native code
- Escape the sandbox (or operate inside it)

---

**Windows Vista enables ASLR**

- Kills (most) predictable images

**Internet Explorer 8 enables DEP**

- Kills heap spraying of code

**Windows 8 adds Force ASLR; IE10 enables it**

- Kills all predictable images

---


**Code heap spraying era** → **Non-ASLR DLL era** → **Arbitrary read/write era**
User mode exploit mitigations
64-bit browsing by default

Heap spraying has been a standard technique used by nearly every browser exploit

Heap spraying example from Metasploit [1]

```javascript
var memory = new Array();
function sprayHeap(shellcode, heapSprayAddr, heapBlockSize) {
    var index;
    var heapSprayAddr_hi = (heapSprayAddr >> 16).toString(16);
    var heapSprayAddr_lo = (heapSprayAddr & 0xffff).toString(16);
    while (heapSprayAddr_hi.length < 4) { heapSprayAddr_hi = "0" + heapSprayAddr_hi; }
    while (heapSprayAddr_lo.length < 4) { heapSprayAddr_lo = "0" + heapSprayAddr_lo; }
    var retSlide = unescape("%u" + heapSprayAddr_hi + "%u" + heapSprayAddr_lo);
    while (retSlide.length < heapBlockSize) { retSlide += retSlide; }
    retSlide = retSlide.substring(0, heapBlockSize - shellcode.length);
    var heapBlockCnt = (heapSprayAddr - heapBlockSize) / heapBlockSize;
    for (index = 0; index < heapBlockCnt; index++) {
        memory[index] = retSlide + shellcode;
    }
}
```


Majority of Windows 10 devices are running a 64-bit version of Windows & Edge

Attackers must have an additional information disclosure

Tactic | Applies to | First shipped
--- | --- | ---
Breaking exploitation techniques | Edge on Windows 10 | July, 2015 (Windows 10 RTM)
Control Flow Guard

Modern exploits typically rely on hijacking control-flow through an indirect call

Example control-flow hijack via indirect call to a ROP gadget[1]

```c
/* Corrupt sound object vtable ptr */
while (1) {
    if (this.s[index][i] == 0xdeadbeef && this.s[index][j+0x99] == 0x1234) {
        soundobjref = this.s[index][j+0x99];
        dec = soundobjref-cvasdr-1;
        this.s[index][dec/+4] = cvasdr=2*4+4*4;
        break;
    }
    j++;
}
/* Run Payload */
this.sound.toString();
```

With CFG in place, traditional ROP gadgets and other invalid functions cannot be called indirectly

Compile time

Runtime

void Foo(...) {
    // SomeFunc is address-taken
    // and may be called indirectly
    Object->FuncPtr = SomeFunc;
}

Metadata is automatically added to the image which identifies functions that may be called indirectly

void Bar(...) {
    // Compiler-inserted check to
    // verify call target is valid
    _guard_check_icall(Object->FuncPtr);
    Object->FuncPtr(xyz);
}

A lightweight check is inserted prior to indirect calls which will verify that the call target is valid at runtime

Image Load

• Update valid call target data with metadata from PE image

Process Start

• Map valid call target data

Indirect Call

• Perform O(1) validity check
  • Terminate process if invalid target

[1] https://github.com/rapid7/metasploit-framework/blob/abd76c50000e75bcac0616b606cd8568e1df5927f/externalsource/exploits/CVE-2014-0322/AxKploit.as

Tactic | Applies to | First shipped
--- | --- | ---
Breaking exploitation techniques | Edge on Windows 10 and IE11 on Windows 8.1+ | November, 2014 (Windows 8.1 Update 3)
# Control Flow Guard Bypasses & Enhancements

Like all mitigations, CFG has by design limitations that place constraints on its overall effectiveness:
- ✓ Return addresses are not protected
- ✓ Valid functions can be called out of context
- ✓ “Fail-open” design for compatibility

Since shipping CFG, researchers have identified bypasses and additional enhancements have been made:

<table>
<thead>
<tr>
<th>Bypass</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-enlightened Just-in-Time (JIT) compilers can be abused</td>
<td>Mitigated in latest version of Edge on Windows 10 (Chakra, Adobe Flash, and WARP)</td>
</tr>
<tr>
<td>Multiple non-instrumented indirect calls reported to our Mitigation Bypass Bounty</td>
<td>Mitigated in latest version of Edge on Windows 10</td>
</tr>
<tr>
<td>Calling sensitive APIs out of context</td>
<td>NtContinue/longjmp – mitigated for all CFG enabled apps on Windows 10</td>
</tr>
<tr>
<td></td>
<td>VirtualProtect/VirtualAlloc – mitigated in latest version of Edge on Windows 10</td>
</tr>
<tr>
<td></td>
<td>LoadLibrary – mitigated in latest version of Edge on Windows 10 via code integrity</td>
</tr>
<tr>
<td></td>
<td>WinExec – mitigated in Edge on Windows 10 anniversary edition via child process policy</td>
</tr>
<tr>
<td>Corrupting return addresses on the stack</td>
<td>Known limitation that we intend to address with new technology (e.g. with Intel CET)</td>
</tr>
</tbody>
</table>

We are continuing to explore ways to improve CFG to more strongly prevent control-flow hijacking.
Code integrity & image load restrictions

Exploits can attempt to inject and run arbitrary code by causing a malicious DLL to be loaded.

