

# 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**

# 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](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!

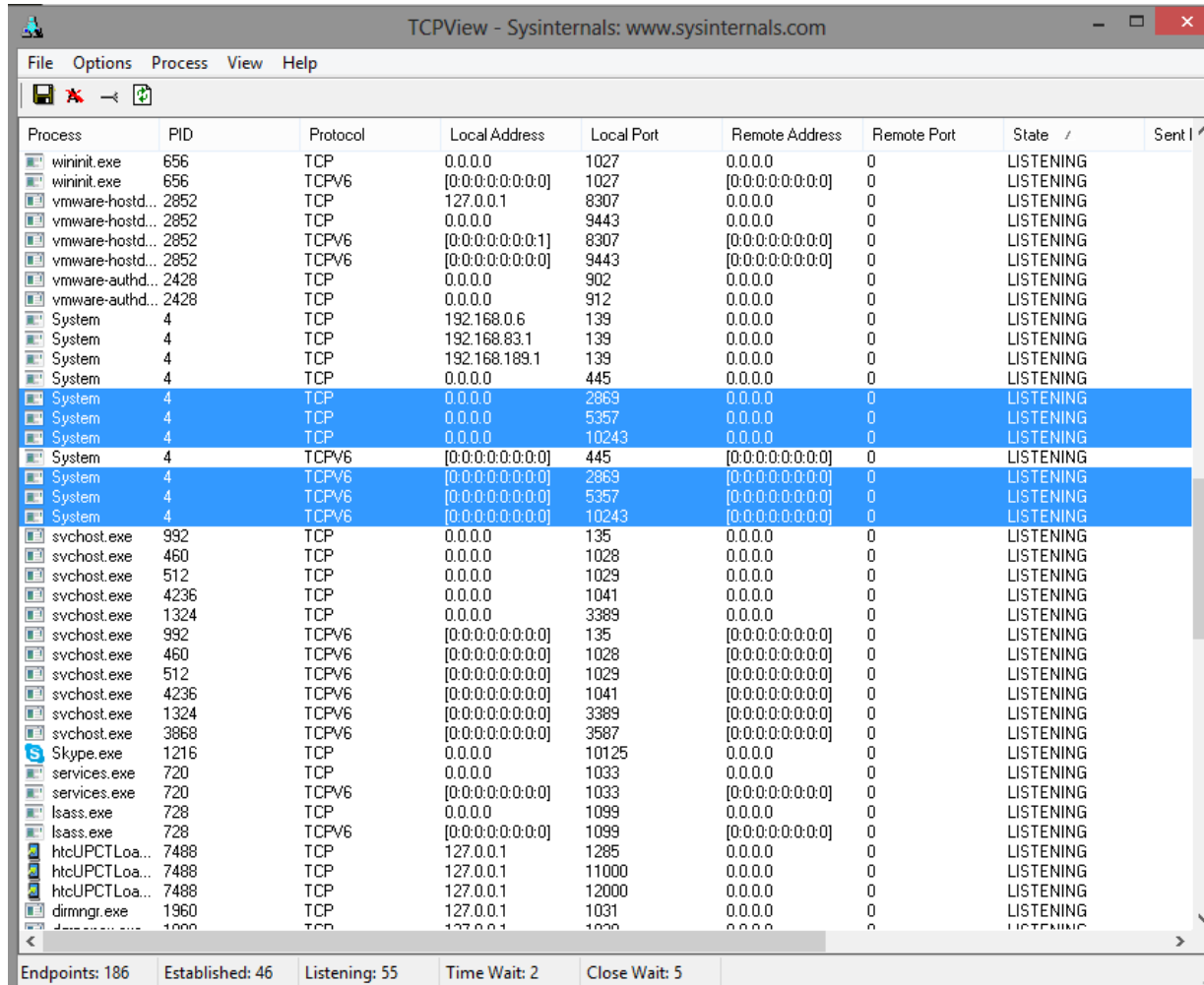
# Finding Web Services

- ▶ 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
<b>2869/tcp</b>	<b>open</b>	<b>http</b>	<b>Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)</b>
<b>5357/tcp</b>	<b>open</b>	<b>http</b>	<b>Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)</b>
<b>10243/tcp</b>	<b>open</b>	<b>http</b>	<b>Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)</b>

# Identifying Web Services

► What process owns 2869, 5357, 10243?



