# Untangling Windows 8 Web Services

# Introduction

HD Moore

- Metasploit founder and chief architect
- Chief research officer for Rapid7
- Head of Rapid7 Labs

Twitter: **@hdmoore**

Email: **hdm@rapid7.com**

RAPID7

# Windows & Web Services

- Windows and Web Services
  - Windows has been moving to SOAP-based management
  - Vista, 7, 8 support remote administration through WinRM
  - Also applies to Server 2008 and Server 2012

- WinRM overlaps with DCOM and DCERPC
  - Remote access to WMI and “psexec” style functionality
  - Full support already in Metasploit

# Windows Development

- Windows Communication Foundation
  - WCF is a set of common APIs, introduced in .NET 4.5
  - Defines transport, encoding, eventing, and security
  - Primarily used to build SOAP and REST services
  - Widely used by Windows enterprise developers
  - More details in Brian Holyfield's presentation

https://www.owasp.org/images/6/6c/Attacking_WCF_Web_Services-Brian_Holyfield.pdf

# Web Services: Not Just for Servers

- Web services have invaded our desktop systems
  - Developer adoption of WCF has driven micro-services
  - "Service Oriented Applications" is the buzzword
  - Every Windows Vista+ desktop has web services
- Little security information on these services
  - Developer documentation touches on some of it
  - No real external security review and discussion
  - Lack of information usually points to bad things ☺

# Identifying & Auditing Web Services

- The tried and true network service audit model
  - Identify what network services are listening
  - Find the processes that own those services
  - List out the EXEs and DLLs are involved
  - Reverse engineer those binaries
  - Obtain packet captures
  - Write some test code
  - Attach a debugger
  - Find bugs?
  - Profit!

# Finding Web Services

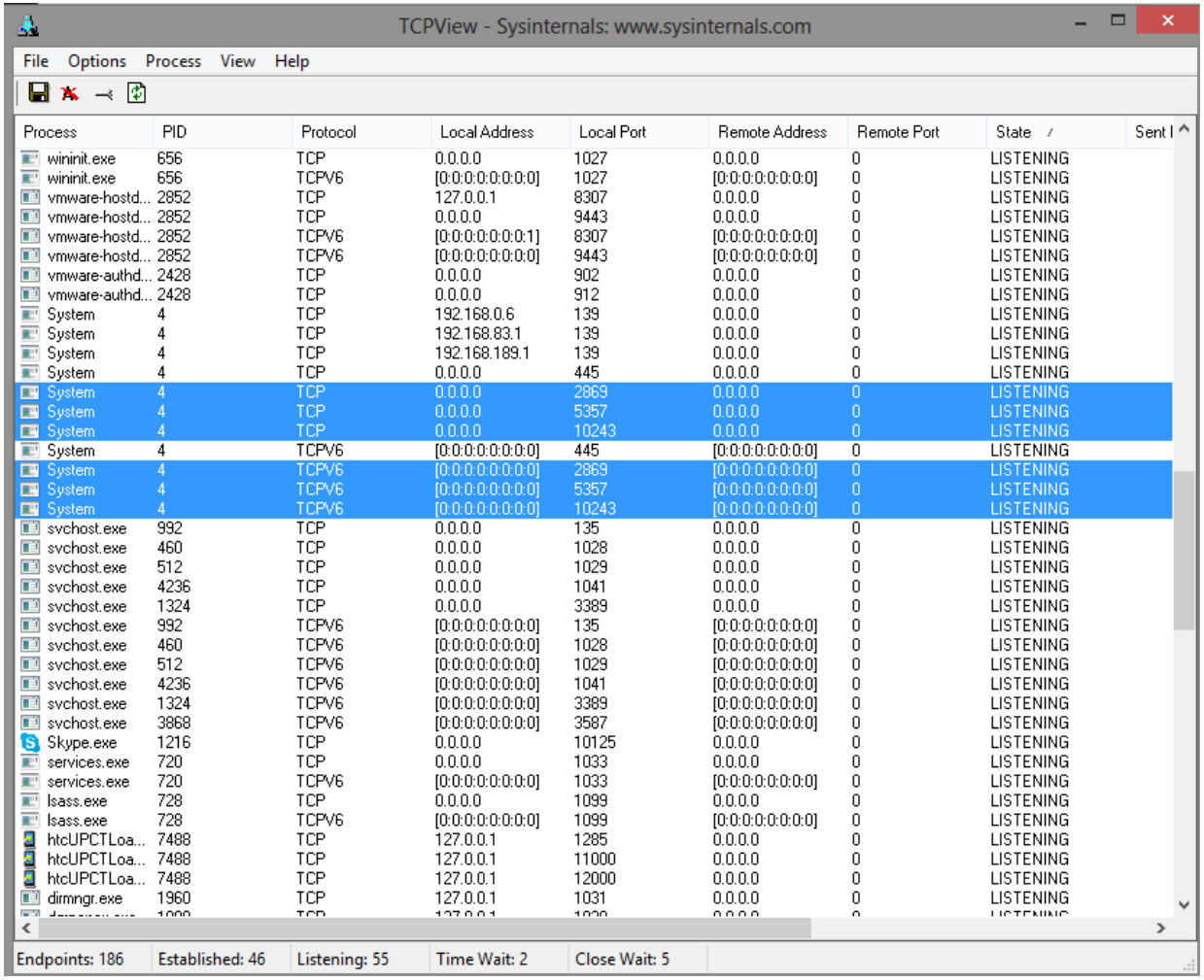
- Start off with a nmap -sSV -p1-65535 --min-rate=2500
  - Microsoft HTTPAPI/2.0 is a good indicator

```
PORT      STATE SERVICE       VERSION
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn
445/tcp   open  netbios-ssn
1027/tcp  open  msrpc         Microsoft Windows RPC
1028/tcp  open  msrpc         Microsoft Windows RPC
1029/tcp  open  msrpc         Microsoft Windows RPC
2869/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
5357/tcp  open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
10243/tcp open  http          Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
```

