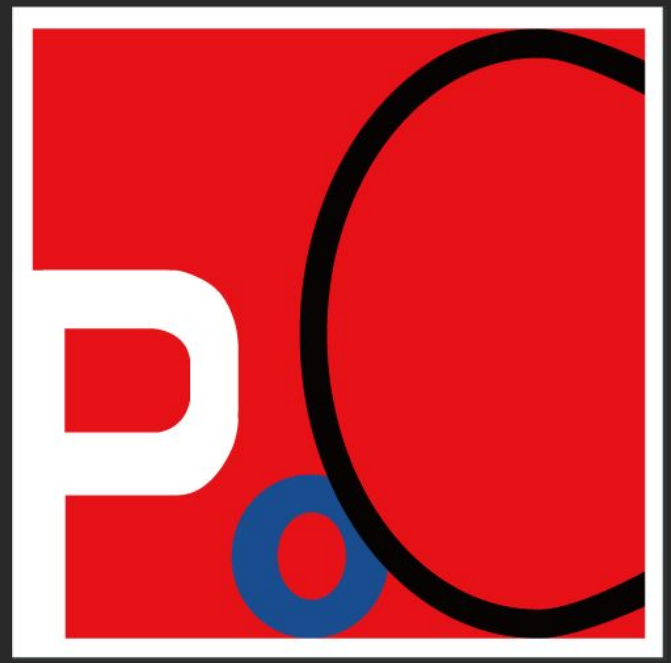
Building More Windows RPC Tooling for Security Research

James Forshaw, Google Project Zero



Who Am I?

- Researcher @ GPZ
- Specialize in Windows
- Writer of Tools!

"Never met a logical vulnerability I didn't like."

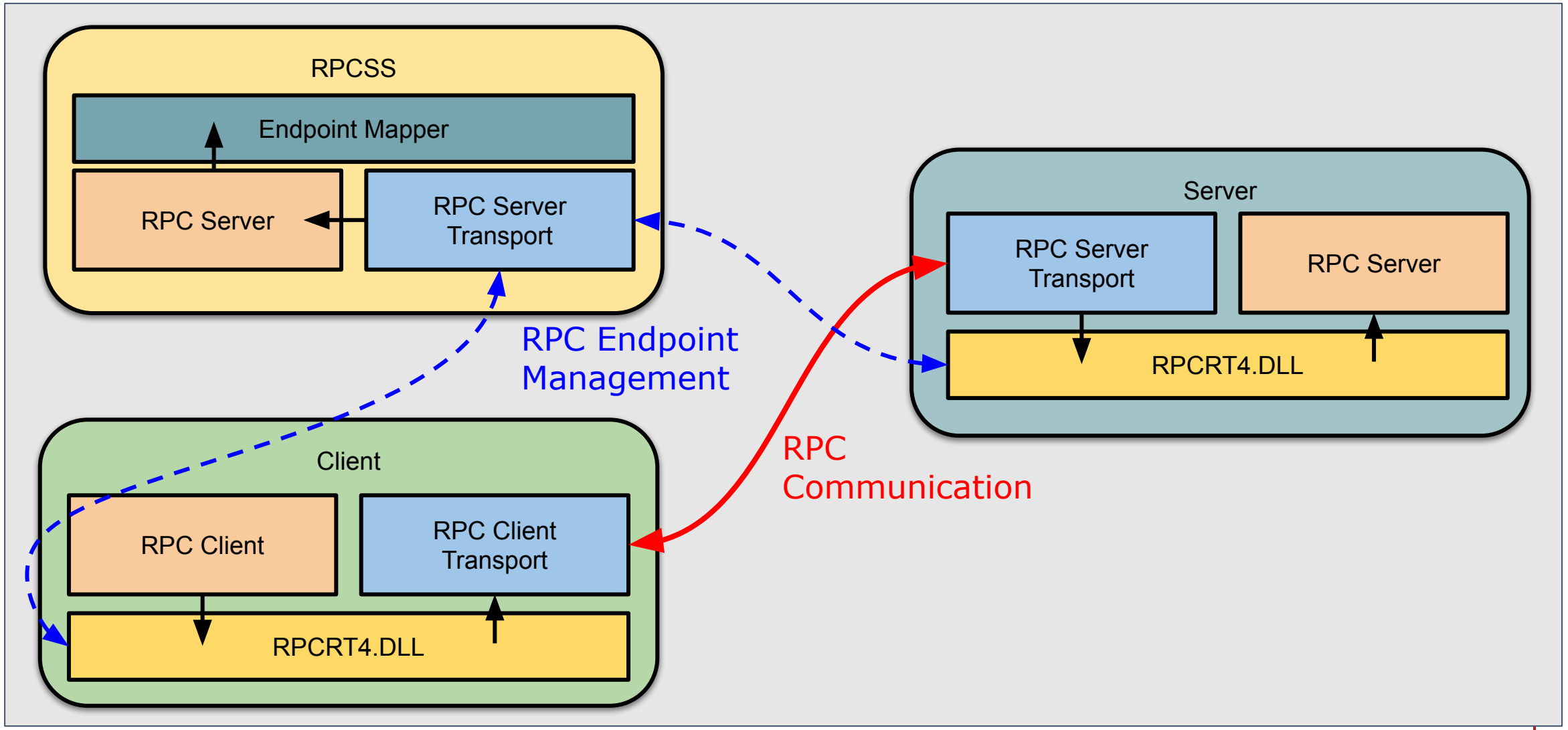
I've Been Here Before (in 2019)

James Forshaw, "Reimplementing Local RPC in .NET"/p>

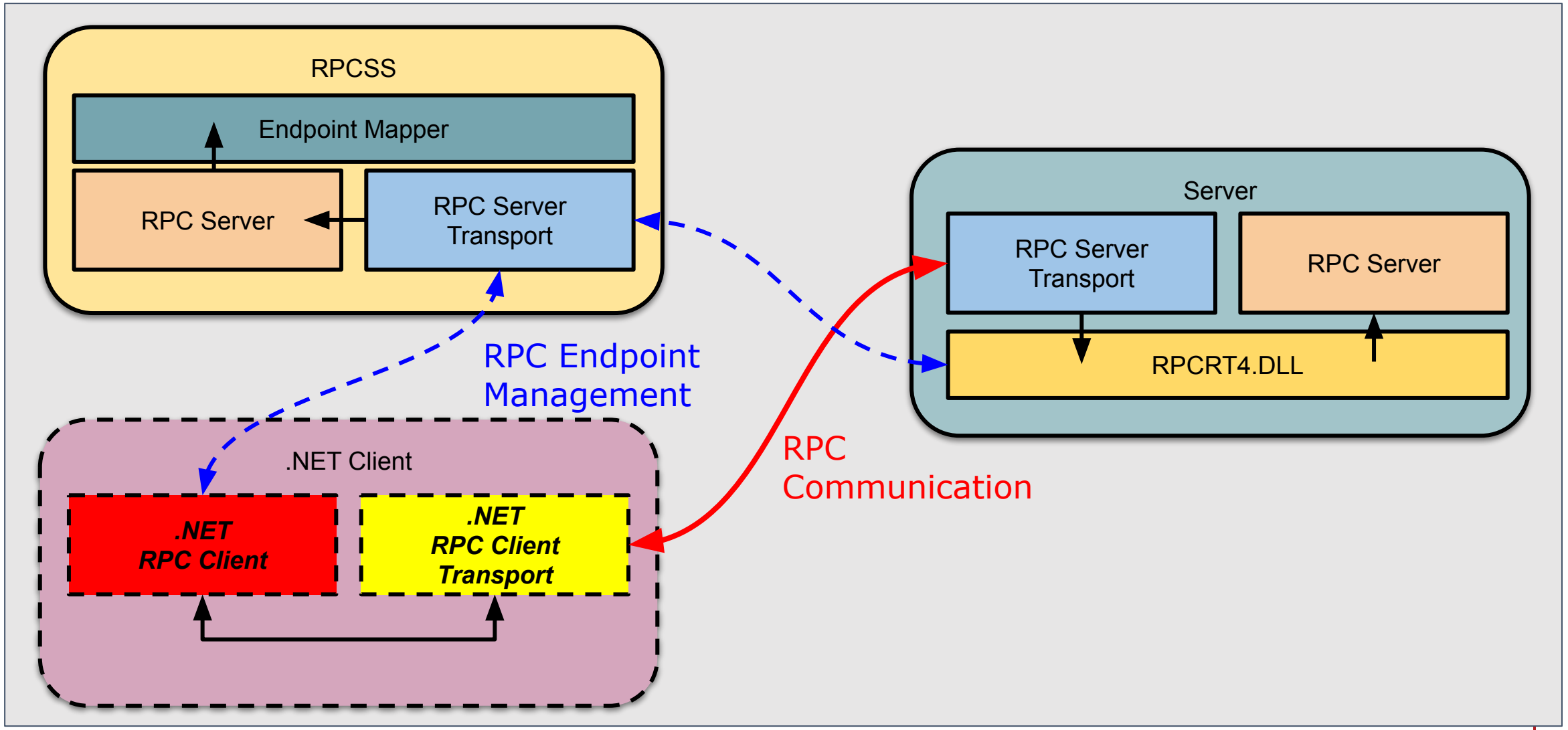
James Forshaw is a security researcher in Google's Project Zero. He has been involved with computer hardware and software security for over 10 years looking at a range of different platforms and applications. With a great interest in logical vulnerabilities he's been listed as the #1 researcher for MSRC, as well as being a Pwn2Own and Microsoft Mitigation Bypass bounty winner. He has spoken at a number of security conferences including Black Hat USA, CanSecWest, Bluehat, HITB, and Infiltrate.