The screenshot shows the TCPView application window with a list of listening ports. The following table represents the data visible in the application:

Process	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port	State
wininit.exe	656	TCP	0.0.0.0	1027	0.0.0.0	0	LISTENING
wininit.exe	656	TCPV6	[0:0:0:0:0:0:0:0]	1027	[0:0:0:0:0:0:0:0]	0	LISTENING
vmware-hostd...	2852	TCP	127.0.0.1	8307	0.0.0.0	0	LISTENING
vmware-hostd...	2852	TCP	0.0.0.0	9443	0.0.0.0	0	LISTENING
vmware-hostd...	2852	TCPV6	[0:0:0:0:0:0:0:1]	8307	[0:0:0:0:0:0:0:0]	0	LISTENING
vmware-hostd...	2852	TCPV6	[0:0:0:0:0:0:0:0]	9443	[0:0:0:0:0:0:0:0]	0	LISTENING
vmware-authd...	2428	TCP	0.0.0.0	902	0.0.0.0	0	LISTENING
vmware-authd...	2428	TCP	0.0.0.0	912	0.0.0.0	0	LISTENING
System	4	TCP	192.168.0.6	139	0.0.0.0	0	LISTENING
System	4	TCP	192.168.83.1	139	0.0.0.0	0	LISTENING
System	4	TCP	192.168.189.1	139	0.0.0.0	0	LISTENING
System	4	TCP	0.0.0.0	445	0.0.0.0	0	LISTENING
System	4	TCP	0.0.0.0	2869	0.0.0.0	0	LISTENING
System	4	TCP	0.0.0.0	5357	0.0.0.0	0	LISTENING
System	4	TCP	0.0.0.0	10243	0.0.0.0	0	LISTENING
System	4	TCPV6	[0:0:0:0:0:0:0:0]	445	[0:0:0:0:0:0:0:0]	0	LISTENING
System	4	TCPV6	[0:0:0:0:0:0:0:0]	2869	[0:0:0:0:0:0:0:0]	0	LISTENING
System	4	TCPV6	[0:0:0:0:0:0:0:0]	5357	[0:0:0:0:0:0:0:0]	0	LISTENING
System	4	TCPV6	[0:0:0:0:0:0:0:0]	10243	[0:0:0:0:0:0:0:0]	0	LISTENING
svchost.exe	992	TCP	0.0.0.0	135	0.0.0.0	0	LISTENING
svchost.exe	460	TCP	0.0.0.0	1028	0.0.0.0	0	LISTENING
svchost.exe	512	TCP	0.0.0.0	1029	0.0.0.0	0	LISTENING
svchost.exe	4236	TCP	0.0.0.0	1041	0.0.0.0	0	LISTENING
svchost.exe	1324	TCP	0.0.0.0	3389	0.0.0.0	0	LISTENING
svchost.exe	992	TCPV6	[0:0:0:0:0:0:0:0]	135	[0:0:0:0:0:0:0:0]	0	LISTENING
svchost.exe	460	TCPV6	[0:0:0:0:0:0:0:0]	1028	[0:0:0:0:0:0:0:0]	0	LISTENING
svchost.exe	512	TCPV6	[0:0:0:0:0:0:0:0]	1029	[0:0:0:0:0:0:0:0]	0	LISTENING
svchost.exe	4236	TCPV6	[0:0:0:0:0:0:0:0]	1041	[0:0:0:0:0:0:0:0]	0	LISTENING
svchost.exe	1324	TCPV6	[0:0:0:0:0:0:0:0]	3389	[0:0:0:0:0:0:0:0]	0	LISTENING
svchost.exe	3868	TCPV6	[0:0:0:0:0:0:0:0]	3587	[0:0:0:0:0:0:0:0]	0	LISTENING
Skype.exe	1216	TCP	0.0.0.0	10125	0.0.0.0	0	LISTENING
services.exe	720	TCP	0.0.0.0	1033	0.0.0.0	0	LISTENING
services.exe	720	TCPV6	[0:0:0:0:0:0:0:0]	1033	[0:0:0:0:0:0:0:0]	0	LISTENING
lsass.exe	728	TCP	0.0.0.0	1099	0.0.0.0	0	LISTENING
lsass.exe	728	TCPV6	[0:0:0:0:0:0:0:0]	1099	[0:0:0:0:0:0:0:0]	0	LISTENING
htcUPCTLoa...	7488	TCP	127.0.0.1	1285	0.0.0.0	0	LISTENING
htcUPCTLoa...	7488	TCP	127.0.0.1	11000	0.0.0.0	0	LISTENING
htcUPCTLoa...	7488	TCP	127.0.0.1	12000	0.0.0.0	0	LISTENING
dimngr.exe	1960	TCP	127.0.0.1	1031	0.0.0.0	0	LISTENING
dimngr.exe	1960	TCP	127.0.0.1	1030	0.0.0.0	0	LISTENING