RAPID7

# Identifying Web Services

- What process owns 2869, 5357, 10243?




RAPID7

# Web Services meet Windows Kernel

- The System process owns the listening ports
  - Why is the kernel handling web services?
  - How do we find the real process?

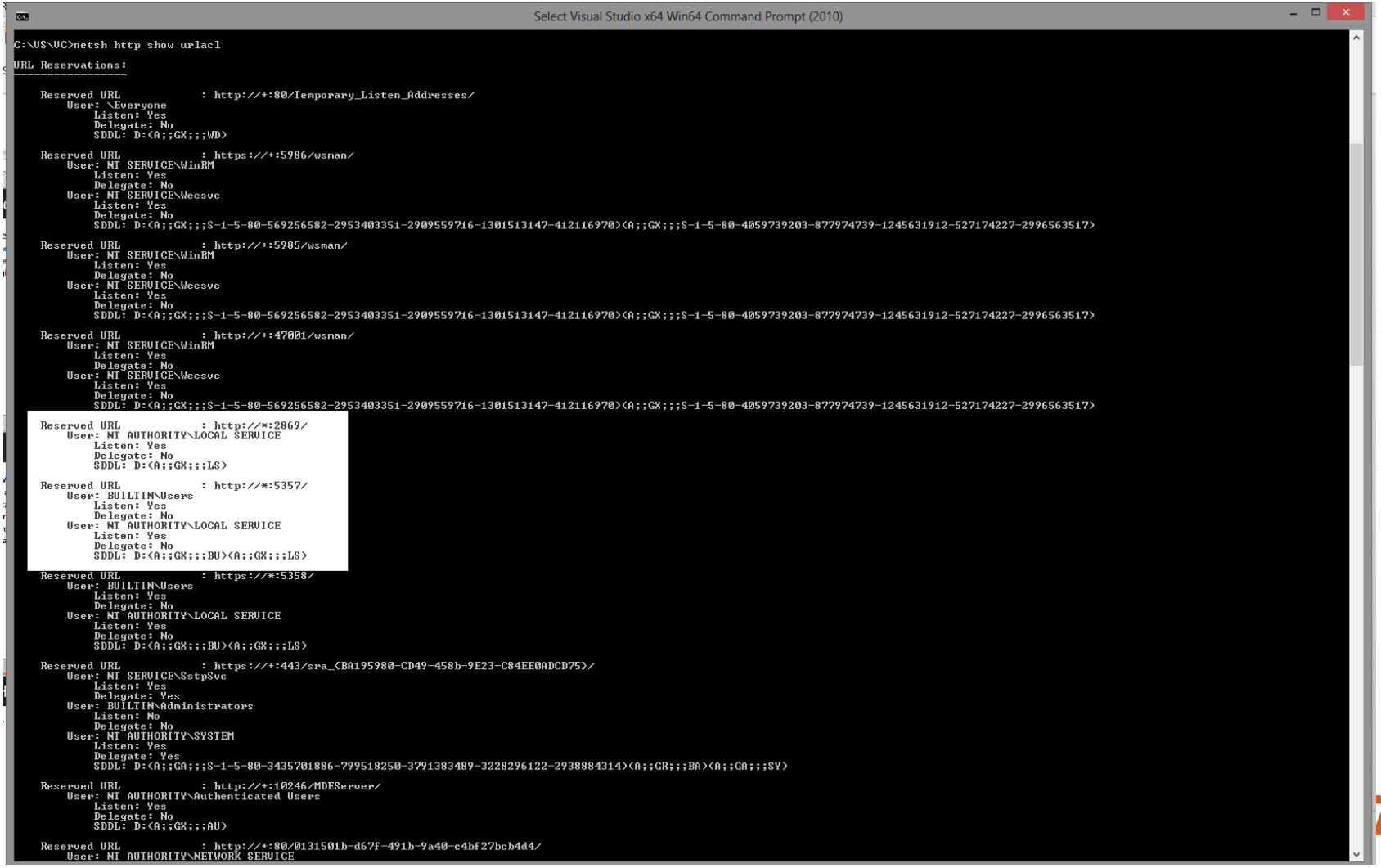
**WHAT IS THIS DEVIL MAGIC?!?!**

# HTTP.sys

- Windows runs a network driver to handle HTTP services
  - HTTP.sys implements a user-land API through httpapi.dll
  - Apps can reserve specific ports, paths, and vhosts
  - The kernel will deliver incoming requests
  - Badness gets filtered out within ring zero
  - Paths are handled through reservations
  - Each reservation has an ACL

