Microsoft

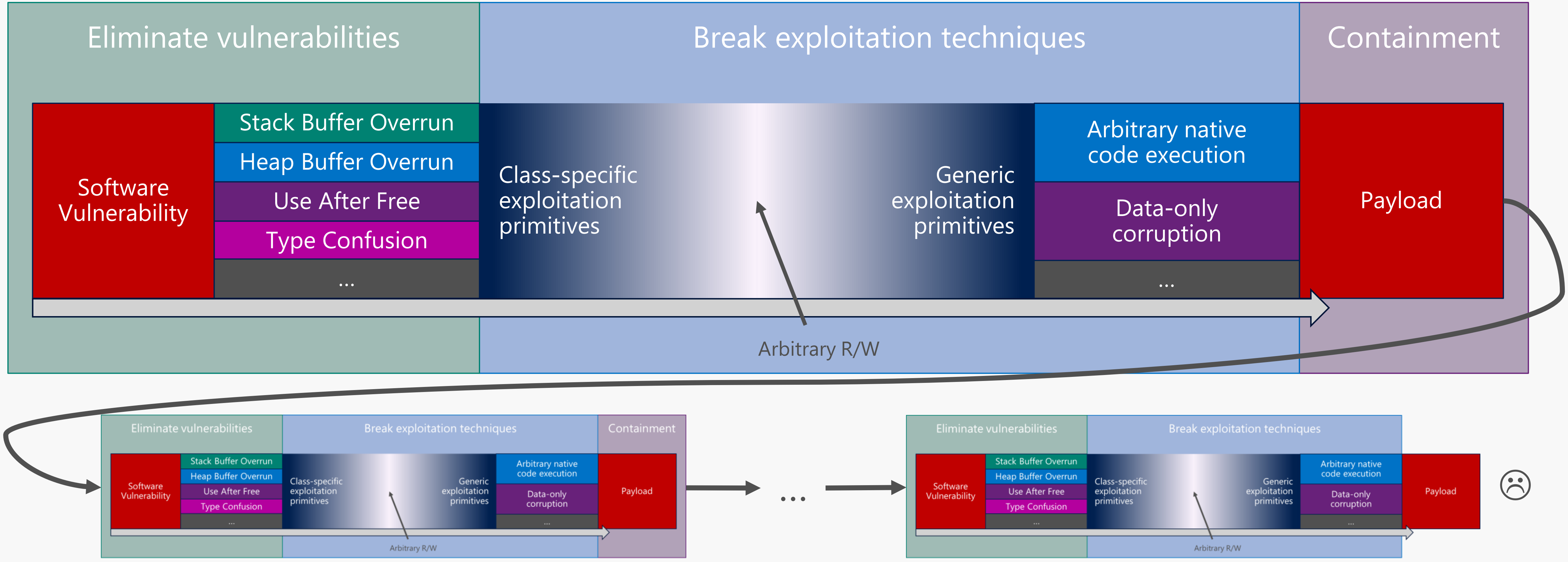
Trends and challenges in the vulnerability mitigation landscape

Matt Miller (@epakskape)
Microsoft Security Response Center (MSRC)

13th USENIX Workshop on Offensive Technologies (WOOT '19)
August 12th, 2019

Approaching the problem of vulnerability mitigation

Attackers transform software vulnerabilities into tools for delivering a payload to a target device



Attackers typically need to elevate privileges

At some point, we lose containment as a defense

This means applying the same defenses to privileged attack surfaces

This leaves eliminating vulnerabilities & breaking techniques

Microsoft's vulnerability mitigation strategy for the past 10+ years

Strategy	Make it difficult and costly to find, exploit, & leverage vulnerabilities	
Tactics	Eliminate vulnerabilities	NULL deref protection, Mem GC, attack surface reduction, …
	Break exploitation techniques	GS, ASLR, DEP, ACG, CIG, CFG, …
	Contain damage & prevent persistence	AppContainer & Virtualization
	Limit the window of time to exploit	Mature detection & response capabilities

We've been at this for a while – how has the vulnerability threat landscape evolved?

Microsoft vulnerability & exploitation trends

Statistician disclaimer: small numbers ahead, the word trends is used loosely ☺

Defining our scope

Vulnerabilities reported to Microsoft are typically addressed in one of two ways

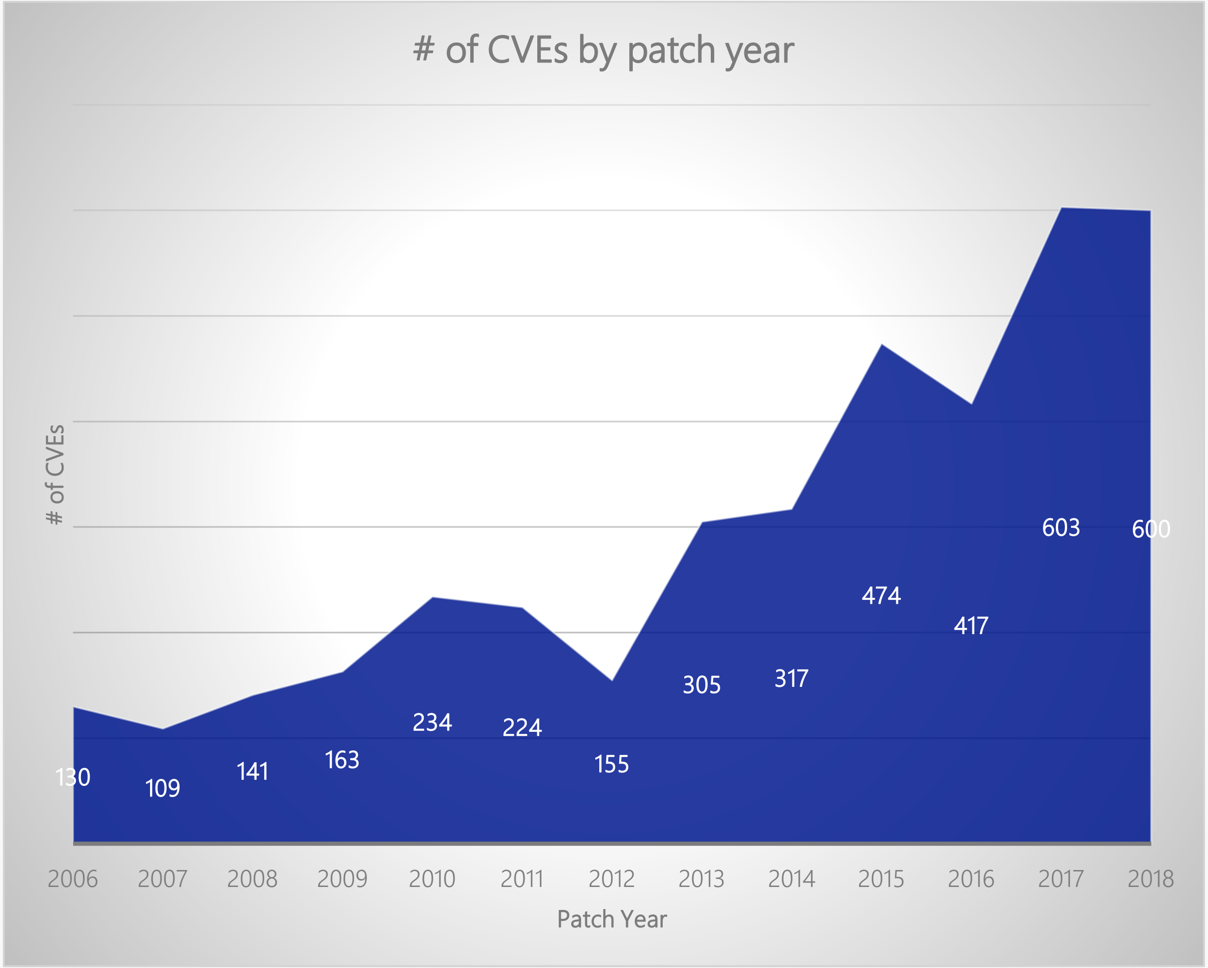
Vulnerability report to secure@microsoft.com → Open MSRC case → Magic Happens ☺ → Online Service Update / Endpoint Software Update