At the bottom of the window, the following statistics are displayed:

Endpoints: 186	Established: 46	Listening: 55	Time Wait: 2	Close Wait: 5
----------------	-----------------	---------------	--------------	---------------



# 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

```

C:\> netsh http show urlacl
URL Reservations:

Reserved URL      : http://*:80/Temporary_Listen_Addresses/
User: \Everyone
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;WD)

Reserved URL      : https://*:5986/usman/
User: NT SERVICE\WinRM
Listen: Yes
Delegate: No
User: NT SERVICE\Wscntfrg
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;S-1-5-80-569256582-2953403351-2909559716-1301513147-412116970)(A;;GX;;;S-1-5-80-4059739203-877974739-1245631912-527174227-2996563517)

Reserved URL      : http://*:5985/usman/
User: NT SERVICE\WinRM
Listen: Yes
Delegate: No
User: NT SERVICE\Wscntfrg
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;S-1-5-80-569256582-2953403351-2909559716-1301513147-412116970)(A;;GX;;;S-1-5-80-4059739203-877974739-1245631912-527174227-2996563517)

Reserved URL      : http://*:47001/usman/
User: NT SERVICE\WinRM
Listen: Yes
Delegate: No
User: NT SERVICE\Wscntfrg
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;S-1-5-80-569256582-2953403351-2909559716-1301513147-412116970)(A;;GX;;;S-1-5-80-4059739203-877974739-1245631912-527174227-2996563517)

Reserved URL      : http://*:2869/
User: NT AUTHORITY\LOCAL SERVICE
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;LS)

Reserved URL      : http://*:5357/
User: BUILTIN\Users
Listen: Yes
Delegate: No
User: NT AUTHORITY\LOCAL SERVICE
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;BU)(A;;GX;;;LS)

Reserved URL      : https://*:5358/
User: BUILTIN\Users
Listen: Yes
Delegate: No
User: NT AUTHORITY\LOCAL SERVICE
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;BU)(A;;GX;;;LS)

Reserved URL      : https://*:443/sra_(BA195980-CD49-458b-9E23-C84EE0ADC075)/
User: NT SERVICE\SstpSvc
Listen: Yes
Delegate: Yes
User: BUILTIN\Administrators
Listen: No
Delegate: No
User: NT AUTHORITY\SYSTEM
Listen: Yes
Delegate: Yes
SDDL: D:(A;;GA;;;S-1-5-80-3435701886-799518250-3791383489-3228296122-2938884314)(A;;GR;;;BA)(A;;GA;;;SY)

Reserved URL      : http://*:10246/MDEServers/
User: NT AUTHORITY\Authenticated Users
Listen: Yes
Delegate: No
SDDL: D:(A;;GX;;;AU)

Reserved URL      : http://*:80/0131501b-d67f-491b-9a40-c4bf27bc4d4/
User: NT AUTHORITY\NETWORK SERVICE