# Inspecting HTTP.sys URL Reservations

- C:\> netsh http show urlacl



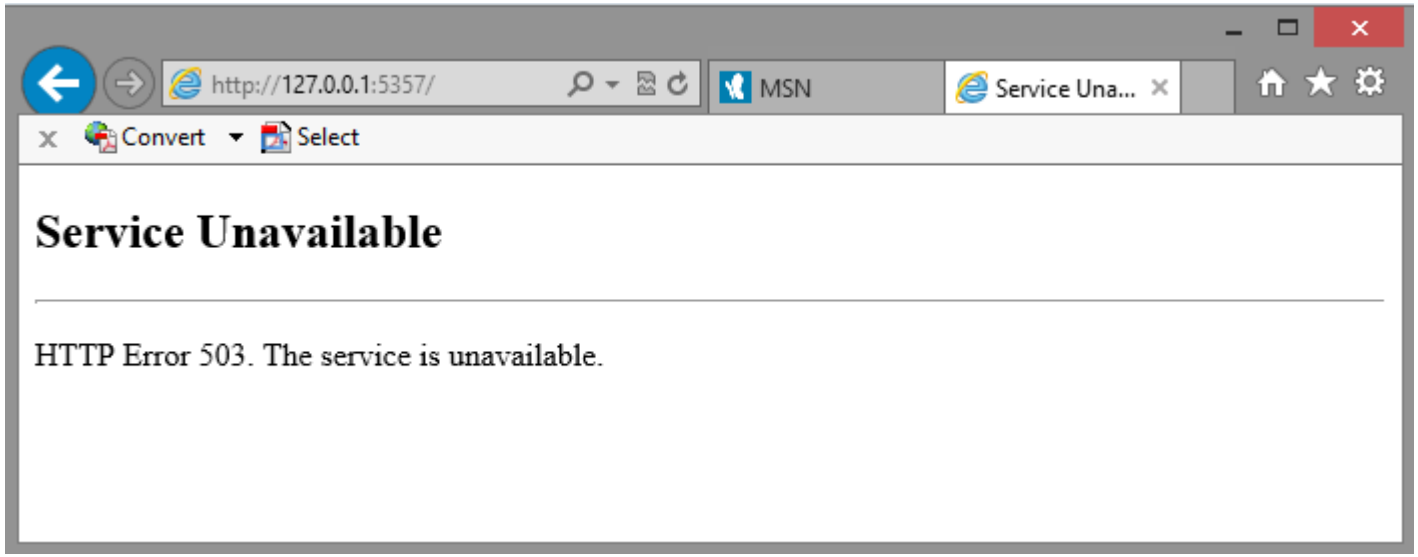
# Helpful URL ACL Entries

Reserved URL : **https://+:5986/wsman/**
User: NT SERVICE\WinRM
Listen: Yes
Delegate: No

Reserved URL : **http://127.0.0.1:47873/help/**
User: \Everyone
Listen: Yes

Reserved URL : **https://+:10245/WMPNSSv4/**
User: NT SERVICE\WMPNetworkSvc
Listen: Yes