[Abstract]
==========
Finding privilege escalation in local Windows RPC servers is the new hotness. Unfortunately the standard Microsoft tooling only generates code for C/C++ which presents a problem for anyone wanting to write proof-of-concepts in a .NET language such as C# or PowerShell.
This presentation will go through the various tasks I undertook to implement a working including:
- Assessing the best approaches to implementing an RPC client in .NET.
- Reverse engineering the APIs to identify the low-level ALPC implementation.
- Implementing NDR parsing and serialization.
- PowerShell Integration.
The presentation will finish up with some details one of the bugs I discovered with the new tooling. The tooling itself will be available to all.

MSRPC "Native" Architecture



MSRPC "With .NET Tooling" Architecture



Where we got to.

- Parsed NDR "bytecode" from server executables
- Generate pseudo code to inspect parsed NDR
- Implemented transport for ALPC
- Generate C# RPC clients on the fly from bytecode
- Found various bugs

Carve out the RPC services in RPCSERVER.EXE

```
PS> $rpc_server = Get-RpcServer "rpcserver.exe"
```

Format an RPC endpoint as pseudo C#.

```
PS> $rpc_server | Format-RpcServer
```

Generate and compile a client for a parsed RPC server.

```
PS> $client = Get-RpcClient $rpc_server
```

Connect an RPC client, try and lookup ALPC port from Endpoint Mapper

```
PS> Connect-RpcClient $client
```

Connect an RPC client, try and lookup ALPC port by brute force.

```
PS> Connect-RpcClient $client -FindAlpcPort
```

Demo

"Possible" Future Work

- Implement parsing NDR64 byte code and NDR64 wire format.
- Support Pipes and some misc other types.
- Implement asynchronous support.
- Implement transports for Named Pipes and TCP.
- Add server support.

"Possible" Future Work

- Implement parsing NDR64 byte code and NDR64 wire format.
- Support Pipes and some misc other types.
- ~~Implement asynchronous support.~~
- Implement transports for Named Pipes and TCP
  - *Hyper-V sockets*
- ~~Add server support.~~
- *Linux and macOS support*
- *Improved RPC Server Discovery*
- *IDL file output*

NDR64

NDR Was Mostly Documented

RPC NDR Format Strings

NDR Engine: 32-bit Interpreter

This document describes the format string descriptors, sometimes referred to as MOPs, for the 32-bit NDR engine. This section describes changes associated with the evolution from the **–Oi** interpreter to the **–Oif** interpreter layer, as well as additions related to pipes and asynchronous support.

This document describes current Microsoft Interface Definition Language (MIDL) technology from the engine perspective, and the current NDR engine.

https://learn.microsoft.com/en-us/windows/win32/rpc/rpc-ndr-format-strings

Interface Definition Language (IDL)

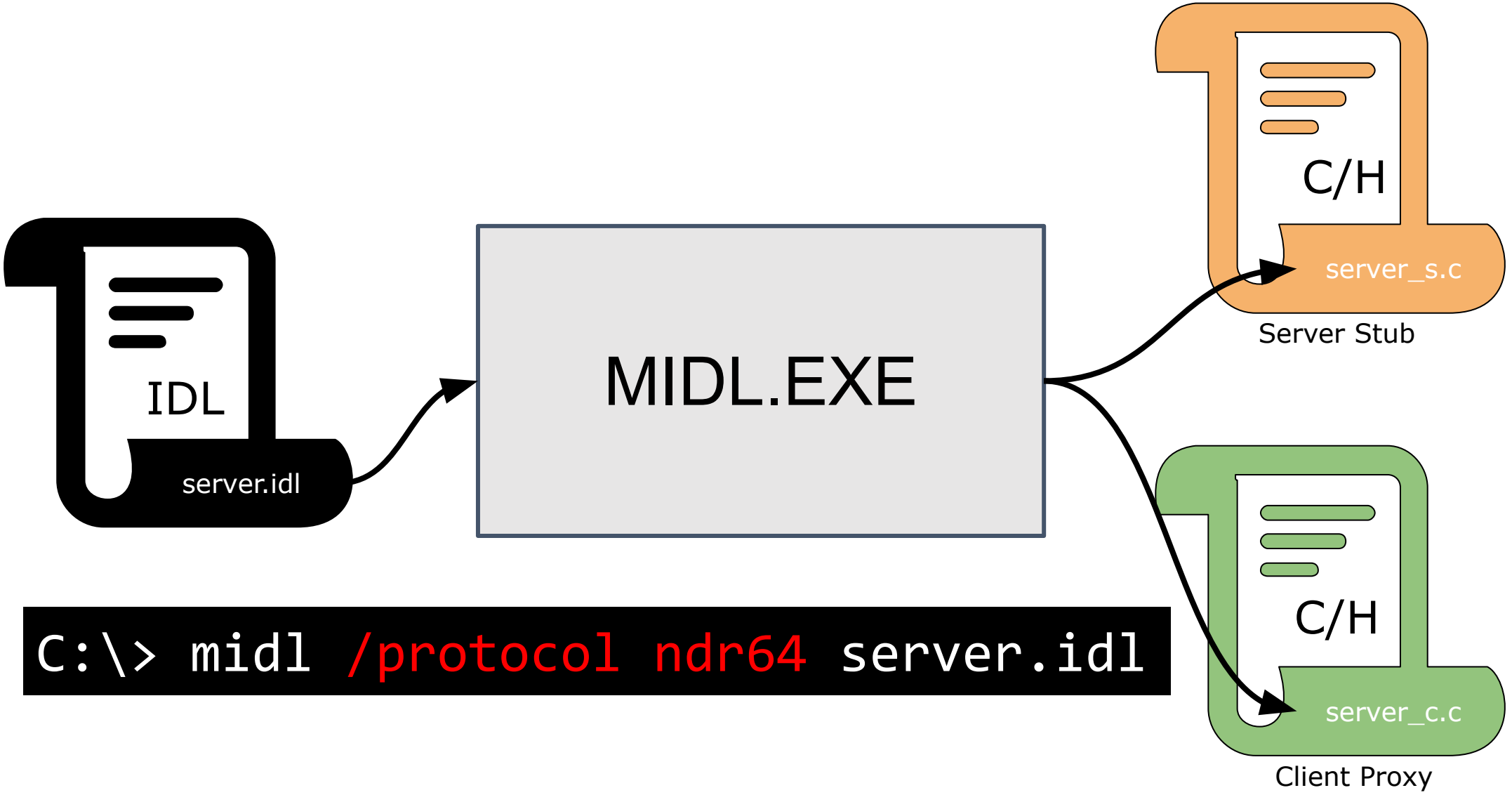
```
typedef struct _MYSTRUCT {
    DWORD a;
    [string] const wchar_t* b;
} MYSTRUCT;

[
    uuid(4870536E-23FA-4CD5-9637-3F1A1699D3DC),
    version(1.0),
]
interface RpcServer
{
    int Func([in] handle_t hBinding, [out] MYSTRUCT* abc);
}
```