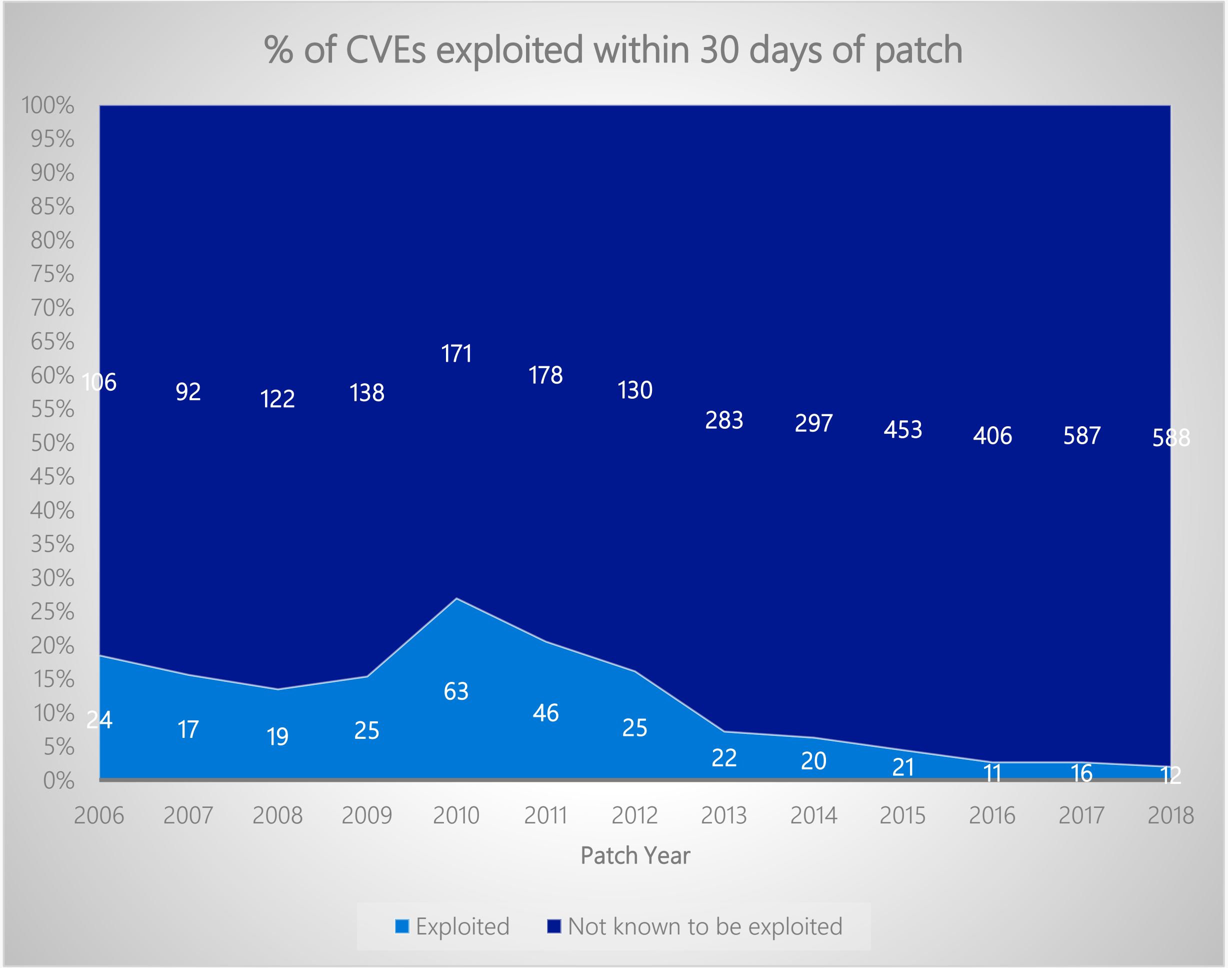
In 2018, ~54% of reported vulnerabilities were addressed via a software update

~85% of those vulnerabilities were Remote Code Execution (RCE), Elevation of Privilege (EOP), or Information Disclosure (ID)

Today we'll be focusing on Microsoft **RCE, EOP, and ID** vulnerabilities (CVEs) addressed via a **software update**