# Not Helpful




Reserved URL : **http://*:2869/**

User: NT AUTHORITY\LOCAL SERVICE

Listen: Yes

Reserved URL : **http://*:5357/**

User: BUILTIN\Users

Listen: Yes

RAPID7

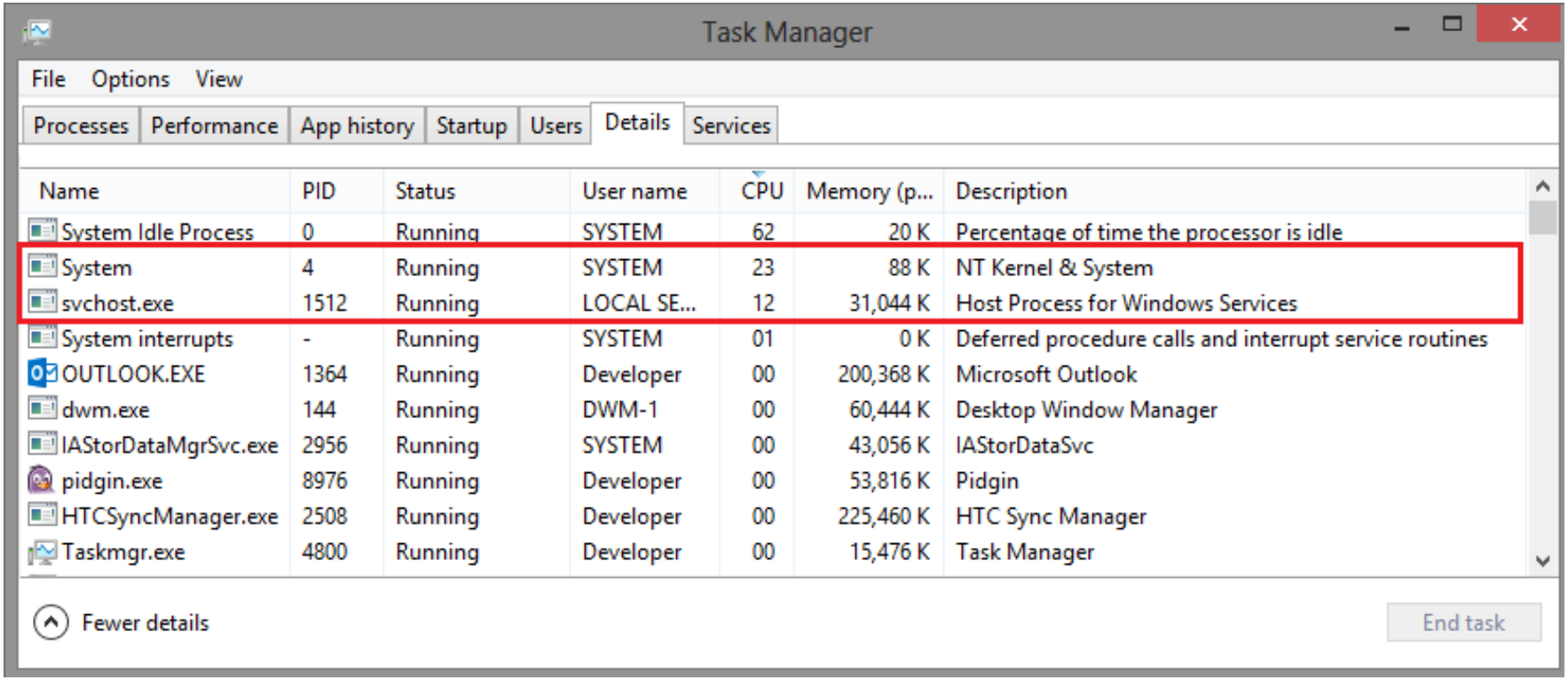
# Around MSDN in 80 Hours

- Applications have reservations for specific URLs
  - Reservations are not the same as active URL handlers
  - Quite a few of those reservations have no service
  - WinRM is reserved (/wsman/), not on by default