RPC Interface GUID and Version

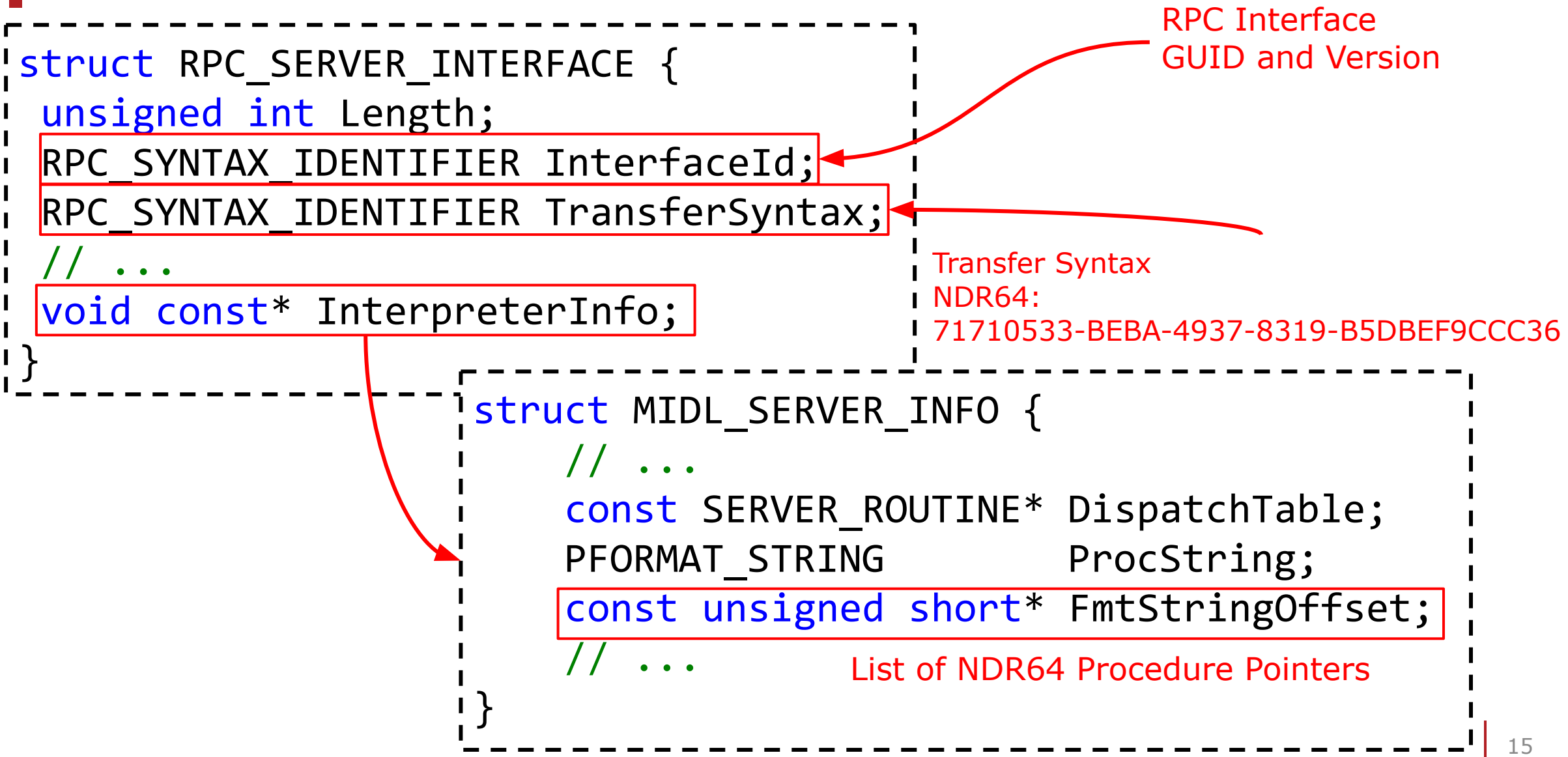
RPC functions

MIDL Compiler

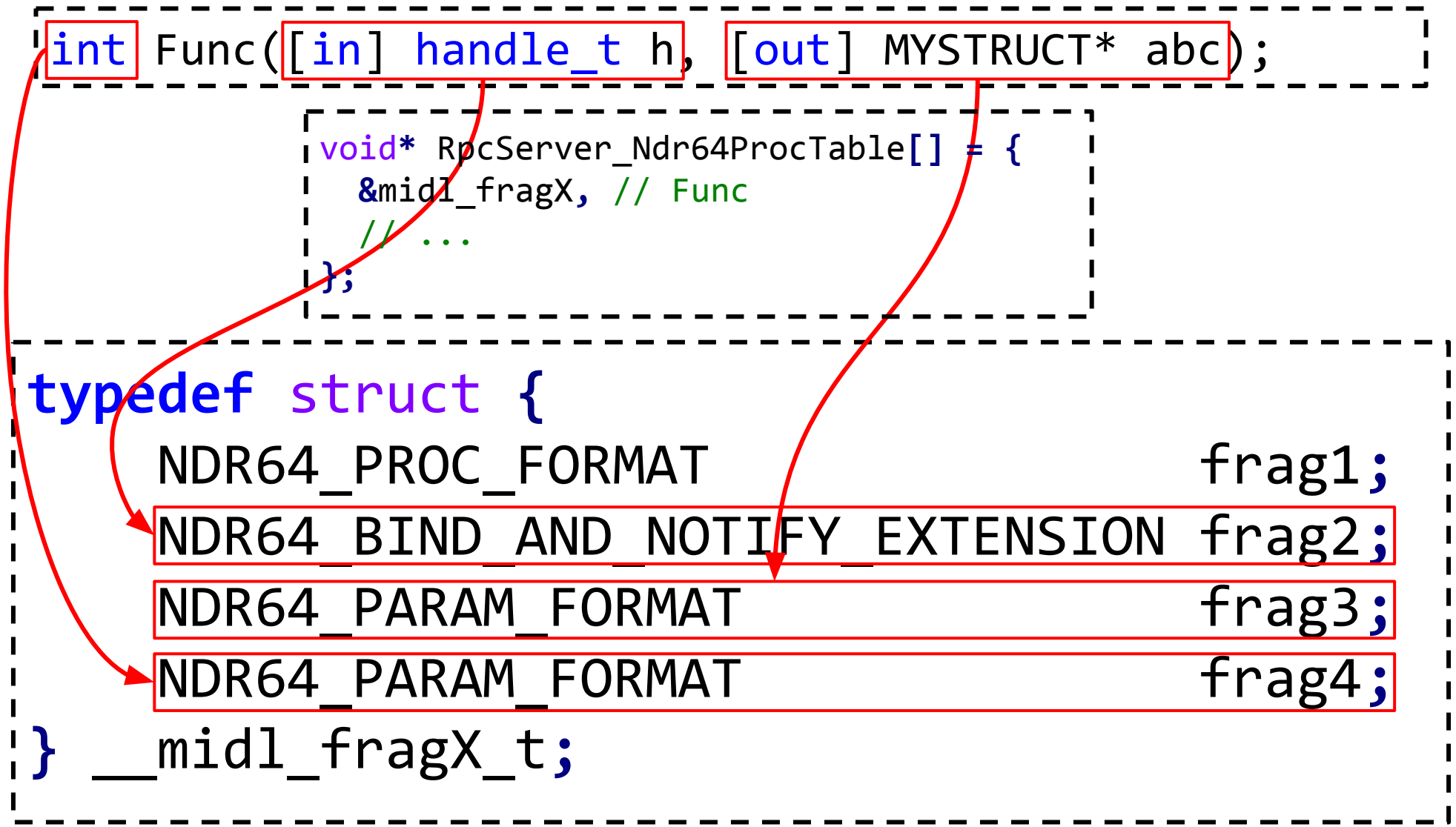



```
C:\> midl /protocol ndr64 server.idl
```

Auto-generated Server Definition Information



Example NDR64 Bytecode Structure



Procedure Parameters

```
struct NDR64_PARAM_FORMAT {
    PNDR64_FORMAT       Type;
    NDR64_PARAM_FLAGS   Attributes;
    NDR64_UINT16        Reserved;
    NDR64_UINT32        StackOffset;
};
```

in/out etc.