More vulnerabilities fixed, fewer known exploits

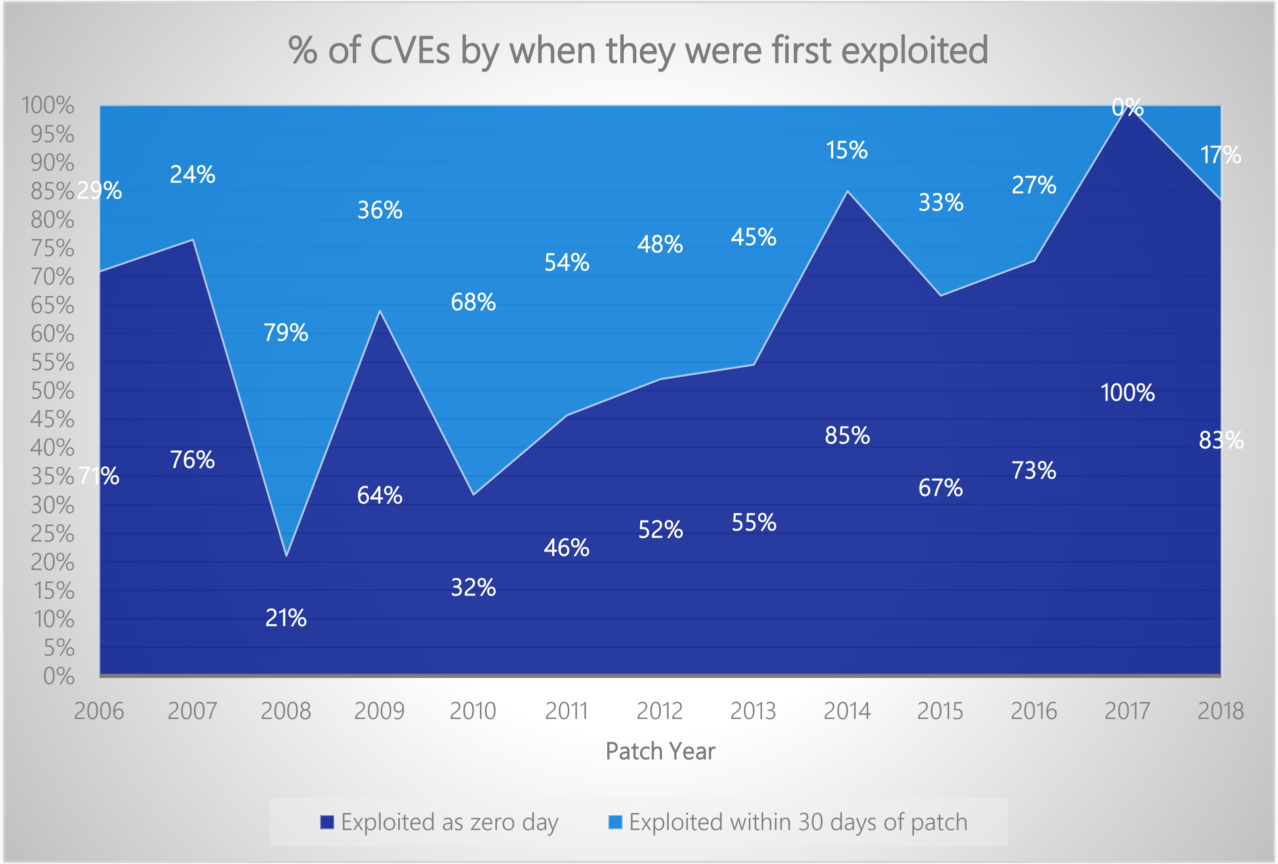


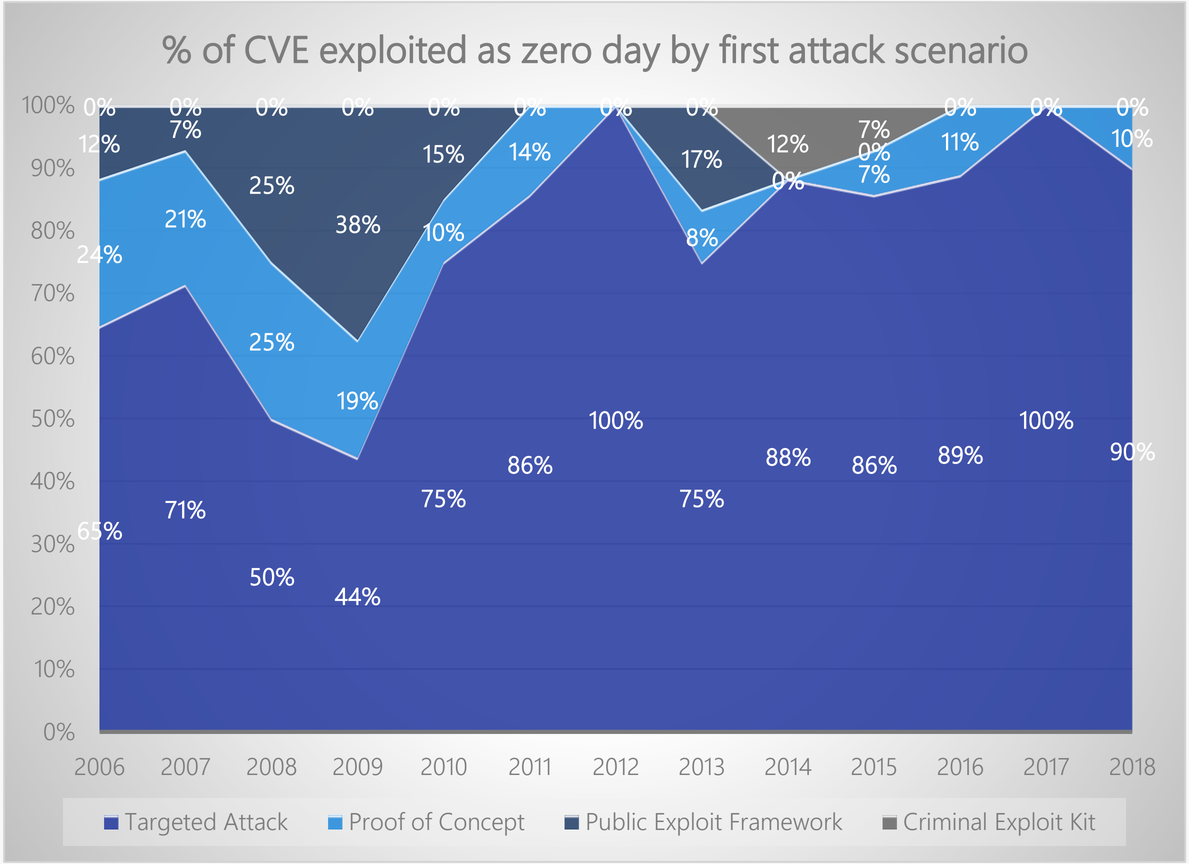


On the surface, risk appears to be *increasing*

But known actualized risk appears to be *decreasing*

When a vulnerability is exploited…





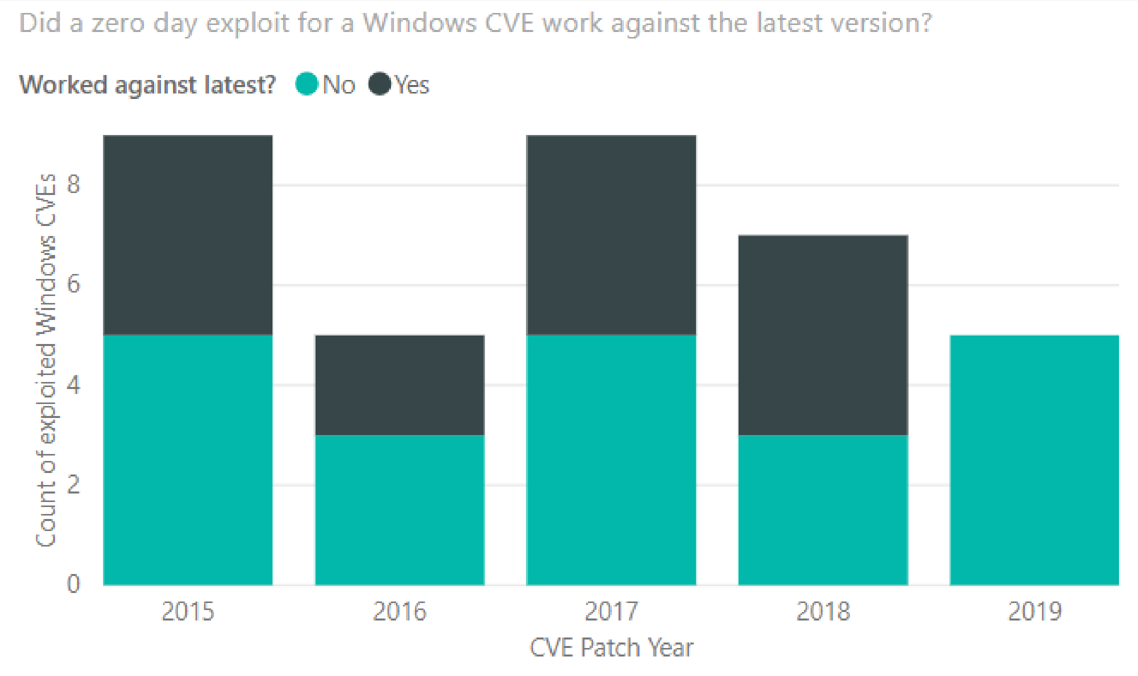
If a vulnerability is exploited, it is most likely going to be exploited as zero day

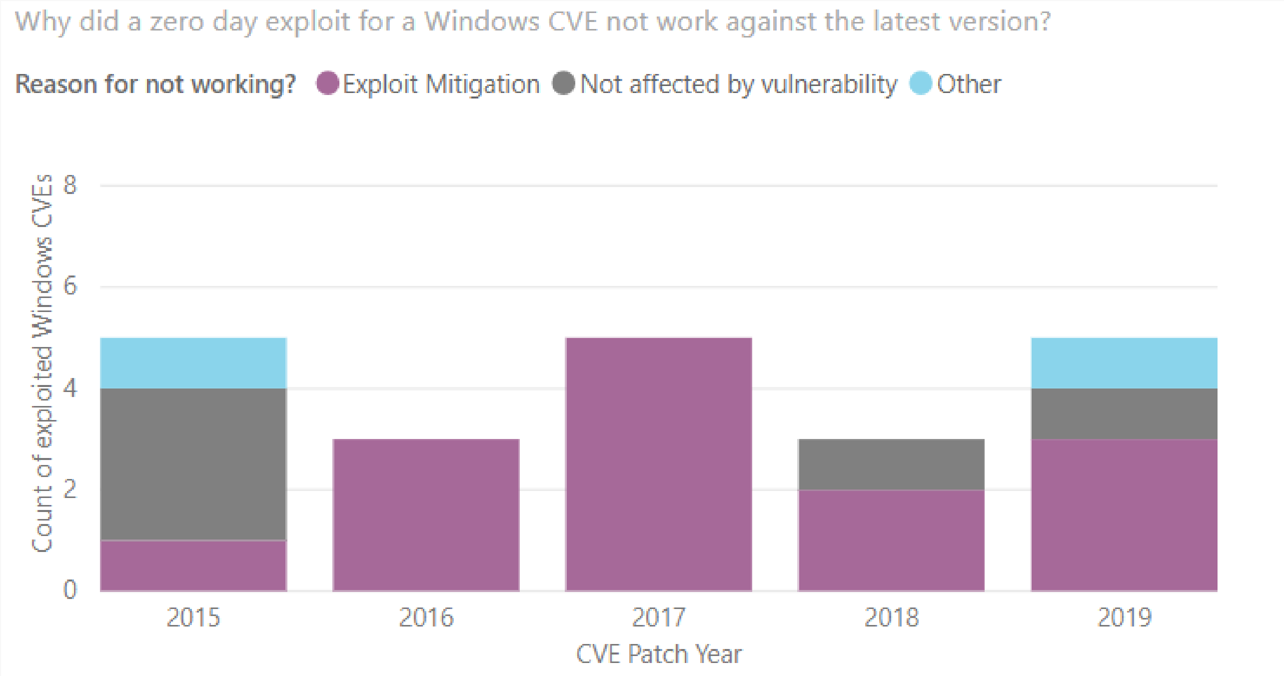
It is now uncommon to see a non-zero-day exploit released within 30 days of a patch being available