  - Reservations also show alternate endpoints
- Reservations are a good start, but not enough
  - We need to figure out what URLs are actually used
  - Brute force? Dir buster? WinDBG breakpoints?
  - Tracing frameworks? Hooks in HTTPAPI.dll?
  - All of this is time intensive…

# Finding the Process

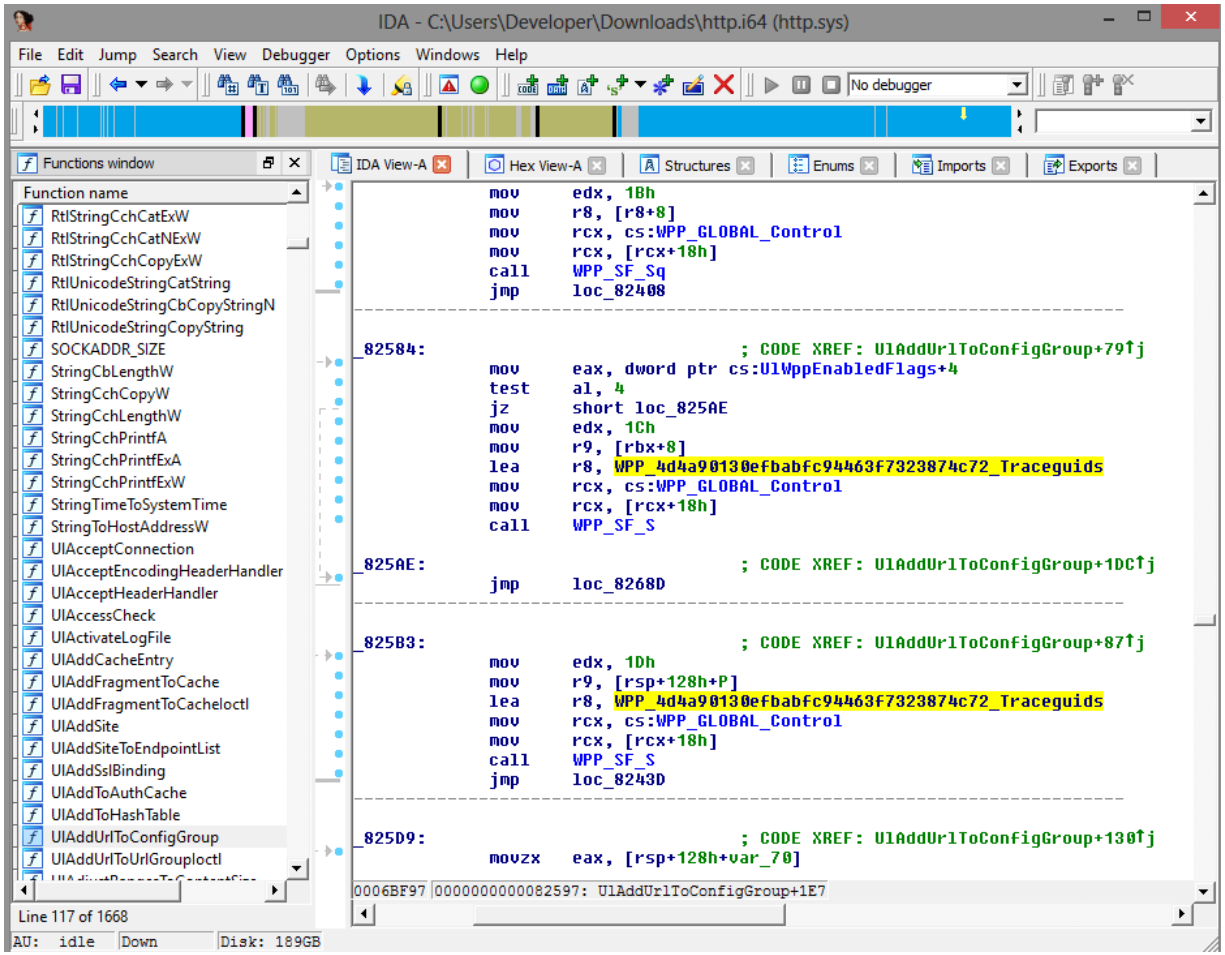
- Hooks and debugging require us to find the process
  - Sometimes brute force is the right approach

