



Lifting the Fog of War

Monitoring, Identifying and Mitigating MS-RPC Based Threats

whoami

Stiv Kupchik

Security Researcher

Akamai

@kupsul 

Background in DFIR and Windows internals

Agenda

- ❏ MS-RPC introduction and overview
- ❏ ETW introduction and overview
- ❏ Using ETW to detect MS-RPC based attacks
- ❏ Supplementing defense with RPC Filters

It's
everywhere
in the
network



Command and Scripting Interpreter	Account Manipulation	Abuse Elevation Control Mechanism	Abuse Elevation Control Mechanism	Adversary-in-the-Middle	Account Discovery	Exploitation of Remote Services	Adversary-in-the-Middle	Application Layer Protocol	Automated Exfiltration	Account Access Removal
Exploitation for Client Execution	BITS Jobs	Access Token Manipulation	Access Token Manipulation	Brute Force	Application Window Discovery	Internal Spearphishing	Archive Collected Data	Communication Through Removable Media	Data Transfer Size Limits	Data Destruction
Inter-Process Communication	Boot or Logon Autostart Execution	Boot or Logon Autostart Execution	BITS Jobs	Credentials from Password Stores	Browser Information Discovery	Lateral Tool Transfer	Audio Capture	Data Encoding	Exfiltration Over Alternative Protocol	Data Encrypted for Impact
Native API	Boot or Logon Initialization Scripts	Boot or Logon Initialization Scripts	Debugger Evasion	Exploitation for Credential Access	Debugger Evasion	Remote Service Session Hijacking	Automated Collection	Data Obfuscation	Exfiltration Over C2 Channel	Data Manipulation
Scheduled Task/job	Browser Extensions	Create or Modify System Process	Doofuscate/Decode Files or Information	Forced Authentication	Device Driver Discovery	Remote Services	Browser Session Hijacking	Dynamic Resolution	Exfiltration Over Other Network Medium	Defacement
Shared Modules	Compromise Client Software Binary	Domain Policy Modification	Direct Volume Access	Forge Web Credentials	Domain Trust Discovery	Replication Through Removable Media	Clipboard Data	Encrypted Channel	Exfiltration Over Physical Medium	Disk Wipe
Software Deployment Tools	Create Account	Escape to Host	Domain Policy Modification	Input Capture	File and Directory Discovery	Software Deployment Tools	Data from Configuration Repository	Fallback Channels	Exfiltration Over Web Service	Endpoint Denial of Service
System Services	Create or Modify System Process	Event Triggered Execution	Execution Guardrails	Modify Authentication Process	Group Policy Discovery	Taint Shared Content	Data from Information Repositories	Ingress Tool Transfer	Scheduled Transfer	Firmware Corruption
User Execution	Event Triggered Execution	Exploitation for Privilege Escalation	Exploitation for Defense Evasion	Multi-Factor Authentication Interception	Network Service Discovery	Use Alternate Authentication Material	Data from Local System	Multi-Stage Channels		Inhibit System Recovery
Windows Management Instrumentation	External Remote Services	Hijack Execution Flow	File and Directory Permissions Modification	Multi-Factor Authentication Request Generation	Network Share Discovery		Data from Network Shared Drive	Non-Application Layer Protocol		Network Denial of Service
	Hijack Execution Flow	Process Injection	Hide Artifacts	Network Sniffing	Network Sniffing		Data from Removable Media	Non-Standard Port		Resource Hijacking
	Modify Authentication Process	Scheduled Task/job	Hijack Execution Flow	OS Credential Dumping	Password Policy Discovery		Data Staged	Protocol Tunneling		Service Stop
	Office Application Startup	Valid Accounts	Impair Defenses	Steal or Forge Authentication Certificates	Peripheral Device Discovery		Email Collection	Proxy		System Shutdown/Reboot
	Pre-OS Boot		Indicator Removal	Steal or Forge Kerberos Tickets	Permission Groups Discovery		Input Capture	Remote Access Software		
	Scheduled Task/job		Indirect Command Execution	Steal Web Session Cookie	Process Discovery		Screen Capture	Traffic Signaling		
	Server Software Component		Masquerading	Unsecured Credentials	Query Registry		Video Capture	Web Service		
	Traffic Signaling		Modify Authentication Process		Remote System Discovery					
	Valid Accounts		Modify Registry		Software Discovery					
			Modify System Image		System Information Discovery					
			Network Boundary Bridging		System Location Discovery					
			Obfuscated Files or Information		System Network Configuration Discovery					
			Pre-OS Boot		System Network Connections Discovery					
			Process Injection		System Owner/User Discovery					
			Reflective Code Loading		System Service Discovery					
			Rogue Domain Controller		System Time Discovery					
			Rootkit		Virtualization/Sandbox Evasion					

... and is involved in many parts of the attack matrix

RPC Attacks are Hard to Detect

- RPC is another layer of encapsulation that you need to peel
 - Deep packet inspection is expensive, usually only connection metadata

RPC Attacks are Hard to Detect

- RPC is another layer of encapsulation that you need to peel
 - Deep packet inspection is expensive, usually only connection metadata
- Most RPC servers are running under svchost or protected processes
 - Difficult to match traffic to process or RPC server

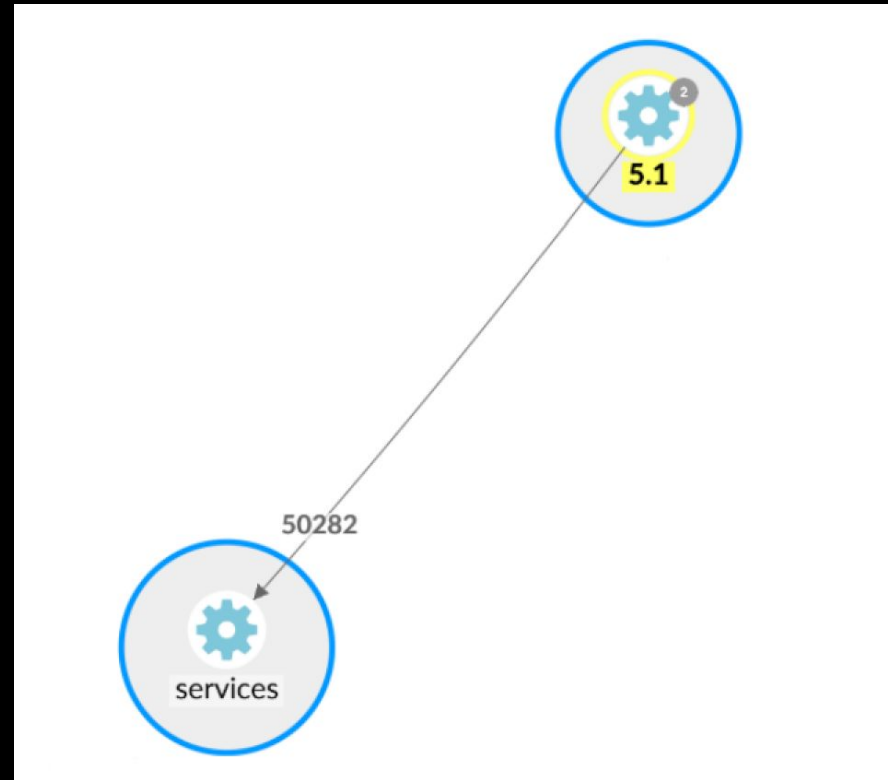
RPC Attacks are Hard to Detect

- RPC is another layer of encapsulation that you need to peel
 - Deep packet inspection is expensive, usually only connection metadata
- Most RPC servers are running under svchost or protected processes
 - Difficult to match traffic to process or RPC server
- RPC traffic can occur over ephemeral ports
 - Can't create FW rules in advance

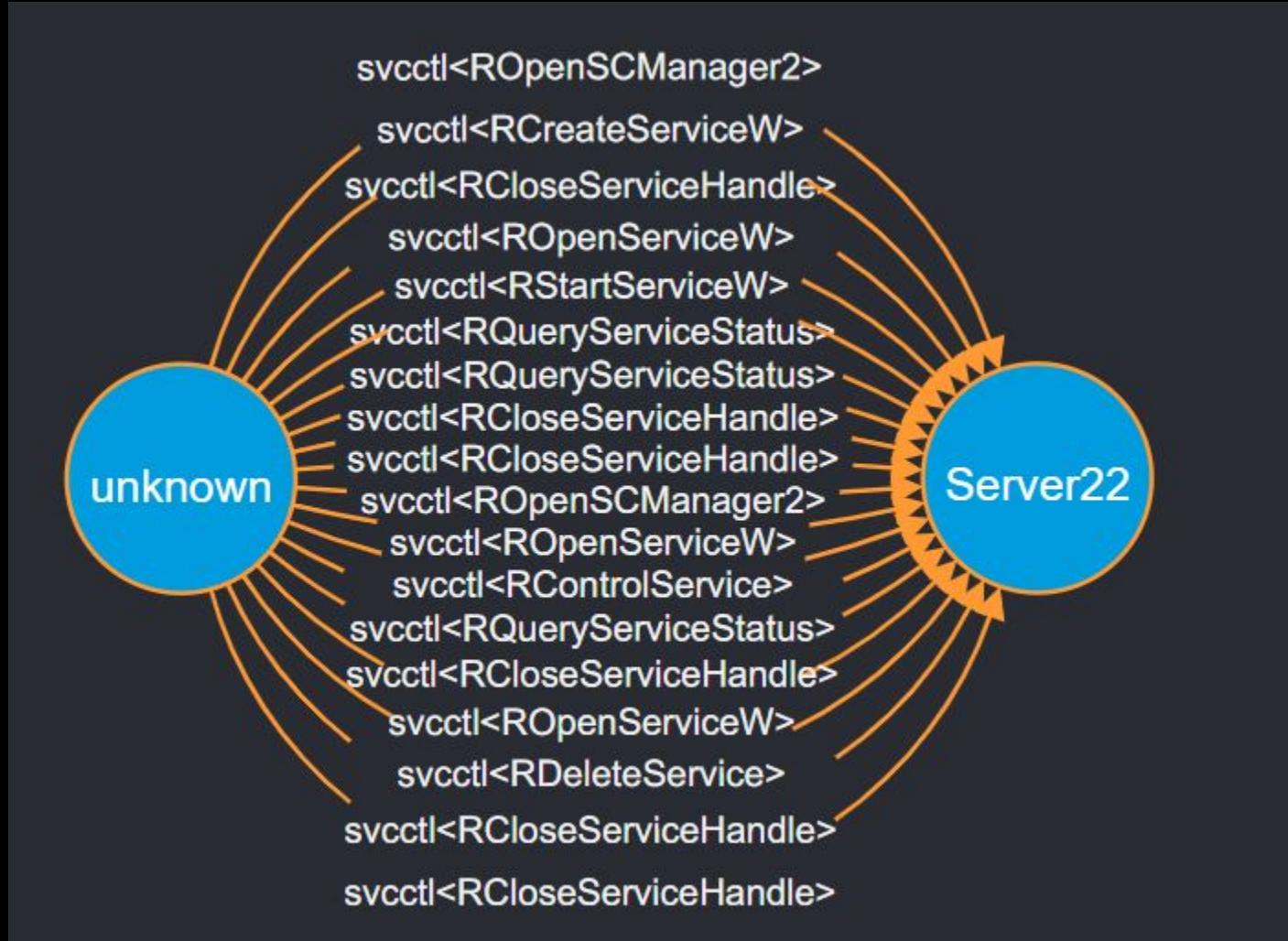
Not Many (documented) Defense Options

- RPC Filters in the Windows Firewall
- ETW monitors aimed at researchers
 - [RpcMon](#) by CyberArk
 - [RpcInvestigator](#) by TrailOfBits
- [RPC Firewall](#) by ZeroNetworks
 - requires process injection & hooks

RPC without visibility



RPC with visibility



MS-RPC Overview

Terminology you'll soon master

- Interface
- {M}IDL
- Transport
- Endpoint
- Binding

The RPC Client-Server Model

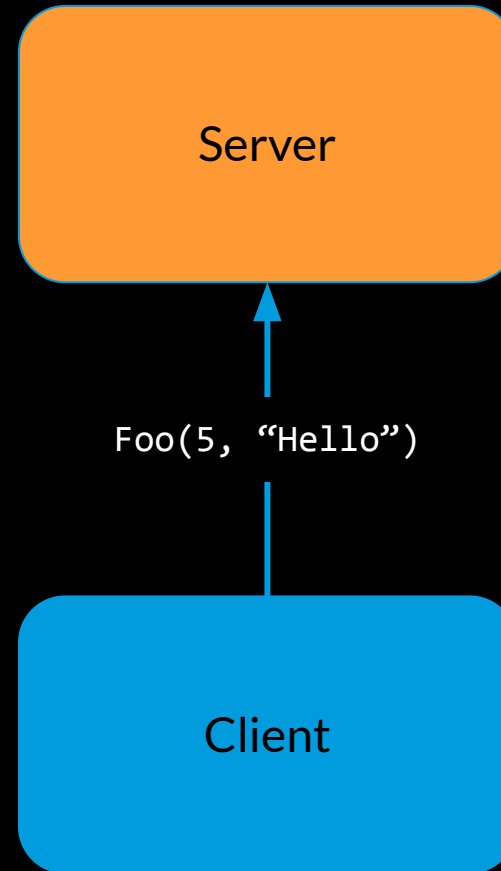


```
graph TD; Server[Server] --- Client[Client];
```

