Lifting the Fog of War

Monitoring, Identifying and Mitigating MS-RPC Based Threats

whoami

Stiv Kupchik

Security Researcher

Akamai

@kupsul 

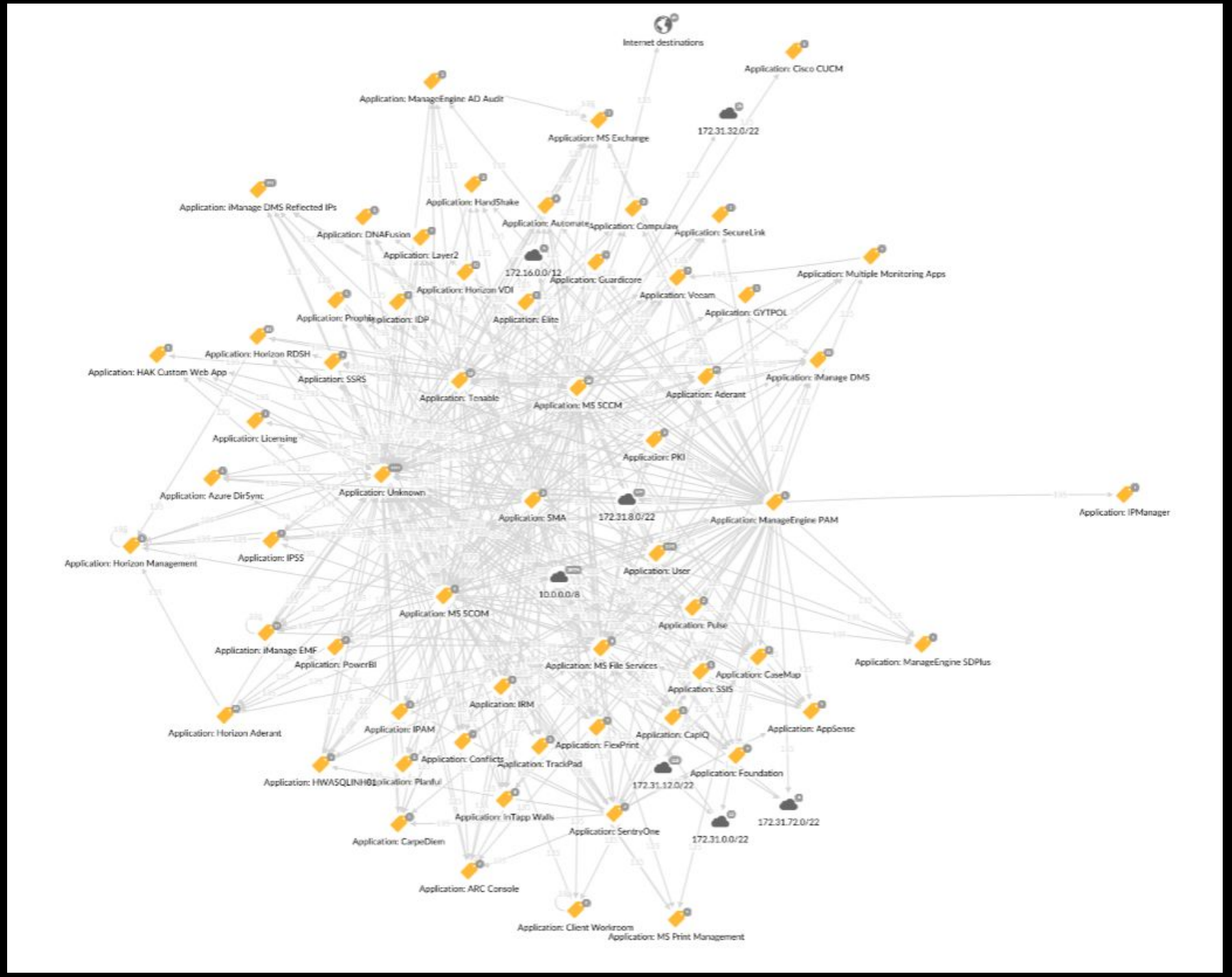
Background in DFIR and Windows internals