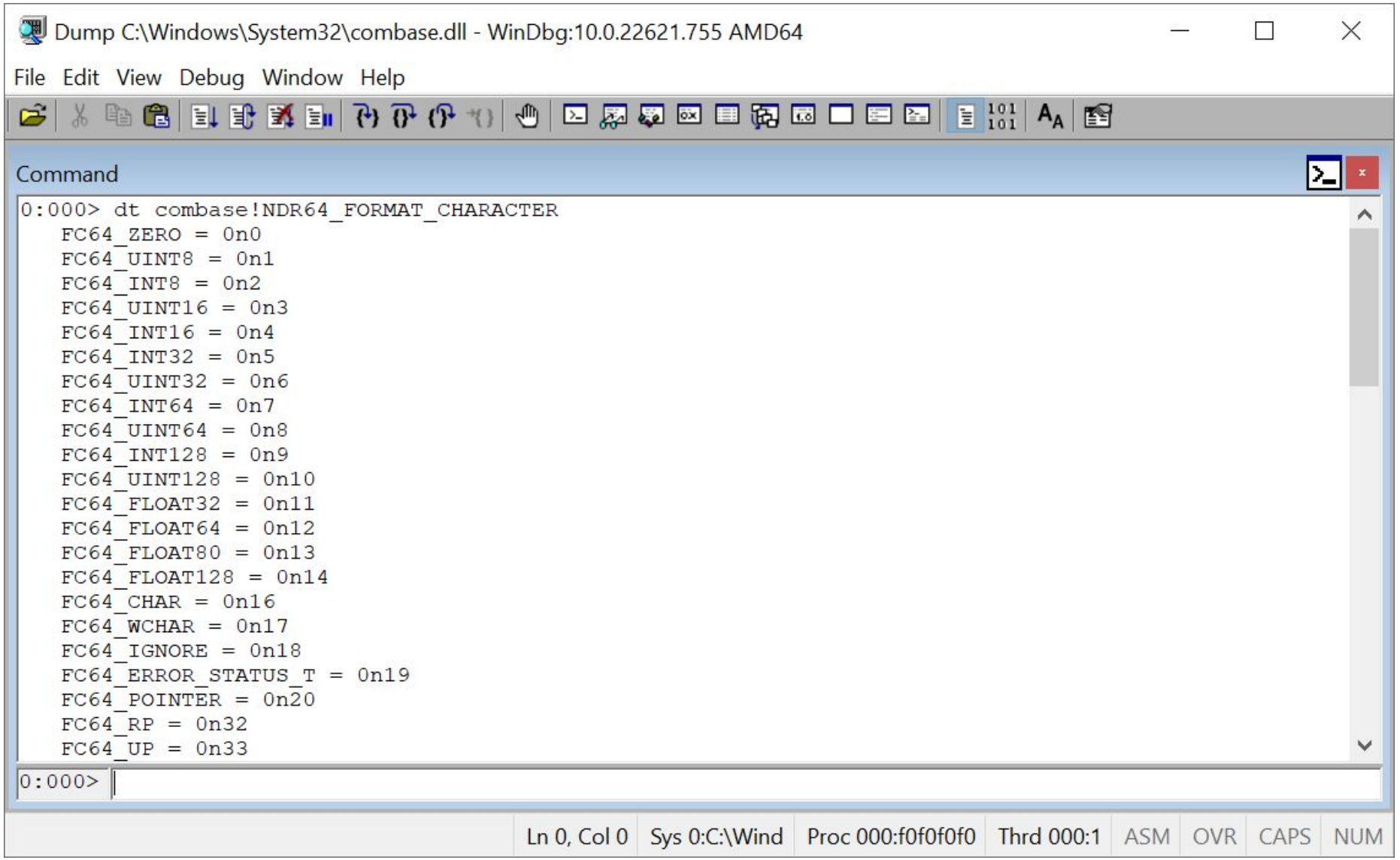
```
const NDR64_FORMAT_CHAR __midl_fragY
    = 0x5 /* FC64_INT32 */;
```

Not in any headers!

```
struct __midl_fragZ_t {
    NDR64_BOGUS_STRUCTURE_HEADER_FORMAT frag1;
    NDR64_SIMPLE_MEMBER_FORMAT          frag2;
    NDR64_MEMPAD_FORMAT                 frag3;
    NDR64_SIMPLE_MEMBER_FORMAT          frag4;
};
```

In "ndr64types.h"

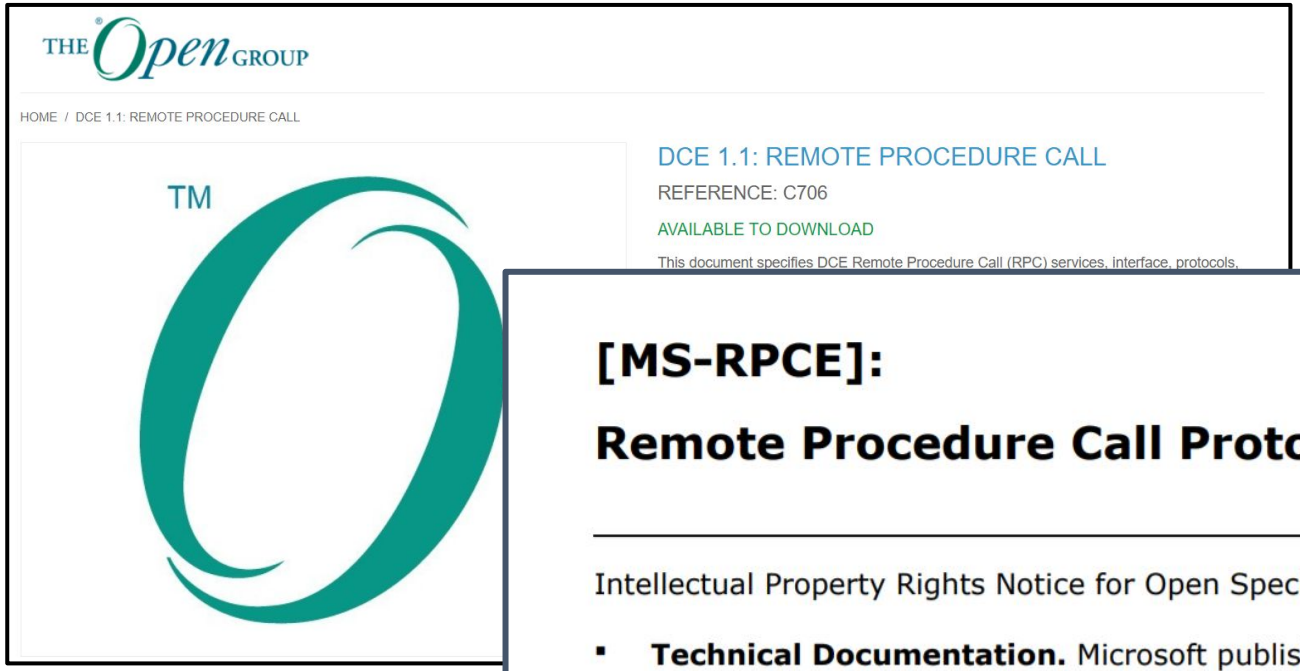
Private COM Symbols Saves the Day (Again!)



Get Parsed NDR64 Procedures

```
PS> $rpc_server = Get-RpcServer "rpcserver.exe"
PS> $rpc_server.Ndr64Procedures
Name                : Func
Params              : {_hProcHandle, ...}
ReturnValue         : FC64_INT32 ...
Handle              : FC64_BIND_PRIMITIVE ...
...
```

Implementing NDR64 Wire Protocol



[MS-RPCE]:

Remote Procedure Call Protocol Extensions

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.