Server

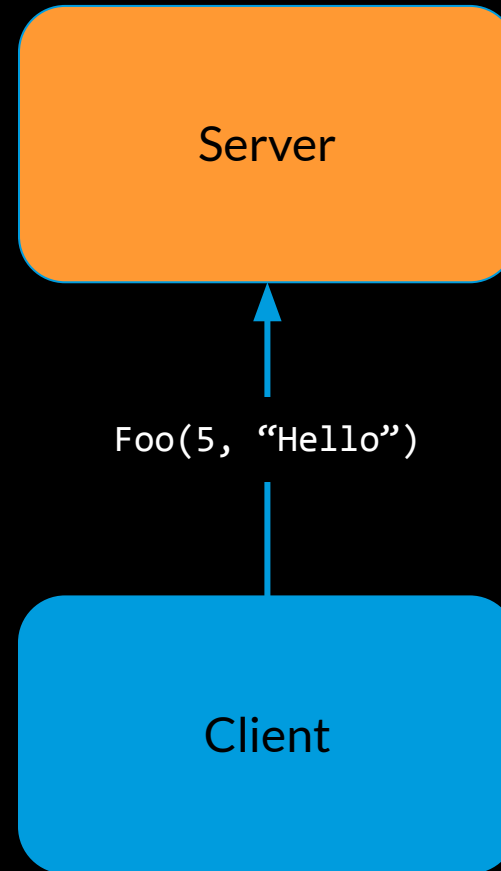
Client

The RPC Client-Server Model

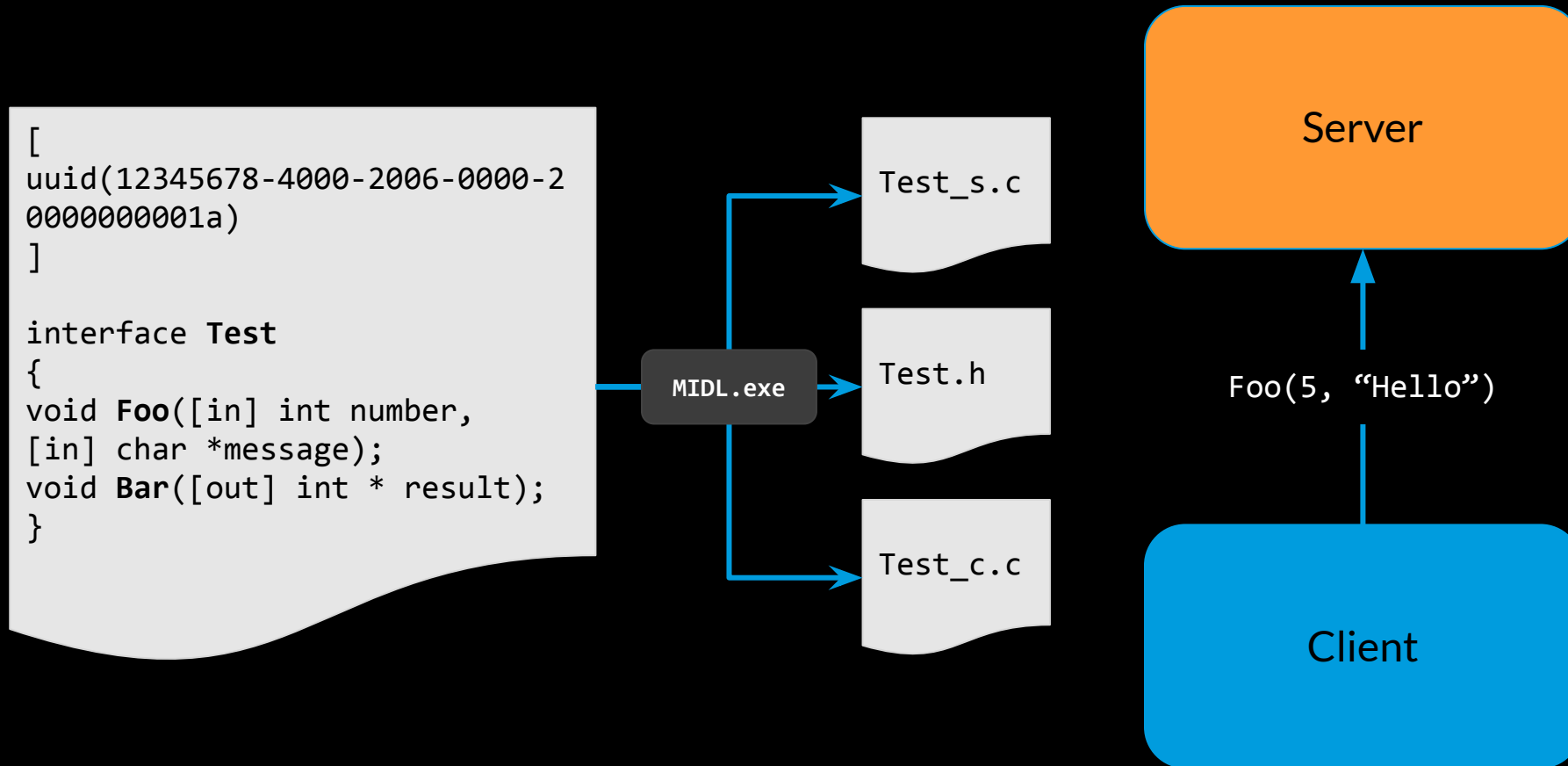


The RPC Client-Server Model

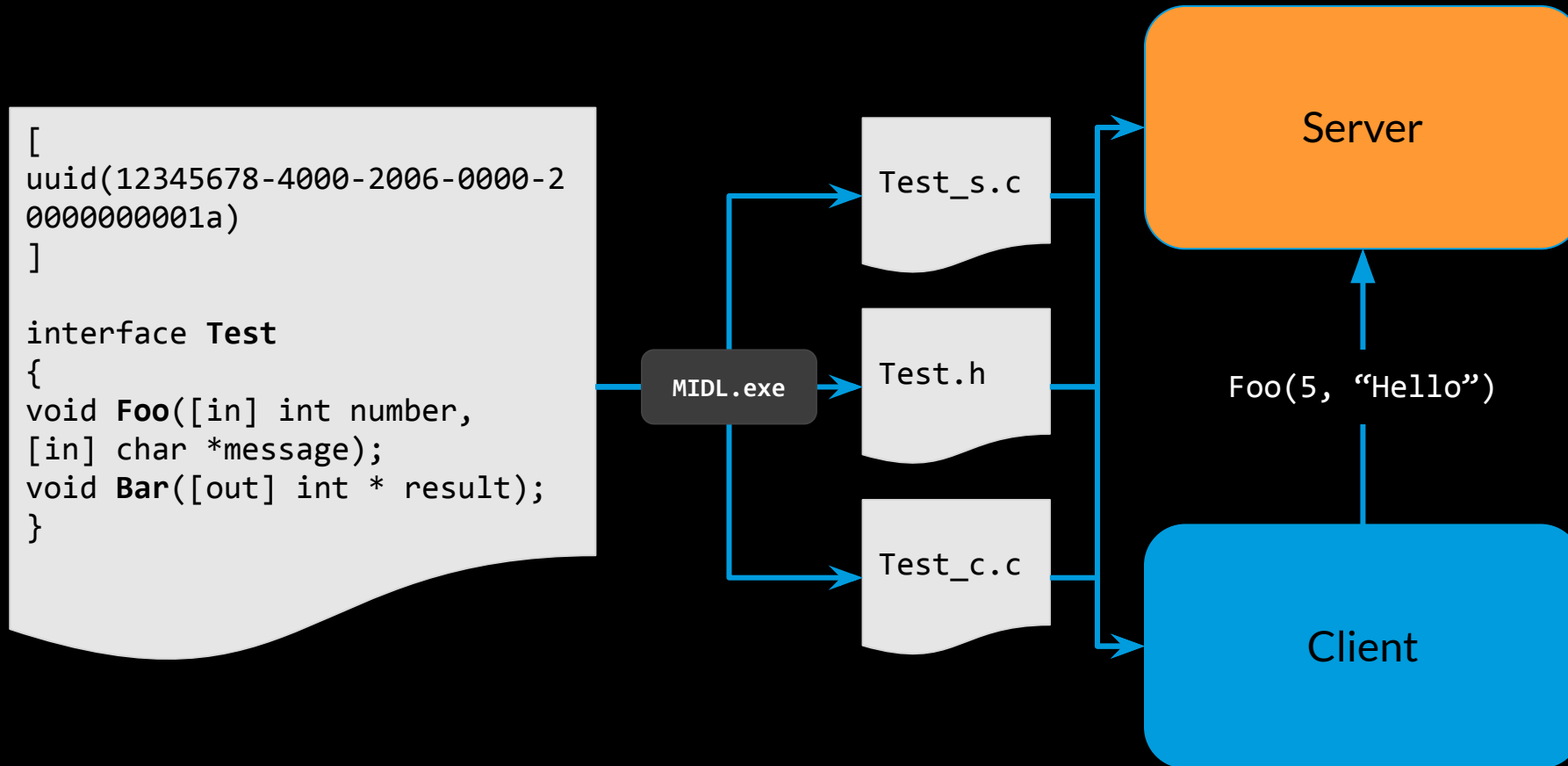
```
[  
  uuid(12345678-4000-2006-0000-2  
  0000000001a)  
]  
  
interface Test  
{  
  void Foo([in] int number,  
  [in] char *message);  
  void Bar([out] int * result);  
}
```



The RPC Client-Server Model



The RPC Client-Server Model

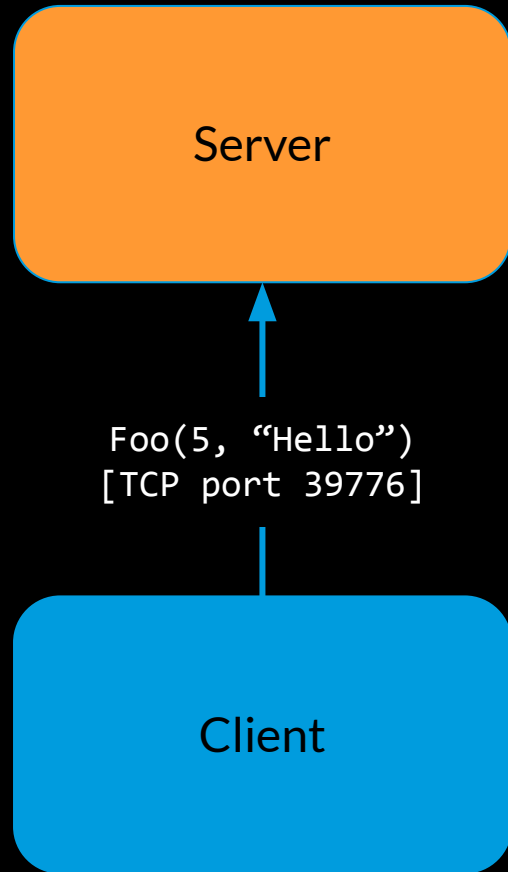


Endpoints

- The server registers an *endpoint* using a certain *transport*

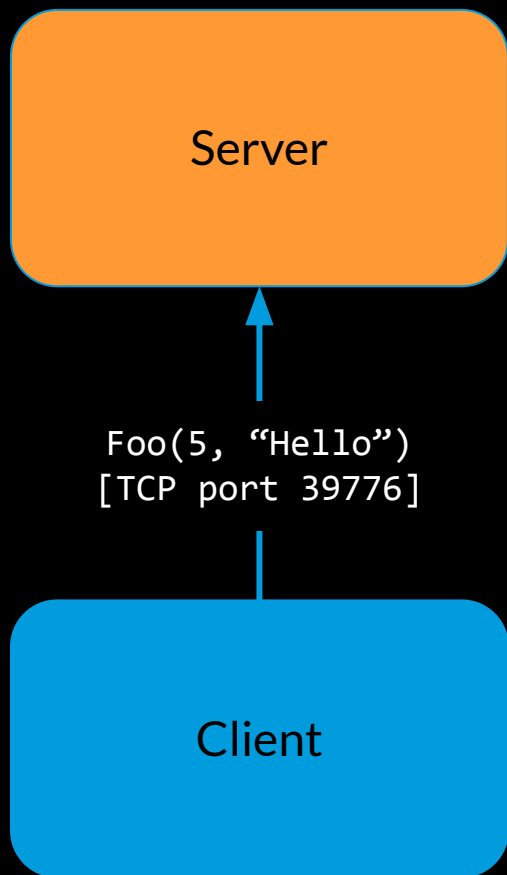
Transports	Protocol Sequence	Endpoints
TCP	ncacn_ip_tcp	<port number>
Named pipe	ncacn_np	<pipe name>
UDP	ncadg_ip_udp	<port number>
ALPC	ncalrpc	<ALPC port>
HTTP	ncacn_http	<hostname>
Hyper-V socket	ncacn_hvsocket	<UUID>