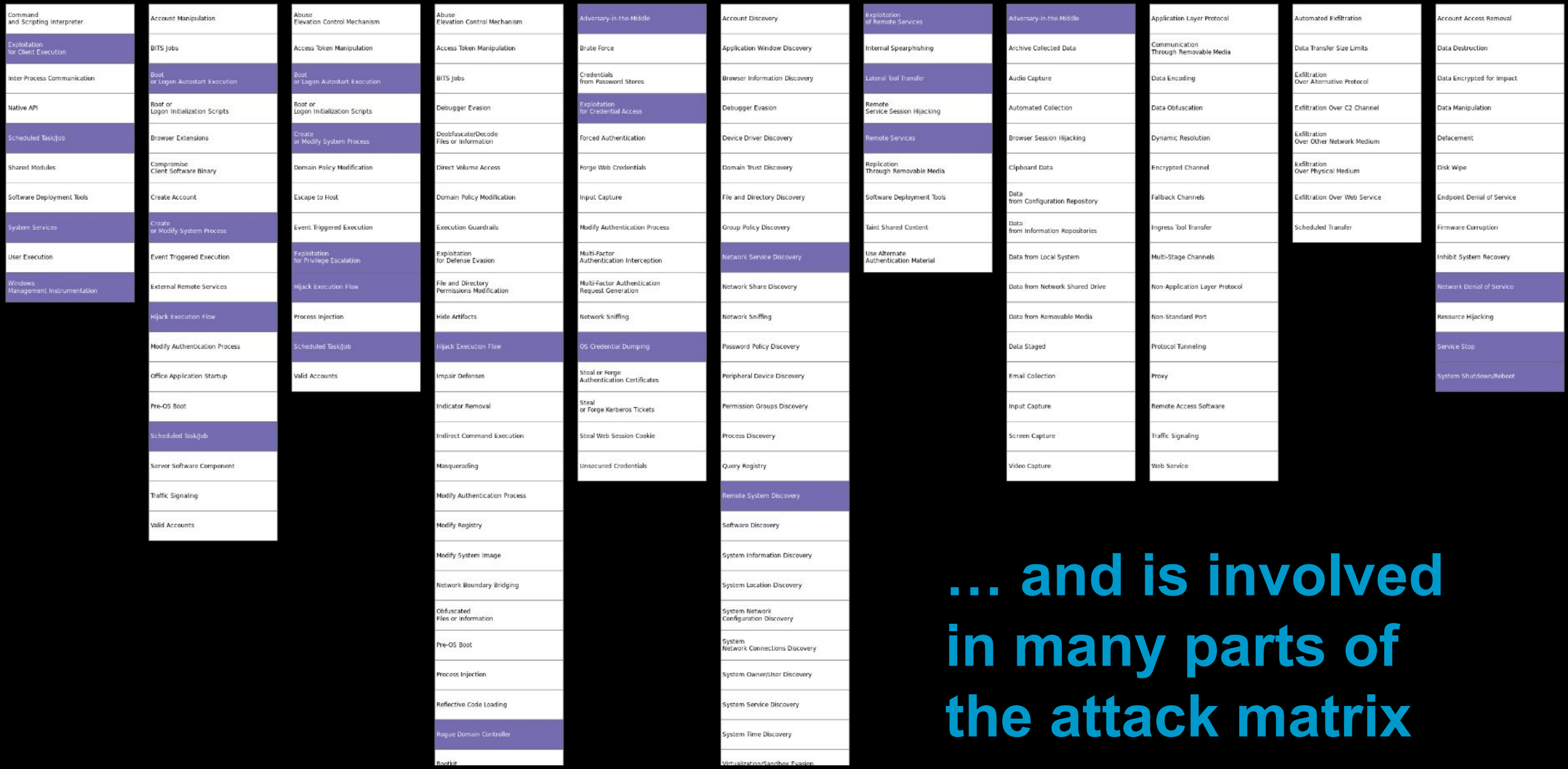
Agenda

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

#define RPC

It’s everywhere in the network





… 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

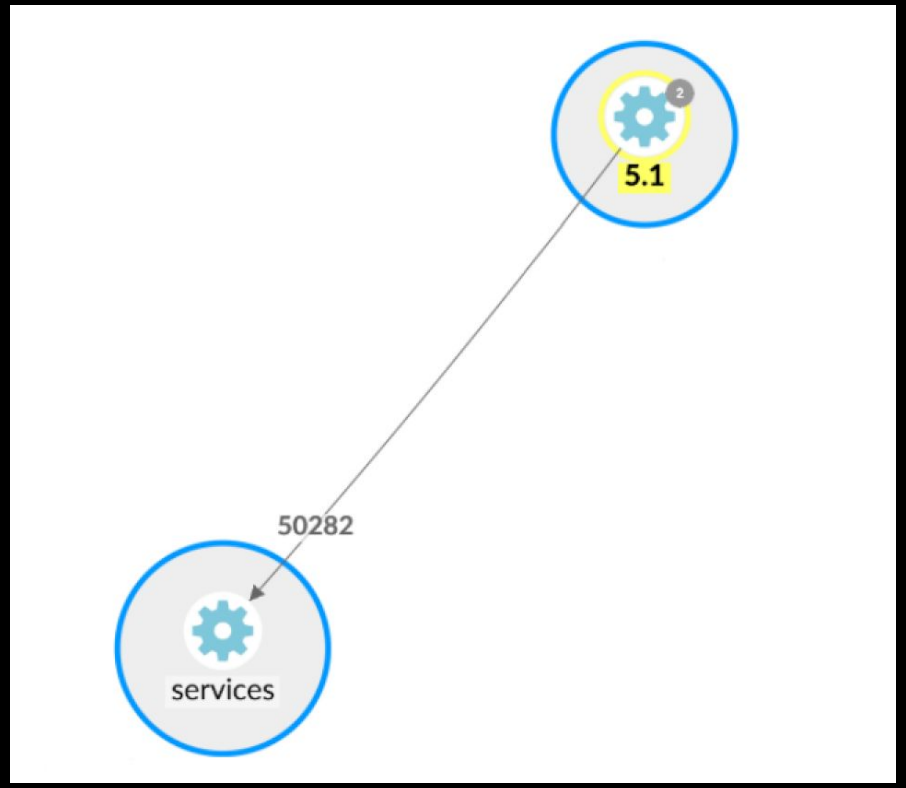
RPC Attacks are Hard to Detect

- 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

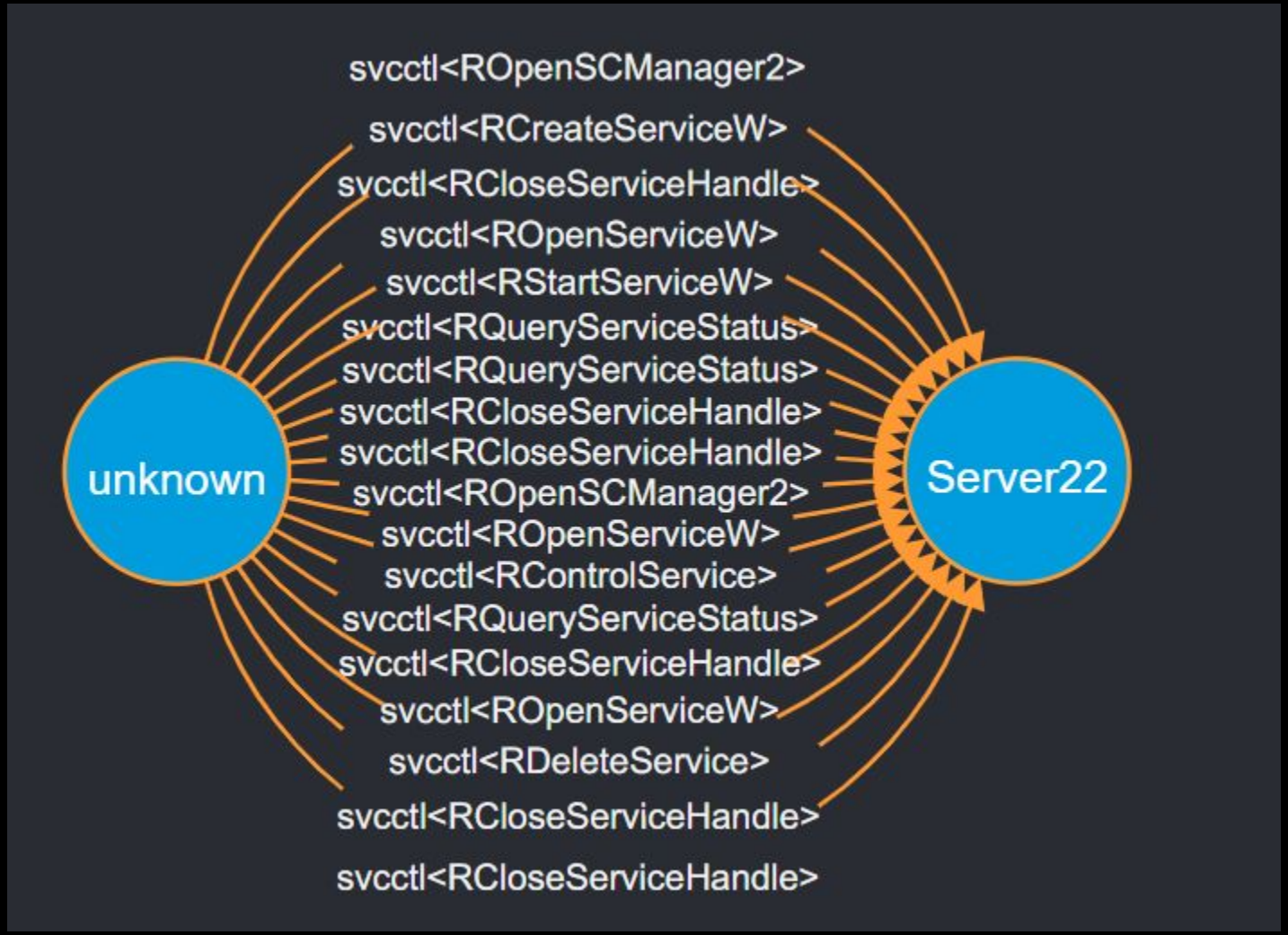
Not Many (documented) Defense Options

- 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

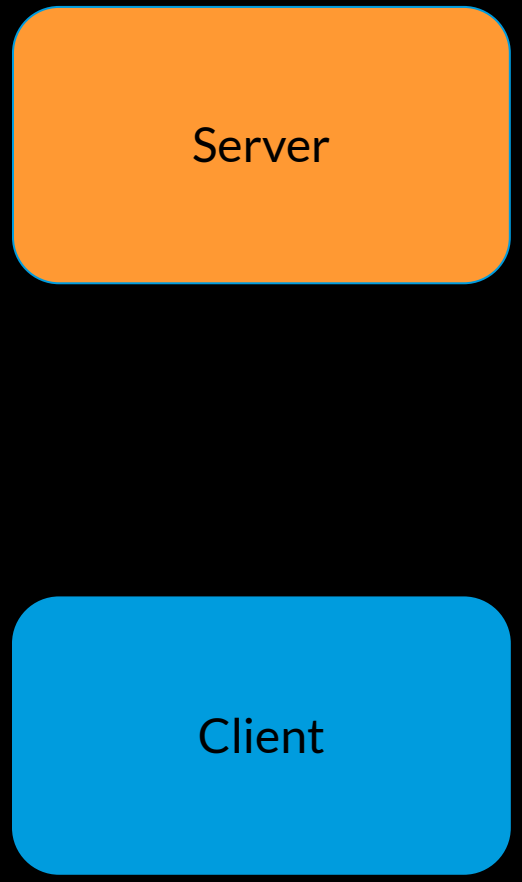


MS-RPC Overview

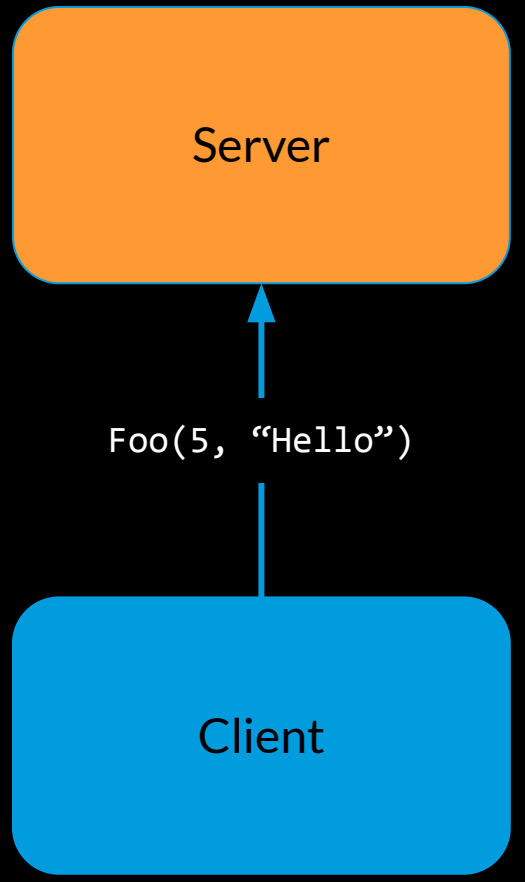
Terminology you'll soon master

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

The RPC Client-Server Model

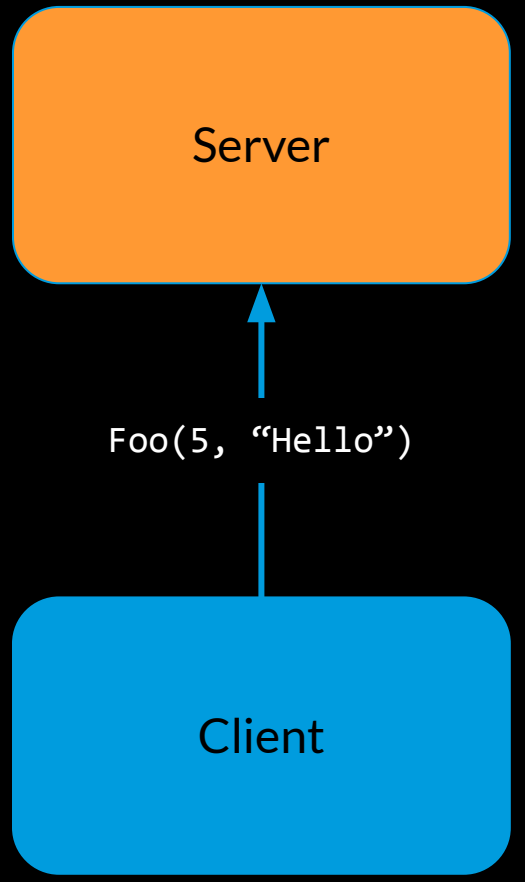


The RPC Client-Server Model

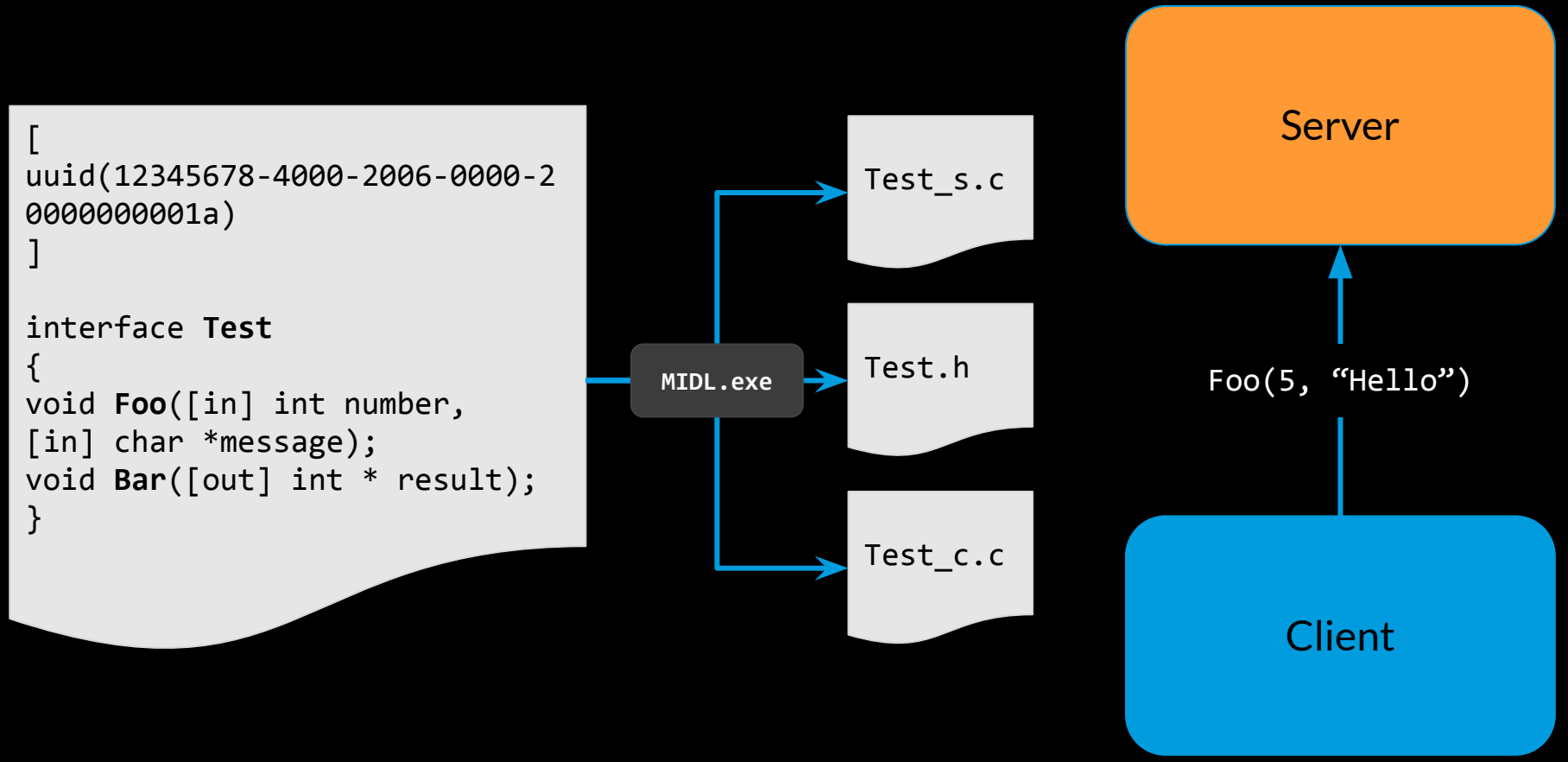