Windows 10 allows processes to enable code integrity and image load restrictions to prevent malicious DLLs from being loaded.

**“LoadLibrary” via JavaScript**

1. Download a DLL by XMLHttpRequest object, the file will be temporarily saved in the cache directory of IE;
2. Use “Scripting.FileSystemObject” to search the cache directory to find that DLL;
3. Use “Scripting.FileSystemObject” to create a directory named “System32”, copy the DLL into that directory, and named it as “shell32.dll”;
4. Modify the “SystemRoot” environment variable of current process via “WScript.Shell” object to the upper directory of the “System32” directory created just now;
5. Create “Shell.Application” object, trigger to loading “%SystemRoot%\System32\shell32.dll”.

Example of such an attack provided by Yang Yu @ Black Hat USA 2014

**DLL loading restrictions supported by Windows 10**

- Only properly signed images can be loaded (Microsoft, WHQL, Store, or DRM signed)
- Binaries on remote devices (UNC/WebDAV) cannot be loaded

An additional benefit: these restrictions help prevent unwanted DLLs from being injected into processes that enable them

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Applies to</th>
<th>First shipped</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breaking exploitation techniques</td>
<td>Edge on Windows 10 and opt-in for other apps</td>
<td>November, 2015 (Windows 10 1511 update)</td>
</tr>
</tbody>
</table>
Dynamic code restrictions

Nearly all exploits rely on creating new executable code pages that contain their shellcode

Windows 10 allows processes to enable dynamic code generation restrictions which imposes W^X invariants

Code is immutable

Code pages cannot become writable via VirtualProtect or initially allocated as WX

Data cannot become code

Data pages cannot become executable via VirtualProtect or initially allocated as WX

Combined with image load restrictions, this prevents all forms of unsigned arbitrary code injection within a process

Tactic | Applies to | First shipped
--- | --- | ---
Breaking exploitation techniques | Opt-in by process on Windows 8.1+ | August, 2013 (Windows 8.1 RTM)
Kernel mode exploit mitigations
Windows 10 Virtualization-Based Security (VBS)

Hyper-V on Windows 10 enables a suite of robust protection features for the host and guest kernels.

Virtual Secure Mode (VSM) architecture

Mitigations enabled by Hyper-V & VSM

- **Hyper Guard**
  - Prevents modification of key MSRs, control registers, and descriptor table registers
  - Example: SMEP cannot be disabled

- **Hypervisor-Enforced Code Integrity (HVIC)**
  - Only properly signed kernel pages can become executable

- Robust even if an attacker can perform arbitrary read/write in VTL0 kernel

---

Tactic | Applies to | First shipped
--- | --- | ---
Breaking exploitation techniques | Windows 10 with Hyper-V enabled | July, 2015 (Windows 10 RTM)

### Windows Kernel 64-bit ASLR Improvements

**Predictable kernel address space layout has made it easier to exploit certain types of kernel vulnerabilities**

#### 64-bit kernel address space layout is now dynamic

<table>
<thead>
<tr>
<th>Linear Address</th>
<th>PML4</th>
<th>Directory ptr</th>
<th>Directory</th>
<th>Table</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

##### System region PML4 entries are randomized
- Non-paged pool
- Paged pool
- System cache
- PFN database
- Page tables
- ... and so on

#### Various address space disclosures have been fixed

- Page table self-map and PFN database are randomized
  - Dynamic value relocation fixups are used to preserve constant address references

- SIDT/SGDT kernel address disclosure is prevented when Hyper-V is enabled
  - Hypervisor traps these instructions and hides the true descriptor base from CPL>0

- GDI shared handle table no longer discloses kernel addresses

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Applies to</th>
<th>First shipped</th>
</tr>
</thead>
</table>
## Enabling opt-in mitigations

