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 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

- **Training**
  - Core training

- **Requirements**
  - Define quality gates/bug bar
  - Analyze security and privacy risk

- **Design**
  - Attack surface analysis
  - Threat Modeling

- **Implementation**
  - Specify tools
  - Enforce banned functions
  - Static analysis

- **Verification**
  - Dynamic/Fuzz testing
  - Verify threat models/attack surface

- **Release**
  - Response plan
  - Final security review
  - Release archive

- **Response**
  - Response execution

~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
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)

- Attack Surface Discovery
  - System Calls
  - COM/RPC
  - Drivers
  - WinRT

- Runtime Analysis
  - Install Build
  - Enable All Features

- Attack Surface Data

Output to Power BI
## Automated Discovery

### Attack Surface Requiring Action

<table>
<thead>
<tr>
<th>Attack surface</th>
<th>Raw #</th>
<th>Diff from last OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WinRT</td>
<td>731</td>
<td>+46</td>
</tr>
<tr>
<td>RPC</td>
<td>164</td>
<td>+33</td>
</tr>
<tr>
<td>COM servers</td>
<td>231</td>
<td>+15</td>
</tr>
<tr>
<td>Device drivers</td>
<td>142</td>
<td>+0</td>
</tr>
<tr>
<td>System calls</td>
<td>1737</td>
<td>+84</td>
</tr>
</tbody>
</table>
Enabling Developers to Fuzz like a Boss

Fuzzing used to be a manual process

- Attack surface is identified manually
- Developer manually writes test harness to exercise coverage
- 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

MSRD
- Fuzzer Test Harness
  - LibFuzzer
  - Syzkaller
  - SAGE
  - Etc.

Runtime monitoring
- Crash Analysis
- Automated Crash Triage
- ASan

Fuzz Target or Build
Symbols
Test Harness
Corpus/meta data

Hyperscale Fuzzing

Azure DevOps
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
Developer using Azure DevOps

GIT PR → OS Repo → Official Build

Static Analysis Build:
- OACR/PreFast
- Semmle

Static Analysis bug filed

Release Readiness

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;
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

5 billion
Threat Detections Per-Month

11+
Zero-day exploits tracked

Med-High Pri
Critical Surface
No known attacks

Low Pri
Medium Surface
No Known Attacks

Medium Pri
Important Surface
Known to be attacked

Highest Pri
Critical Surface
Known to be attacked

700+
MSRC cases from 100+ finders

Community
Bounty
Bluehat 😊
Hyper-V: a case study

Virtual Machine Worker Process (VMWP.exe)

Host Compute Service (VMCompute.exe)

Containerd (Kubernetes/Dockerd)

VHDPMEM tool (vhdconvert.exe)

Kernel (Root Partition)

Hypervisor
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

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
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

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
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
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.

```rust
/// 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.

```c
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

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

/// 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);
}
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()`

```rust
def breakpoint(&mut self, franzia: &mutFranzia) {
    let rip = franzia.vm().regs().rip();
    if rip == self.end_case_bp {
        // call case done
        franzia.queue_new_fuzz_case(ResetReason::EndCondition);
    }
}
```
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

13 Days

Time-to-Bug

Total

20 Days

Time-to-Exploit
Exploit Primitive 1

Memory Write (DejaBlue)
Custom serialization layer
Heap smash with everything controlled!

```cpp
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);
```
Memory Read
Paired RCE with externally reported info leak (thanks!)
Fastpath performance enhancement
Leaks uninitialized heap data

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

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();
};

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
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 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