Demo

Implementing OSF Transports

OSF is Slightly Different from ALPC Transport

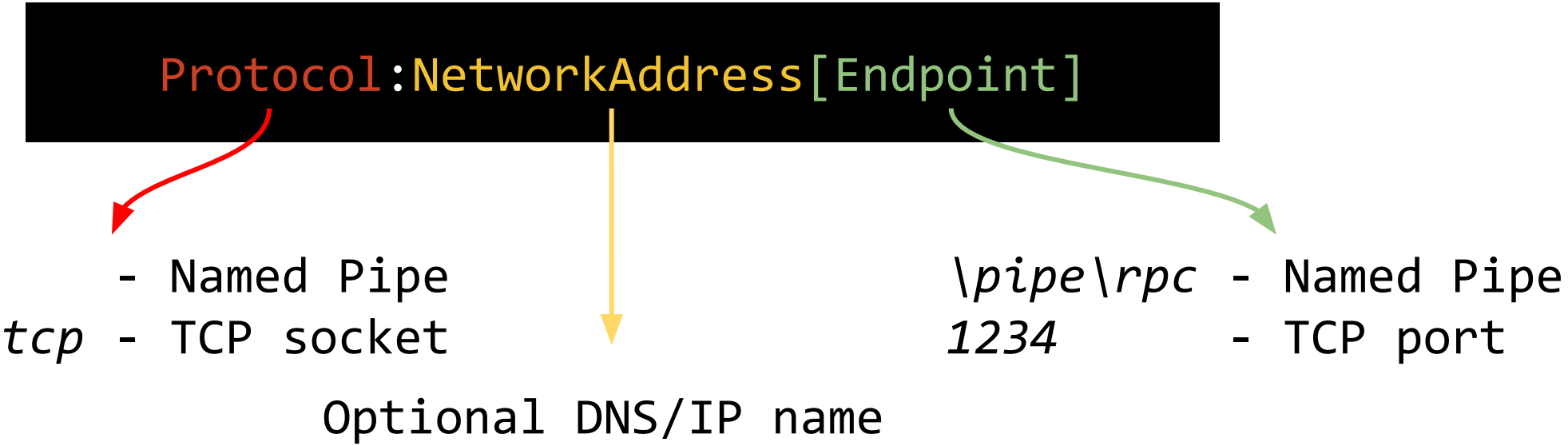
ALPC

```
struct LRPC_BIND_MESSAGE {
  LRPC_HEADER Header;
  int RpcStatus;
  RPC_SYNTAX_IDENTIFIER Interface;
  TransferSyntaxSetFlags TransferSyntaxSet;
  short DceNdrSyntaxIdentifier;
  short Ndr64SyntaxIdentifier;
  short FakeNdr64SyntaxIdentifier;
  bool RegisterMultipleSyntax;
  bool UseFlowId;
  long FlowId;
  int ContextId;
}
```

OSF

```
struct bind_pdu {
  byte rpc_vers = 5;
  byte rpc_vers_minor;
  byte PTYPE = bind;
  byte pfc_flags;
  byte packed_drep[4];
  ushort frag_length;
  ushort auth_length;
  uint call_id;
  ushort max_xmit_frag;
  ushort max_recv_frag;
  uint assoc_group_id;
  ContextList p_context_elem;
}
```

RPC Binding Strings




```
PS> $b = Get-RpcStringBinding "ncacn_np" "\pipe\rpc"
PS> Connect-RpcClient $client -StringBinding $b
```

```
PS> Connect-RpcClient $client -ProtocolSequence
"ncacn_np" "\pipe\rpc" -NetworkAddress "server.com"
```

Security Context Multiplexing

- Microsoft extension (see MS-RPCE §3.3.1.5.4)
- Allows you to negotiate multiple security context for the same connection and switch them for each call.
- Could have interesting behaviors!

```
PS> Add-RpcClientSecurityContext $client
-AuthenticationLevel PacketPrivacy
```

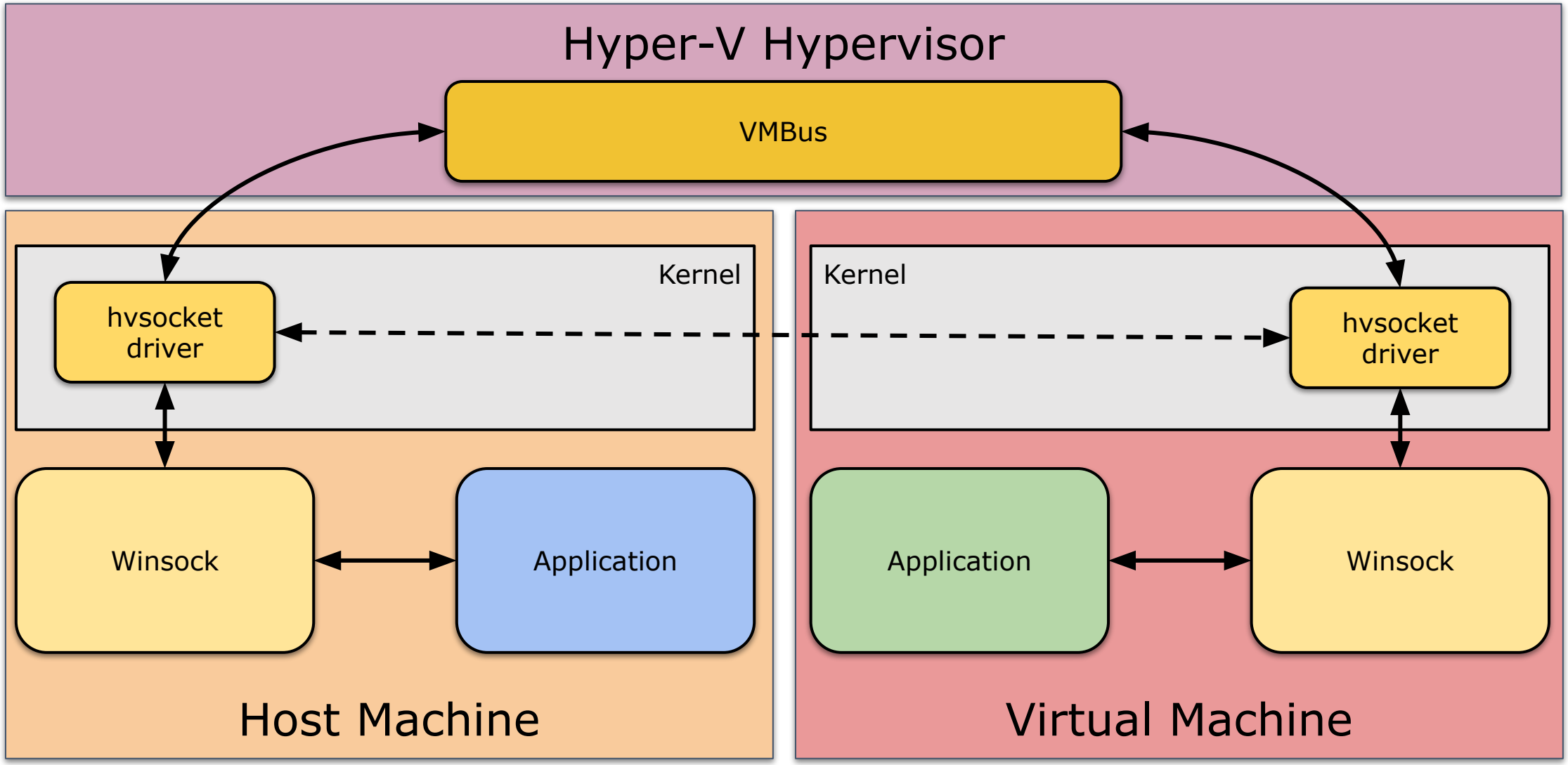
Demo

Do We Support All Useful Protocols?

```
PS> Get-RpcEndpoint | Group ProtocolSequence
Count Name
----- ----
   19 ncacn_ip_tcp
  396 ncalrpc
   15 ncacn_np
    1 ncacn_hvsocket
```

What's this?

Hyper-V Sockets



Hyper-V Socket RPC Binding Strings

```
ncacn_hvsocket:VMID[SYSID]
```

```
VM ID to connect to
Empty      : Loopback
"parent"   : Parent host
"silohost": Parent silo
GUID       : VM by ID.
```

```
GUID of service to connect to.
Endpoint Mapper:
DA32E281-383E-49A1-900A-AF3B74B90B0E
```

```
PS> Get-HyperVSocketTable -Listener
```

Demo

Cross Platform Support

PowerShell is Cross Platform

PowerShell

Welcome to the PowerShell GitHub Community! PowerShell is a cross-platform (Windows, Linux, and macOS) automation and configuration tool/framework that works well with your existing tools and is optimized for dealing with structured data (e.g. JSON, CSV, XML, etc.), REST APIs, and object models. It includes a command-line shell, an associated scripting language, and a framework for processing cmdlets.