$ ab -n 1000000 -c 50 http://192.168.0.6:5357/



# IDA Pro + HTTP.sys

- IDA Pro helped uncover a better solution
  - `HTTP.sys!AddUrlToConfigGroup` has built-in tracing!




RAPID7

# Event Tracing + HTTP.sys

- Fight Windows kernel magic with more magic
  - Event Tracing for Windows is amazing

**C:\> logman start httptrace -p Microsoft-Windows-HttpService 0xFFFF -o trace.etl -ets**

**C:\> net stop upnphost**

**C:\> net start upnphost**

**.. Do some UPnP stuff ..**

**C:\> logman stop httptrace -ets**

**C:\> tracerpt.exe trace.etl of CSV -o httptrace.csv**

| Event Name | Type | Event ID | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| EventTrace | Header | 0 | | 83952134 | | 9200 | | 8 | | 1.30166E+17 | |
| Microsoft-Windows-HttpService | RemUrl | 32 | "http://*:2869/upnp/eventing/" | | | | | | | | |
| **Microsoft-Windows-HttpService** | **AddUrl** | **31** | **"http://*:2869/upnp/eventing/"** | **0x0** | | | | | | | |
| Microsoft-Windows-HttpService | ConnConnect | 21 | | 16 | "192.168.0.6:2869" | | | 16 | "192.168.0.10:54775" | | |
| Microsoft-Windows-HttpService | ConnIdAssgn | 22 | 0xFE000006600001AB | | 0xFFFFFA8042D97BB0 | | | | | | |
| Microsoft-Windows-HttpService | RecvReq | 1 | 0xFE000006600001AB | | | | 16 | "192.168.0.10:54775" | | | |
| Microsoft-Windows-HttpService | Parse | 2 | | 1 | "http://192.168.0.6:2869/upnp/eventing/vuhkhxybrb" | | | | | | |
| Microsoft-Windows-HttpService | Deliver | 3 | 0xFE000006800001AC | | | | 0 | "<<unnamed>>" | "http://192.168.0.6:2869/upnp/eventing/vuhkhxybrb" | | |

RAPID7

# Event Tracing is Awesome

- HttpService tracing logs everything using HTTP.sys
  - Easily monitor IIS-based web applications & web services
  - Uncovers fun features (and vulns) in all sorts of applications
  - CSV output makes detailed analysis easy

- Run "logman providers" for a full list of targets
  - Microsoft-Windows-AppLocker
  - Microsoft-Windows-Bluetooth-HidBthLE
  - Microsoft-Windows-Crypto-RNG
  - Microsoft-Windows-Firewall
  - Microsoft-Windows-NDIS-PacketCapture

# Tracing WinHTTP Client Requests

- Sometimes you need to trace HTTP client code as well
  - Many processes are both clients and services (UPnP, WSD)
  - Logging all HTTP client actions is really useful
  - This survives a reboot and acts globally

**C:\> netsh winhttp set tracing trace-file-prefix="C:\Temp\" level=verbose **
**format=ansi state=enabled max-trace-file-size=1073741824**