Well-Known Endpoints

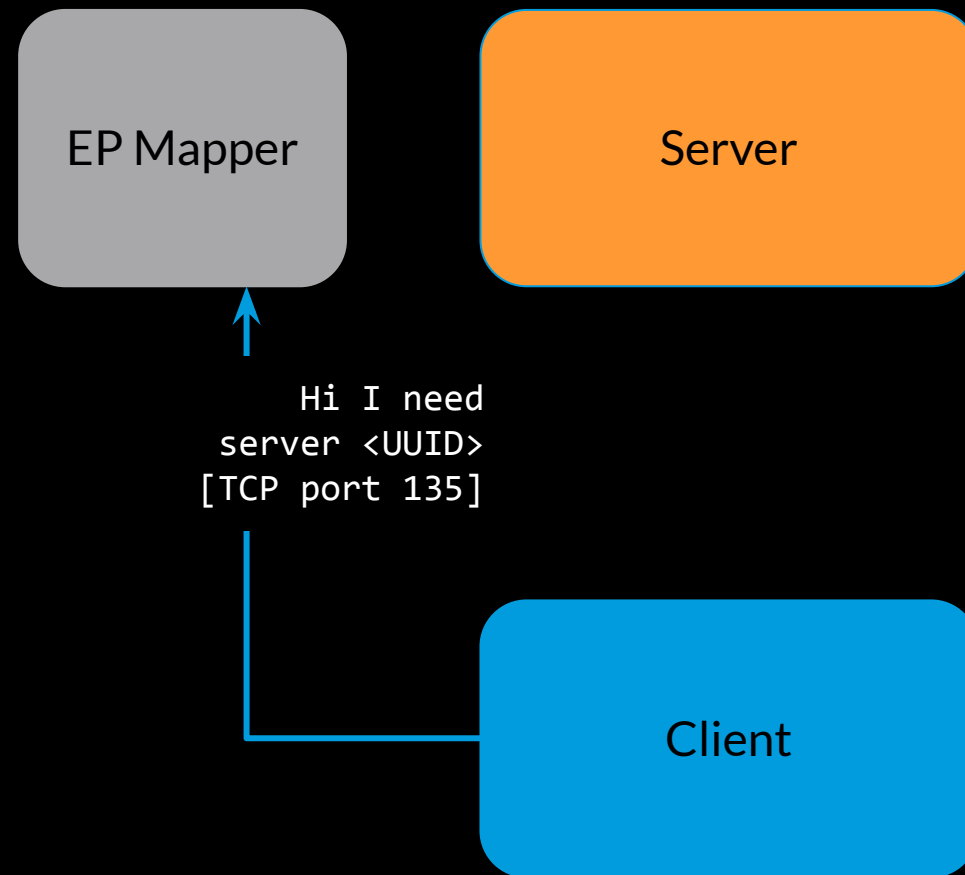


Dynamic Endpoints

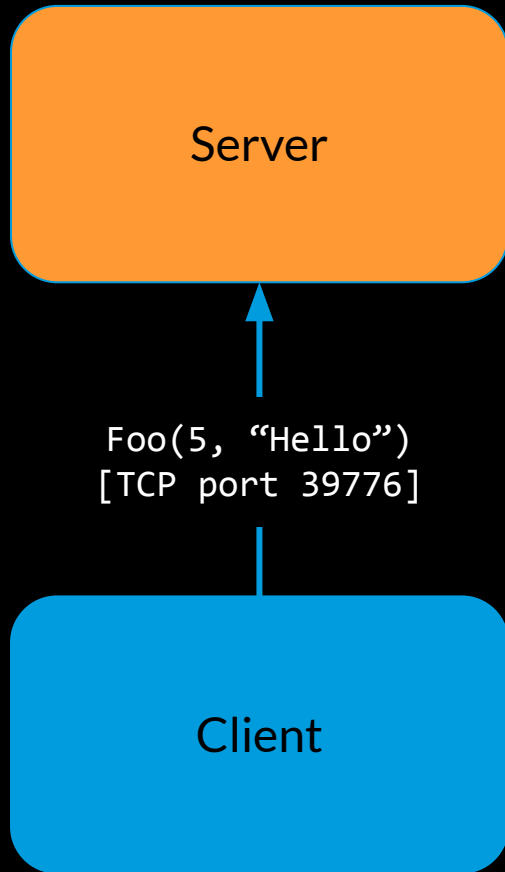
Well-Known Endpoints



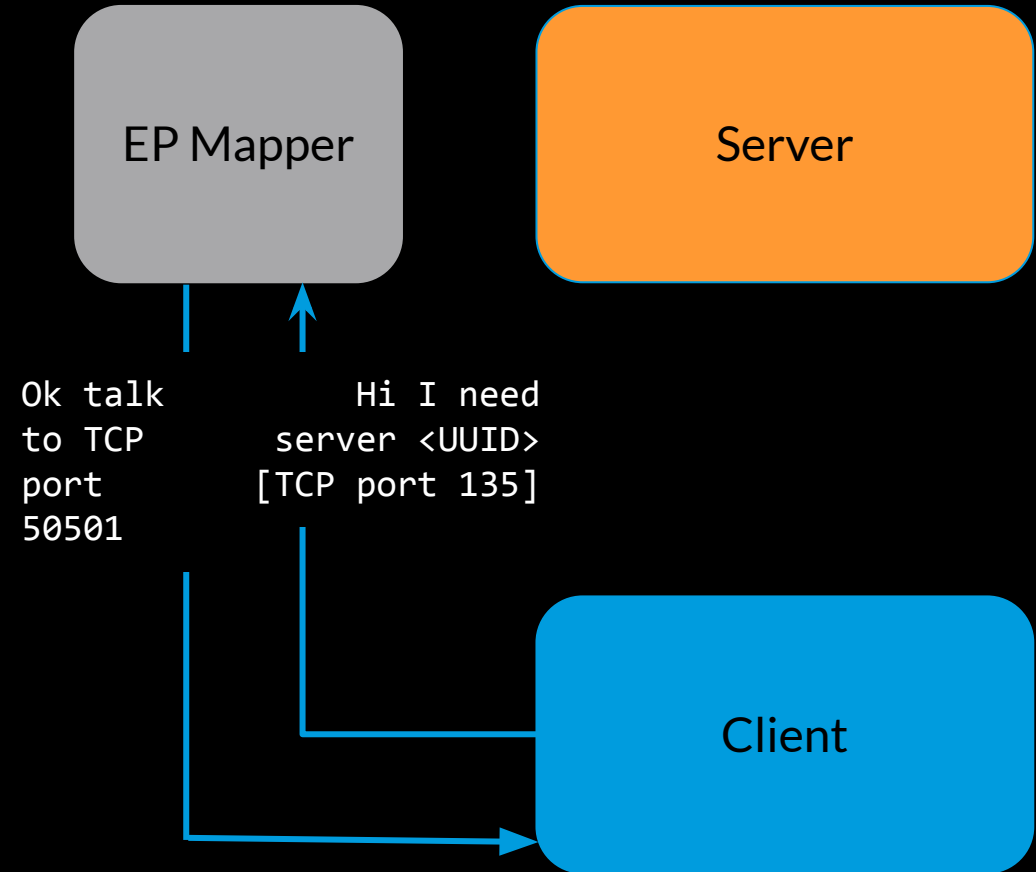
Dynamic Endpoints



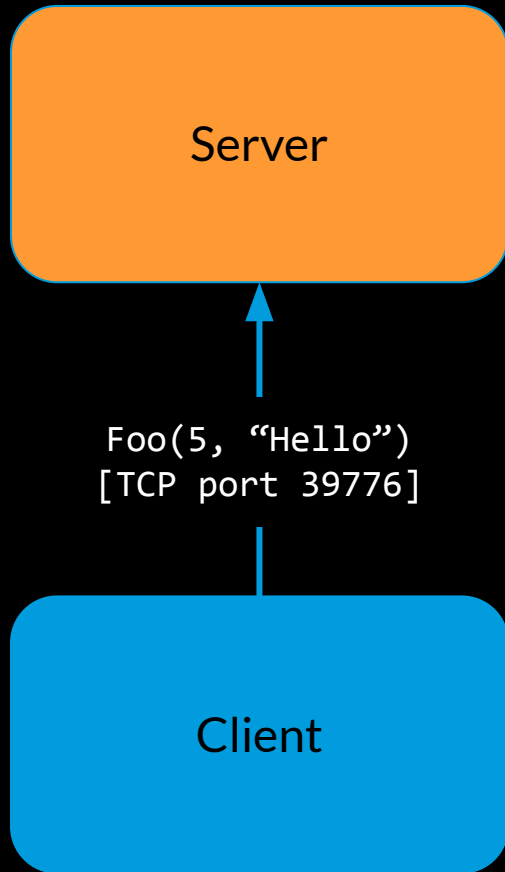
Well-Known Endpoints



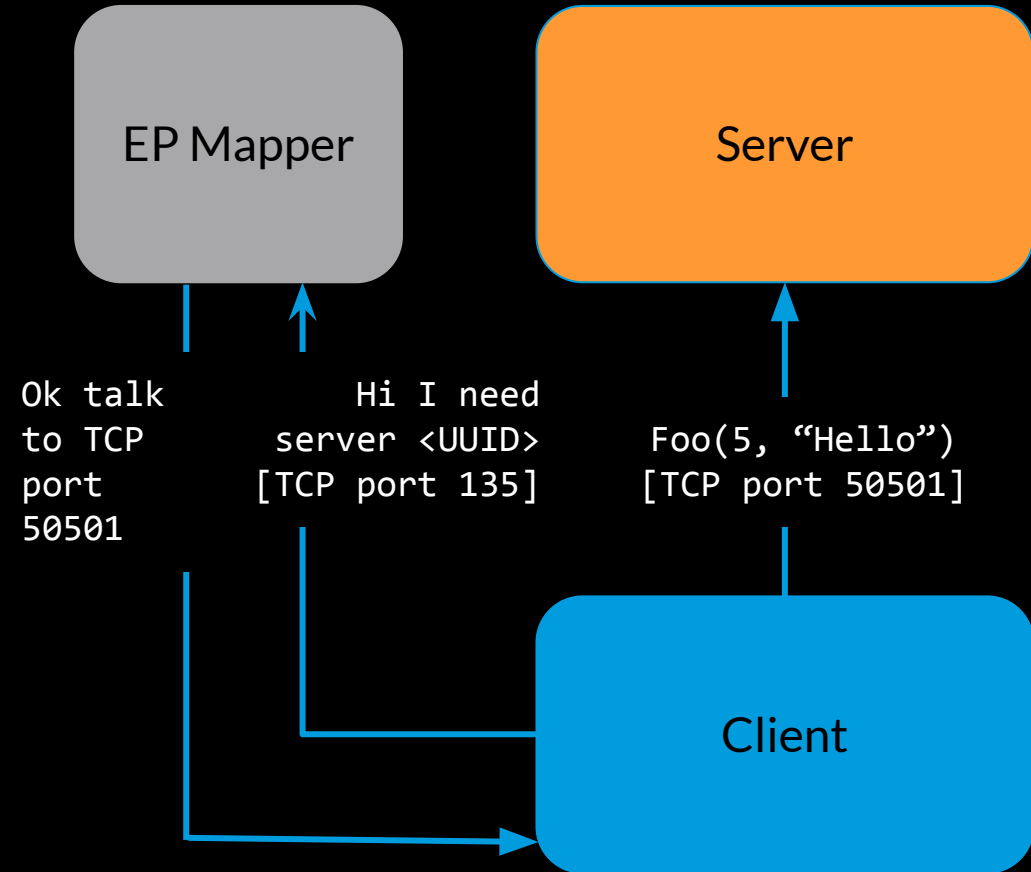
Dynamic Endpoints



Well-Known Endpoints



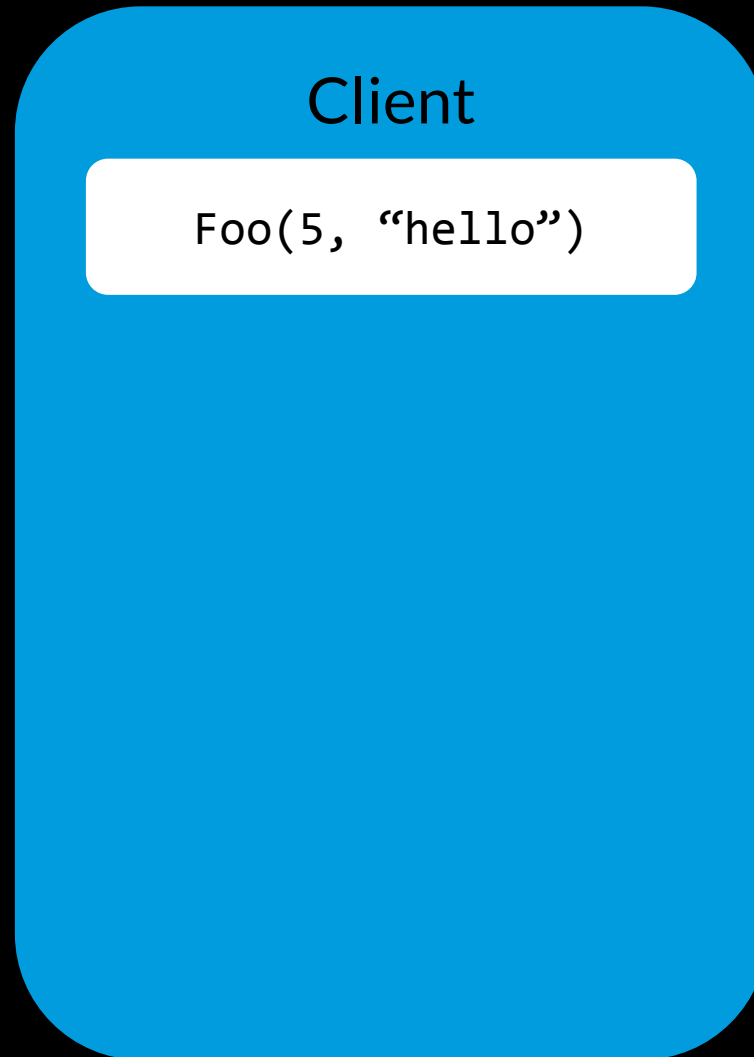
Dynamic Endpoints



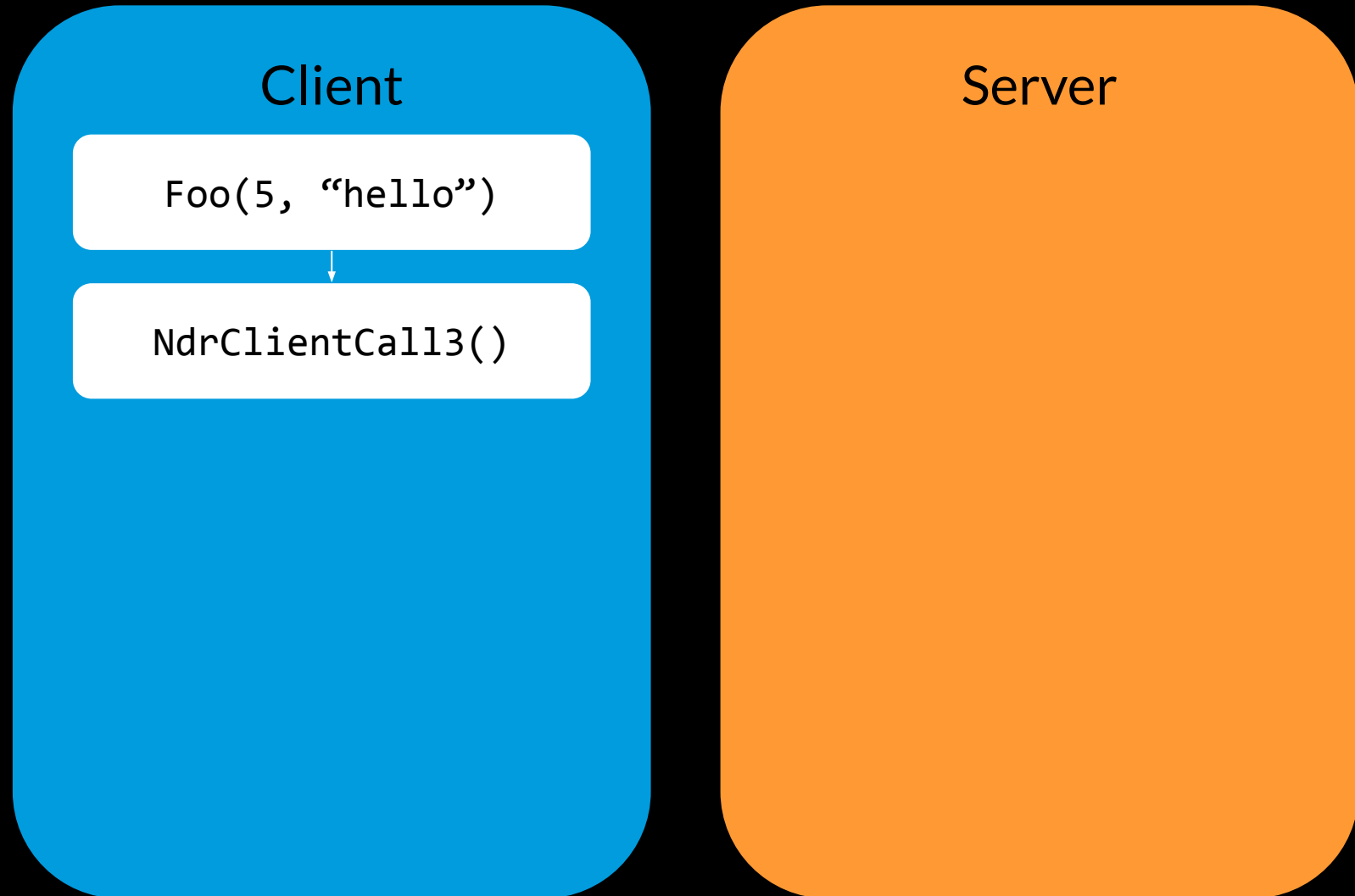
Binding

- The representation of a session between a client and a server
 - Practically, a handle
 - Client and server can manipulate binding data using designated functions
 - Used for authentication (among other things)

