Reverse Engineering Windows AFD.sys

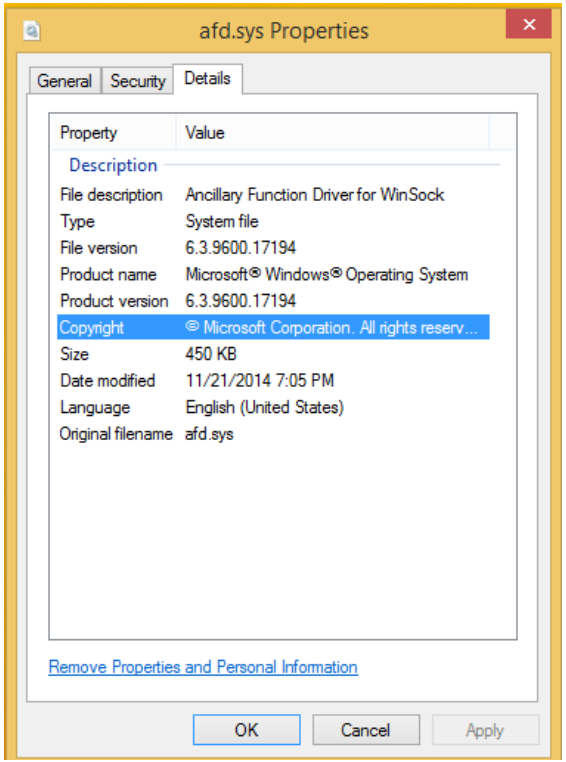
Steven Vittitoe
Google
@bool101
bool@google.com

Outline

- Why AFD.sys
- Winsock overview
- Interesting findings
- Input to AFD.sys
- Analysis
- Fuzzing
- Future

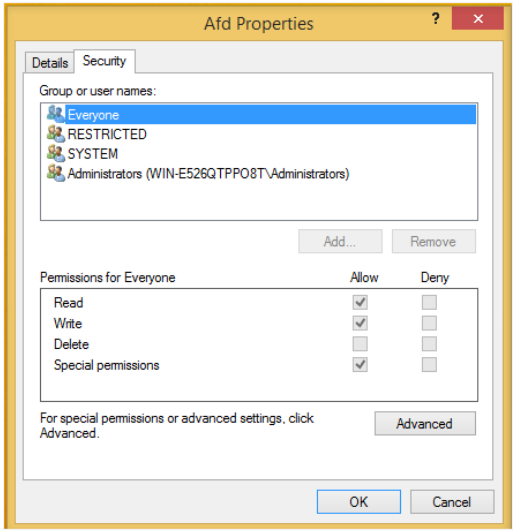
What is AFD.sys?

- Default kernel module
- Ancillary Function Driver
- Ring 0 entrypoint for Winsock
- Required for socket() calls
- Not all network comms use it:
  - winhttp wininet
  - webdav mrxsmb



Why AFD.sys?

- Sandbox accessibility
  - Chrome YES
  - Adobe Reader YES
  - IE EPM YES



- History of bugs:
  - CVE-2011-2005 CVE-2012-0148
  - CVE-2013-3887 CVE-2014-1767

Goals

- Project Zero's goal:

  "Make 0-days hard(er)"