The RPC Client-Server Model

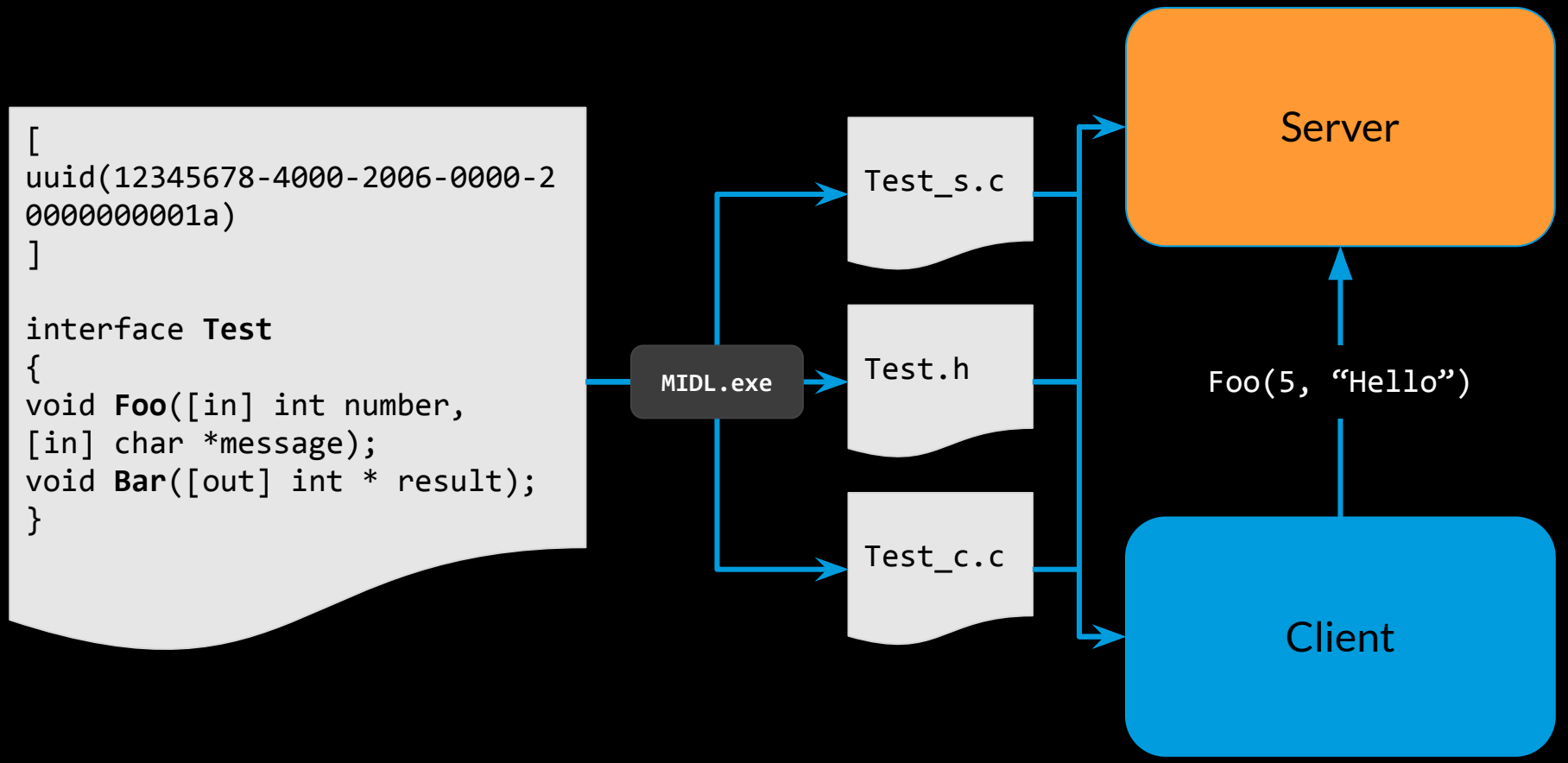
```
[
uuid(12345678-4000-2006-0000-2
0000000001a)
]

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



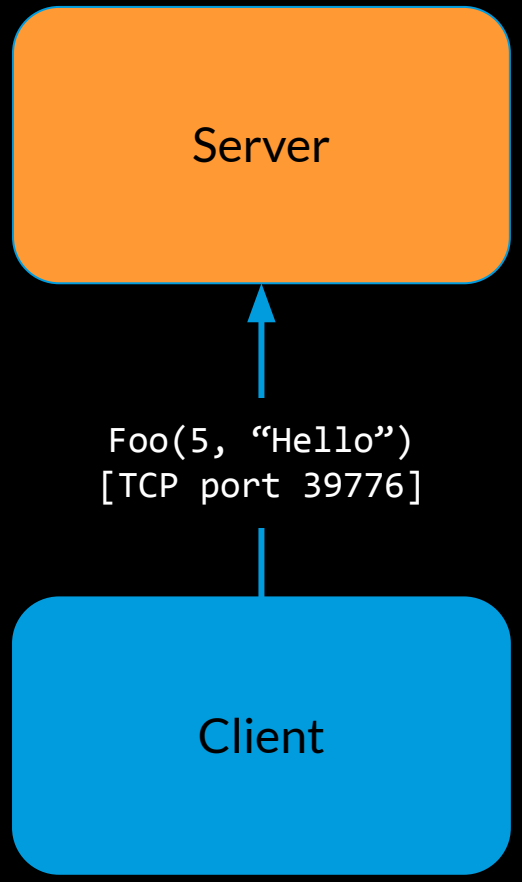
Endpoints

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

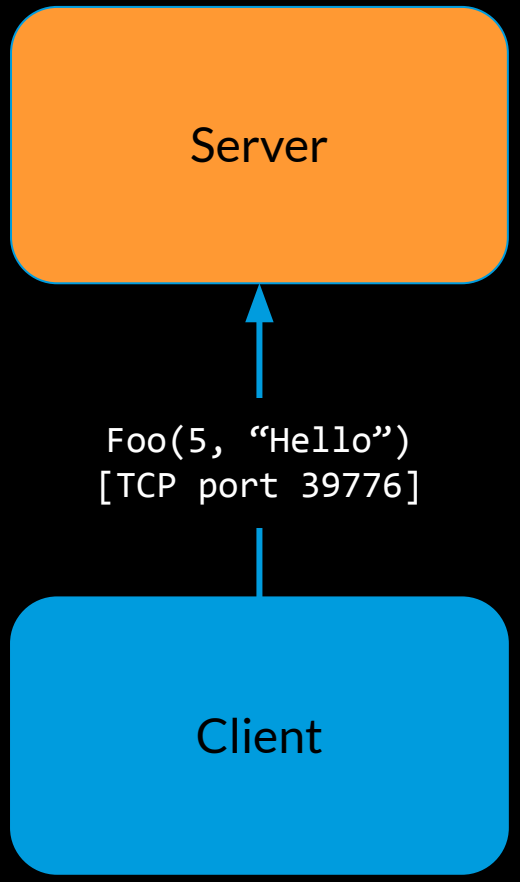
Transports	Protocol Sequence	Endpoints
TCP	`ncacn_ip_tcp`	<port number>
Named pipe	`ncacn_np`	<pipe name>
UDP	`ncadg_ip_udp`	<port number>
ALPC	`ncalrpc`	<ALPC port>
HTTP	`ncacn_http`	<hostname>
Hyper-V socket	`ncacn_hvsocket`	<UUID>

Well-Known Endpoints

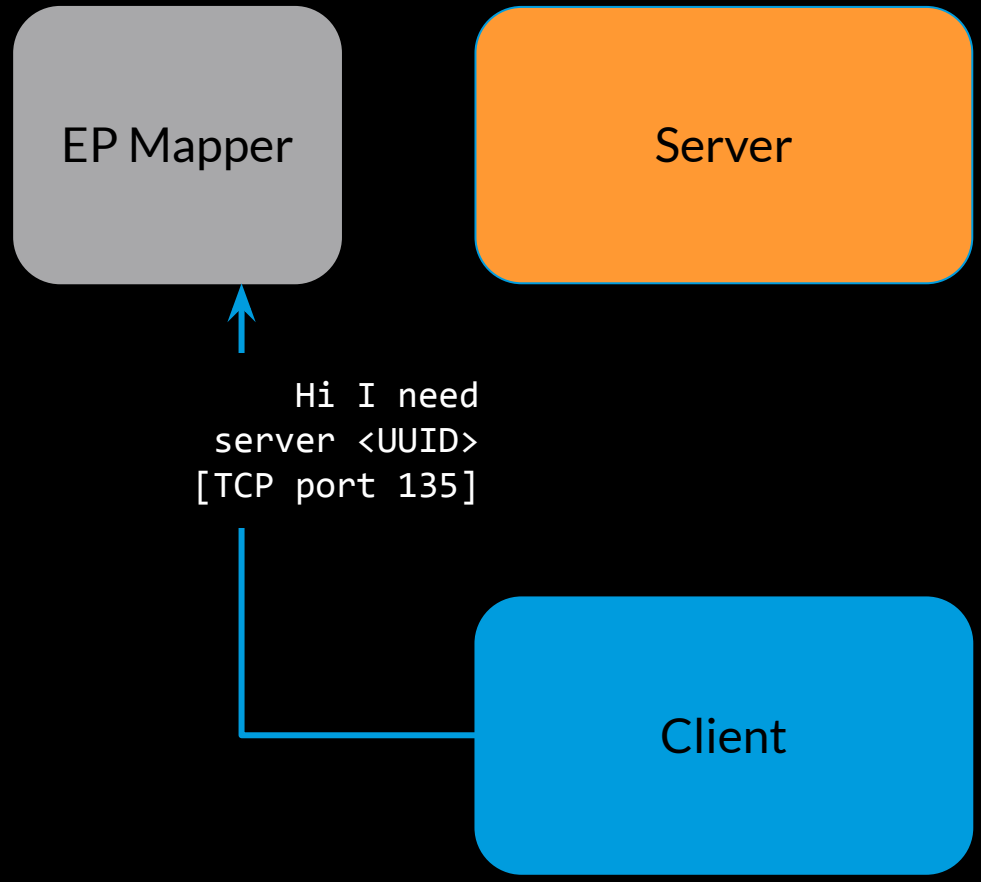
Dynamic Endpoints



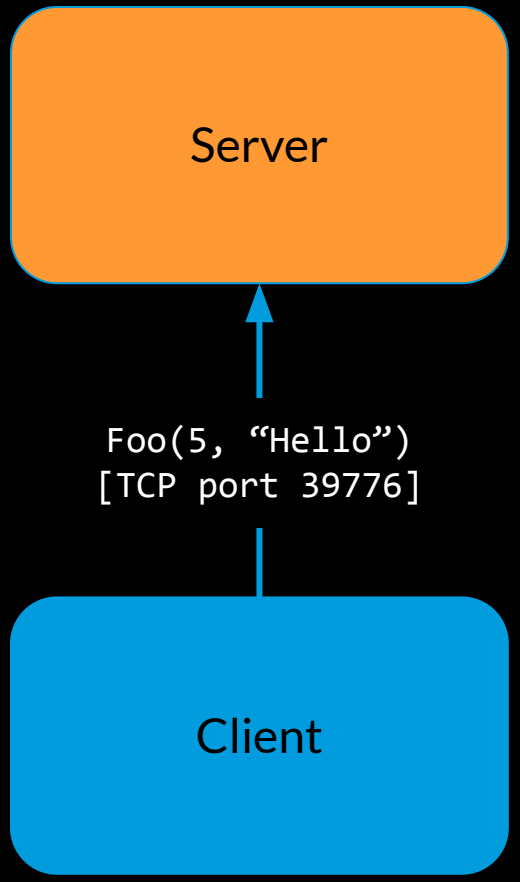
Well-Known Endpoints



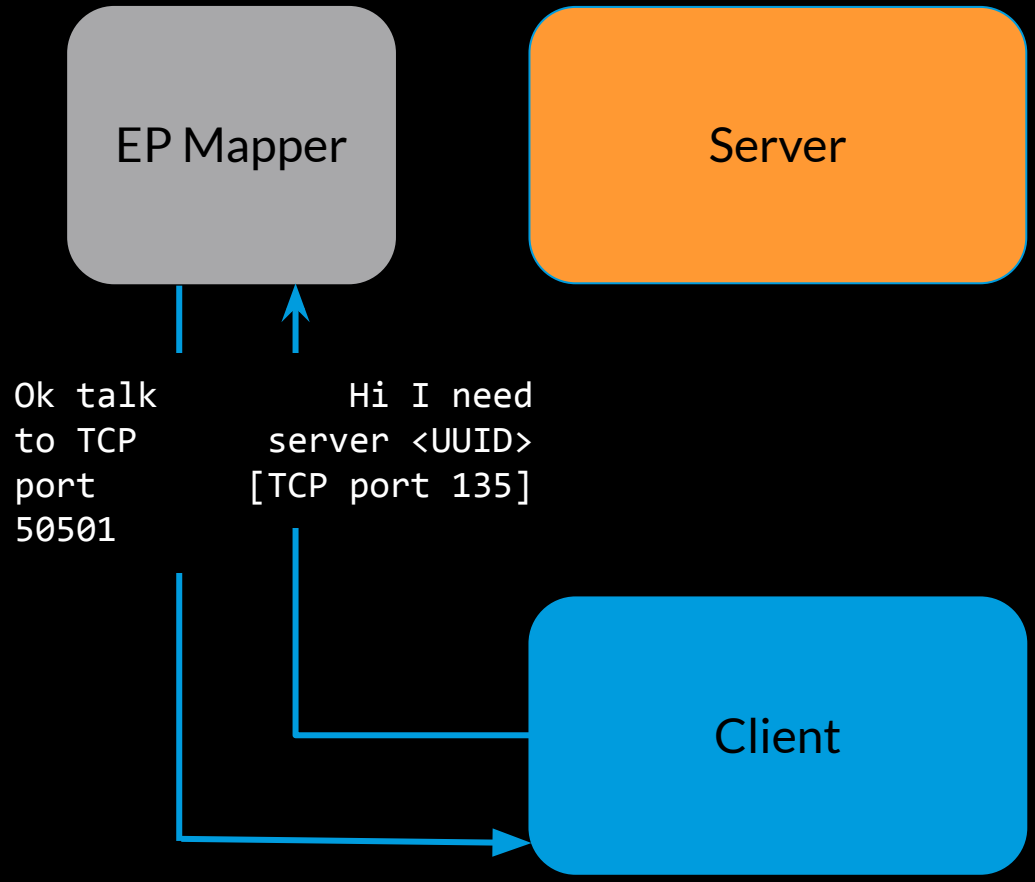
Dynamic Endpoints



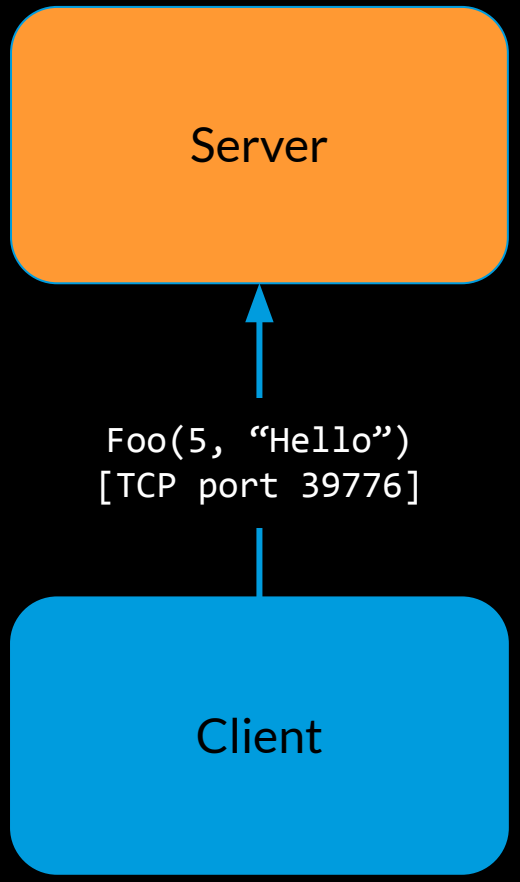