**.. Wait for the client to do things ..**

**C:\> netsh winhttp set tracing state=disabled**

# Tracing WinHTTP for Great Success

- NetSH tracing happens to be a great rootkit
  - C:\> netsh trace start scenario=internetclient
  - Logs all requests and responses by default
  - SSL is captured clear-text

16:51:47.898 ::*0000004* :: WinHttpWriteData(0x36aae0, 0x11aa7c4, 658, 0x0)
16:51:47.899 ::*0000004* :: <<<<-------- HTTP stream follows below ---------->>>>
**<?xml version="1.0" encoding="utf-8" ?>**
**<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"**
**xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">**
**<soap:Header><wsa:To>urn:uuid:dbe17c74-3b21-4f52-addc-b84b444f73a0</wsa:To>**
**<wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/Get</wsa:Action>**
**<wsa:MessageID>urn:uuid:8506ac50-3646-4621-96806f484d87909</wsa:MessageID>**
**<wsa:ReplyTo>**
**<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous</wsa:Address>**
**</wsa:ReplyTo>**
**<wsa:From><wsa:Address>urn:uuid:b32467b5-e7ee-4ae3-8a8e-f5aa417c23b6</wsa:Address></wsa:From>**
**</soap:Header>**
**<soap:Body></soap:Body>**
**</soap:Envelope>**
16:51:47.899 ::*0000004* :: <<<<-------- End ---------->>>>

RAPID7

# Windows 8 and TCP Port 5357

- Leverage all of the steps we just covered
  - Identify the process by spamming HTTP requests to it
  - Determine what service this is related to using WinDbg
  - Trace HttpAddUrl to identify accessible endpoints
  - Sniff traffic going to and from the service

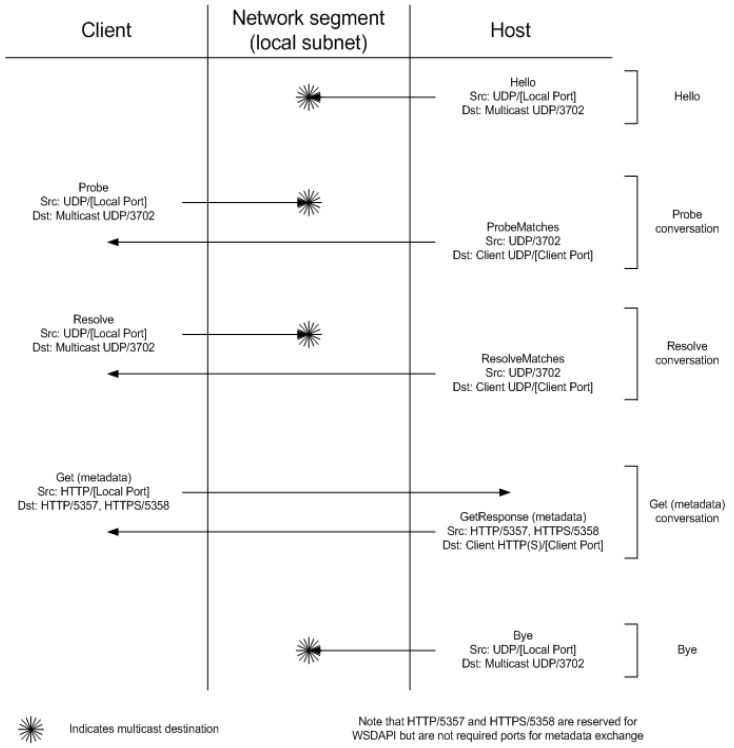
# Web Services for Devices

- TCP Port 5357 is the SOAP endpoint for WSD / DPWS
  - This is a UPnP-like protocol for talking to network devices
  - Primarily used for network printers and scanners
  - WSDL for these two profiles is well-published
- UDP Port 3702 is required to speak WSD
  - The 5357 web service generates two random GUID URLs
  - We have to discover these through UDP 3702 probes
  - The protocol format is SOAP-over-UDP (!)
  - Designed for the local network only

# Web Services for Devices

- WSD provides discovery and metadata extraction

1. Device sends HELLO to multicast
2. Clients send a discovery probe
3. Device replies with SOAP URL
4. Client downloads XML metadata
5. Client registers an event listener
6. Client calls a device API over SOAP
7. Device calls back to client event listener
8. Device responds with data
9. Device sends BYE when shutting down