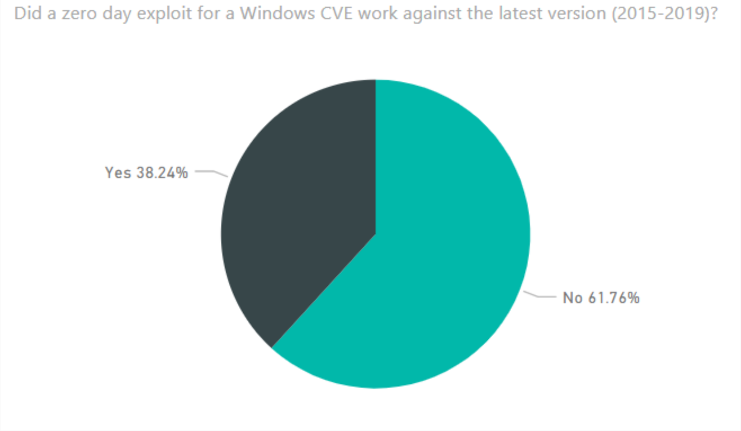
When a vulnerability is exploited as zero day, it is most likely to first be used in a targeted attack

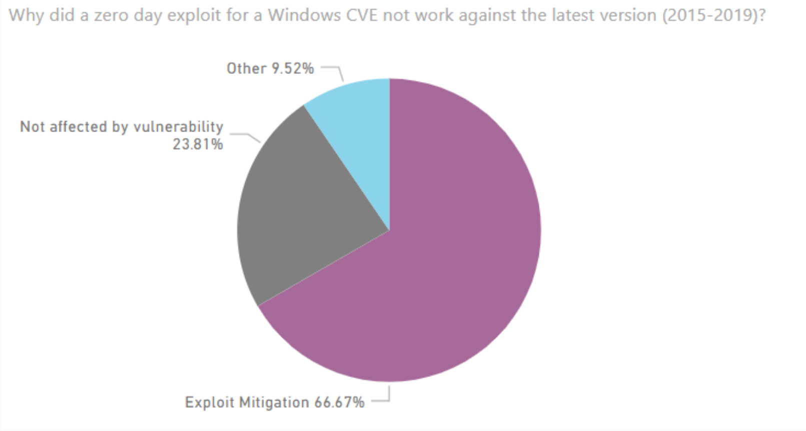
Older software versions are typically targeted by exploits

Older software versions are typically targeted





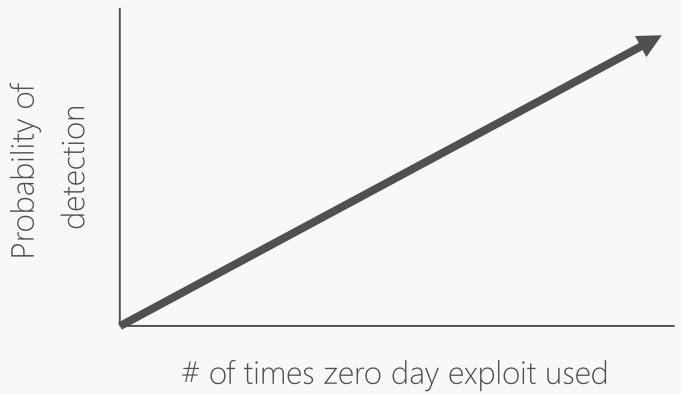




What about the zero day exploits we don’t know about?

It is challenging to effectively estimate the number of unknown zero day exploits

Hypothesis: increased exploit costs drive selective use



- ✓ **Probability of detection increases with zero day use**
  - Attackers are incentivized to minimize use
  - Targets that detect zero day may alert vendor
- ✓ **Selective use reduces downstream supply**
  - Many actors lack means and capability to acquire

Assertion: economics of the zero day market have shifted

- ✓ **Windows 10 is always up to date**
  - Poor ROI for exploiting patched vulnerabilities
  - Rapid evolution of defensive technologies
- ✓ **Mass-market exploit kits have struggled to maintain supply**
  - Decrease in reusable & public exploits
  - Cost to acquire exceeds expected ROI
- ✓ **Market shifted toward social engineering**
  - Macros, phishing, tech support scams, pw spraying, …

MAGNITUDE ACTOR ADDS A SOCIAL ENGINEERING SCHEME FOR WINDOWS 10

MARCH 08, 2017 Kafeine

https://www.proofpoint.com/us/threat-insight/post/magnitude-actor-social-engineering-scheme-windows-10

Widespread attacks are now the exception, not the norm

Server-side exploits

IIS, SQL, DCOM

Productivity app exploits

Office, Adobe Reader

Browser-based exploits

Internet Explorer, Adobe Flash, Java, ActiveX controls

Widespread attacks via exploits are now uncommon
- ✓ Pervasive sandboxing
- ✓ Strong mitigations
- ✓ Regular updates