<table>
<thead>
<tr>
<th>Mitigation</th>
<th>How to opt-in</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Flow Guard</td>
<td>Compile and link with /guard:cf (requires Visual Studio 2015 Update 2+)</td>
</tr>
<tr>
<td>Image load restrictions</td>
<td>• SetProcessMitigationPolicy with ProcessImageLoadPolicy</td>
</tr>
<tr>
<td></td>
<td>• UpdateProcThreadAttribute with PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY</td>
</tr>
<tr>
<td>Code integrity restrictions</td>
<td>• SetProcessMitigationPolicy with ProcessSignaturePolicy</td>
</tr>
<tr>
<td></td>
<td>• UpdateProcThreadAttribute with PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY</td>
</tr>
<tr>
<td>Dynamic code restrictions</td>
<td>• SetProcessMitigationPolicy with ProcessDynamicCodePolicy</td>
</tr>
<tr>
<td></td>
<td>• UpdateProcThreadAttribute with PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY</td>
</tr>
<tr>
<td>Child process restrictions</td>
<td>• UpdateProcThreadAttribute with PROC_THREAD_ATTRIBUTE_CHILD_PROCESS_POLICY</td>
</tr>
<tr>
<td>HVCI</td>
<td>• Set HypervisorEnforcedCodeIntegrity (REG_DWORD) to 1 in</td>
</tr>
<tr>
<td></td>
<td>HKLM\SYSTEM\CurrentControlSet\Control\DeviceGuard</td>
</tr>
</tbody>
</table>

For more information: [https://aka.ms/setprocessmitigationpolicy](https://aka.ms/setprocessmitigationpolicy)
Containing damage & preventing persistence

We assume that we won’t be able to prevent exploitation in all cases, so we look for ways to effectively isolate and contain the damage if a vulnerability is successfully exploited.

**Goal:** Increase attacker cost of effectively leveraging an exploit for a vulnerability
AppContainer (AC)

AppContainer provides strong sandboxing and isolation for applications on Windows

- Store apps all run within an AC
- Font parsing is now done in user mode within an AC
- Edge uses a multi-AC design for isolation
- Adobe Flash has now been moved to its own AC
- Win32k system call filtering is enabled for Edge

AppContainer Properties:
- Security boundary: Microsoft will address vulnerabilities that can violate AC security boundary
- Capability-based resource access: Network, file, registry, and device access are restricted (both read and write)
- Locked down process: No symbolic links, reduced attack surface, and various mitigations on by default

Tactic | Applies to | First shipped
--- | --- | ---
Containing damage & preventing persistence | Multiple applications | August, 2012 (Windows 8)
Limiting the window of opportunity to exploit

Assuming all else fails, we look to have effective tools and processes to limit the scope and window of opportunity for attackers to leverage an exploit for a vulnerability

**Goal:** Minimize an attacker’s return on investment from the use of an exploit for a vulnerability
Reducing the attacker’s window of opportunity

**Rapidly Respond**
- Mobilize engineering teams to quickly understand and develop a fix for a vulnerability

**Rapidly Protect**
- Use SmartScreen and other technologies to protect customers from in-the-wild attacks
- Enable Microsoft Active Protection Program (MAPP) partners to protect the broader ecosystem

**Rapidly Update**
- Broadly deploy and install security updates to quickly minimize the affected population size
Conclusion
Hardware assists for helping to mitigate control-flow hijacking & ROP

Control-flow Enforcement Technology (CET)

- Indirect branch tracking via ENDBRANCH
- Return address protection via shadow stack

Measuring the impact of our strategy so far

We’ve made measurable progress on improving customer safety

- The number of Microsoft vulnerabilities exploited within 30 days of a patch has continued to decline Y/Y despite increases in the number of vulnerabilities being addressed each year.

- In the last two years, no zero day exploits for Microsoft RCE vulnerabilities have been found in-the-wild that work against Internet Explorer 11 on Windows 8.1+

- Since releasing Edge one year ago, there have been no zero day exploits found in-the-wild targeting Edge.

Windows 10 and Edge are always up-to-date and offer strong defenses against modern threats.

Our data-driven and red team assisted approach strongly positions us to identify & deliver impactful mitigations.
Microsoft Edge RCE on WIP Bounty Program

- Submit a **remote code execution (RCE) vulnerability for Microsoft Edge**
- The bugs reported should be on the **most recent Windows Insider Preview slow build**
- Program duration is **August 4 2016 to May 15 2017**

Only eligible reported on the latest WIP slow builds will be paid a bounty
- This continues our effort in finding bugs in the earlier stages of development
- Microsoft will pay up to $1,500 for the first external report received on an internally known issue
- Microsoft will pay up to $15,000 on a previously unknown RCE on Edge

<table>
<thead>
<tr>
<th>Vulnerability type</th>
<th>Functioning Exploit</th>
<th>Proof of concept</th>
<th>Report Quality</th>
<th>Payout range (USD) *</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Code Execution in Microsoft Edge on recent builds of WIP slow</td>
<td>Required</td>
<td>Required</td>
<td>High</td>
<td>Up to $15,000</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Required</td>
<td>High</td>
<td>Up to $6,000</td>
</tr>
<tr>
<td></td>
<td>No</td>
<td>Required</td>
<td>Low</td>
<td>Up to $1,500</td>
</tr>
</tbody>
</table>