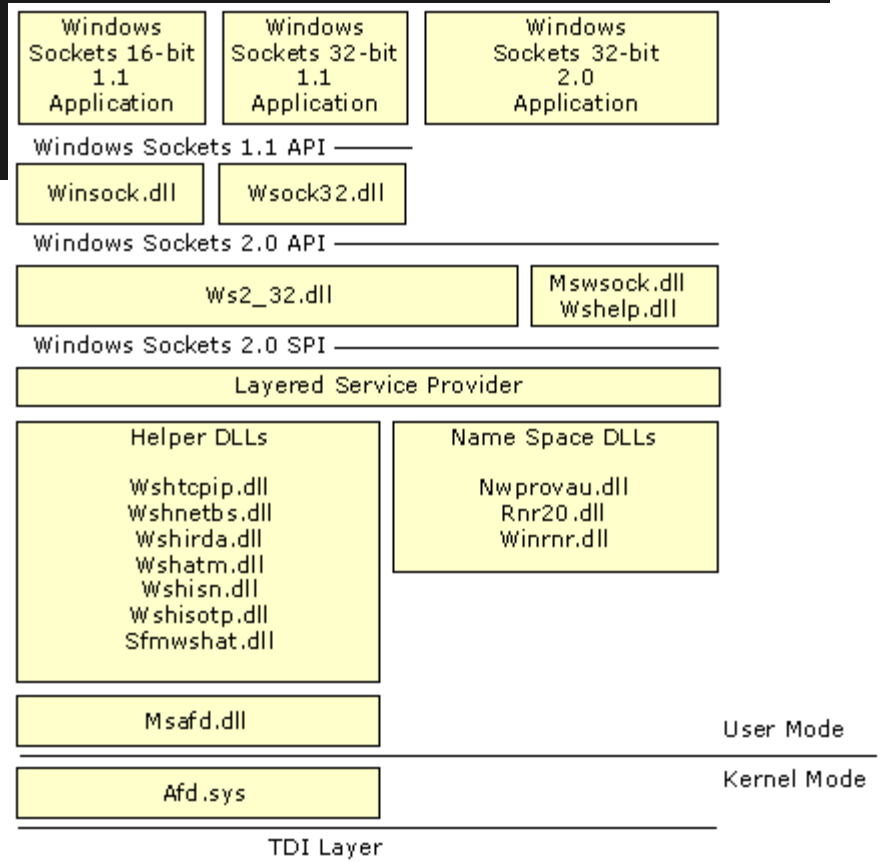
- Strengthen Sandboxes
  - Widely adopted strategy
  - Increase attacker cost
  - Ways to escape:
    - Logic errors (broker process)
    - Bugs in syscalls / win32k.sys
    - Bugs in accessible devices!

Why AFD.sys?

- Cannot be disabled until Windows 8
  - Even then not easy to disable
- Complexity and accessibility
  - AFD.sys size ~500KB
    - win32k.sys is 3.1MB
    - most kernel drivers < 100KB
  - 70 IOCTL’s reachable from \\Device\Afd\Endpoint
  - Handles everything from TCP/IP to SAN

Winsock

- socket(AF_INET) call
  1. ws2_32 (2 fn)
  2. mswsock (4 fn)
  3. wshtcpip (1 fn)
  4. mswsock (IOCTL)



AFD is a translator

- AFD acts as a server to user mode Winsock
  - Abstracts multiple protocols
  - Ends up relaying to:
    - Transport Driver Interface (TDI)
    - Winsock Kernel (WSK)
- Serves kernel mode clients as a WSK provider (internal IOCTL)

First Glance

- DbgPrint
  - Normally removed in release builds?
  - 23 xrefs in Win7
  - 113 xrefs in Win8
- 74/279 import DbgPrint* (~25%)
  - Event Tracing for Windows (ETW) extensively used
  - Helpful in RE efforts

Registry

- Several configurations pulled from registry:
  - HKLM\System\CCS\Services\Afd
    - Buffer sizes
    - DisableRawSecurity - admin raw sockets
    - DefaultSendWindow
    - AfdReadRegistry() populates _AfdConfigInfo
- Few are “Volatile” configurations
  - Change notification registered