Well-Known Endpoints



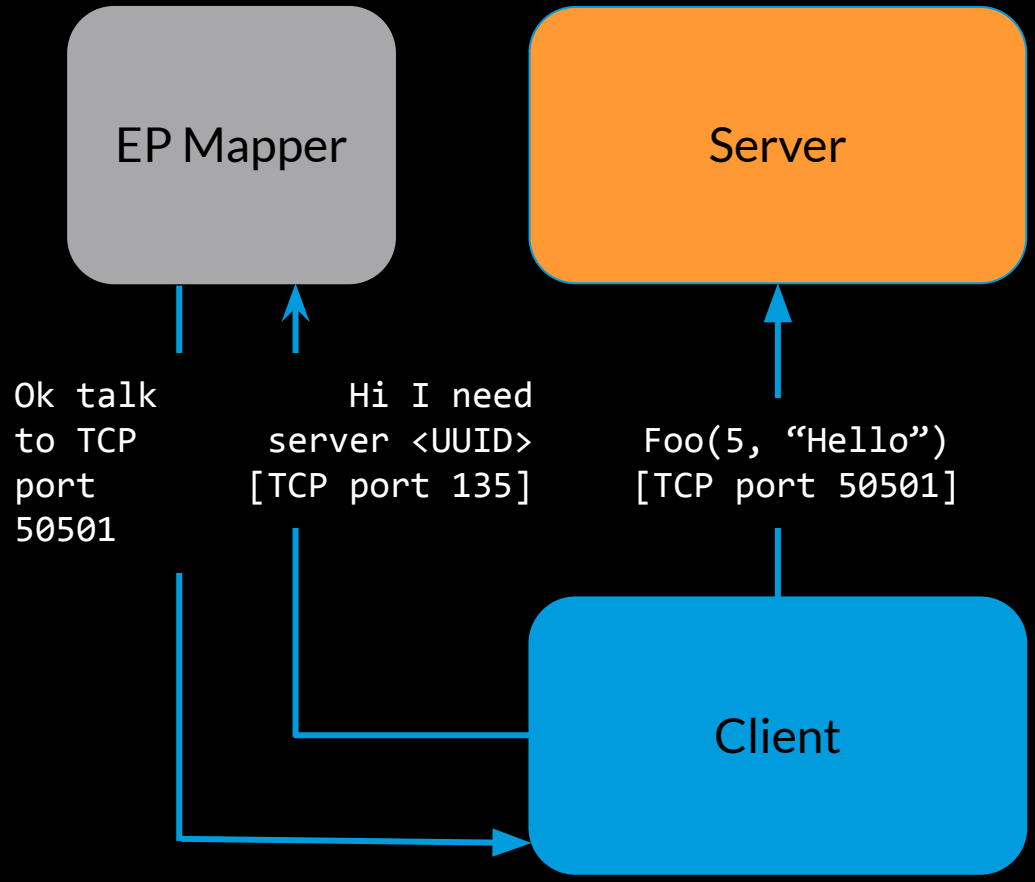
Dynamic Endpoints



Well-Known Endpoints



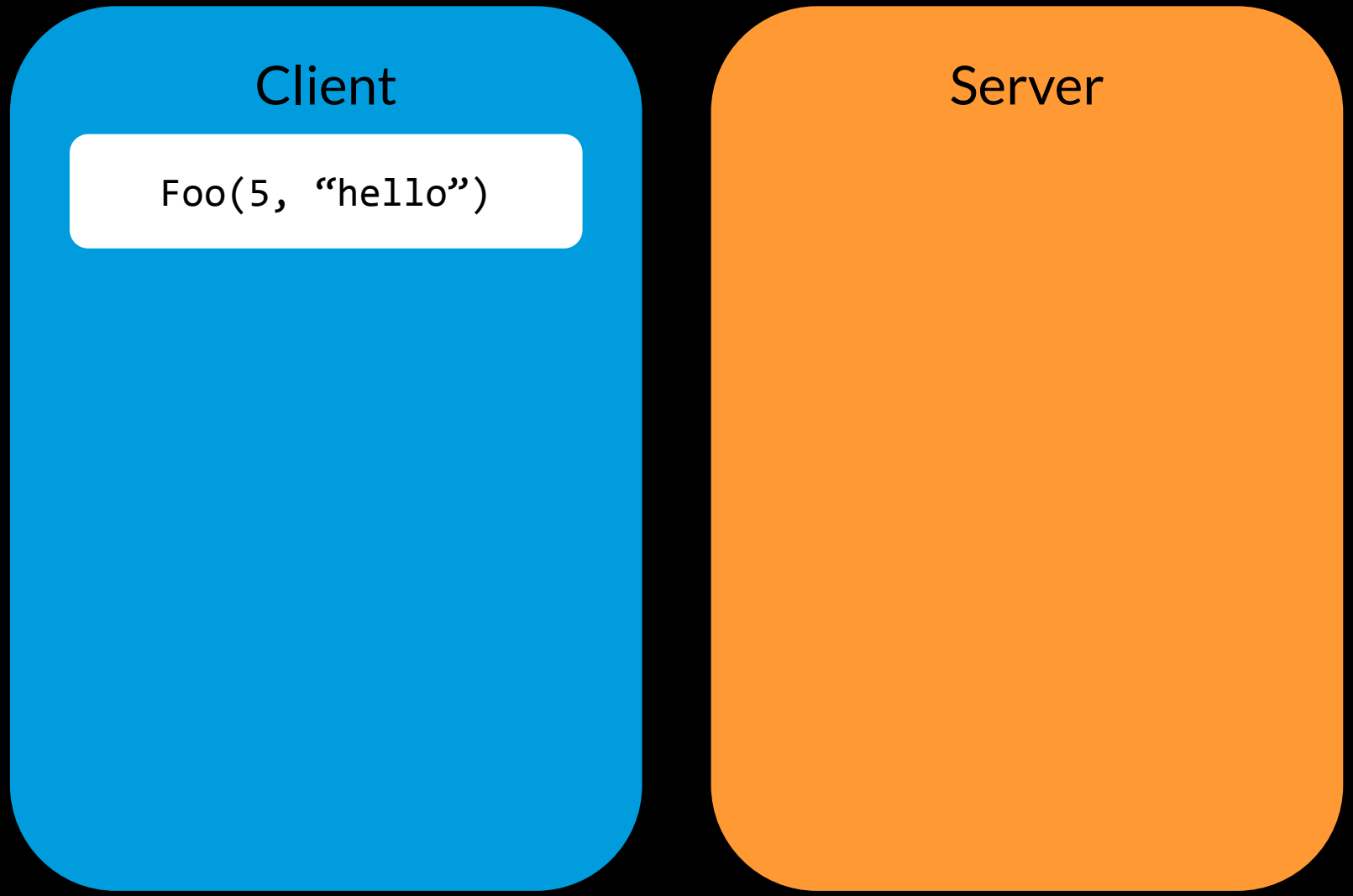
Dynamic Endpoints



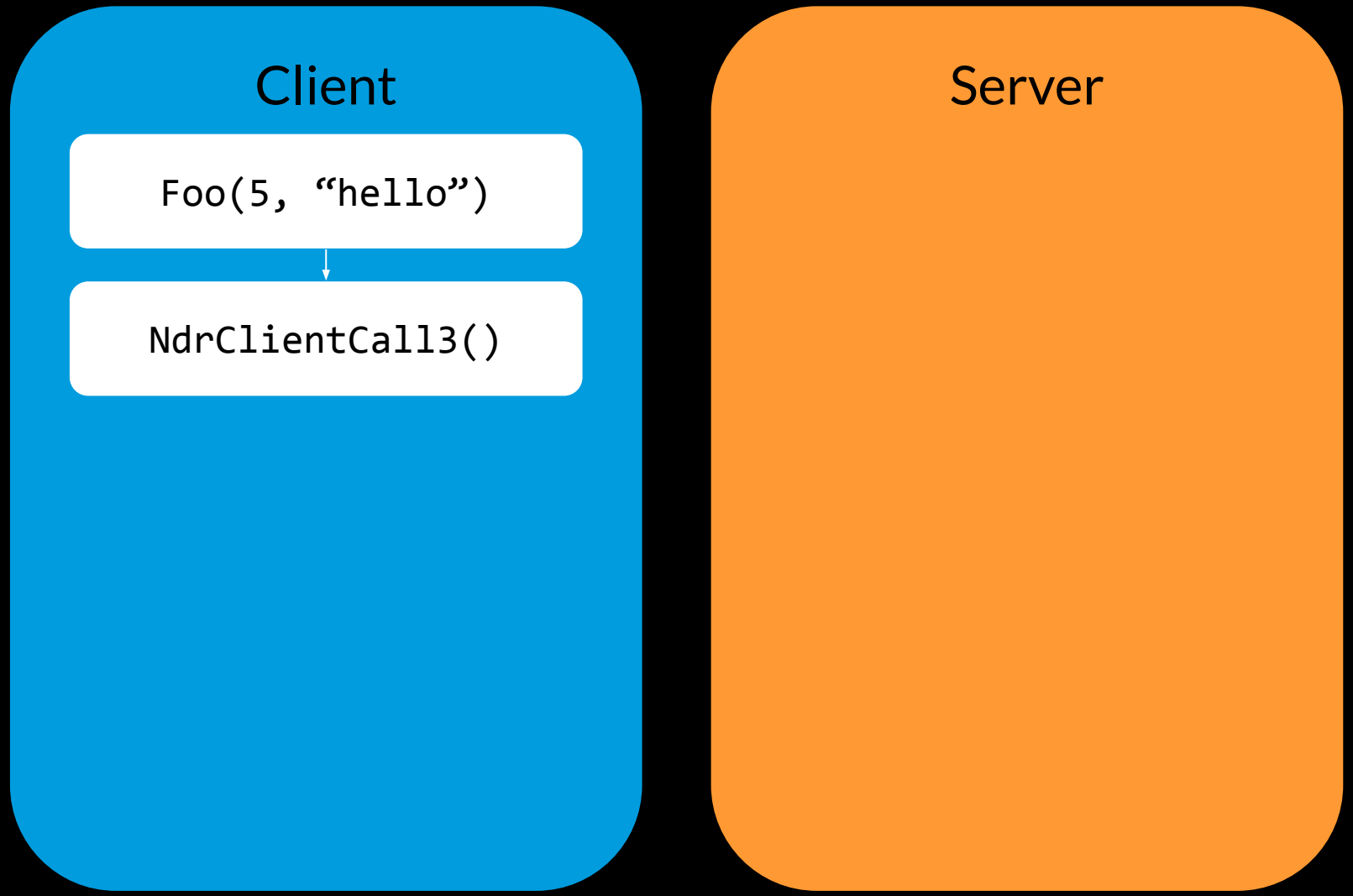
Binding

- 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)

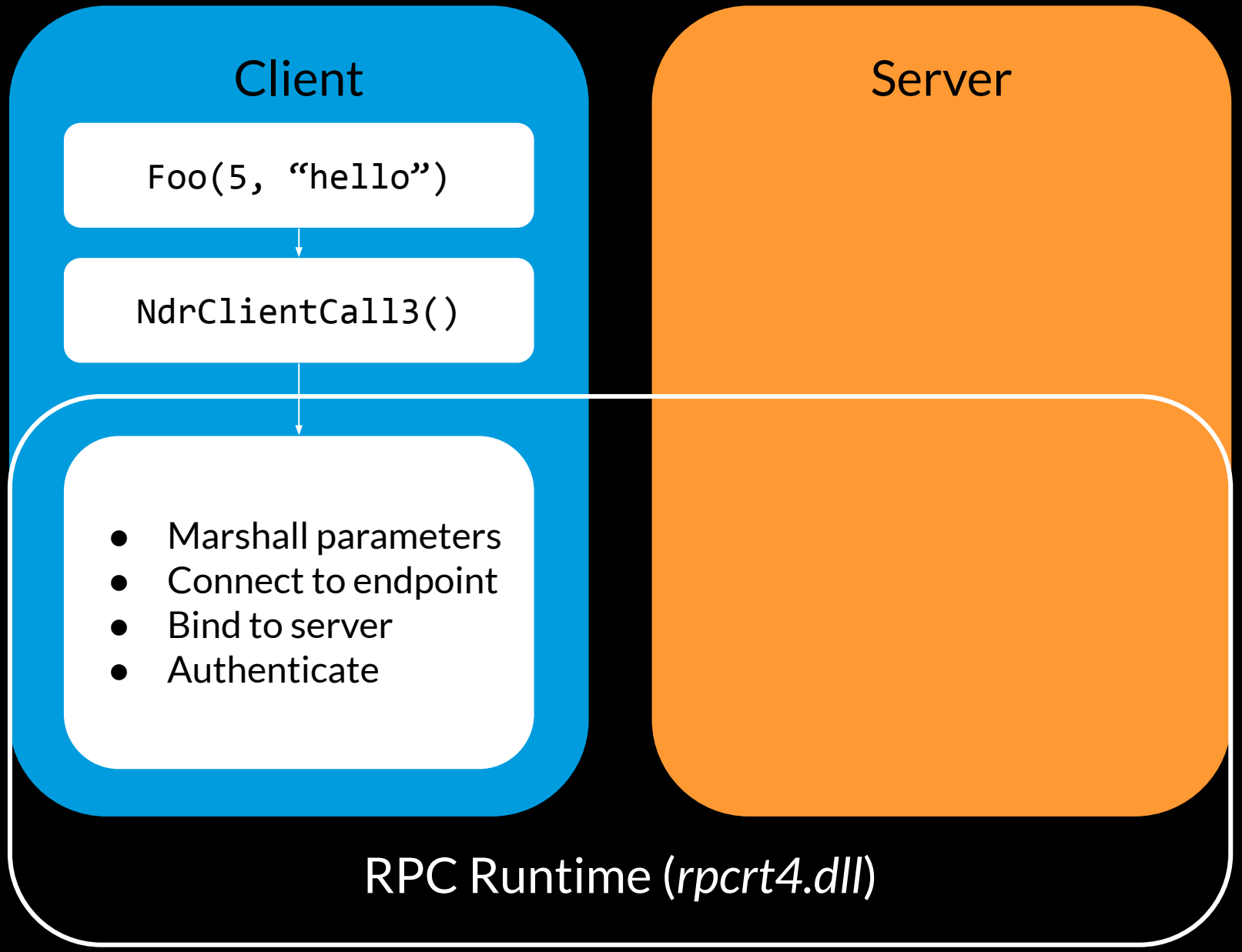
An RPC Call's Flow



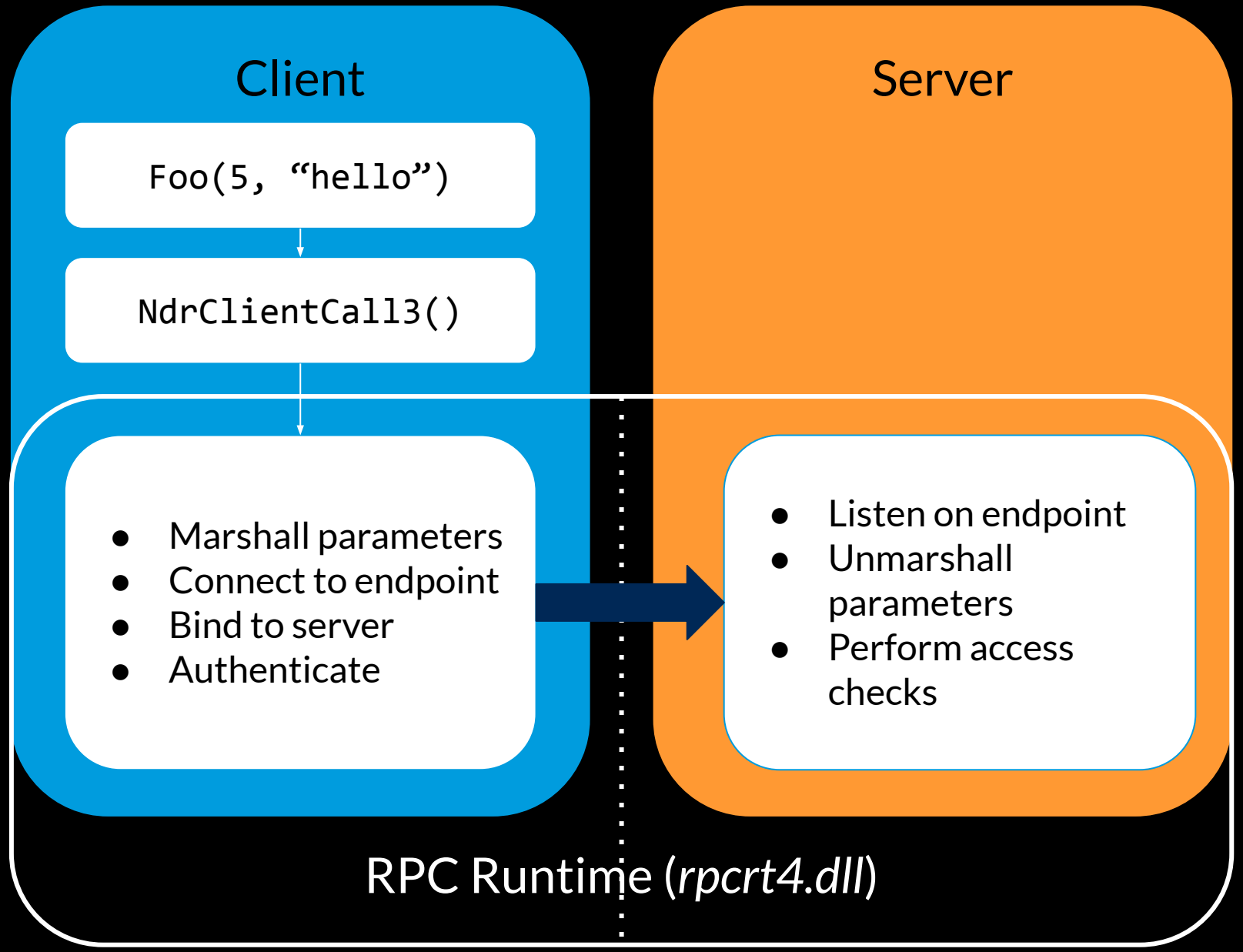
An RPC Call's Flow



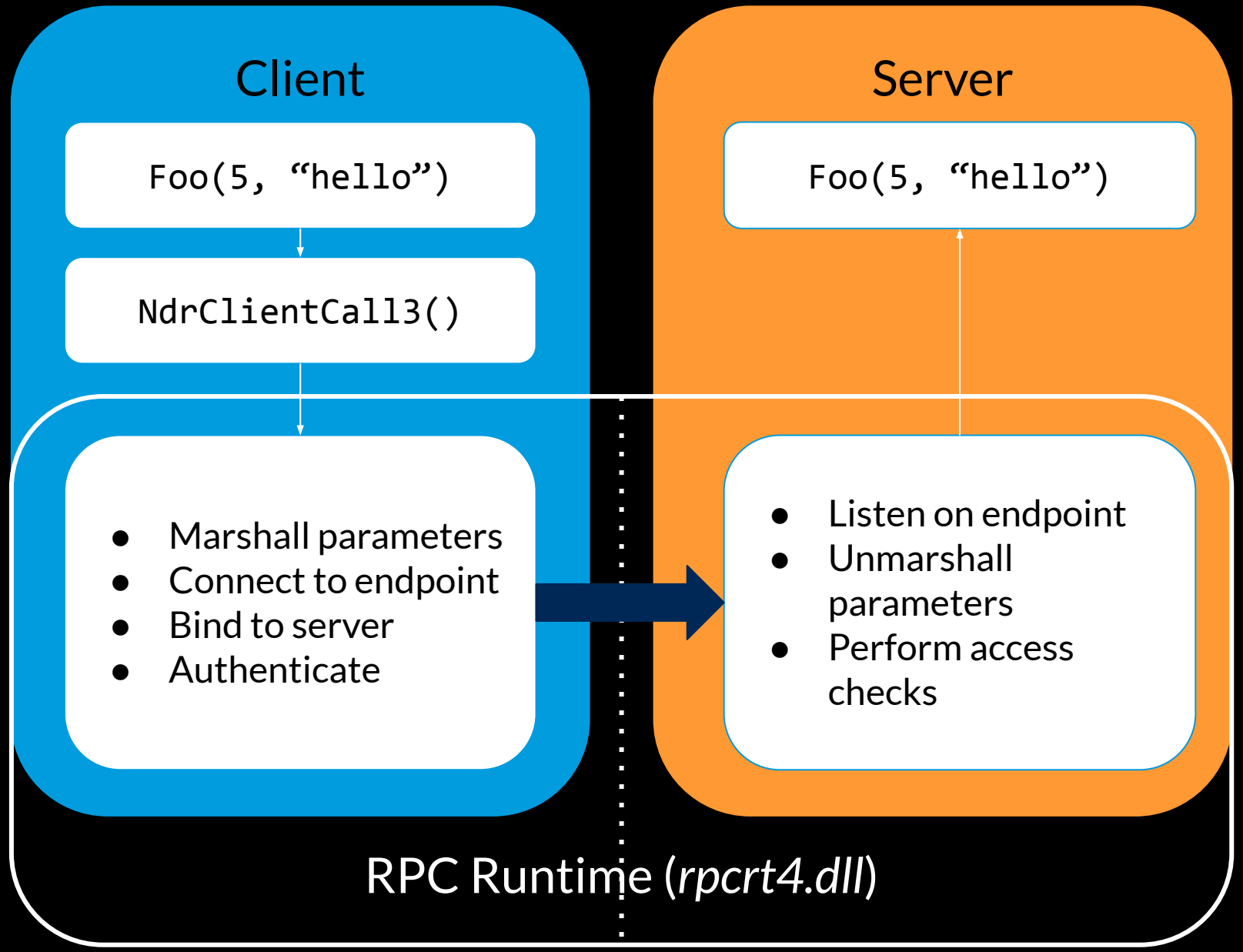
An RPC Call's Flow



An RPC Call's Flow



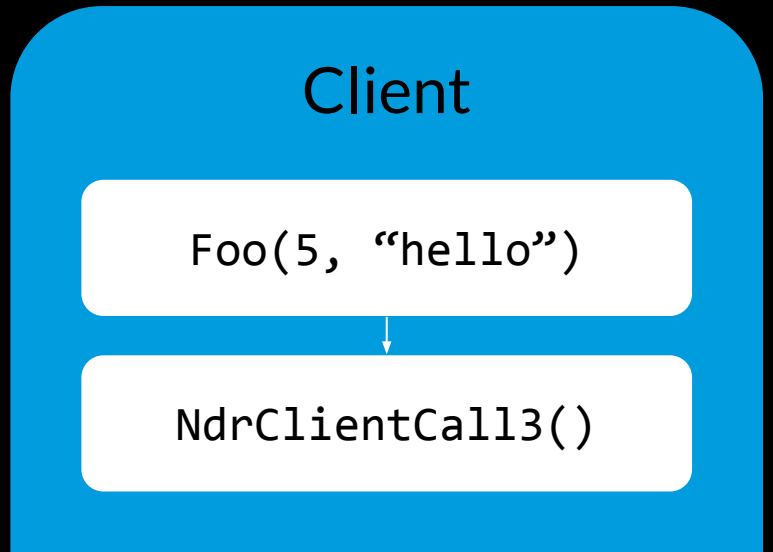
An RPC Call's Flow



Zooming In

IDL:

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



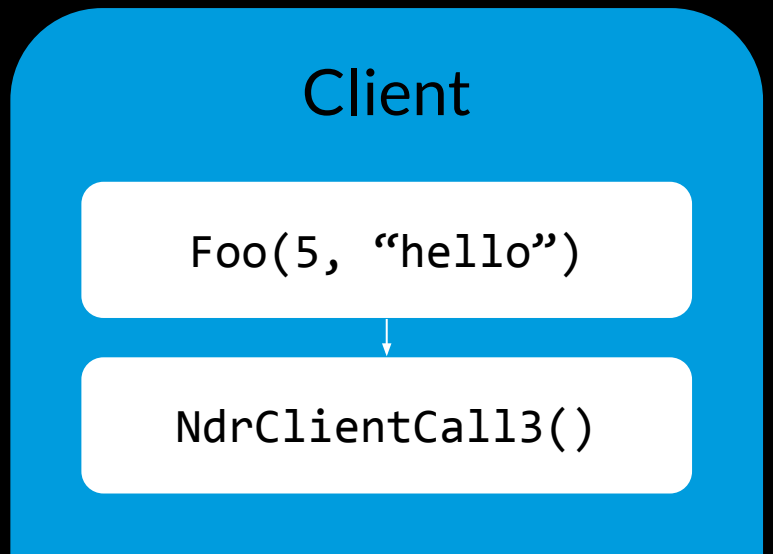
Zooming In

IDL:

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

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);
}
```