Changes Required

- Can't use ALPC for obvious reasons
- Linux does support Hyper-V sockets but only exposes a "VSOCK" interface
- Reimplement any code which uses Windows APIs
  - Parsing string bindings
  - Endpoint mapper
- Implement managed NTLM/Kerberos/Negotiate authentication
- TCP works out to the box
- Named Pipes needs a managed SMB client
  - Well I'll just write my own :)

Demo

Pipes and Other Types

Pipes

```
typedef pipe char EFS_EXIM_PIPE;

long EfsRpcReadFileRaw(
 [in]  PEXIMPORT_CONTEXT_HANDLE   hContext,
 [out] EFS_EXIM_PIPE            * EfsOutPipe
);

long EfsRpcWriteFileRaw(
 [in]  PEXIMPORT_CONTEXT_HANDLE   hContext,
 [in]  EFS_EXIM_PIPE            * EfsInPipe
);
```

Define a pipe type using a certain base type.

Use for input or output

C Implementation

```
typedef struct pipe_EFS_EXIM_PIPE {
 void (*pull)(char* state, char* buf,
   unsigned long esize, unsigned long * ecount);
 void (*push)(char* state, char* buf,
   unsigned long ecount);
 void (*alloc)(char* state, unsigned long bsize,
    long** buf, unsigned long* bcount );
    char * state;
} EFS_EXIM_PIPE;
```

Push and pull data on demand.

.NET Implementation

Exposed as a raw array.

```
int EfsRpcReadFileRaw(NdrContextHandle hContext,
                      out byte[] EfsOutPipe) {
  CheckSynchronousPipeSupport();
  _Marshal_Helper m = new();
  m.WriteContextHandle(hContext);
  _Unmarshal_Helper u = SendReceive();
  EfsOutPipe = u.ReadPipeArray<byte>();
  return u.ReadInt32();
}
```

Only supported on OSF protocols (ALPC needs async support)

Read synchronously from stream.

Context Handles

```
typedef [context_handle] void* CONTEXT_HANDLE;
```

- Default
  - Can be passed to any RPC interface in the same process
- Strict
  - Can only be passed to the same RPC interface
- Type Strict
  - Can only be passed to the same RPC interface and must have the same MIDL type.

```
PS> $rs | Select-RpcServer -NonStrictContextHandle
```

Credential Guard RPC Interfaces

```
PS> Get-RpcServer C:\windows\system32\LsaIso.exe
Name       UUID                                 Ver Procs EPs Service Running
----       ----                                 --- ----- --- ------- -------
LsaIso.exe a3e5af3e-8a33-4737-af6e-bc1f8ecee4bf 1.0 5     0           False
LsaIso.exe 39730ec4-82ea-4fdf-8a45-c408e393e212 1.0 2     0           False
LsaIso.exe eda3c9e4-0d4c-4bb7-b612-0e89d4f0607d 1.0 1     0           False
LsaIso.exe 57cce375-4430-47a6-bb96-2cad0d2fd140 1.0 26    0           False
LsaIso.exe 9cfeead6-6135-4fcf-831a-fd3b236023f8 1.0 33    0           False
LsaIso.exe 45527ae0-2a7d-4cec-b214-739f4159c392 1.0 19    0           False
LsaIso.exe 1707e621-44e3-4f54-bb7d-c537eabb55a5 1.0 3     0           False

PS> Get-Item 'NtObject:\RPC Control\LSA_ISO_RPC_SERVER'
Name                TypeName
----                --------
LSA_ISO_RPC_SERVER ALPC Port
```

BCrypt RPC Implementation CVE-2022-34705

```
NTSTATUS BCryptIumOpenAlgorithmProvider(BCRYPT_ISO_OBJECT **obj,
  LPCWSTR pszAlgId, ULONG dwFlags) {
  BCRYPT_ALG_HANDLE hAlgorithm;
  BCryptOpenAlgorithmProvider(&hAlgorithm, pszAlgId,
                              NULL, dwFlags);
  *obj = AllocateBCryptIsoObject(hAlgorithm);
  return STATUS_SUCCESS;
}
```

BCrypt RPC Implementation CVE-2022-34705

```
NTSTATUS BCryptIumOpenAlgorithmProvider(BCRYPT_ISO_OBJECT **obj,
  LPCWSTR pszAlgId, ULONG dwFlags) {
  BCR
  BCr

  *ob
  ret
}
```

```
BCRYPT_ISO_OBJECT *AllocateBCryptIsoObject(
            BCRYPT_ALG_HANDLE hAlgorithm) {
  BCRYPT_ISO_OBJECT* result = LocalAlloc(0,
                                sizeof(BCRYPT_ISO_OBJECT));
  result->Magic = 'BIOM';
  result->hAlgorithm = hAlgorithm;
  return result;
}
```

BCrypt RPC Implementation CVE-2022-34705

```
NTSTATUS BCryptIumCloseAlgorithmProvider(BCRYPT_ISO_OBJECT *obj,
                                         ULONG dwFlags) {
  if (!obj || obj->Magic == 'BIOM') {
    return STATUS_INVALID_HANDLE;
  }
  BCryptCloseAlgorithmProvider(obj->hAlgorithm, dwFlags);
  LocalFree(obj);
  return STATUS_SUCCESS;
}
```

BCrypt RPC Interface Definition CVE-2022-34705

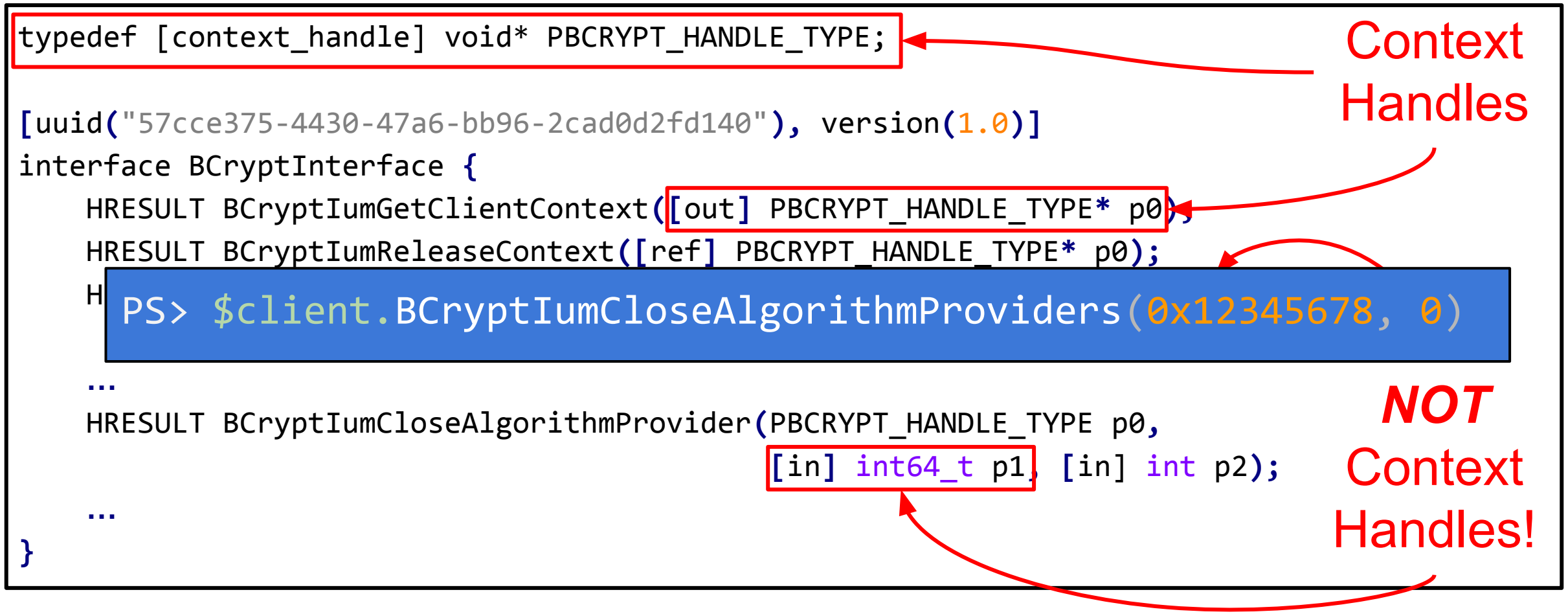
```
typedef [context_handle] void* PBCRYPT_HANDLE_TYPE;

[uuid("57cce375-4430-47a6-bb96-2cad0d2fd140"), version(1.0)]
interface BCryptInterface {
    HRESULT BCryptIumGetClientContext([out] PBCRYPT_HANDLE_TYPE* p0);
    HRESULT BCryptIumReleaseContext([ref] PBCRYPT_HANDLE_TYPE* p0);
    HRESULT BCryptIumOpenAlgorithmProvider([in] handle_t p1, [out] int64_t* p1,
         [in] wchar_t* p2,  [in] wchar_t* p3,  [in] int p4);
    …
    HRESULT BCryptIumCloseAlgorithmProvider(PBCRYPT_HANDLE_TYPE p0,
                                            [in] int64_t p1, [in] int p2);
    …
}
```