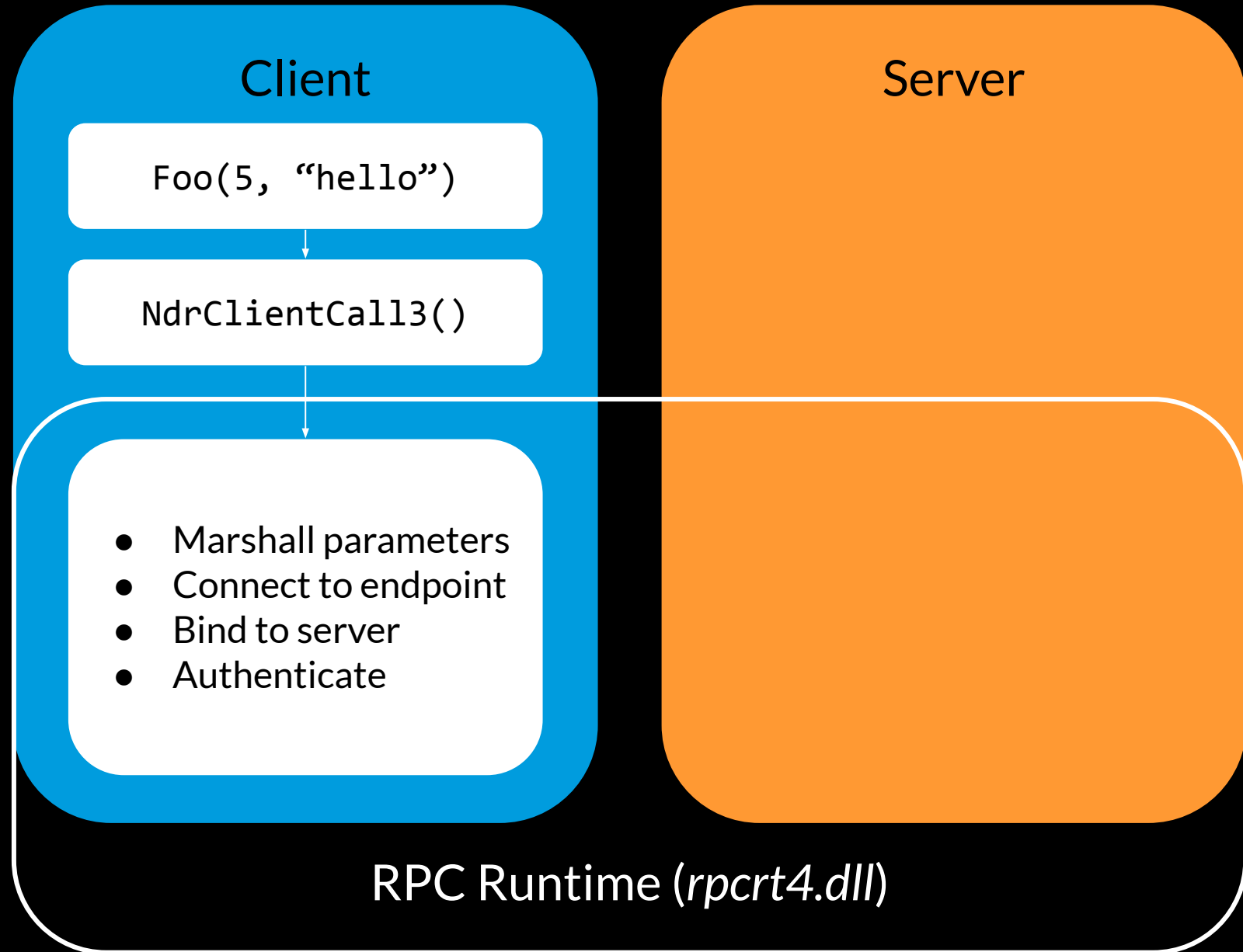
An RPC Call's Flow



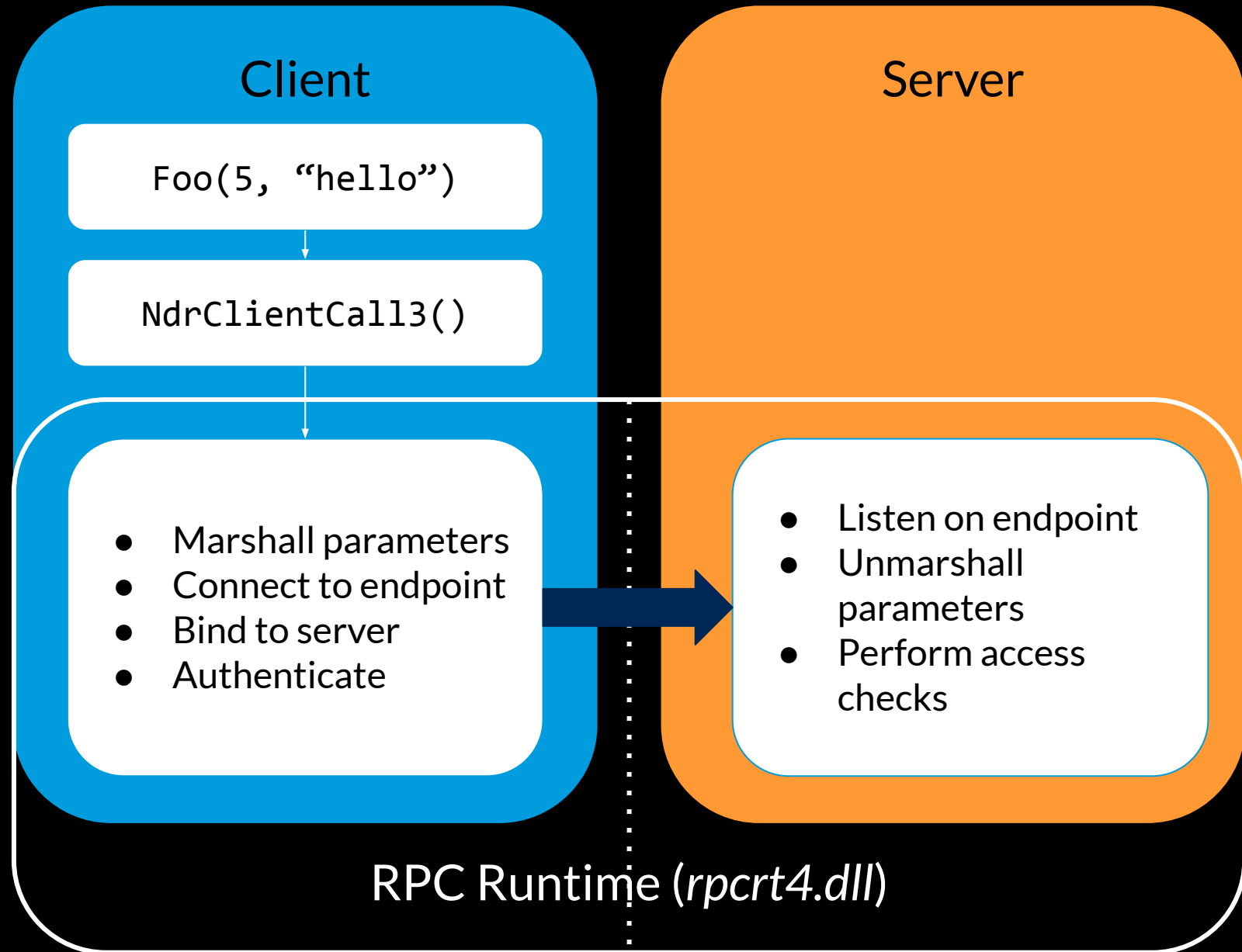
An RPC Call's Flow



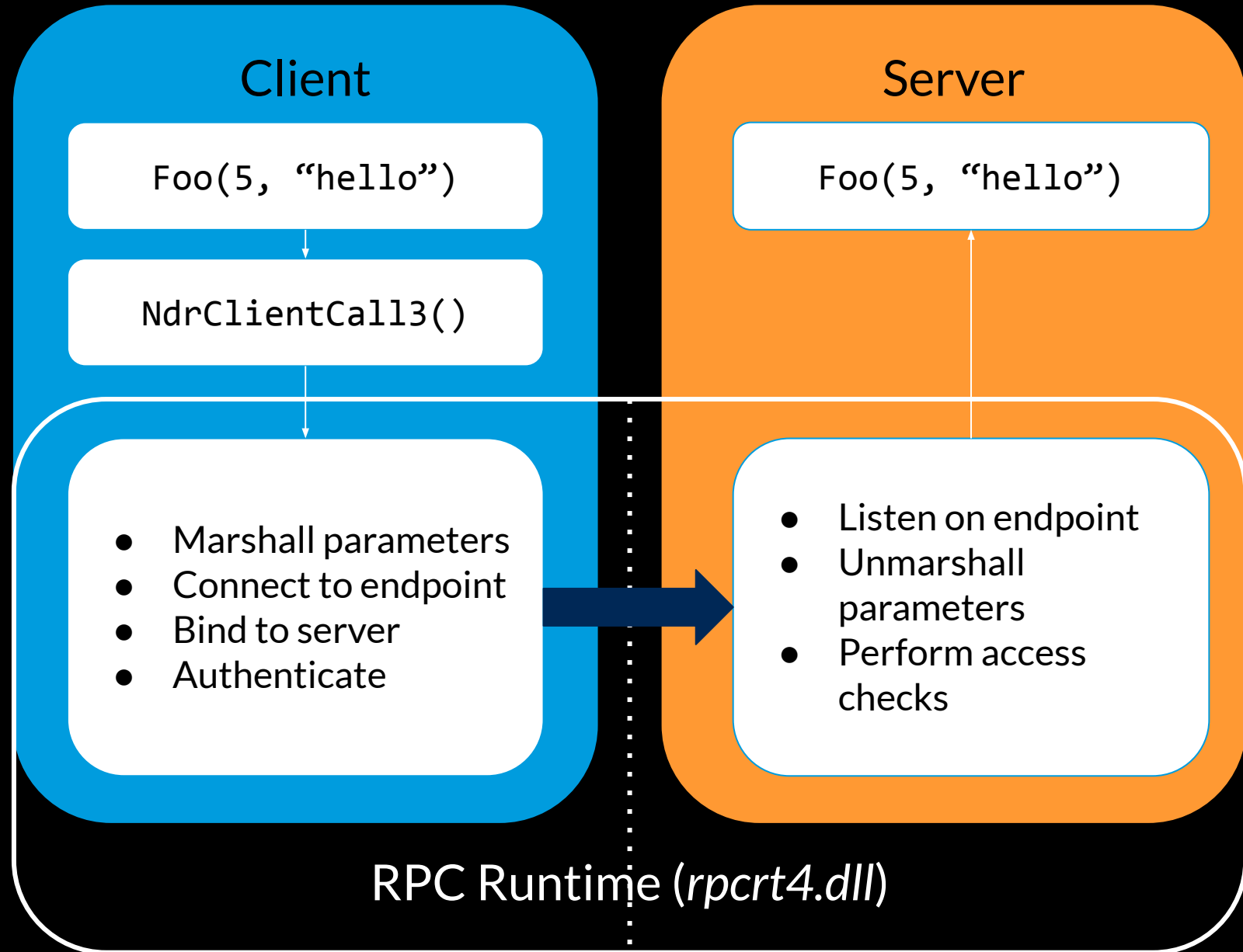
An RPC Call's Flow



An RPC Call's Flow



An RPC Call's Flow



Zooming In

IDL:

```
void Foo([in] int number,  
         [in] char* message);
```

Client

Foo(5, "hello")



NdrClientCall3()

Zooming In

IDL:

```
void Foo([in] int number,  
         [in] char* message);
```

MIDL.exe



Test_c.c:

```
void Foo(  
    handle_t IDL_handle,  
    int number,  
    unsigned char *message) {  
  
    NdrClientCall3(  
        (PMIDL_STUBLESS_PROXY_INFO  
        )&Test_ProxyInfo, 0, 0,  
        IDL_handle, number, message);  
    }
```

Client

Foo(5, "hello")

NdrClientCall3()



Zooming In

IDL:

```
void Foo([in] int number,  
         [in] char* message);
```

MIDL.exe



Test_c.c:

```
void Foo(  
    handle_t IDL_handle,  
    int number,  
    unsigned char *message) {
```

Client


Foo(5, "hello")

NdrClientCall3()



```
NdrClientCall3(  
    (PMIDL_STUBLESS_PROXY_INFO  
    )&Test_ProxyInfo, 0, 0,  
    IDL_handle, number, message);  
}
```

Opnum



Quick Recap

- Interface – describes server functionality [UUID]
- Transport – the communication medium [protocol sequence]
- Endpoint – destination to connect to [port, pipe name, etc.]
- Binding – represents a client-server session [binding handle]

RPC Visibility

#define ETW

- Event Tracing for Windows ([ETW](#)) is a built-in tracing and logging mechanism

#define ETW

- Event Tracing for Windows ([ETW](#)) is a built-in tracing and logging mechanism
- Provider-consumer model
 - Providers define a [schema](#) for their events so consumers can parse them
 - Both providers and consumers need to register with the ETW

#define ETW

- Event Tracing for Windows ([ETW](#)) is a built-in tracing and logging mechanism
- Provider-consumer model
 - Providers define a [schema](#) for their events so consumers can parse them
 - Both providers and consumers need to register with the ETW
- UM logic implemented in ntdll, transfers control to kernel

Provider <> Consumer

Application process

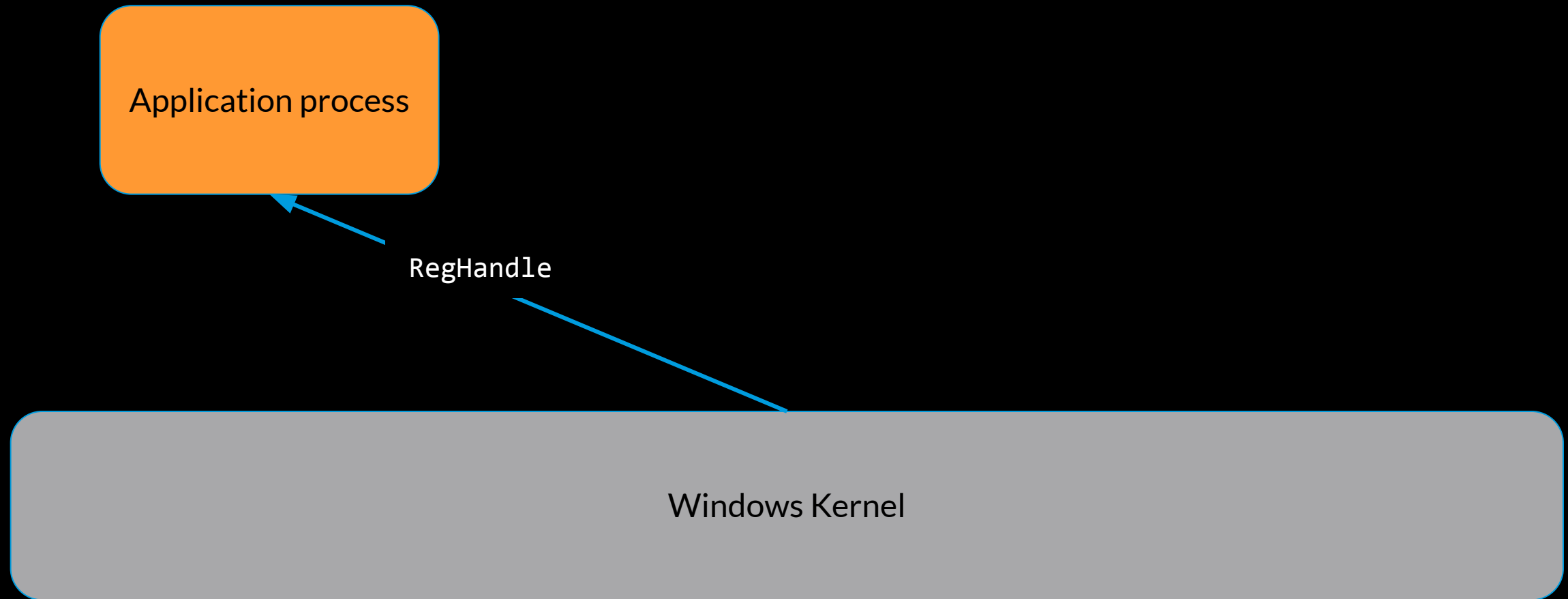
```
graph TD; A[Application process] -- "EtwEventRegister(DEADBEEF-BAAD-DEAD-BEEF-BAADF00D)" --> B[Windows Kernel];