Zooming In

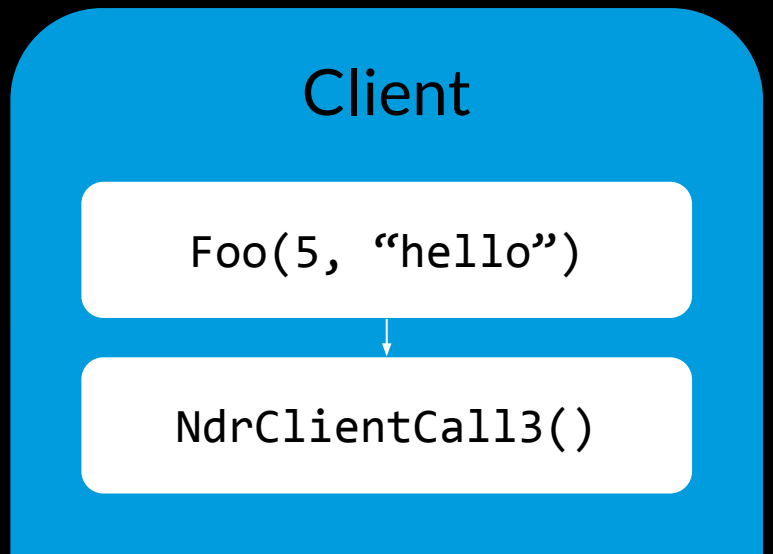
IDL:

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

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);
}
```

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

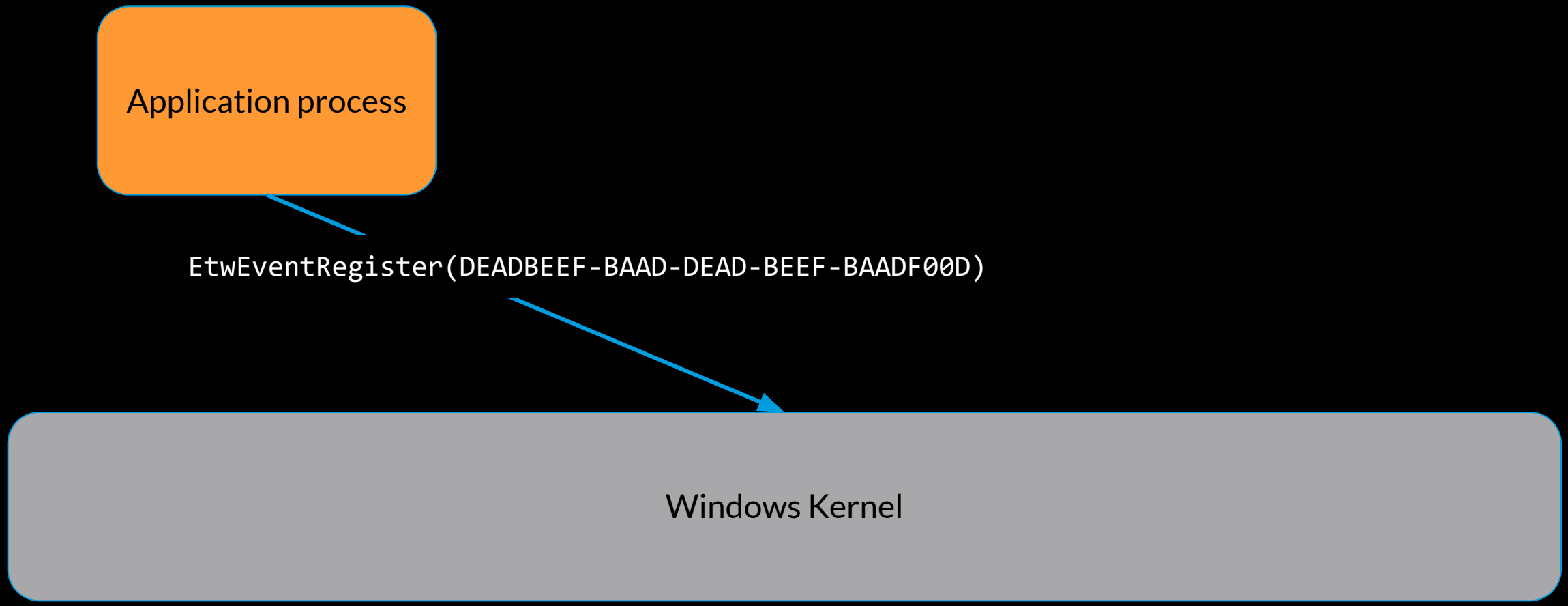
#define ETW

- 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

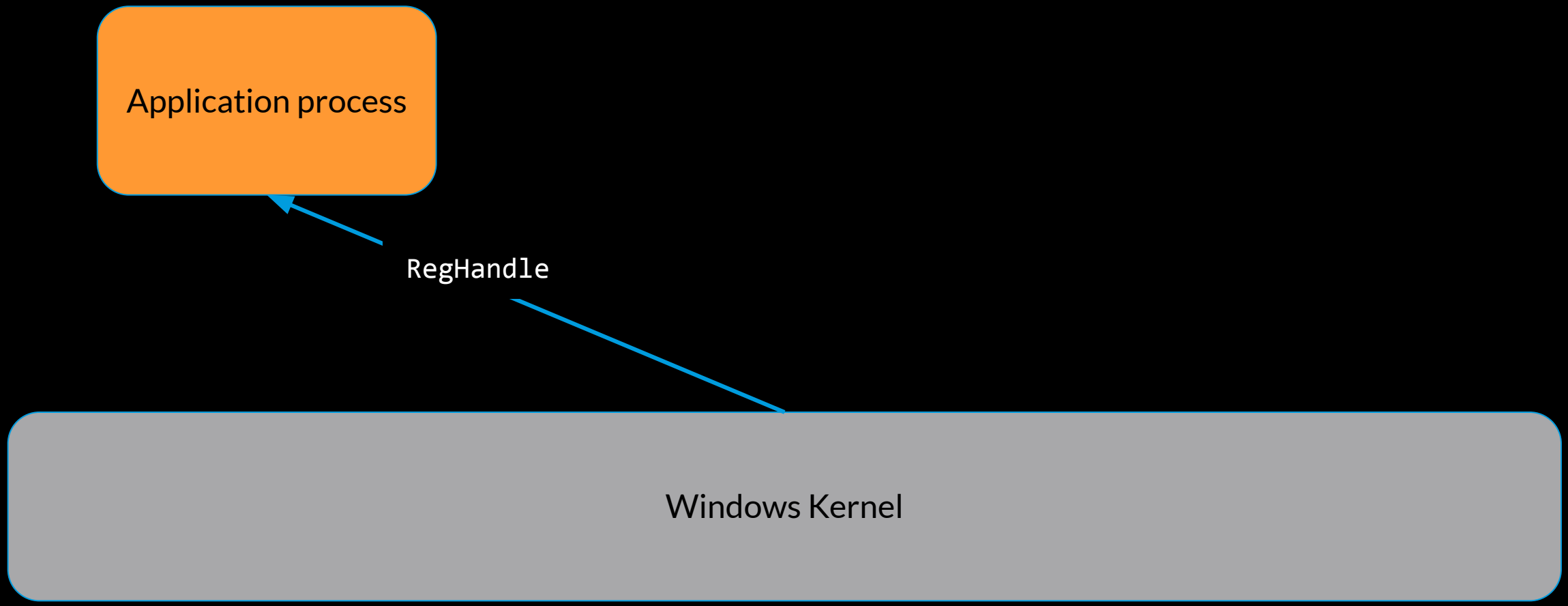
#define ETW

- 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

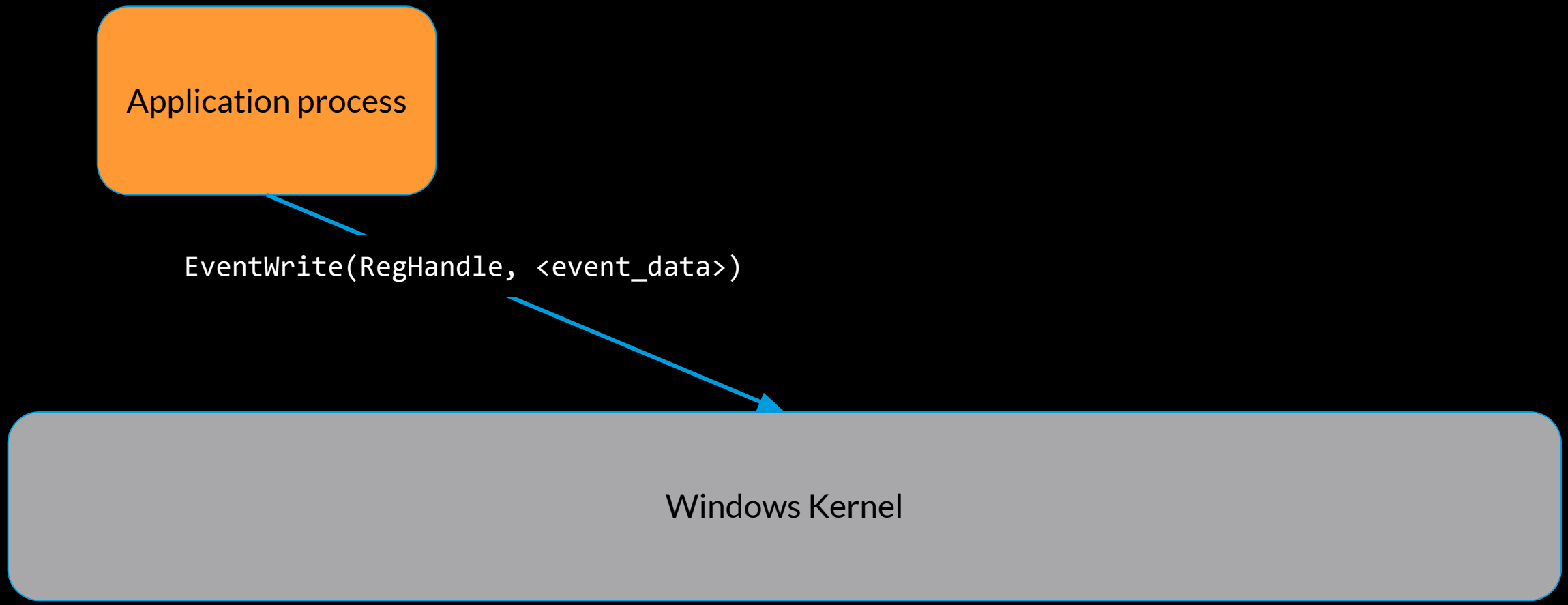
Provider <> Consumer



Provider <> Consumer



Provider <> Consumer



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

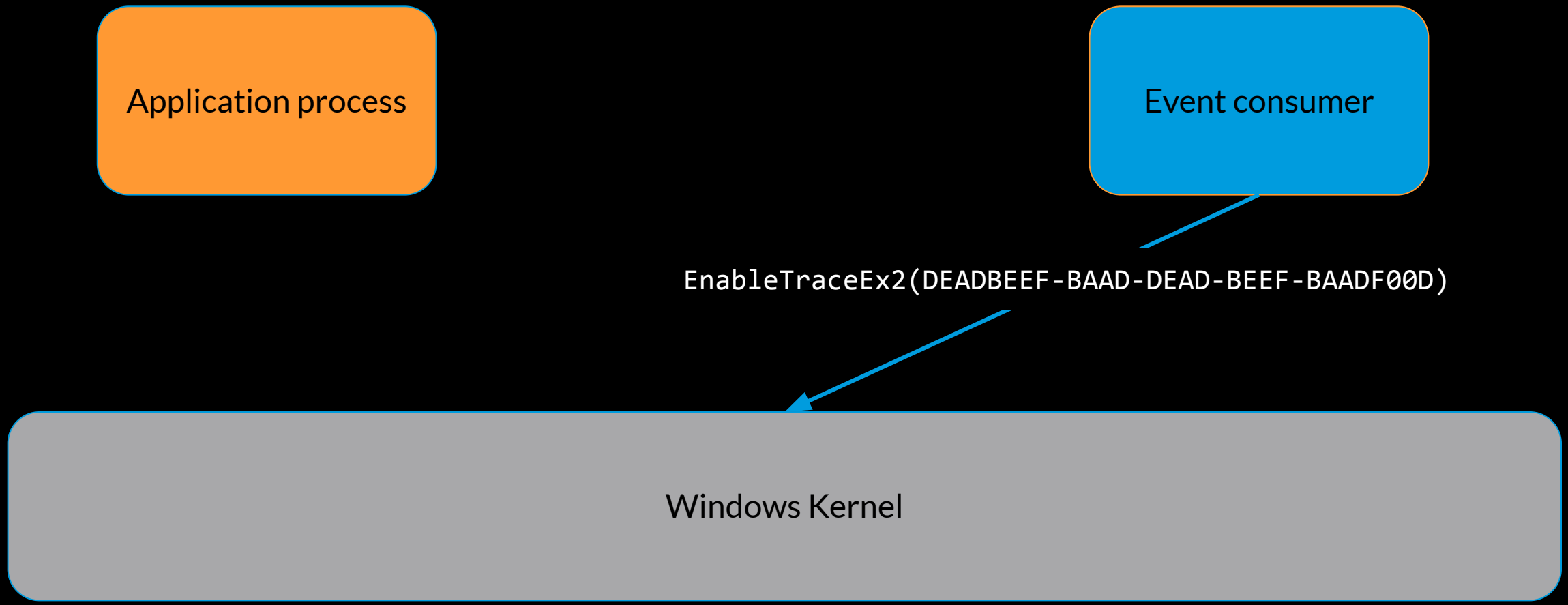
Context is Important

- 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

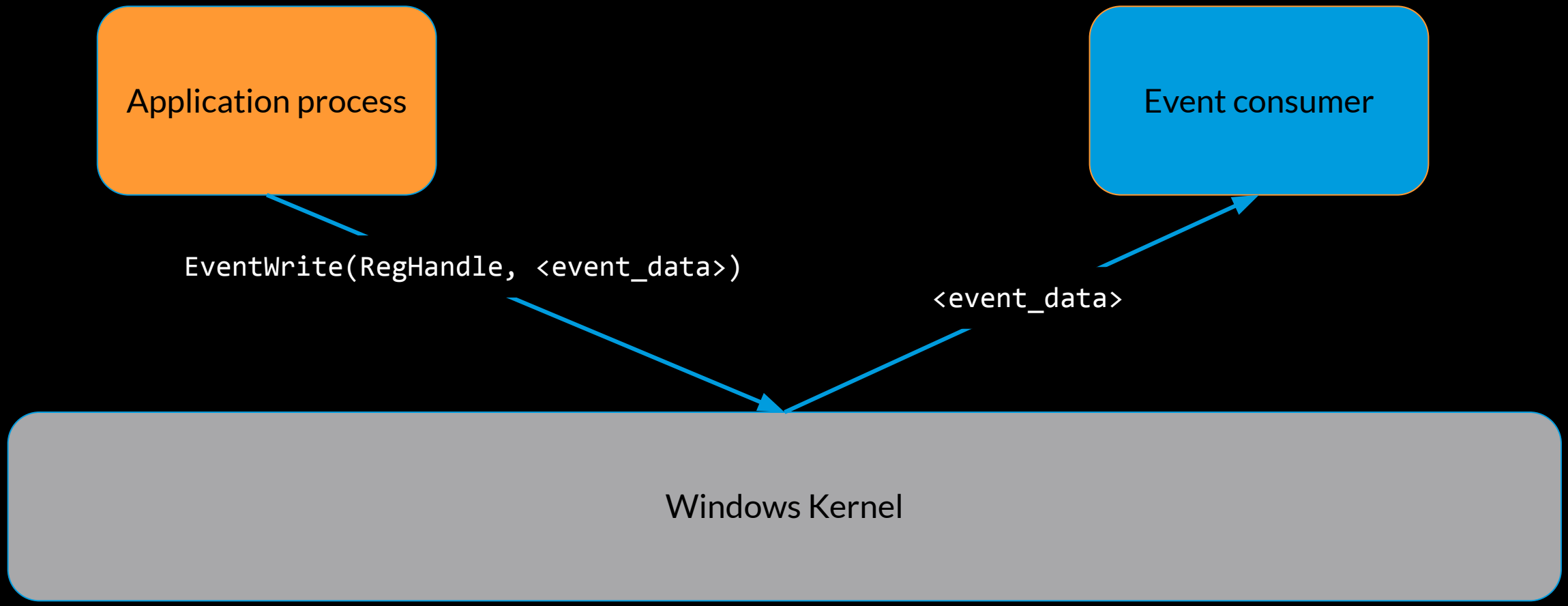
```
void Penablecallback(
    [in] LPCGUID SourceId,
    [in] ULONG IsEnabled,
    [in] UCHAR Level,
    [in] ULONGLONG MatchAnyKeyword,
    ULONGLONG MatchAllKeyword,
    [in, optional] PEVENT_FILTER_DESCRIPTOR FilterData,
    [in, optional] PVOID CallbackContext
)
```

Provider <> Consumer

Provider <> Consumer



Provider <> Consumer



RPC ETW Provider

- Microsoft-Windows-RPC[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 )
      McTemplateU0jqqzzzqqq_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 )
      McTemplateU0jqqzzzqqq_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

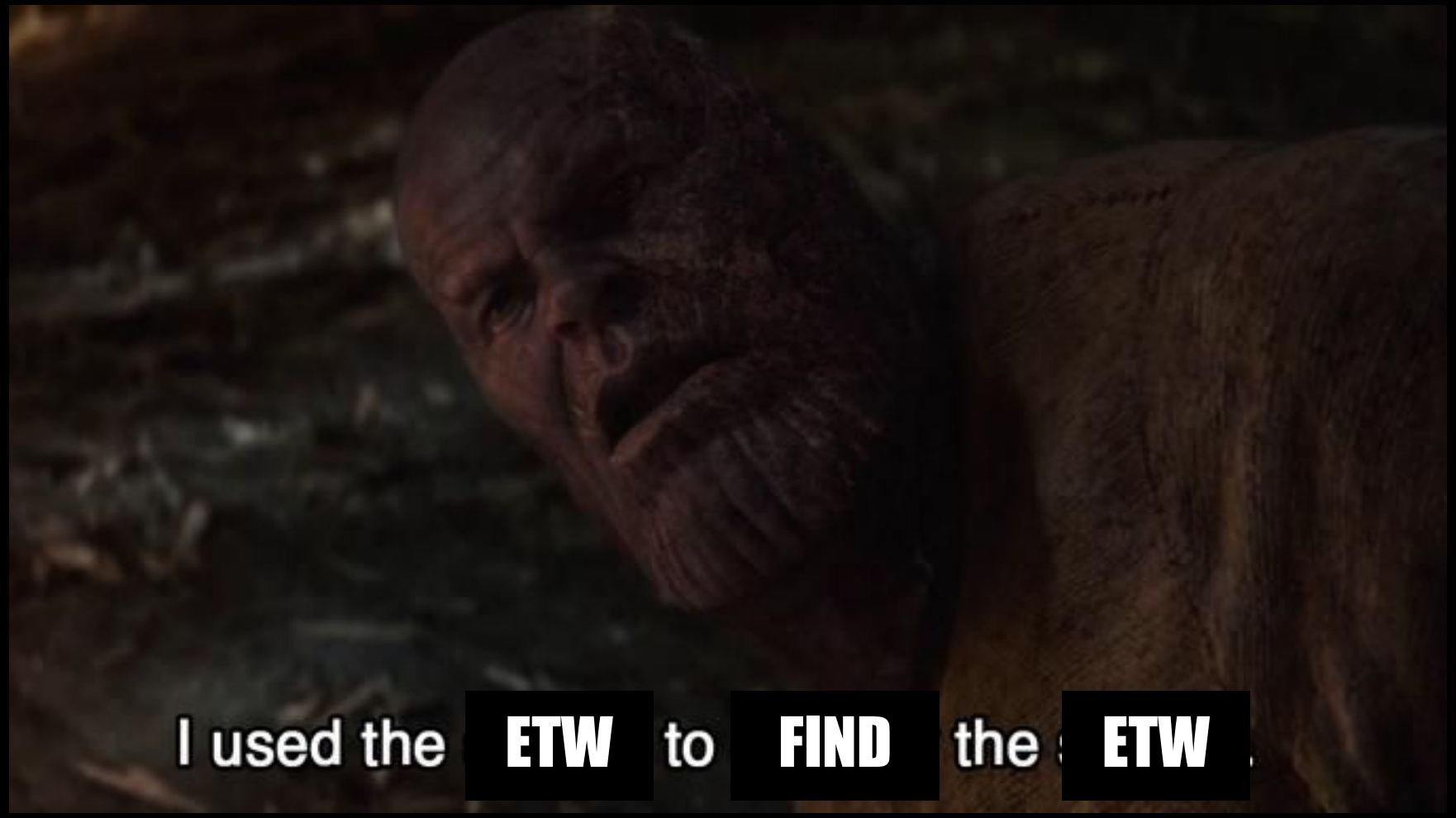
Just Do Client Calls, Dummy…

- 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

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-TCPIP.xml

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

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-TCPIP.xml

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>
```