Exceptions exist, e.g. WannaCry

2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018

Windows XP SP2 Ships

Firewall on-by-default & DEP

Windows Vista Ships

ASLR

Office 2010 Ships

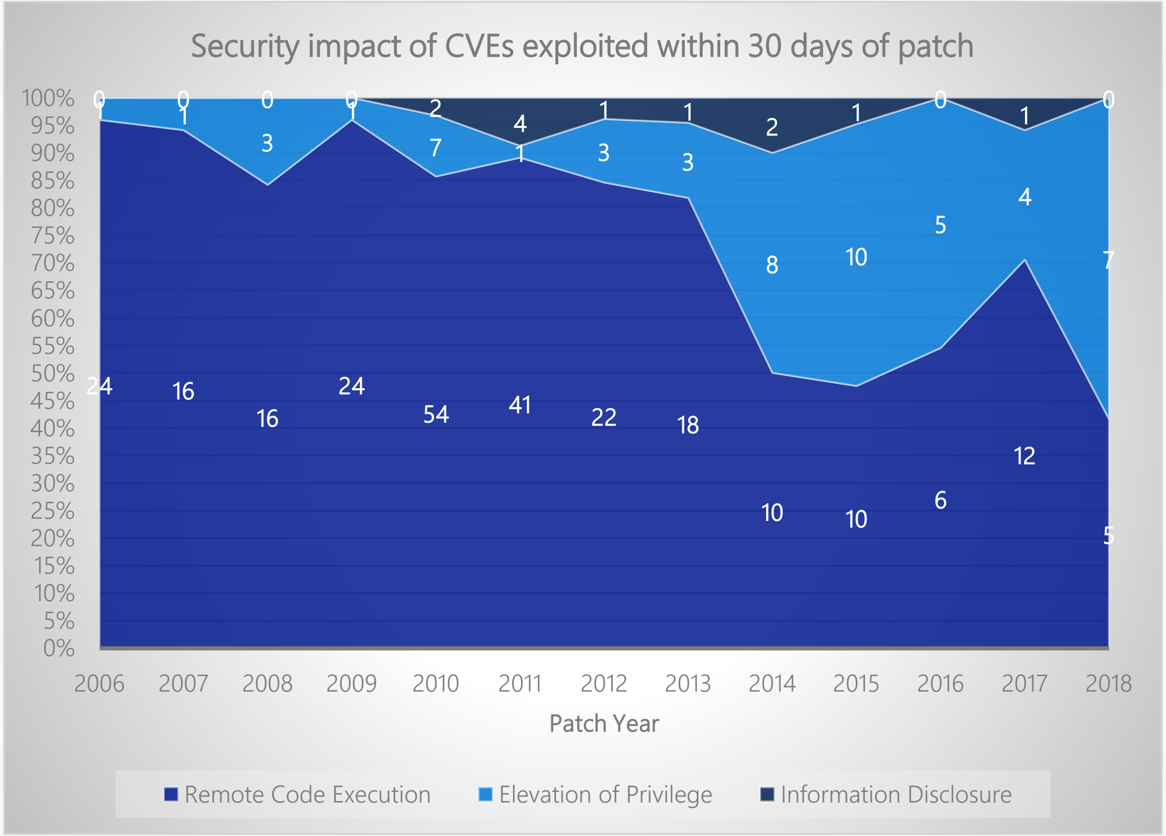
DEP, ASLR, & Protected View

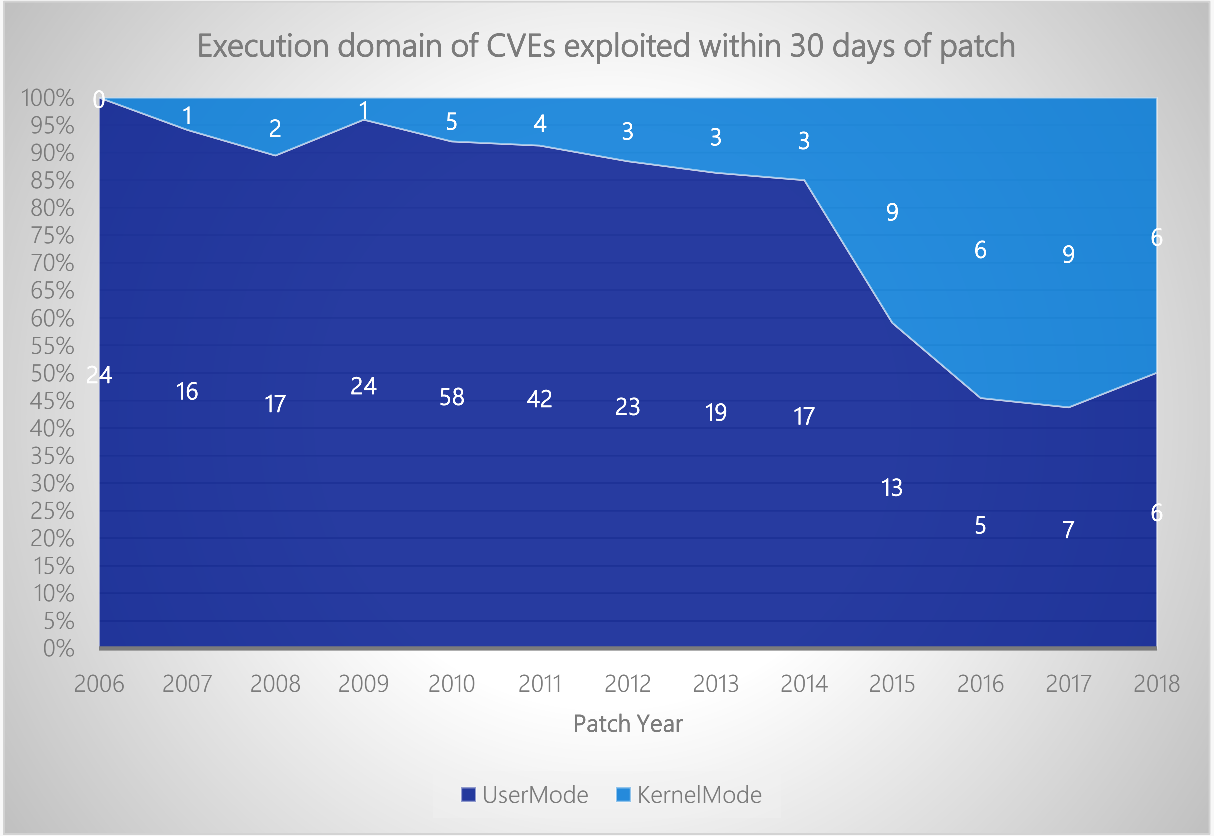
Edge ships

Sandbox, ASLR, DEP, CFG, etc

The echoes of pervasive sandboxing

The prevalence of sandboxes has increased the need for a sandbox escape

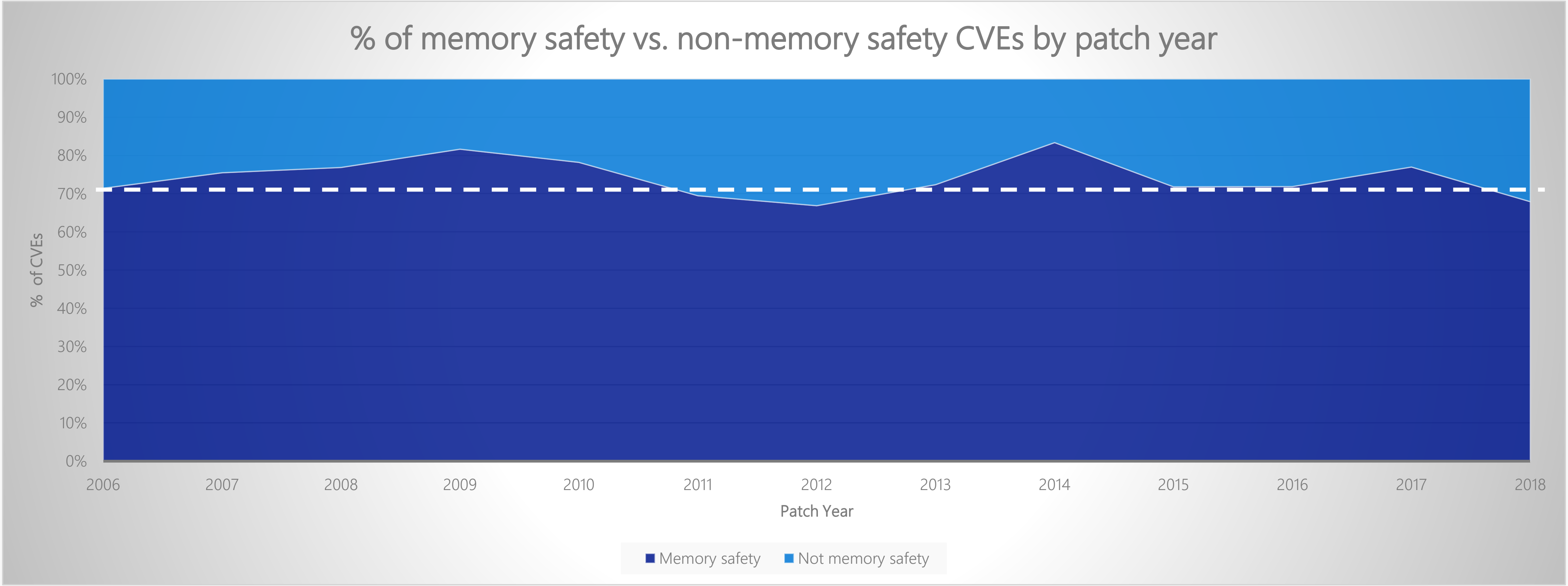




Since ~2014, we've seen an increase in EOP exploits in-the-wild, largely focused on kernel mode vulnerabilities