```

The diagram illustrates the interaction between an application process and the Windows kernel. An orange rounded rectangle at the top is labeled 'Application process'. A blue arrow points from the bottom of this rectangle to the text 'EtwEventRegister(DEADBEEF-BAAD-DEAD-BEEF-BAADF00D)'. Another blue arrow points from this text to a large grey rounded rectangle at the bottom labeled 'Windows Kernel'.

EtwEventRegister(DEADBEEF-BAAD-DEAD-BEEF-BAADF00D)

Windows Kernel

Provider <> Consumer



Provider <> Consumer

Application process

```
graph TD; A[Application process] -- "EventWrite(RegHandle, <event_data>)" --> B[Windows Kernel];
```

The diagram illustrates the interaction between an application process and the Windows kernel. An orange rounded rectangle labeled 'Application process' is positioned at the top left. A blue arrow points from the bottom of this rectangle to a light blue rounded rectangle labeled 'Windows Kernel' at the bottom. The arrow is labeled with the function call 'EventWrite(RegHandle, <event_data>)'.

EventWrite(RegHandle, <event_data>)

Windows Kernel

Provider <> Consumer

Application process



<event_data>

Windows Kernel

Context is Important

- Since events need to hop the kernel boundary, it is a waste to send them when no consumers are tracing events

Context is Important

- Since events need to hop the kernel boundary, it is a waste to send them when no consumers are tracing events
- Providers can define a callback to the kernel, to be notified about the state of the provider

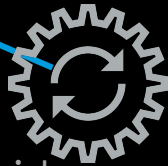
Context is Important

- Since events need to hop the kernel boundary, it is a waste to send them when no consumers are tracing events
- Providers can define a callback to the kernel, to be notified about the state of the provider
- If there are no consumers, the provider is considered disabled and can skip event writing

```
void Penablecallback(  
    [in] LPCGUID SourceId,  
    [in] ULONG IsEnabled,  
    [in] UCHAR Level,  
    [in] ULONGLONG MatchAnyKeyword,  
    ULONGLONG MatchAllKeyword,  
    [in, optional] PEVENT_FILTER_DESCRIPTOR FilterData,  
    [in, optional] PVOID CallbackContext  
)
```

Provider <> Consumer

Application process



Provider enabled?