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-SMBServer.xml

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>
```

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-SMBServer.xml

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>
```

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-SMBServer.xml

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>
```

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-SMBServer.xml

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>
```

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-SMBServer.xml

SMB ETW

event 500 — Smb2ConnectionAcceptStart

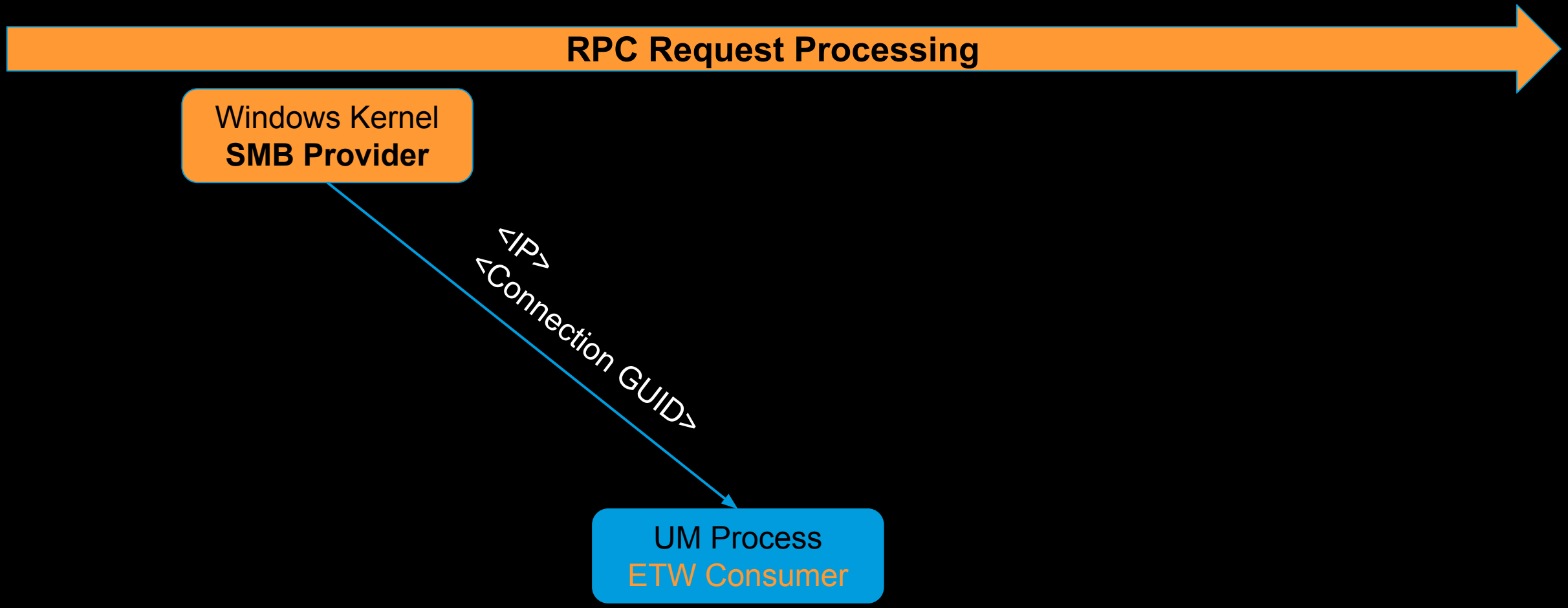
```
<template tid="Smb2ConnectionAcceptStart">
    <data name="ConnectionGUID"
          inType="win:GUID"/>
    <data name="AddressLength"
          inType="win:UInt32"/>
    <data name="Address"
          inType="win:Binary"
          length=”AddressLength”/>
</template>
```