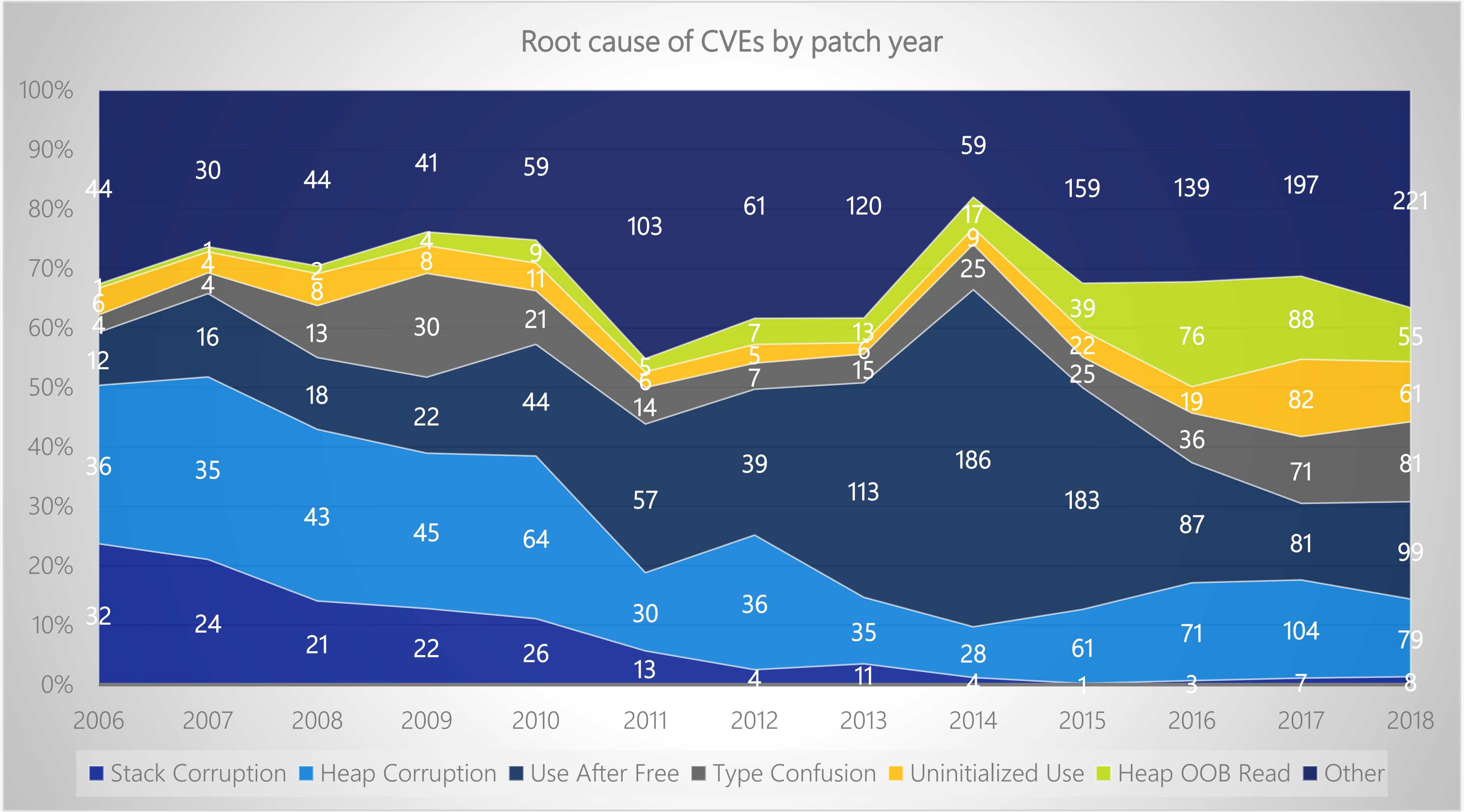
Memory safety issues remain dominant

We closely study the root cause trends of vulnerabilities & search for patterns



~70% of the vulnerabilities addressed through a security update each year continue to be memory safety issues

Drilling down into root causes



Stack corruptions are essentially dead

Use after free spiked in 2013-2015 due to web browser UAF, but was mitigated by Mem GC

Heap out-of-bounds read, type confusion, & uninitialized use have generally increased

Spatial safety remains the most common vulnerability category (heap out-of-bounds read/write)

Top root causes since 2016:

#1: heap out-of-bounds

#2: use after free

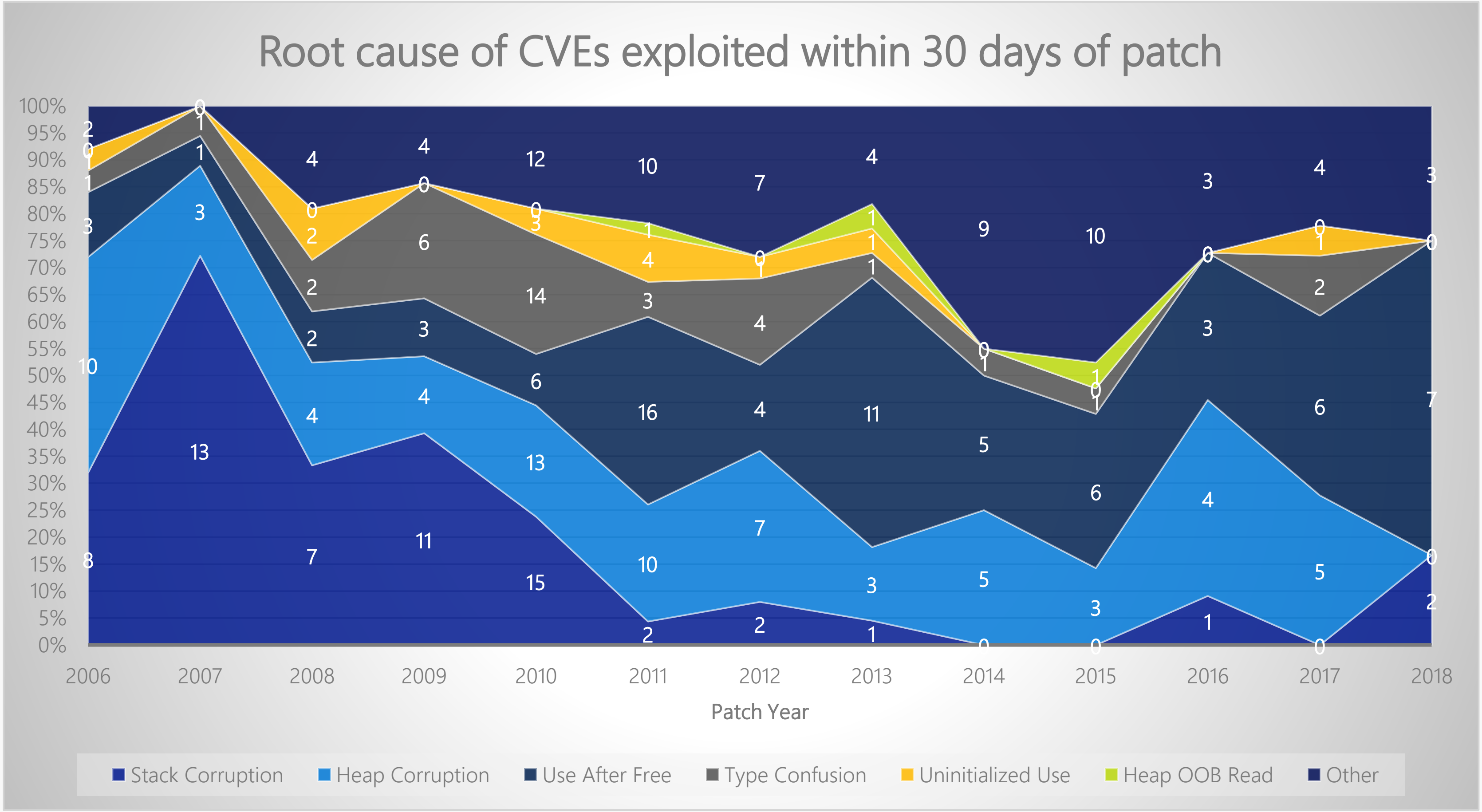
#3: type confusion

#4: uninitialized use

Note: CVEs may have multiple root causes, so they can be counted in multiple categories

Root causes of exploited vulnerabilities

The root cause of exploited vulnerabilities provide hints on attacker preference & ease of exploitability