Windows Kernel

Provider <> Consumer

Application process

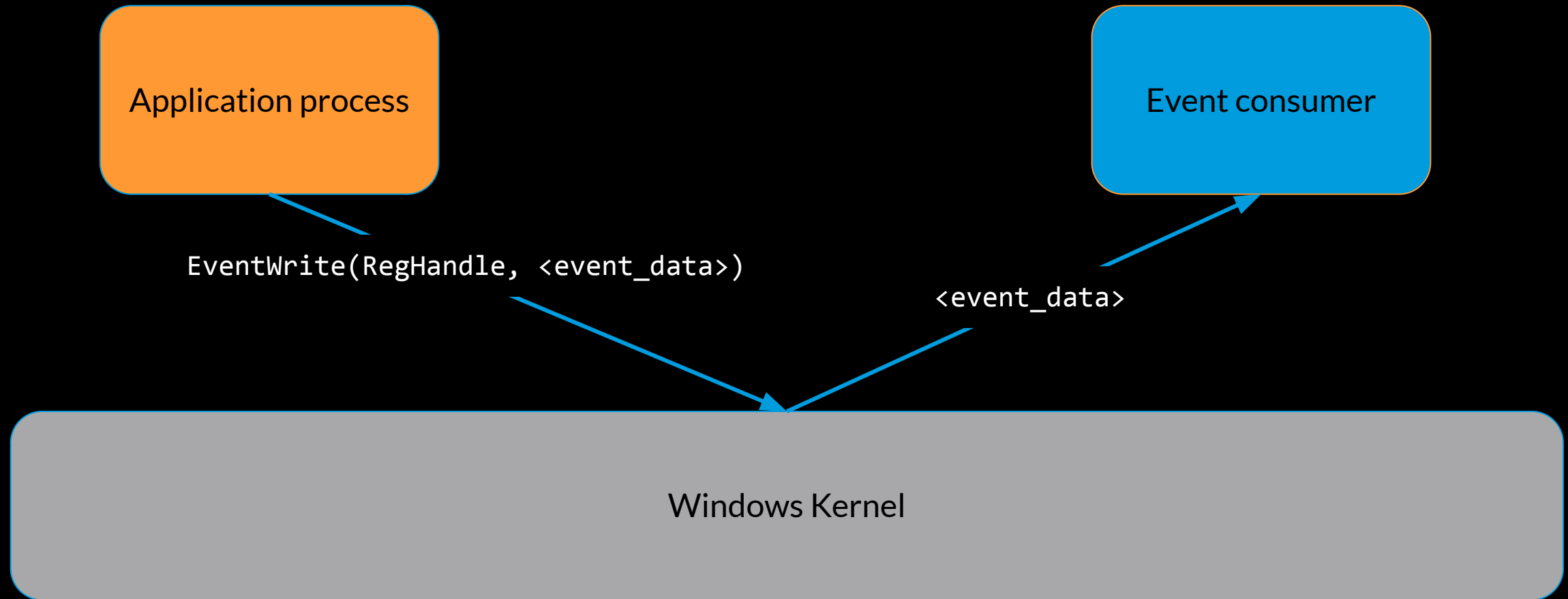
Event consumer

`EnableTraceEx2(DEADBEEF-BAAD-DEAD-BEEF-BAADF00D)`

Windows Kernel

```
graph TD; AP[Application process]; EC[Event consumer]; WK[Windows Kernel]; EC -- "EnableTraceEx2(DEADBEEF-BAAD-DEAD-BEEF-BAADF00D)" --> WK;
```


Provider <> Consumer



RPC ETW Provider

- [Microsoft-Windows-RPC](https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win7-7600/Microsoft-Windows-RPC.xml)¹ {6ad52b32-d609-4be9-ae07-ce8dae937e39}
- Implemented in the runtime *rpcrt4.dll*
 - Since event routing is handled in the kernel, multiple processes can write to the same provider
- 13 different events
 - Event ids 5,7 — Client call start/stop
 - Event ids 6,8 — Server call start/stop

¹ <https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win7-7600/Microsoft-Windows-RPC.xml>

Call Start Schema

```
<template tid="RpcServerCallStartArgs_V1">
  <data name="InterfaceUuid" inType="win:GUID"/>
  <data name="ProcNum" inType="win:UInt32"/>
  <data name="Protocol" inType="win:UInt32"/>
  <data name="NetworkAddress" inType="win:UnicodeString"/>
  <data name="Endpoint" inType="win:UnicodeString"/>
  <data name="Options" inType="win:UnicodeString"/>
  <data name="AuthenticationLevel" inType="win:UInt32"/>
  <data name="AuthenticationService" inType="win:UInt32"/>
  <data name="ImpersonationLevel" inType="win:UInt32"/>
</template>
```

Call Start Schema

```
<template tid="RpcServerCallStartArgs_V1">  
  <data name="InterfaceUuid" inType="win:GUID"/>  
  <data name="ProcNum" inType="win:UInt32"/>  
  <data name="Protocol" inType="win:UInt32"/>  
  <data name="NetworkAddress" inType="win:UnicodeString"/>  
  <data name="Endpoint" inType="win:UnicodeString"/>  
  <data name="Options" inType="win:UnicodeString"/>  
  <data name="AuthenticationLevel" inType="win:UInt32"/>  
  <data name="AuthenticationService" inType="win:UInt32"/>  
  <data name="ImpersonationLevel" inType="win:UInt32"/>  
</template>
```

Additional Useful Event Headers

- ActivityId — a UUID that can be used to track event chains (Call start, stop, error)
- ProcessId — the PID of the process that sent the trace event
- Timestamp — the time the trace event was generated

Attack Detection?

Server Trace Events Are Lacking

```
if ( (Microsoft_Windows_RPCEnableBits & 2) != 0 )
    McTemplateU0jqzzzzqqq_EtwEventWriteTransfer(
        *(__QWORD *)(*((__QWORD *)this + 38) + 80i64),
        (__int64)&RpcServerCallStartEvent,          // Event descriptor
        (__int64)v4 + 84,                            // Interface UUID
        *(__DWORD *)this + 95),                     // Opnum
        v25[48],                                     // Transfer protocol id
        0i64,                                         // Network address
        *(const wchar_t **)(*(__QWORD *)(*((__QWORD *)this + 38) + 80i64) + 32i64), // Endpoint
        0i64,                                         // Options
        v25[32],                                     // Authentication level
        v25[36],                                     // Authentication service
        0);                                           // Impersonation level
```

OSF_SCALL::DispatchHelper

Server Trace Events Are Lacking

```
if ( (Microsoft_Windows_RPCEnableBits & 2) != 0 )
    McTemplateU0jqzzzzqqq_EtwEventWriteTransfer(
        *(__QWORD *)(*((__QWORD *)this + 38) + 80i64),
        (__int64)&RpcServerCallStartEvent,          // Event descriptor
        (__int64)v4 + 84,                            // Interface UUID
        *(__DWORD *)this + 95,                       // Opnum
        v25[48],                                     // Transfer protocol id
        0i64,                                         // Network address
        *(const wchar_t **)(*(__QWORD *)(*((__QWORD *)this + 38) + 80i64) + 32i64), // Endpoint
        0i64,                                         // Options
        v25[32],                                     // Authentication level
        v25[36],                                     // Authentication service
        0);                                           // Impersonation level
```

OSF_SCALL::DispatchHelper

Just Do Client Calls, Dummy...

- We're looking for malicious traffic, assume client is in the attacker's control
- Attackers can:

Just Do Client Calls, Dummy...

- We're looking for malicious traffic, assume client is in the attacker's control
- Attackers can:
 - Tamper with ETW events via hooks/memory manipulation

Just Do Client Calls, Dummy...

- We're looking for malicious traffic, assume client is in the attacker's control
- Attackers can:
 - Tamper with ETW events via hooks/memory manipulation
 - Use their own machine outside of our control + local proxy (no ETW then)

Just Do Client Calls, Dummy...

- We're looking for malicious traffic, assume client is in the attacker's control
- Attackers can:
 - Tamper with ETW events via hooks/memory manipulation
 - Use their own machine outside of our control + local proxy (no ETW then)
 - Generate RPC traffic without the OS (i.e Impacket)

ETW to the Rescue, Again



TCP ETW

event 1017 — TcpAcceptListenerComplete

```
<template tid="TcpAccpetListenerRouteLookupFailureArgs">  
  <data name="LocalAddressLength" inType="win:UInt32"/>  
  <data name="LocalAddress" inType="win:Binary" length="LocalAddressLength"/>  
  <data name="RemoteAddressLength" inType="win:UInt32"/>  
  <data name="RemoteAddress" inType="win:Binary" length="RemoteAddressLength"/>  
  ...  
</template>
```

Address binary field := AF, IP, port

TCP ETW

event 1017 — TcpAcceptListenerComplete

```
<template tid="TcpAccpetListenerRouteLookupFailureArgs">  
  <data name="LocalAddressLength" inType="win:UInt32"/>  
  <data name="LocalAddress" inType="win:Binary" length="LocalAddressLength"/>  
  <data name="RemoteAddressLength" inType="win:UInt32"/>  
  <data name="RemoteAddress" inType="win:Binary" length="RemoteAddressLength"/>  
  ...  
</template>
```

Address binary field := AF, IP, port

SMB ETW

event 500 — Smb2ConnectionAcceptStart

```
<template tid="Smb2ConnectionAcceptStart">  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="AddressLength"  
    inType="win:UInt32"/>  
  <data name="Address"  
    inType="win:Binary"  
    length="AddressLength"/>  
</template>
```


SMB ETW

event 500 — Smb2ConnectionAcceptStart

```
<template tid="Smb2ConnectionAcceptStart">  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="AddressLength"  
    inType="win:UInt32"/>  
  <data name="Address"  
    inType="win:Binary"  
    length="AddressLength"/>  
</template>
```

SMB ETW

event 500 — Smb2ConnectionAcceptStart

```
<template tid="Smb2ConnectionAcceptStart">  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="AddressLength"  
    inType="win:UInt32"/>  
  <data name="Address"  
    inType="win:Binary"  
    length="AddressLength"/>  
</template>
```

event 8 — Smb2RequestCreate_V2

```
<template tid="Smb2RequestCreate_V2">  
  ...  
  <data name="Filename"  
    inType="win:UnicodeString"/>  
  ...  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="TreeConnectGUID"  
    inType="win:GUID"/>  
</template>
```

SMB ETW

event 500 — Smb2ConnectionAcceptStart

```
<template tid="Smb2ConnectionAcceptStart">  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="AddressLength"  
    inType="win:UInt32"/>  
  <data name="Address"  
    inType="win:Binary"  
    length="AddressLength"/>  
</template>
```

event 8 — Smb2RequestCreate_V2

```
<template tid="Smb2RequestCreate_V2">  
  ...  
  <data name="Filename"  
    inType="win:UnicodeString"/>  
  ...  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="TreeConnectGUID"  
    inType="win:GUID"/>  
</template>
```

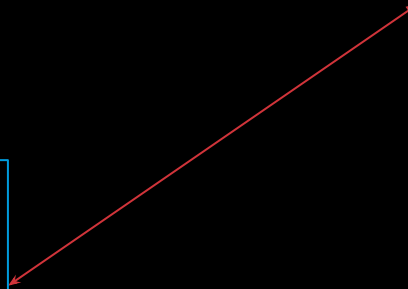
SMB ETW

event 500 — Smb2ConnectionAcceptStart

```
<template tid="Smb2ConnectionAcceptStart">  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="AddressLength"  
    inType="win:UInt32"/>  
  <data name="Address"  
    inType="win:Binary"  
    length="AddressLength"/>  
</template>
```

event 8 — Smb2RequestCreate_V2

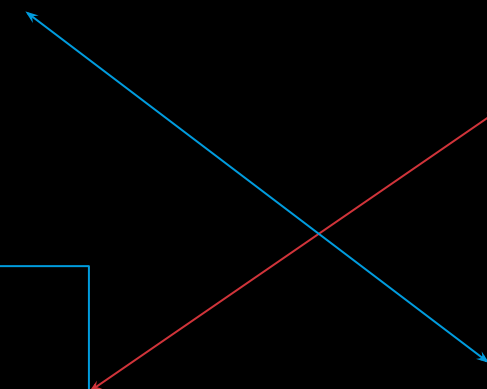
```
<template tid="Smb2RequestCreate_V2">  
  ...  
  <data name="Filename"  
    inType="win:UnicodeString"/>  
  ...  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="TreeConnectGUID"  
    inType="win:GUID"/>  
</template>
```