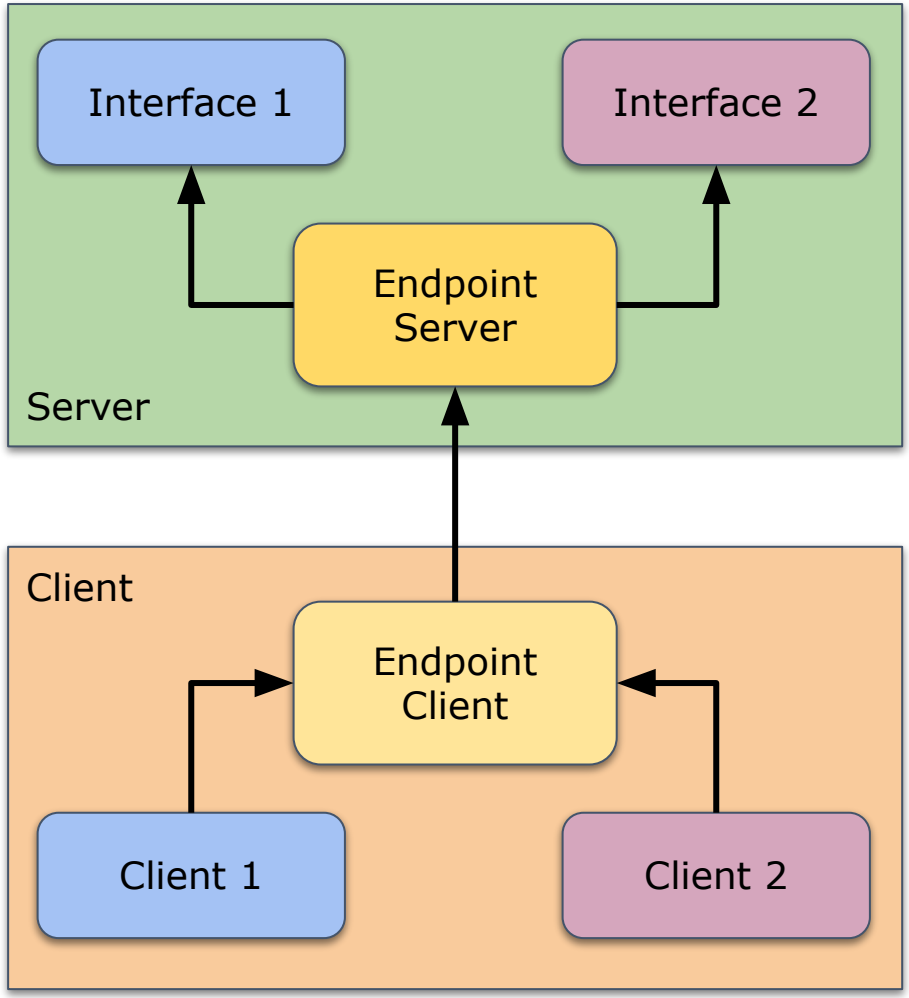
Context Handles

NOT Context Handles!

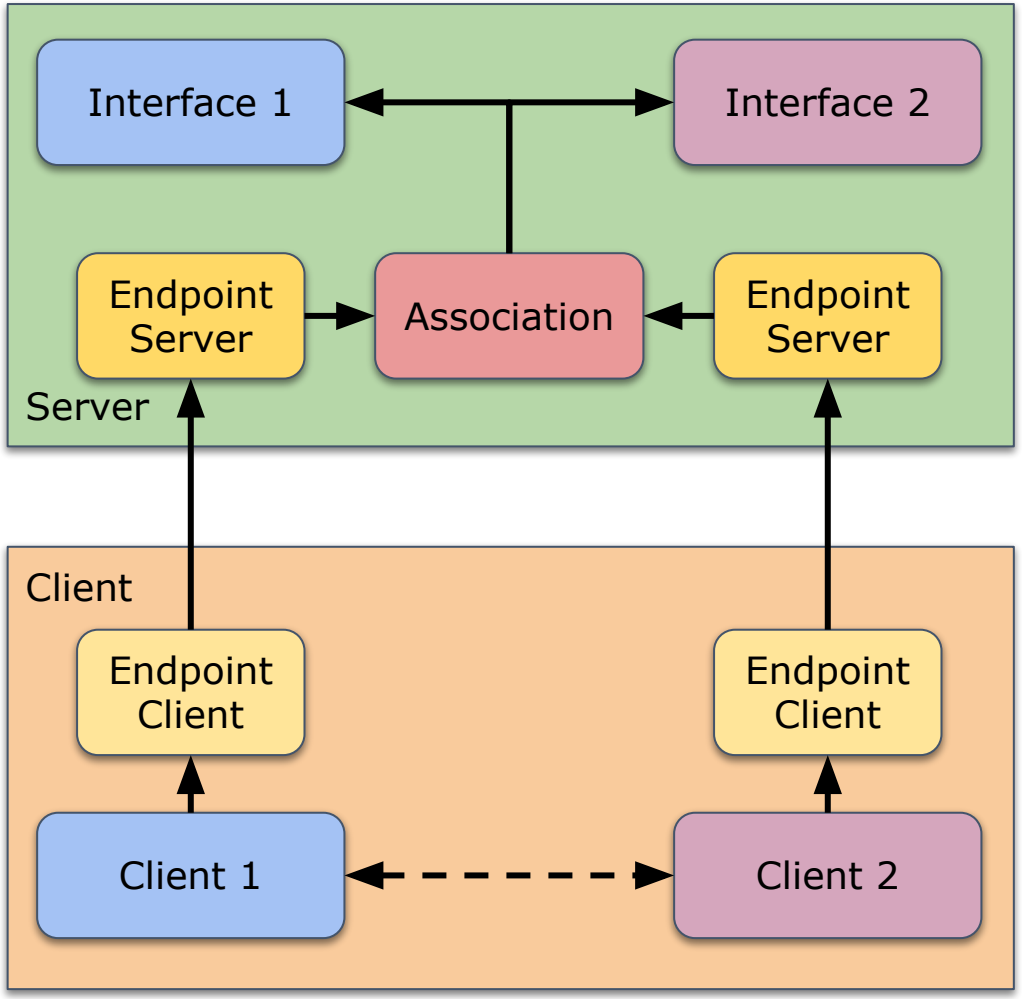
BCrypt RPC Interface Definition CVE-2022-34705



Testing Non-Strict Context Handles



Multiplexing



Association

```
PS> Connect-RpcClient $client1 -ProtocolSequence
"ncacn_np" -NetworkAddress "localhost"

PS> $config = New-RpcClientTransportConfig
-ProtocolSequence "ncacn_np"

PS> $config.AssociationGroupId =
$client1.Transport.AssociationGroupId

PS> Connect-RpcClient $client2 -ProtocolSequence
"ncacn_np" -NetworkAddress "localhost"
-Configuration $config
```

Must use "remote" named pipe.