Use after free and heap corruption continue to be preferably targeted

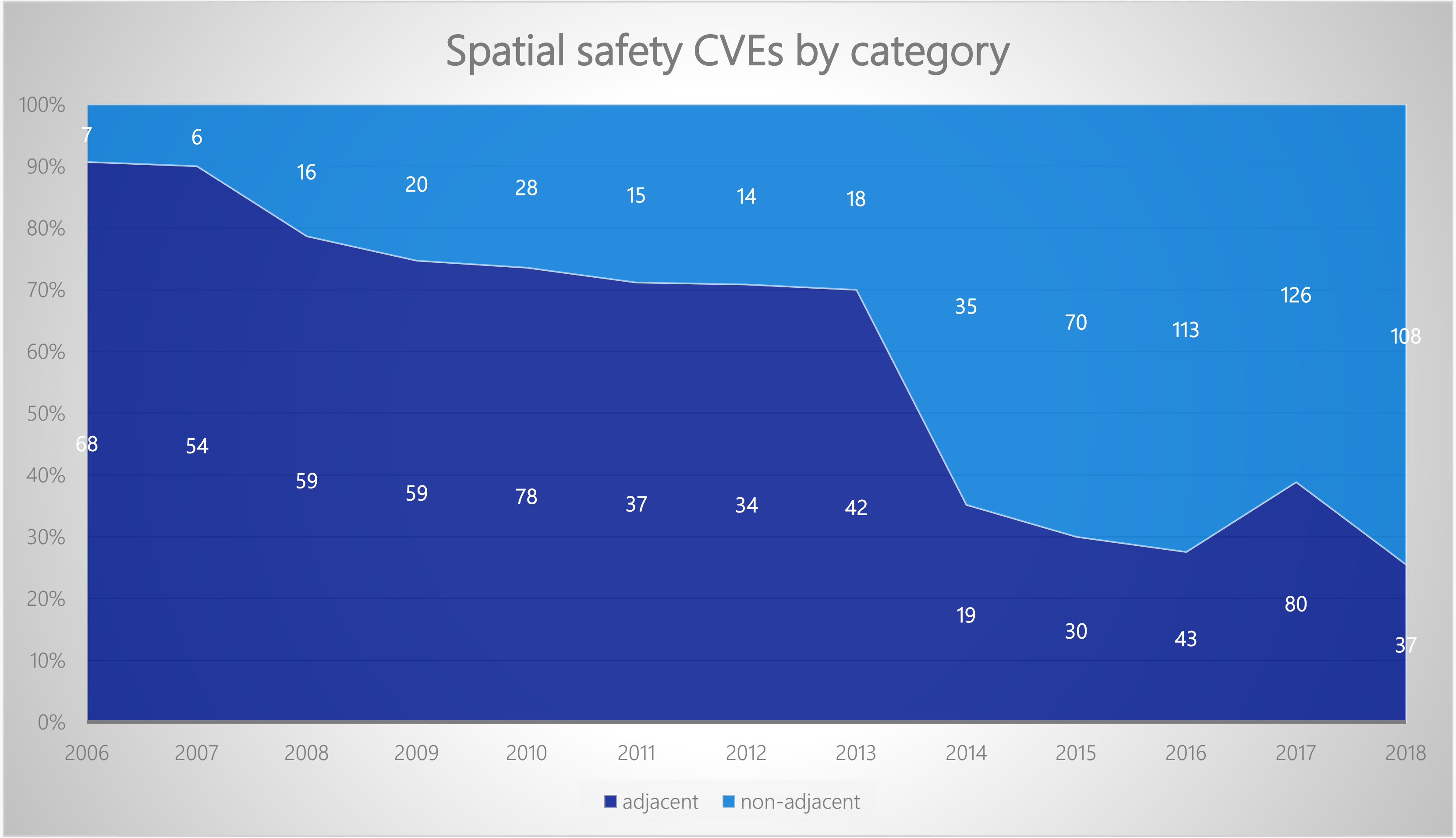
"Other" category consists of a few common types of issues:

- XSS & zone elevation issues
- DLL planting issues
- File canonicalization & symbolic link issues

Note: CVEs may have multiple root causes, so they can be counted in multiple categories

Exploring spatial safety vulnerabilities

Adjacent read/write (buffer overruns) have become less common than non-adjacent (out-of-bounds read/write)



Adjacent spatial safety violation

When the initial out-of-bounds access is always immediately adjacent to an allocation, e.g. displacement is not controllable.

```
memcpy(dst, src, n);
```

Non-adjacent spatial safety violation

When the initial-out-of-bounds access can be beyond the immediate bounds of an allocation, e.g. displacement is controllable.

```
dst[offset] = x; // offset is
                 // controlled
```


Note: CVEs may have multiple root causes, so they can be counted in multiple categories

Challenges with vulnerability mitigation today

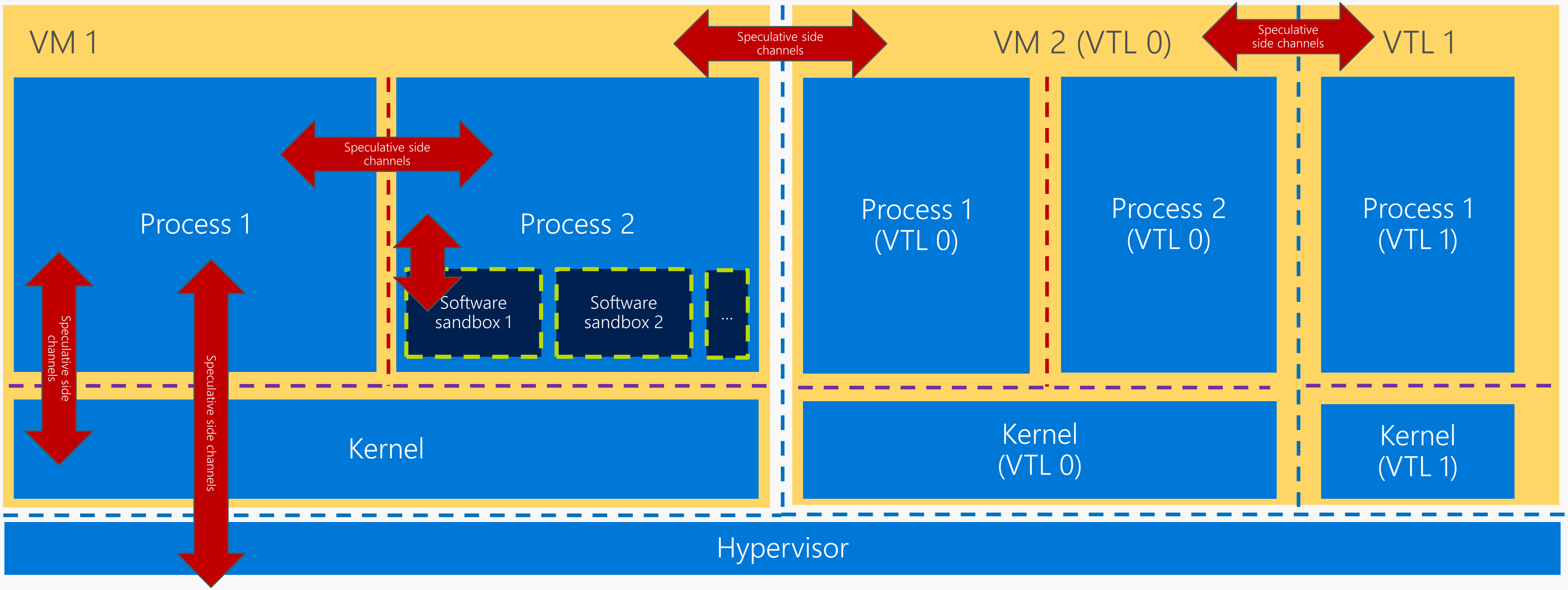
Challenges with eliminating vulnerabilities [1/2]

Most of the vulnerability classes that existed 20+ years ago still exist today	• Some vulnerability classes have been eliminated, but most remain • Developers are still making many of the same mistakes
SDL & training can help, but cognitive load on developers remains high	• Developers are expected to self-identify & prevent vulnerabilities, often without sufficient tools to do so
Software is increasingly developed with disjoint security policies & standards	• Increasing adoption of OSS means inheriting varying security practices • Not reasonable to expect training, SDL, or other policies to be on par
Finding every vulnerability is not scalable or practical	• Hunting for vulnerabilities is necessary today, but insufficient • The number of vulnerabilities found & fixed continues to increase despite decades of effort by many people & tools

Challenges with eliminating vulnerabilities [2/2]

New vulnerability classes have arisen that were not anticipated

- Speculative execution side channels (Spectre, Meltdown, etc) have impacted the security boundaries that software relies on



Challenges with breaking exploitation techniques [1/4]

Since ~2012 we've been pursuing a set of solutions to mitigate arbitrary native code execution

Prevent control-flow hijacking

Control Flow Guard (CFG)

Enforce control flow integrity on indirect calls

Shipped

Shadow stack

Use a separate stack for return addresses

Future (CET[1])

Prevent arbitrary code generation

Code Integrity Guard (CIG)

Images must be signed and arbitrary images cannot be loaded

Shipped

Arbitrary Code Guard (ACG)

Prevent dynamic code generation, modification, and execution

Shipped

CFG+CIG+ACG are enabled on Windows 10 for Edge, Hyper-V VM Worker Process, and the Windows kernel (when HVCI enabled)

How have exploits adapted?