SMB ETW

event 500 — Smb2ConnectionAcceptStart

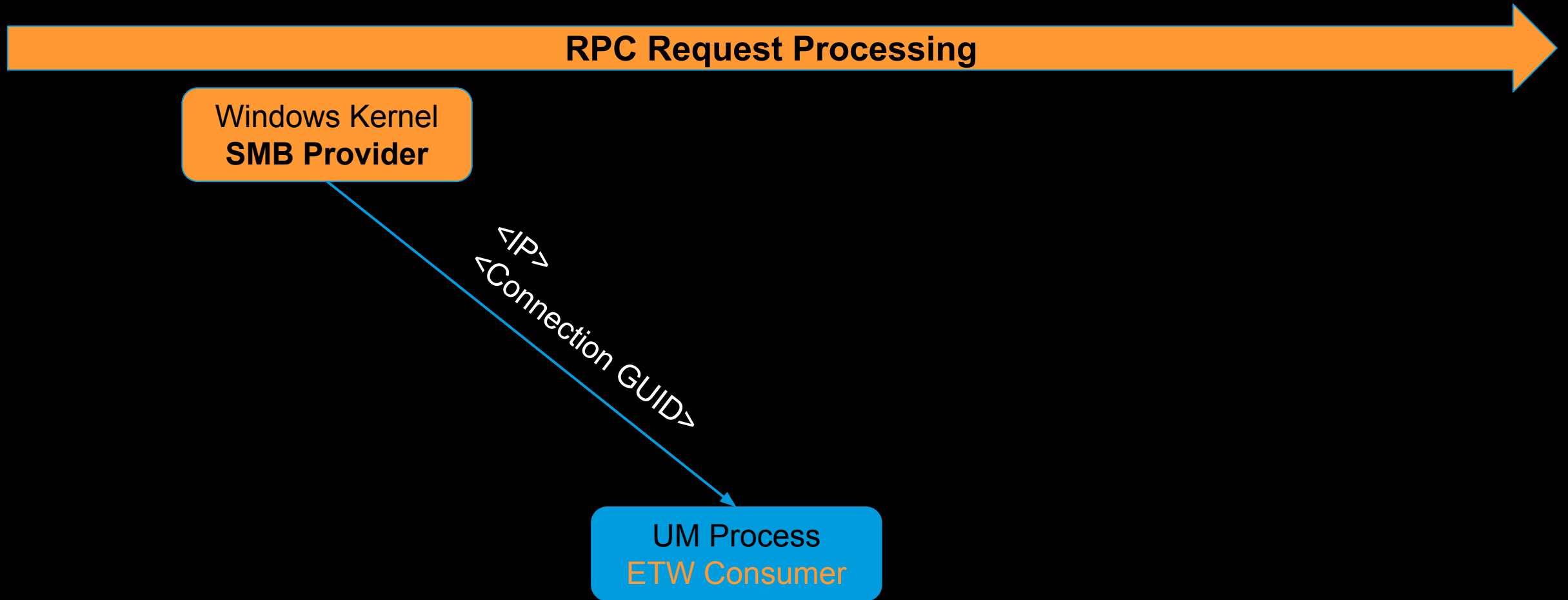
```
<template tid="Smb2ConnectionAcceptStart">  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="AddressLength"  
    inType="win:UInt32"/>  
  <data name="Address"  
    inType="win:Binary"  
    length="AddressLength"/>  
</template>
```



event 8 — Smb2RequestCreate_V2

```
<template tid="Smb2RequestCreate_V2">  
  ...  
  <data name="Filename"  
    inType="win:UnicodeString"/>  
  ...  
  <data name="ConnectionGUID"  
    inType="win:GUID"/>  
  <data name="TreeConnectGUID"  
    inType="win:GUID"/>  
</template>
```

Matching Flows



Matching Flows

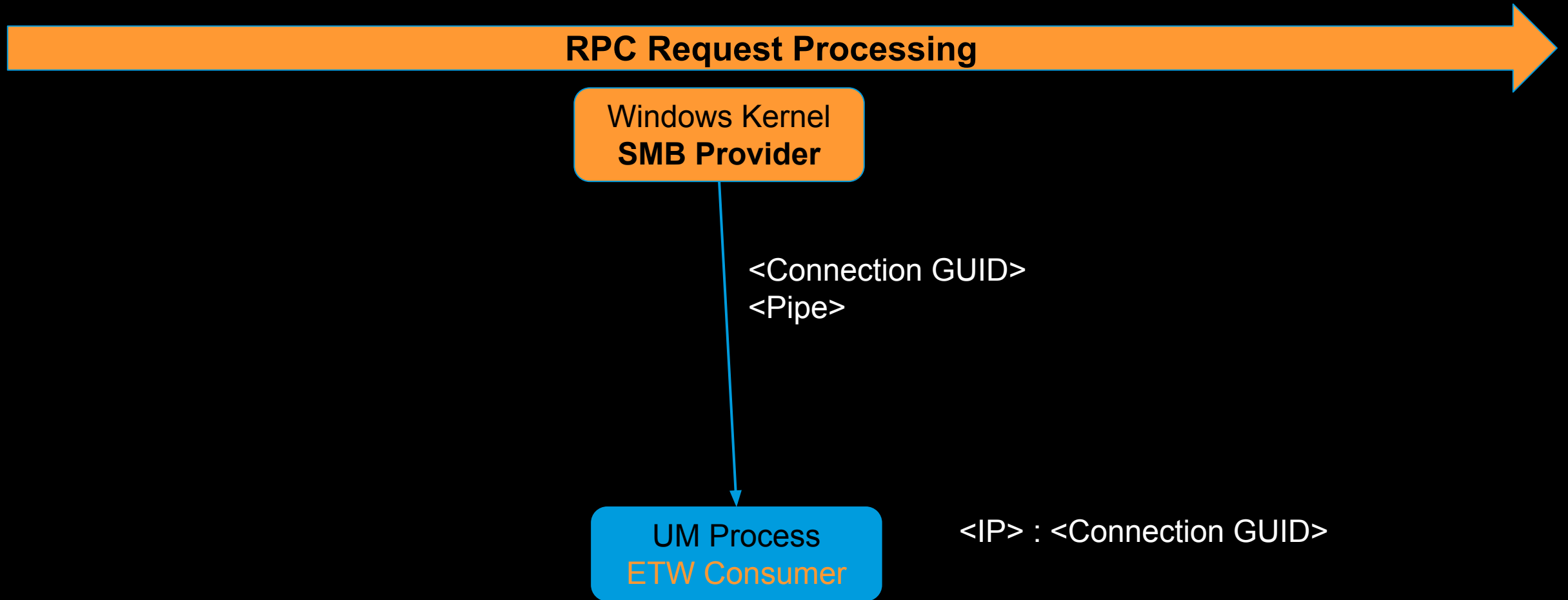


RPC Request Processing

UM Process
ETW Consumer

<IP> : <Connection GUID>

Matching Flows



Matching Flows

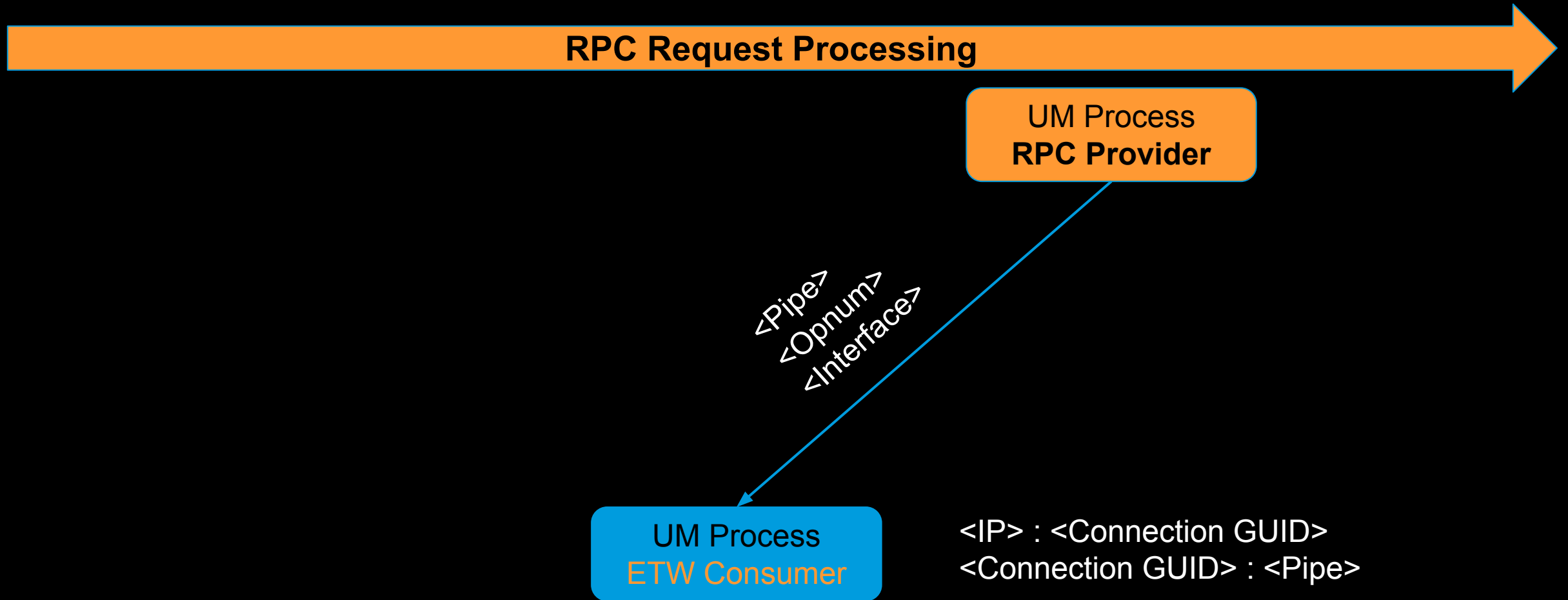


RPC Request Processing

UM Process
ETW Consumer

<IP> : <Connection GUID>
<Connection GUID> : <Pipe>

Matching Flows



Matching Flows



RPC Request Processing

UM Process
ETW Consumer

<IP> : <Connection GUID>
<Connection GUID> : <Pipe>
<Pipe> : <Interface, Opnum>

Matching Flows

Machine **<IP>** connected over **<Pipe>** to request operation **<Opnum>** of the interface **<Interface>**

RPC Visibility

- Python script¹ with pywintrace
- Subscribe to the SMB, TCP and RPC providers
- Send results to Neo4J for easy visualization and querying
 - Could be extended to use other databases

¹https://github.com/akamai/akamai-security-research/rpc_visibility

Attack Detection

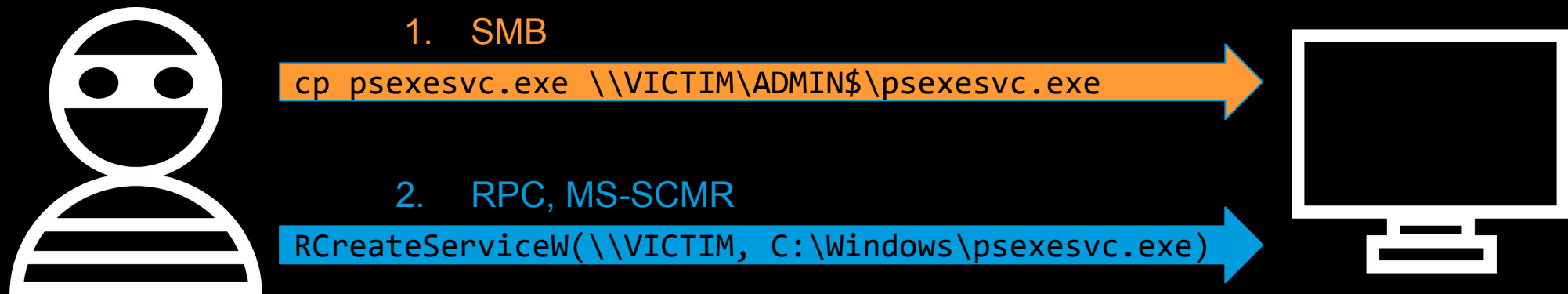
Demo Time

RPC Visibility VS PSEXEC



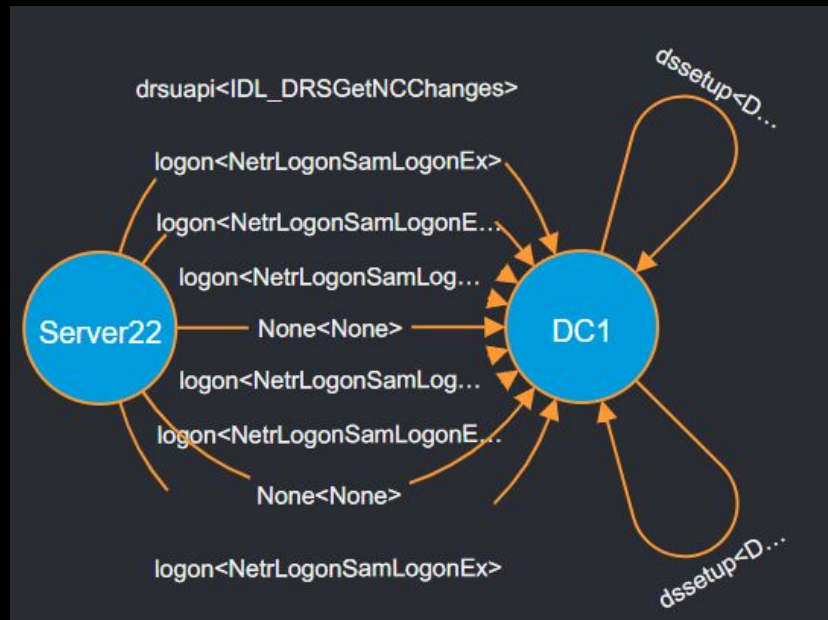
PSEXec

- Both a general name for attack technique and a sysinternals tool
- Copy service PE to remote machine's ADMIN\$ share
- Tell the SCM (using [MS-SCMR](#)) to run a service from the copied binary
 - 0xC — RCreateServiceW