RAPID7

# WSD Discovery

- WSD is designed to operate on the local network only
  - Windows does not respond to unicast probes from local devices
  - Windows does respond to multicast & broadcast probes
  - Windows announces itself as a pub:Computer device
  - Generally doesn't host any services
- WSD is part of Windows Function Discovery
  - This wraps up WSD, UPnP, and other similar protocols
  - Used in the Network Browser to show devices
- WSD-enabled devices often respond to unicast
  - An Internet-connected printer talks WSD just fine

# WSD on the Internet

- Scanning the internet for WSD-enabled systems
  - Probed the entire internet for WSD on port 3702
  - Extracted the TCP port and URL for metadata
  - Downloaded XML metadata when possible
  - Expected to see a ton of printers and scanners
- WSD is a fairly new protocol (2010+)
  - Most network devices do not support it yet
  - Older versions of Windows ignore it
  - Still, interesting results

# WSD on the Internet

**2.5 million** systems responded to the UDP probe

156,000 Printers

30,000 Scanners

20,000 Video Transmitters

**2.36 million Windows PCs**

- This doesn't make any sense given multicast behavior
  - Almost all of these show an external IP in the xAddrs response

    <wsd:XAddrs>**http://aaa.bbb.ccc.ddd:5357/fc031037-375d-4091-8f5f-c44513a0f99b**/</wsd:XAddrs>
  - It looks like the Windows Firewall allows WSD by default
  - Windows PCs on the internet respond to unicast queries!

RAPID7

# WSD on the Internet

- Sampled 80,000 metadata files via TCP port 5357
- Things started to get interesting

<wsdp:ServiceId>**fdid:Provider\Microsoft.Base.Publication/Publication%5CHomeGroup/22b669b37ea6746d29c989d8bd5578f019f0e592.HomeGroupClassifier_HomeGroup_Invitation_ID**</wsdp:ServiceId>

<pub:Resource>\l?xml\sversion="1.0"\sencoding="UTF-16"?>\lHOMEGROUP_RECORD>\lINVITATION>\alt;PEERINVITATION\sVERSION="1.1"\agt;\alt;CLOUDNAME\agt;LinkLocal_ff00::%10/8\alt;/CLOUDNAME\agt;\alt;SCOPE\agt;LINKLOCAL\alt;/SCOPE\a>\lGUIDNAME>{C81BDCF2-FA30-4EB4-ABD9-B7C7693E3BB3}\l/GUIDNAME

>\lOWNER>**Joan**\l/OWNER

>\lOWNERID>22b669b37ea6746d29c989d8bd5578f019f0e592.HomeGroupClassifier\l/OWNERID

>\lOWNERMACHINENAME>**JOAN-PC**\l/OWNERMACHINENAME

>\lLASTCHANGED>129828926058689726\l/LASTCHANGED

>\lHOMEGROUPSIZE>1\l/HOMEGROUPSIZE

>\lADDRESS>[**fe80::f9e4:dbc0:b1f7:bb84%10**]:3587\l/ADDRESS

>\lDIGITALHASH>-----BEGIN\sCERTIFICATE-….

RAPID7

# Windows HomeGroups meet WSD

- Windows 7 & Windows 8 support HomeGroups
  - Light version of Active Directory for consumers
  - ONLY available over link-local IPv6 (TCP 3587)
  - Simplified file and resource sharing
- Windows uses WSD for Resource Publication
  - Used to share HomeGroup invitations on the local network
  - Also advertises what types of things are shared
  - This leaks local-only resource names externally
- The username and machine name are exposed
  - Part of the HomeGroup invitation record
  - The username gets fun…

# Windows 8 Live.com Authentication

- Windows 8 allows live.com usernames for authentication
  - Your live.com email becomes your desktop username
  - Your password is synced with your live.com account

- Putting this all together
  - WSD UDP responds to unicast on external networks
  - Windows leaks HomeGroup invitations via WSD
  - HomeGroup invitations contain the username
  - Windows 8 allows live.com usernames…

# LOL

- A few thousand live.com account email addresses…

<censored for posterity>

RAPID7

# Thanks!

# Q & A

RAPID7