```

# 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

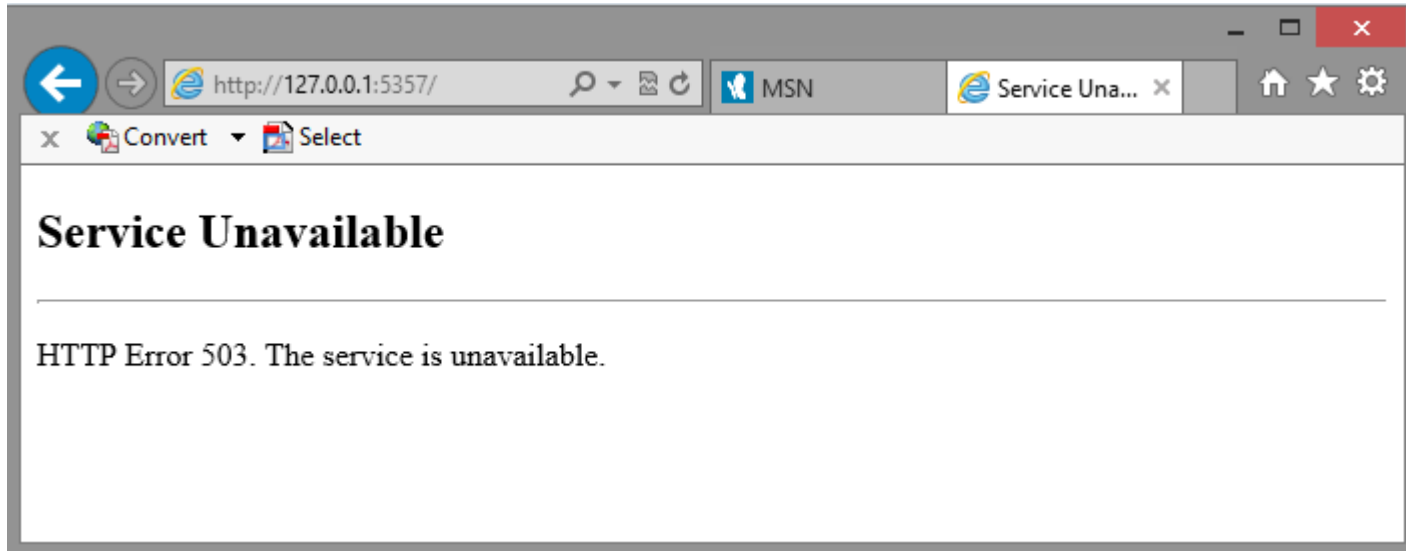
Listen: Yes

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

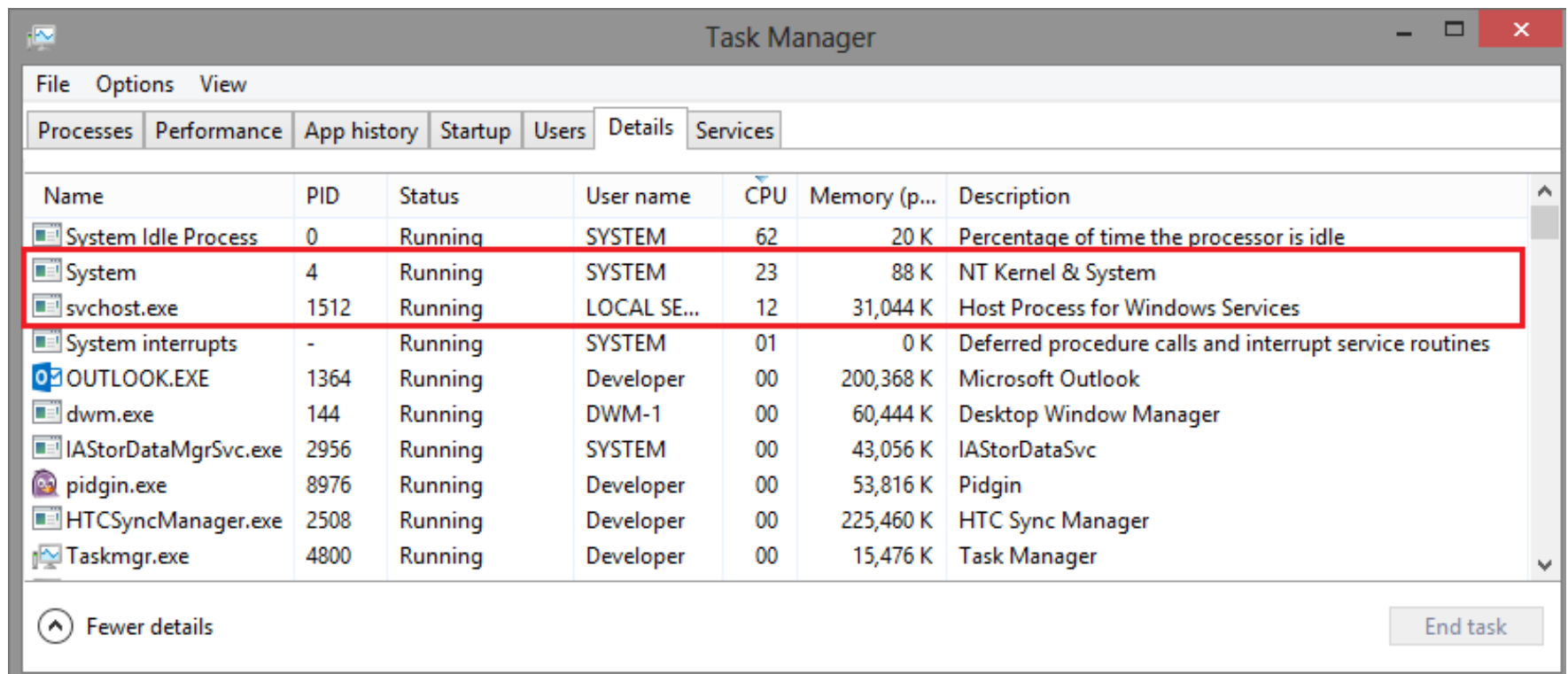
# Around MSDN in 80 Hours

- ▶ 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/
```



The screenshot shows the Windows Task Manager window with the 'Details' tab selected. The process list is as follows:

Name	PID	Status	User name	CPU	Memory (p...	Description
System Idle Process	0	Running	SYSTEM	62	20 K	Percentage of time the processor is idle
System	4	Running	SYSTEM	23	88 K	NT Kernel & System
svchost.exe	1512	Running	LOCAL SE...	12	31,044 K	Host Process for Windows Services
System interrupts	-	Running	SYSTEM	01	0 K	Deferred procedure calls and interrupt service routines
OUTLOOK.EXE	1364	Running	Developer	00	200,368 K	Microsoft Outlook
dwm.exe	144	Running	DWM-1	00	60,444 K	Desktop Window Manager
IAStorDataMgrSvc.exe	2956	Running	SYSTEM	00	43,056 K	IAStorDataSvc
pidgin.exe	8976	Running	Developer	00	53,816 K	Pidgin
HTCSyncManager.exe	2508	Running	Developer	00	225,460 K	HTC Sync Manager
Taskmgr.exe	4800	Running	Developer	00	15,476 K	Task Manager

# IDA Pro + HTTP.sys

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