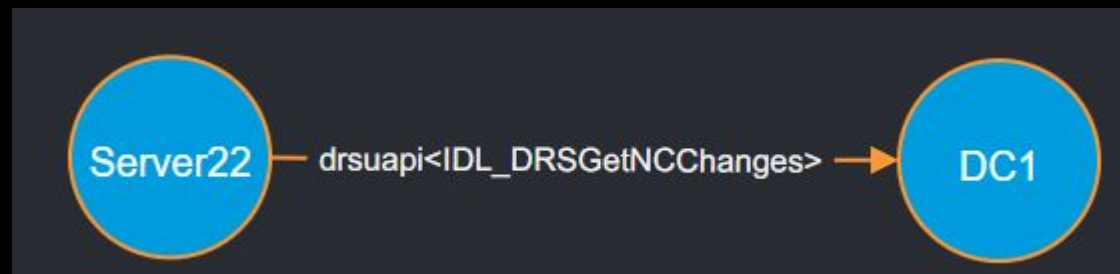
DCSync

- Connect to a DC and pretend to be another DC
 - Request replication of the credential database
- Uses the Directory Replication service ([MS-DRSR](#))
 - 0x3 — IDL_DRSGetNCChanges



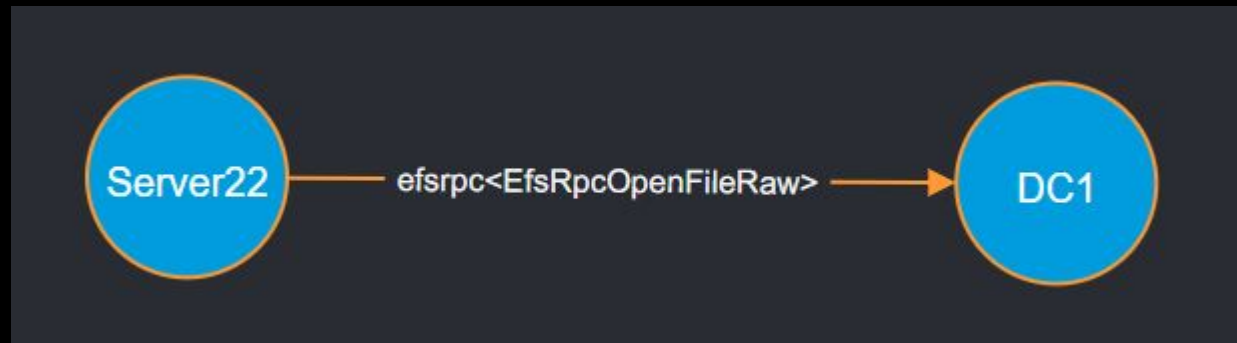
DCSync

- Connect to a DC and pretend to be another DC
 - Request replication of the credential database
- Uses the Directory Replication service ([MS-DRSR](#))
 - 0x3 — IDL_DRSGetNCChanges



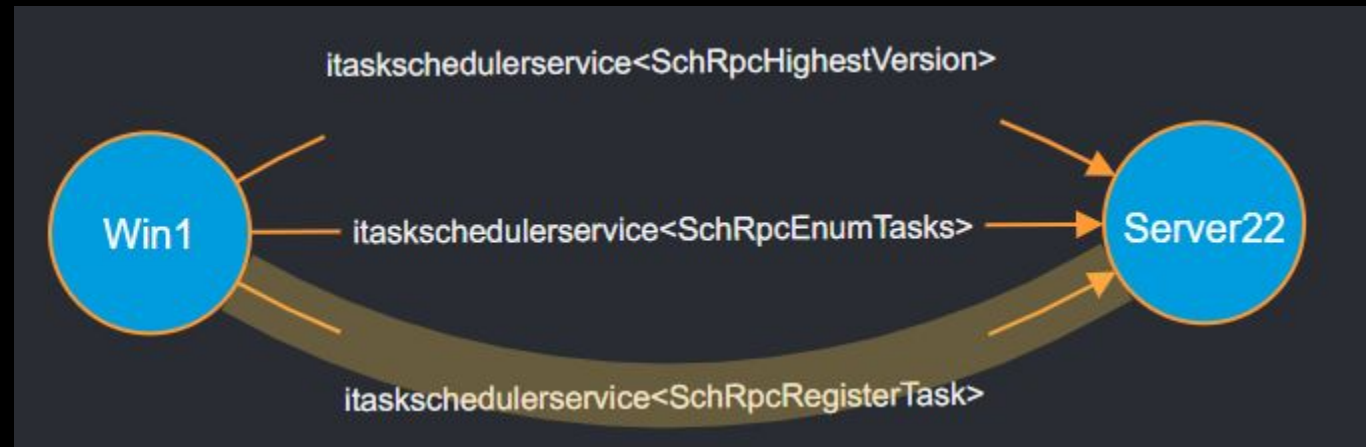
PetitPotam

- Tell the EFS service to open a remote file
 - Triggers an SMB connection with authentication that can be relayed
- Uses the [MS-EFSR](#) interface
 - 0x0 — EfsRpcOpenFileRaw
 - 0x4 — EfsRpcEncryptFileSrv



Task Scheduler

- Create a scheduled task remotely
 - MITRE [T1053.005](#)
- Uses the Task Scheduler Service Remoting protocol [MS-TSCH](#)
 - 0x1 — SchRpcRegisterTask



Remote WMI

- Use WMI to execute a binary remotely
- Implemented over [MS-DCOM](#)
 - Another layer of encapsulation
 - Can't easily tell which operation was requested



Determining WMI maliciousness

- Check the opnums in each malicious operation, compared to benign
 - Heuristic based approach, has to be configured for each technique

WMIC process call create

Interface Name	Opnums
iwbemservices	24, 6
iremunknown2	5, 3
iwbemlevel1login	6, 3
iwbemloginclientid	3
iremotescmactivator	4
iobjectexporter	5

WMIC process get

Interface Name	Opnums
iwbemservices	20
iremunknown2	5, 3
iwbemlevel1login	6, 3
iwbemloginclientid	3
iremotescmactivator	4
iobjectexporter	5
iwbemwcosmartenum	3
iwbemfetchsmartenum	3

Determining WMI maliciousness

- Check the opnums in each malicious operation, compared to benign
 - Heuristic based approach, has to be configured for each technique

WMIC process call create

Interface Name	Opnums
iwbemservices	24, 6
iremunknown2	5, 3
iwbemlevel1login	6, 3
iwbemloginclientid	3
iremotescmactivator	4
iobjectexporter	5

WMIC process get

Interface Name	Opnums
iwbemservices	20
iremunknown2	5, 3
iwbemlevel1login	6, 3
iwbemloginclientid	3
iremotescmactivator	4
iobjectexporter	5
iwbemwcosmartenum	3
iwbemfetchsmartenum	3

Determining WMI maliciousness

- Check the opnums in each malicious operation, compared to benign
 - Heuristic based approach, has to be configured for each technique

WMIC process call create

Interface Name	Opnums
iwbemservices	ExecMethod
	GetObject

WMIC process get

Interface Name	Opnums
iwbemservices	ExecQuery
iwbemwcosmartenum	Next
iwbemfetchsmartenum	GetSmartEnum

Drawbacks

- Still only metadata based detection
- Can't differentiate between “good” and “bad” flows
 - Detection has to be context dependent
 - Will have to be heuristics based analysis
- No blocking, visibility only
- Can't handle DCOM with certainty

Incident Response & Mitigation

#define RPC Filters

- Filtering mechanism part of the Windows Firewall
- Exposed through *netsh* or via WinAPI
- Available since Windows Vista

```
netsh rpc filter>show filter
Listing all RPC Filters.
-----
filterKey: ac65a4a0-0ab9-11ee-b826-8038fb83bd95
displayData.name: RPCFilter
displayData.description: RPC Filter
filterId: 0x2e0773
layerKey: um
weight: Type: FWP_EMPTY Value: Empty
action.type: block
numFilterConditions: 1

filterCondition[0]
    fieldKey: if_uuid
    matchType: FWP_MATCH_EQUAL
    conditionValue: Type: FWP_BYTE_ARRAY16_TYPE Value: 367abb81 35f19844 f09832ad 03100038
```

Layers? Is it a cake?

- Filtering can occur at different parts of the RPC connections
- Pre-defined layers tell the FW where to apply the rule
 - FWPM_LAYER_RPC_UM — connection to interface
 - FWPM_LAYER_RPC_EPMAP — connection to the endpoint mapper
 - FWPM_LAYER_RPC_EP_ADD — new endpoint registration

#define RPC Filters

- Filtering mechanism part of the Windows Firewall
- Exposed through *netsh* or via WinAPI
- Available since Windows Vista

```
netsh rpc filter>show filter
Listing all RPC Filters.
-----
filterKey: ac65a4a0-0ab9-11ee-b826-8038fb83bd95
displayData.name: RPCFilter
displayData.description: RPC Filter
filterId: 0x2e0773
layerKey: um
weight: Type: FWP_EMPTY Value: Empty
action.type: block
numFilterConditions: 1

filterCondition[0]
    fieldKey: if_uuid
    matchType: FWP_MATCH_EQUAL
    conditionValue: Type: FWP_BYTE_ARRAY16_TYPE Value: 367abb81 35f19844 f09832ad 03100038
```

Filter fields

- src & dst IP
- RPC endpoints (port/named pipe)
- RPC interface UUID
- User token

•
•
•

Full list + notes in our [GH](#)

Mitigation strategies

After understanding what RPC traffic is going on in the network (using ETW), we can:

Mitigation strategies

After understanding what RPC traffic is going on in the network (using ETW), we can:

- Block unused RPC interfaces that can be used by attackers
 - If no one is using SCMR, why should it be left open?

Mitigation strategies

After understanding what RPC traffic is going on in the network (using ETW), we can:

- Block unused RPC interfaces that can be used by attackers
 - If no one is using SCMR, why should it be left open?
- Block RPC interfaces once an attack is detected/suspected
 - Emergency anti-lateral movement button

Mitigation strategies

After understanding what RPC traffic is going on in the network (using ETW), we can:

- Block unused RPC interfaces that can be used by attackers
 - If no one is using SCMR, why should it be left open?
- Block RPC interfaces once an attack is detected/suspected
 - Emergency anti-lateral movement button
- Restrict compromised users using RPC filters

Summary


- The RPC ETW provider is a treasure trove of information
- Use it to get the gist of what is going on in the network, and detect potential malicious activity
- Respond to incidents with RPC filters, or use them to mitigate the risk beforehand



Thanks for listening

Questions?

References

- Our  [repository](#) for all things RPC, including the RPC Visibility script
- RPC tools by other researchers
 - [RpcMon](#) by CyberArk
 - [RpcInvestigator](#) by TrailOfBits
 - [RPC Firewall](#), requires process injection & hooks
- [A Definitive Guide to the RPC filters](#), by our research team
- Useful readings:
 - Jonathan Johnson of SpecterOps, [Utilizing RPC Telemetry](#)
 - Carsten Sandker, [Offensive Windows IPC Internals 2: RPC](#)
 - James Forshaw, [How to secure a Windows RPC Server, and how not to.](#)