Inputs

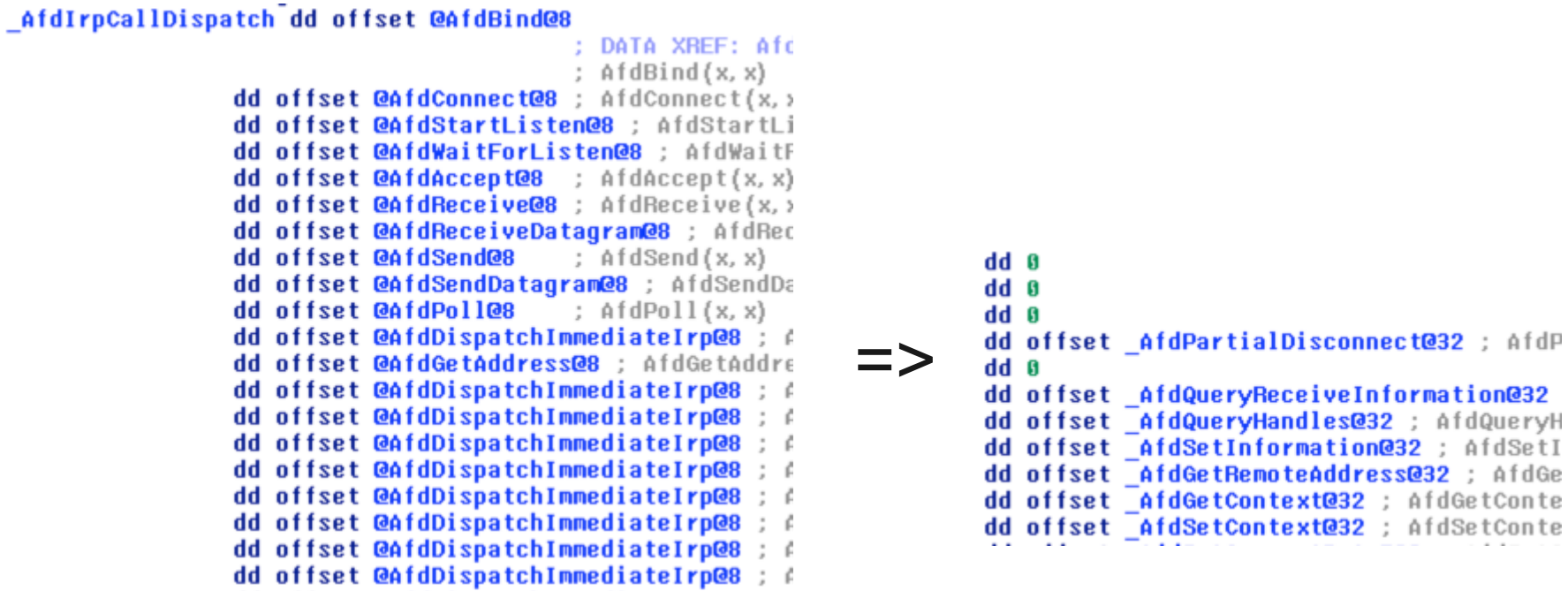
```
memset32(DriverObject->MajorFunction, (int)AfdDispatch, 0x1Cu);
DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = (PDRIVER_DISPATCH)AfdDispatchDeviceControl;
DriverObject->MajorFunction[IRP_MJ_INTERNAL_DEVICE_CONTROL] = (PDRIVER_DISPATCH)AfdWskDispatchInternalDeviceControl;
DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] = (PDRIVER_DISPATCH)AfdEtwDispatch;
DriverObject->FastIoDispatch = (PFAST_IO_DISPATCH)&AfdFastIoDispatch;
DriverObject->DriverUnload = (PDRIVER_UNLOAD)AfdUnload;
```

- IOCTLs
- Plug-n-Play Events
- TDI address changes and filtering
- RPC

IOCTLs

- Easy to find tables
  - AfdIrpCallDispatch - functions
  - AfdIoctlTable - numbers
- Another level of indirection
  - AfdImmediateCallDispatch
  - For routines that always IofCompleteRequest

Immediate Call Dispatch



Static Bug Hunting

- Windows 7 x86
- Basic bottom up static analysis
  - memcpy, memmove, ExAllocatePool*, etc
  - functions with __security_check_cookie xrefs
  - functions with large stack buffers
  - object reference counts
- Script to find unchecked return values
  - ExAllocatePool* (Note: TagPriority raises exception)

Static Bug Hunting

- Manual review of all reachable IOCTLs
  - Not WSK or SAN related IOCTLs
  - Data alignment
  - Proper size restrictions
  - TOCTOU on METHOD_NEITHER IOCTLs
  - Integer under/overflow issues
  - Signed integer issues

Fuzzing

- Preference for static / dynamic analysis
  - Better understanding of target
  - Leads to better fuzzers
- Two weeks fuzz time
  - Single core
  - Simple fuzzer
    - Hit all IOCTLs
    - Usermode buffer mutated in another thread
    - Basic awareness of what was expected data

Future Work

- “Native” AFD library
  - Skip user mode winsock entirely
  - Compile into shellcode for use in a sandbox
  - Feedback into a more intelligent fuzzer
- More fuzzing
  - At scale
  - More expected data structures defined
- Manual review of WSK and SAN functions

Thanks

- Google
- Project Zero
- James Forshaw

Questions

?