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>
```

https://github.com/repnz/etw-providers-docs/blob/master/Manifests-Win10-17134/Microsoft-Windows-SMBServer.xml

Matching Flows

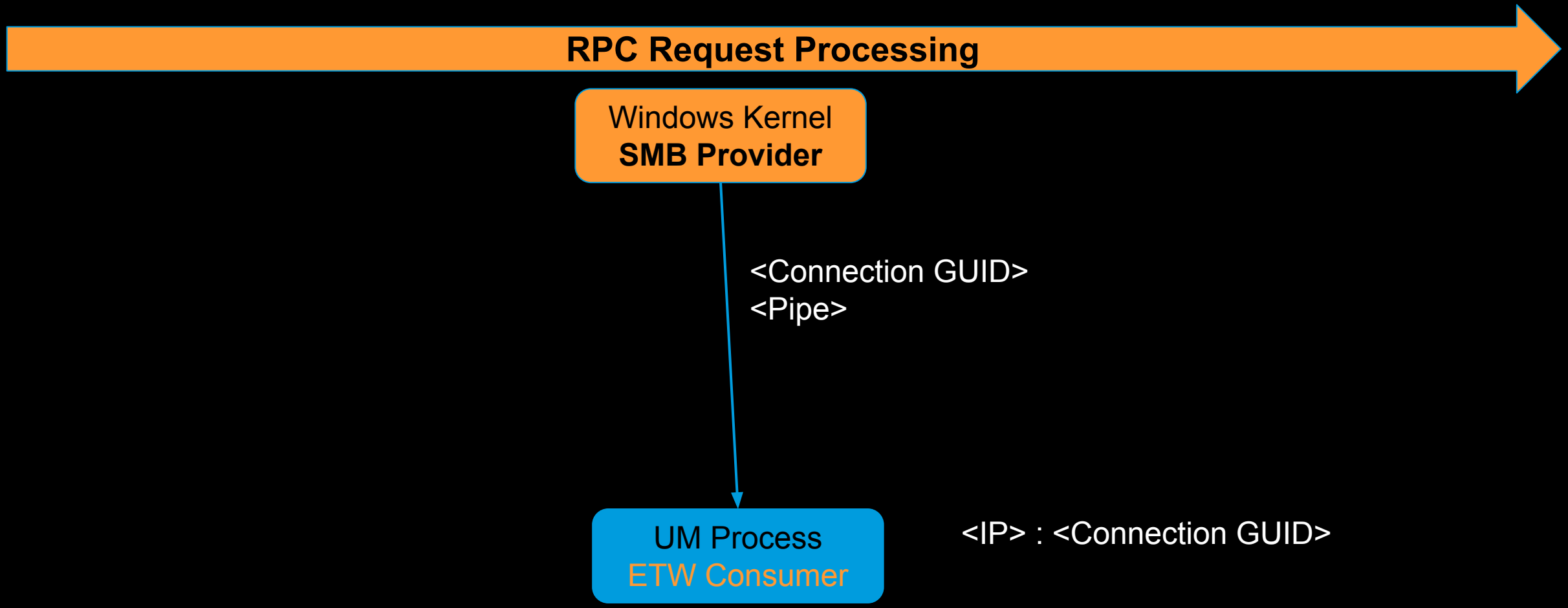


Matching Flows

UM Process
ETW Consumer

<IP> : <Connection GUID>

Matching Flows

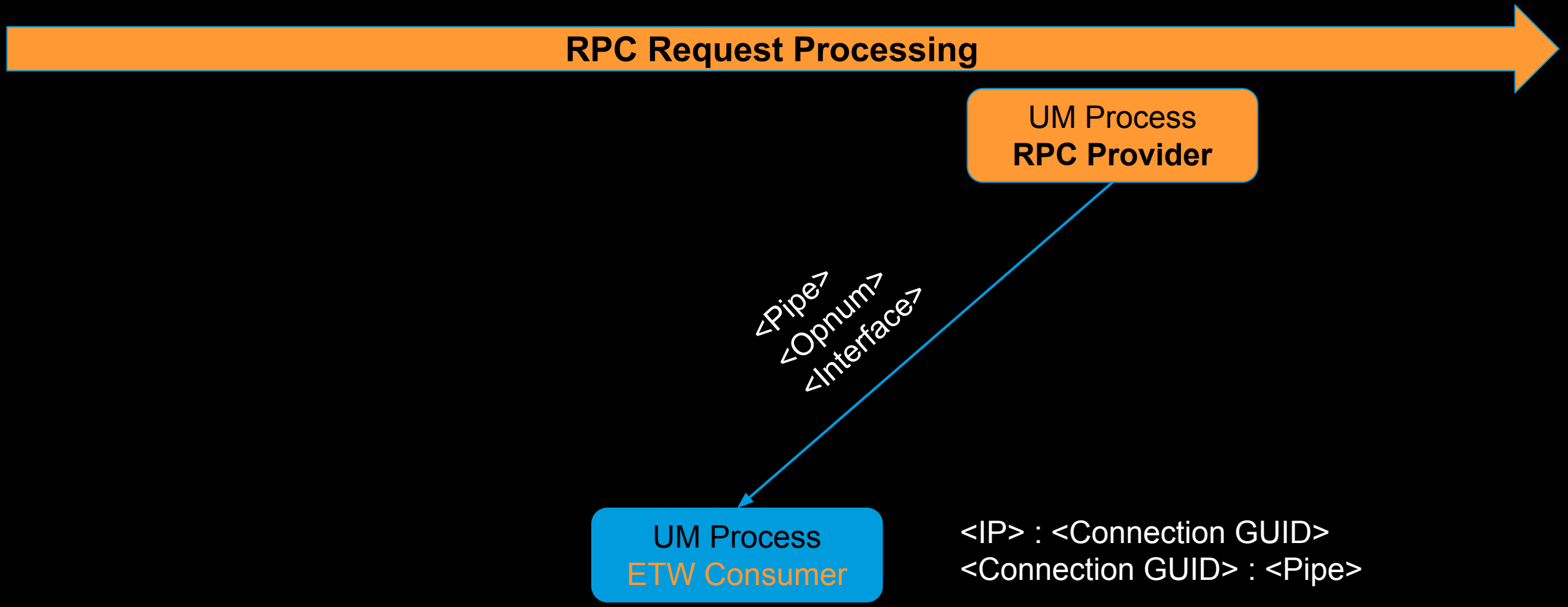


Matching Flows

UM Process
ETW Consumer

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

Matching Flows



Matching Flows

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 script[1] 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

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

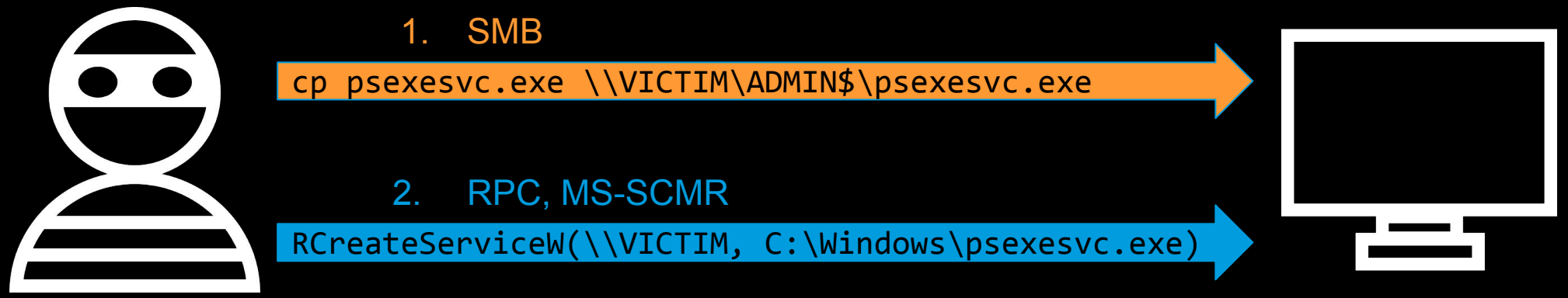
Attack Detection

Demo Time

RPC Visibility VS PSExec

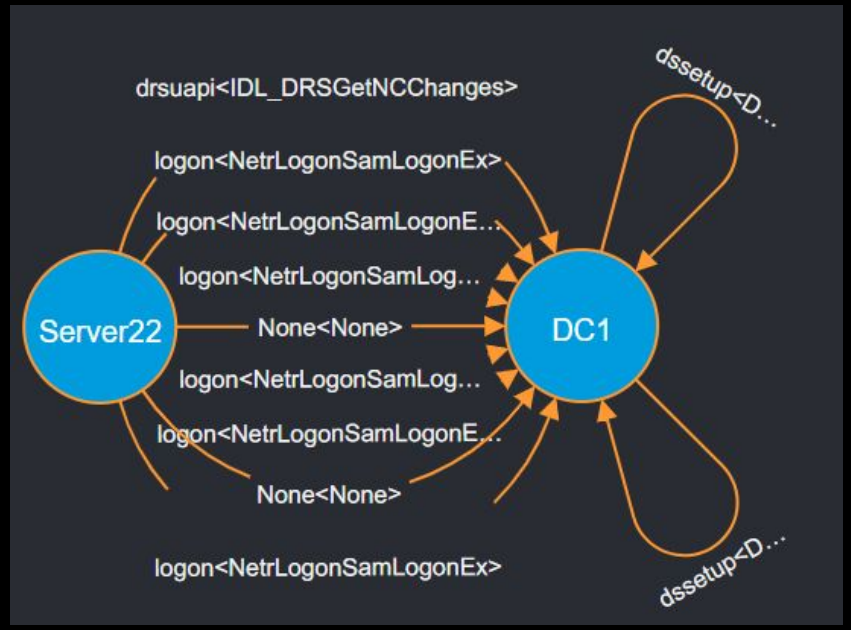
PSExec

- 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



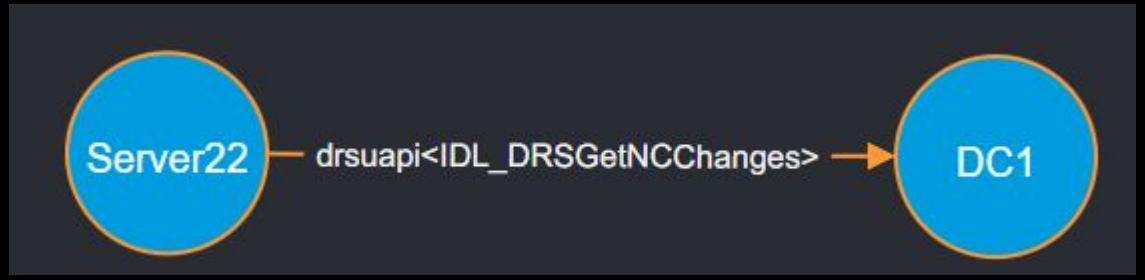
DCSync

- 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



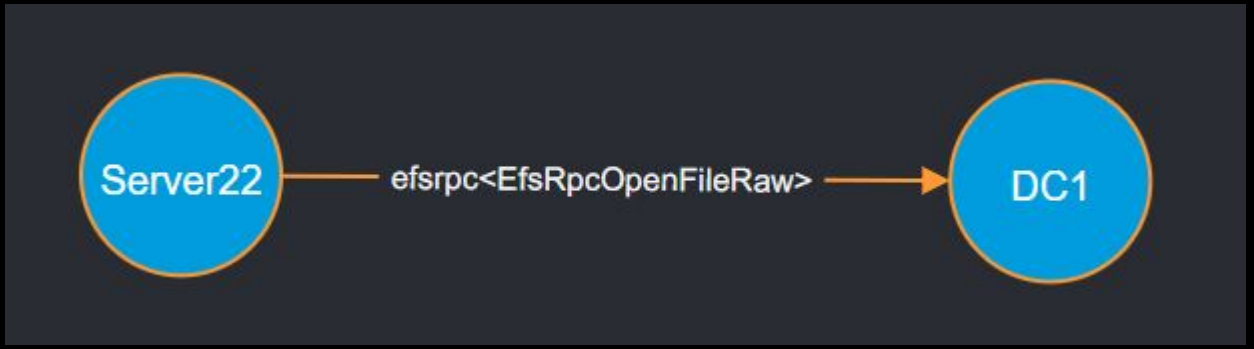
DCSync

- 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



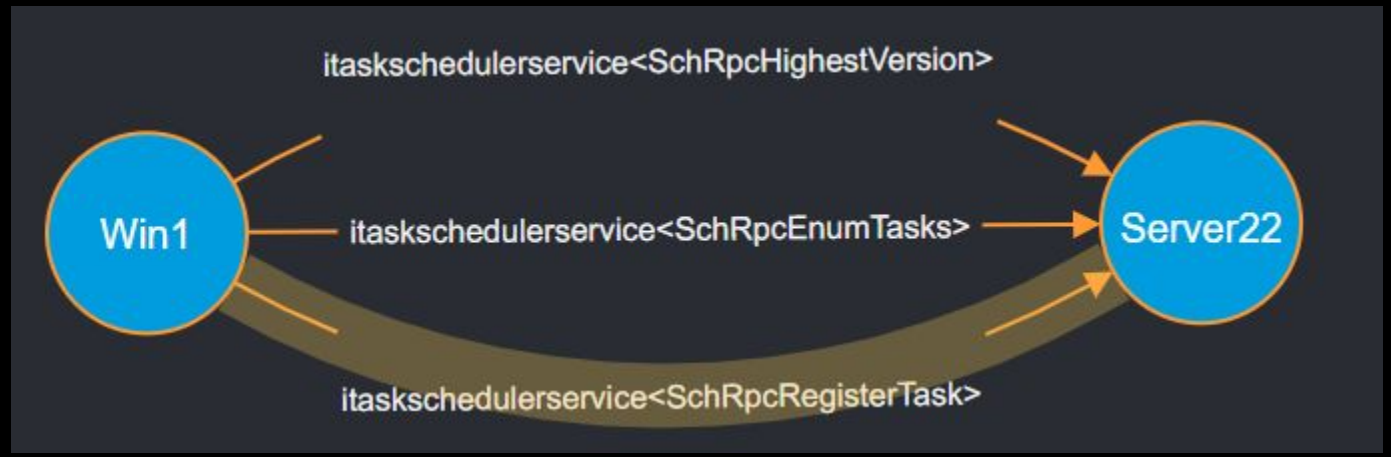
PetitPotam

- 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 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.