```
IDA - C:\Users\Developer\Downloads\http.i64 (http.sys)
File Edit Jump Search View Debugger Options Windows Help
No debugger

Function name
RtlStringCchCatExW
RtlStringCchCatExW
RtlStringCchCopyExW
RtlUnicodeStringCatString
RtlUnicodeStringCbCopyStringN
RtlUnicodeStringCopyString
SOCKADDR_SIZE
StringCbLengthW
StringCchCopyW
StringCchLengthW
StringCchPrintFA
StringCchPrintExA
StringCchPrintExW
StringTimeToSystemTime
StringToHostAddressW
UIAcceptConnection
UIAcceptEncodingHeaderHandler
UIAcceptHeaderHandler
UIAccessCheck
UIActivateLogFile
UIAddCacheEntry
UIAddFragmentToCache
UIAddFragmentToCacheLocl
UIAddSite
UIAddSiteToEndpointList
UIAddSslBinding
UIAddToAuthCache
UIAddToHashTable
UIAddUrlToConfigGroup
UIAddUrlToUrlGroupLocl
UIAddUrlToUrlGroupLocl

mov     edx, 1Bh
mov     r8, [r8+8]
mov     rcx, cs:WPP_GLOBAL_Control
mov     rcx, [rcx+18h]
call    WPP_SF_Sq
jmp     loc_82488

-----
82584:                ; CODE XREF: UIAddUrlToConfigGroup+797j
mov     eax, dword ptr cs:UIWppEnabledFlags+4
test    al, 4
jz     short loc_825AE
mov     edx, 1Ch
mov     r9, [rbx+8]
lea     r8, WPP_4d4a90130efbabfc94463f7323874c72_Traceguids
mov     rcx, cs:WPP_GLOBAL_Control
mov     rcx, [rcx+18h]
call    WPP_SF_S