[1] CET = Intel's Control-flow Enforcement Technology

Challenges with breaking exploitation techniques [2/4]

Most exploits have followed the same general steps since ~2016

Start

Place array base or length at predictable location

Modify array base or length via memory corruption, enabling arbitrary read/write

Use arbitrary read/write to discover DLL base address

Use arbitrary read/write to discover DLL base address & stack address

Construct ROP payload by searching for code sequences in the DLL

Corrupt return address and trigger return to first gadget

Corrupt function pointer and trigger function call to first gadget

Corrupt C++ virtual table pointer and trigger method call to first gadget

Execute ROP payload

Execute arbitrary native code

Corrupt state of security policy

Read sensitive content

End

With CFG+CIG+ACG available, exploits typically focus on:

1. Targets that don't enable them
2. Corrupting return addresses
3. Data-only corruption

Challenges with breaking exploitation techniques [3/4]

Robust, efficient, and compatible control-flow integrity has proven difficult

- CET will address the last major gap related to return address protection
- We've encountered multiple design limitations with CFG that are challenging to address assuming arbitrary read/write at arbitrary times

Limitation	Example
Calling valid functions out of context	Corrupting a function pointer with the address of "system" or other sensitive functions is possible because CFG is coarse-grained today
Modifying memory that is used to create a CPU context	Corrupting data used by the loader, exception handler, unwinder, or set thread context can lead to setting an instruction pointer to a controlled value
Making read-only memory writable	Coercing an application into making read-only pages writable and then corrupting imported functions and other data CFG expects to be read-only
Reusing stale code pointers	Suspending a thread and then resuming it after the code referred to by the instruction pointer has changed
Downgrade attacks	Coercing an application to load a DLL that doesn't have CFG enabled or that has a gap in CFG instrumentation/coverage

See our talk on The Evolution of CFI Attacks and Defenses for more information

https://github.com/Microsoft/MSRC-Security-Research/blob/master/presentations/2018_02_OffensiveCon/The%20Evolution%20of%20CFI%20Attacks%20and%20Defenses.pdf

Challenges with breaking exploitation techniques [4/4]

Data-only corruption remains a common & generally unsolved problem

- At a certain level, mitigating data-only corruption is equivalent to mitigating memory safety all-up
- Isolating all sensitive information and security policies becomes untenable at some point because you eventually lose containment

Array base & length corruption

Exploits typically corrupt an array base or length to enable arbitrary read/write of memory

Robustly preventing this is non-trivial as you need to:

- Protect all arrays
- Protect all pointers to arrays
- Protect all pointers to pointers
- …

This quickly decomposes to needing to solve memory safety

Kernel process token corruption

Windows kernel exploits often corrupt the Token field of an EPROCESS object to elevate privileges to SYSTEM

EPROCESS objects are dynamically allocated, mutable, and frequently accessed

Making the Token field read-only is insufficient because all pointers to the EPROCESS need to be protected as well

This quickly decomposes to needing to solve memory safety or isolate logic & state related to the Token of an EPROCESS

Kernel object ACL corruption

Windows kernel exploits sometimes corrupt the ACL associated with a kernel object to grant unauthorized access

This has the same challenges as protecting kernel process tokens

These are just a few examples of the real-world challenges with mitigating data-only corruption attacks

Challenges with containing damage & preventing persistence

Fine-grained kernel attack surface reduction	• Limiting the kernel system call interface exposed to sandboxed applications in a configurable way is challenging on Windows today • Virtualized containers such as Windows Defender Application Guard (WDAG) or Windows Sandbox can provide good alternatives
Efficient & fine-grained compartmentalization	• Efficiently isolating components within a process is challenging today, e.g. isolating the loader, unwinder, and exception handling logic for CFG • Software designs & implementations are not generally amenable to fine-grained compartmentalization today
Rising tensions between high density and hostile multitenant isolation	• The desire for high density (web browsing, containers) is increasingly at odds with what is necessary to provide hostile multitenant isolation • The overhead of mitigating speculative side channels affects density (e.g. site isolation, core isolation, context switching, etc)

Challenges with limiting the window of time to exploit

Long-term support & the evolving threat landscape

- Long-term supported versions of software do not typically benefit from advancements in hardening & defense
- As the threat landscape evolves, older versions of software have typically become less resilient to modern techniques

OSS, many dependencies, and the software update lifecycle

- Software update supply chains are increasingly tied to components that many apps & services take a dependency on
- Coordinating patch releases amongst all app & service dependencies is an increasing challenge

Strategic shifts & opportunities

What should secure software development look like in the future?

Should it still be easy for developers to make the same mistakes they make today?

Should software & service vendors still be fixing a non-trivial number of vulnerabilities?

Should consumers & businesses still be concerned about the risk associated with software vulnerabilities?

What can we do to get a point where we are “done” with vulnerabilities?

What would it mean to get to "done"?

"done" =~ software vulnerabilities are no longer a significant problem

For customers…	Minimal risk of being attacked via a vulnerability
	Security updates are uncommon and are non-disruptive
For attackers…	Exploitable vulnerabilities are uncommon
	Exploiting vulnerabilities is no longer economically viable
For vendors…	The total cost of secure software dev & support is minimal

We don't need to get to zero vulnerabilities to get to "done"

Design & logic vulnerabilities are more challenging & require more thought

Individual apps & services may get to "done" at different rates

This is a huge challenge, but it is a goal state we need to work toward

Toward getting to done with vulnerabilities

Our mitigation tactics are still relevant today, but our strategic objectives can improve

We want to shift the goal posts from *increasing cost & difficulty* toward *getting to done*

Make unsafe code safer	Transition to safer languages	Safer & more efficient dev
Eliminate common classes of memory safety vulnerabilities • Build time errors • Runtime prevention	Make languages safer (C++) and use safer alternatives (Rust, C#) • Enforcing C++ core guidelines • Practical usage of memory safe languages	Make software development more secure & cost effective • Warning autofix tools • Improve perf of safe code

Focus more on making it durably hard for developers to make mistakes while retaining good perf & dev efficiency

Progress toward this direction

Make unsafe code safer

Joseph Bialek
@JosephBialek

Please join the Windows kernel in wishing farewell to uninitialized plain-old-data structs on the stack. As of today's WIPFast build, any Windows code compiled with /kernel also gets compiled with InitAll, a compiler security feature that initializes POD structs at declaration.

```
898424d8010000 mov     qword ptr [rsp+1D8h],rax <-- 24 byte struct
898424e0010000 mov     qword ptr [rsp+1E0h],rax
898424e8010000 mov     qword ptr [rsp+1E8h],rax
```

10:28 AM · Nov 14, 2018 · Twitter Web Client

150 Retweets **379** Likes

This bug class accounted for 49 vulnerabilities reported to MSRC in 2017-2018 (~4%)

Transition to safer languages

Matt Miller
@epakskape

If you're writing C++ code or reviewing it for vulnerabilities, consider using gsl::span as a safer alternative to using raw pointers to access arrays. It provides a relatively straightforward way to prevent out of bounds read/write memory safety issues.

isocpp/CppCoreGuidelines
The C++ Core Guidelines are a set of tried-and-true guidelines, rules, and best practices about coding in C++ - ...
github.com

10:41 AM · Aug 29, 2018 · Twitter Web Client

We've been adopting span in key code bases (e.g. Hyper-V) and it has already helped eliminate vulnerabilities that were later identified

More progress on the horizon

Make unsafe code safer

Zero initialization of most Windows kernel stack variables

Zero initialization of Windows kernel pool allocations

C++ static downcast protection

Transition to safer languages

A proactive approach to more secure code

Security Research & Defense / By MSRC Team / July 16, 2019

We need a safer systems programming language

Security Research & Defense / By MSRC Team / July 18, 2019

Why Rust for safe systems programming

Security Research & Defense / By MSRC Team / July 22, 2019

https://msrc-blog.microsoft.com/tag/safe-systems-programming-languages/

Some of our focus areas for ongoing R&D

Eliminating common vulnerability classes

Adopting safer languages where it really matters

Efficient & finer-grained compartmentalization

Stronger & more robust exploit mitigations

We believe these focus areas will help us address many of the challenges we are currently facing

It’s important to remember there are other threats

Attack a target environment …	Threats
Through asset compromise	Supply chain
	Physical attacks
Without authorized credentials	Vulnerabilities
	Insecure configuration
With authorized credentials	Identity compromise
	Malicious insiders

This presentation focused on vulnerability mitigation, but other threats are important to mitigate as well

As the cost of exploiting vulnerabilities has gone up, other vectors have increased in favor (e.g. social engineering, password spraying, etc)

Wrapping up

We believe our strategy has had a positive impact & we're continuing to refine it

Many of the challenges we face are relevant to the software industry as a whole

We're excited about making more progress toward getting to done ☺

Fascinated by what you saw? Want to help us make the online world safer?

Report vulnerabilities & mitigation bypasses via our bounty programs!

https://aka.ms/bugbounty