825AE:                ; CODE XREF: UIAddUrlToConfigGroup+1DC7j
jmp     loc_8268D

-----
825B9:                ; CODE XREF: UIAddUrlToConfigGroup+877j
mov     edx, 1Dh
mov     r9, [rsp+128h+P]
lea     r8, WPP_4d4a90130efbabfc94463f7323874c72_Traceguids
mov     rcx, cs:WPP_GLOBAL_Control
mov     rcx, [rcx+18h]
call    WPP_SF_S
jmp     loc_8243D

-----
825D9:                ; CODE XREF: UIAddUrlToConfigGroup+1307j
movzx   eax, [rsp+128h+var_70]

0006BF97 0000000000082597: UIAddUrlToConfigGroup+1E7
```



# 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	<b>AddUrl</b>	<b>31</b>	<b>"http://*:2869/upnp/eventing/"</b>	<b>0x0</b>		
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"

# 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 ----->>>>
```

# 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

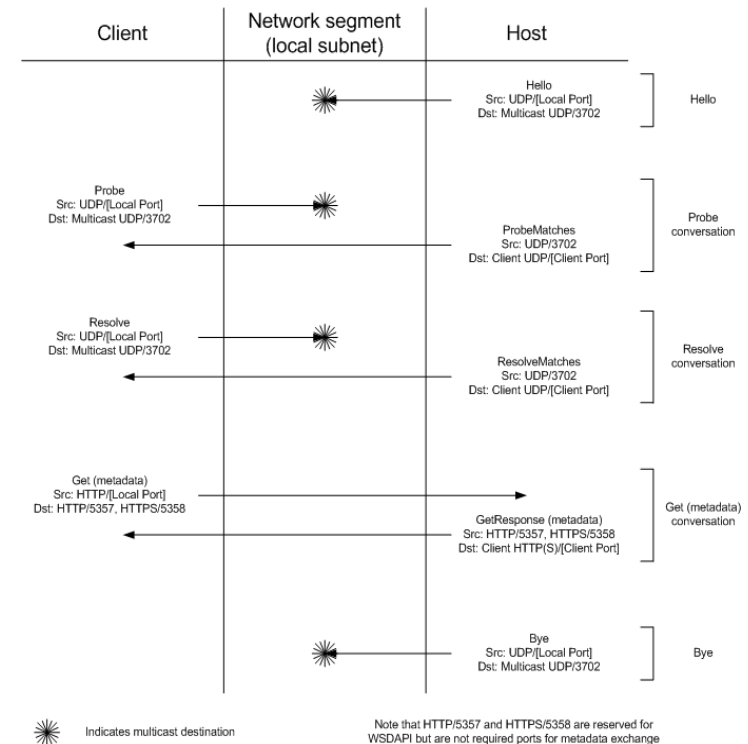
# Web Services for Devices

- ▶ 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



# 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!

# 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/22b669b37e  
a6746d29c989d8bd5578f019f0e592.HomeGroupClassifier_HomeGroup_Invitation_ID</wsdp:Servic  
eld>
```

```
<pub:Resource>\?xml\sversion="1.0"\sencoding="UTF-  
16"?&gt;\IHOMEGROUP_RECORD&gt;\IINVITATION&gt;\alt;PEERINVITATION\sVERSION="1.1"\agt;\alt;CLOUDNAME\agt;LinkLocal_ff00::%10/8\alt  
;/CLOUDNAME\agt;\alt;SCOPE\agt;LINKLOCAL\alt;/SCOPE\agt;\IGUIDNAME&gt;{C81BDCF2-FA30-4EB4-ABD9-B7C7693E3BB3}\I/GUIDNAME
```

```
&gt;\IOWNER&gt;Joan\I/OWNER
```

```
&gt;\IOWNERID&gt;22b669b37ea6746d29c989d8bd5578f019f0e592.HomeGroupClassifier\I/OWNERID
```

```
&gt;\IOWNERMACHINENAME&gt;JOAN-PC\I/OWNERMACHINENAME
```

```
&gt;\ILASTCHANGED&gt;129828926058689726\I/LASTCHANGED
```

```
&gt;\IHOMEGROUPSIZE&gt;1\I/HOMEGROUPSIZE
```

```
&gt;\IADDRESS&gt;[fe80::f9e4:dbc0:b1f7:bb84%10]:3587\I/ADDRESS
```

```
&gt;\IDIGITALHASH&gt;-----BEGIN\sCERTIFICATE-....
```

# 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>

Thanks!

Q & A