Copy association group ID

Type-strict Context Handles

```
typedef [context_handle] void* SERVICE_HANDLE;
typedef [context_handle] void* DATA_HANDLE;
int Test1([out] SERVICE_HANDLE* sh);
int Test2([in] DATA_HANDLE dh);
```

Original IDL

```
int Test1(out NdrContextHandle sh);
int Test2(NdrContextHandle dh);
```

Old Code Gen

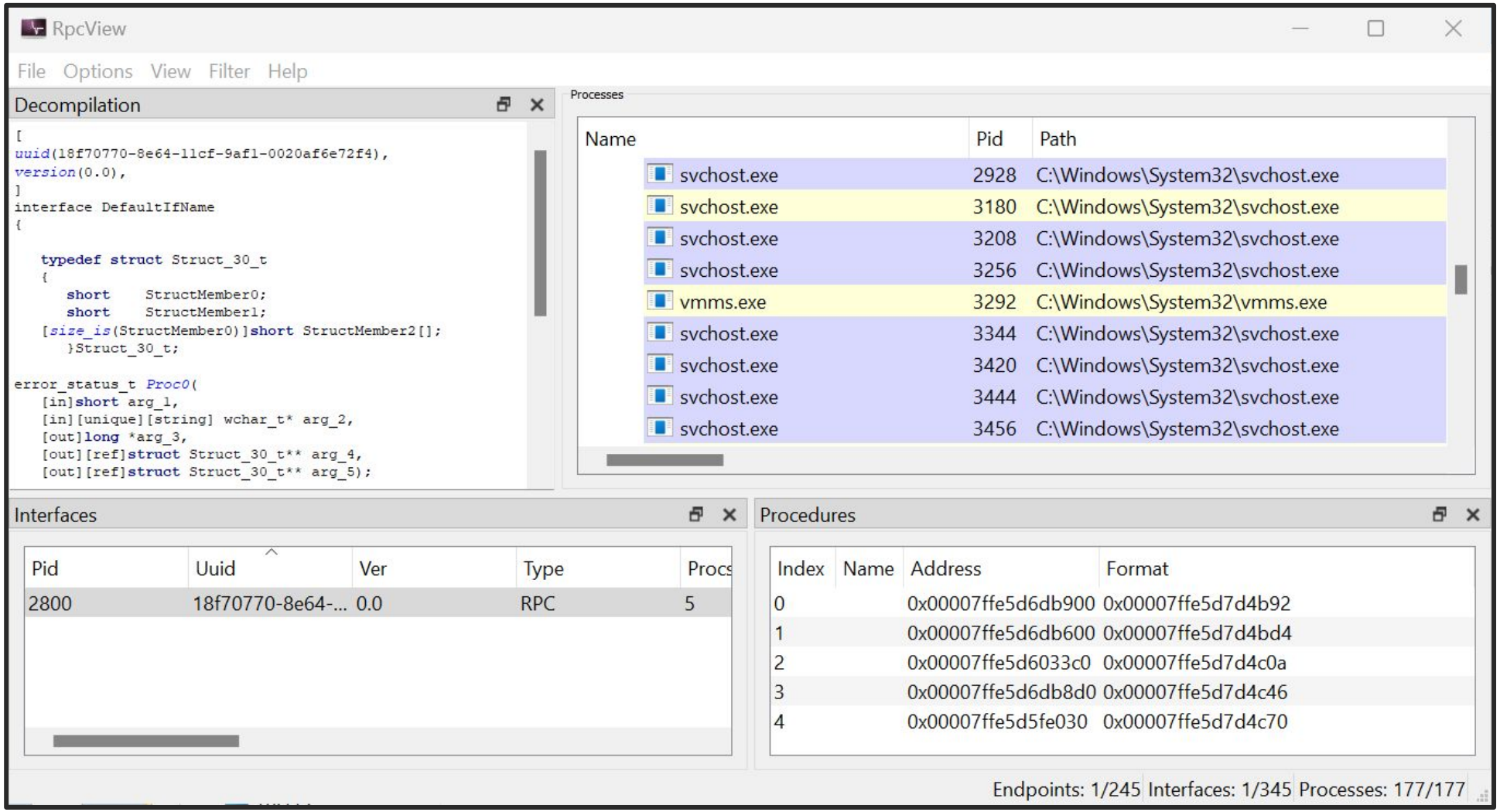
```
int Test1(out TypeStrictContextHandle_1 sh);
int Test2(TypeStrictContextHandle_2 dh);
```

New Code Gen

Demo

Creeping up on RpcView

RpcView



https://github.com/silverf0x/RpcView

Querying Process From a Transport

ncalrpc

```
struct ALPC_SERVER_SESSION_INFORMATION {
    DWORD SessionId;
    DWORD ProcessId;
};

NtAlpcQueryInformation(hPort, AlpcServerSessionInformation, ...)
```

ncacn_np

```
BOOL GetNamedPipeServerProcessId(
  HANDLE Pipe, PULONG ServerProcessId
);
```

ncacn_ip_tcp

```
DWORD GetExtendedTcpTable(
  PVOID  pTcpTable,
  // ...
);
```

Query Endpoint Manager

```
PS> Get-RpcEndpoint | Group {$_.GetServerProcess().ProcessId}
Count Name  Group
----- ----  -----
   94 1968  {[0b1c2170-5732-4e0e-8cd3-d9b16f3b84d7:0.0] ...}
    1 39672 {[09c76598-1491-4810-bbb0-7f403a2ab7ea:5.1] ncalrpc...}
    1 47684 {[09c76598-1491-4810-bbb0-7f403a2ab7ea:5.1] ncalrpc...}
    2 34652 {[c9ac6db5-82b7-4e55-ae8a-e464ed7b4277:1.0] ncalrpc...}
    1 33756 {[a111f1c5-5923-47c0-9a68-d0bafb577901:1.0] ncacn_np...}
    1 29308 {[54f96d15-d9a7-4422-bd32-8b0cebd00400:1.0] ncalrpc...}
   13 7752  {[552d076a-cb29-4e44-8b6a-d15e59e2c0af:1.0] ncalrpc...}
    2 28632 {[0a46e05f-ded7-4890-9dca-8256bb7ad510:2.0]...}
```

Enumerating ALPC Port Handles

```
PS> Get-RpcAlpcServer -IgnoreComInterface
PID   ProcessName       Endpoints Name
---   -----------       --------- ----
9712  sihost.exe        3         \RPC Control\LRPC-...
8864  PowerMgr.exe      1         \RPC Control\BaseModule...
7304  taskhostw.exe     2         \RPC Control\PlaySoundKRpc1
7304  taskhostw.exe     2         \RPC Control\webcache...
19548 dllhost.exe       6         \RPC Control\webplats...
15656 svchost.exe       6         \RPC Control\LRPC-...
```

Parsing All Modules in a Process

```
PS> Get-RpcServer -ProcessId 1234 | Select Name, InterfaceId

Name         InterfaceId
----         -----------
RPCRT4.dll   afa8bd80-7d8a-11c9-bef4-08002b102989
combase.dll 18f70770-8e64-11cf-9af1-0020af6e72f4
combase.dll 00000131-0000-0000-c000-000000000046
combase.dll 00000143-0000-0000-c000-000000000046
combase.dll e1ac57d7-2eeb-4553-b980-f80c69a9e0f7
combase.dll 69c09ea0-4a09-101b-ae4b-08002b349a02
WS2_32.dll   048cf666-ab42-42b4-8975-1357018decb3
SSPICLI.DLL 4f32adc8-6052-4a04-8701-293ccf2096f0
```

Getting Services with RPC Interface Triggers

```
PS> Get-Win32Service | ? { $_.Triggers.Count -gt 0 }
PS> $t = [...RpcInterfaceServiceTriggerInformation]
PS> foreach($s in $ss) {
    $ts = $s.Triggers | % {$_.GetType()}
    if ($t -in $ts) {
        "<$($s.Name)>"
        $s.Triggers.InterfaceId | Out-String
    }
}

<Appinfo>
201ef99a-7fa0-444c-9399-19ba84f12a1a
5f54ce7d-5b79-4175-8584-cb65313a0e98
...
```

Formatting RPC Servers

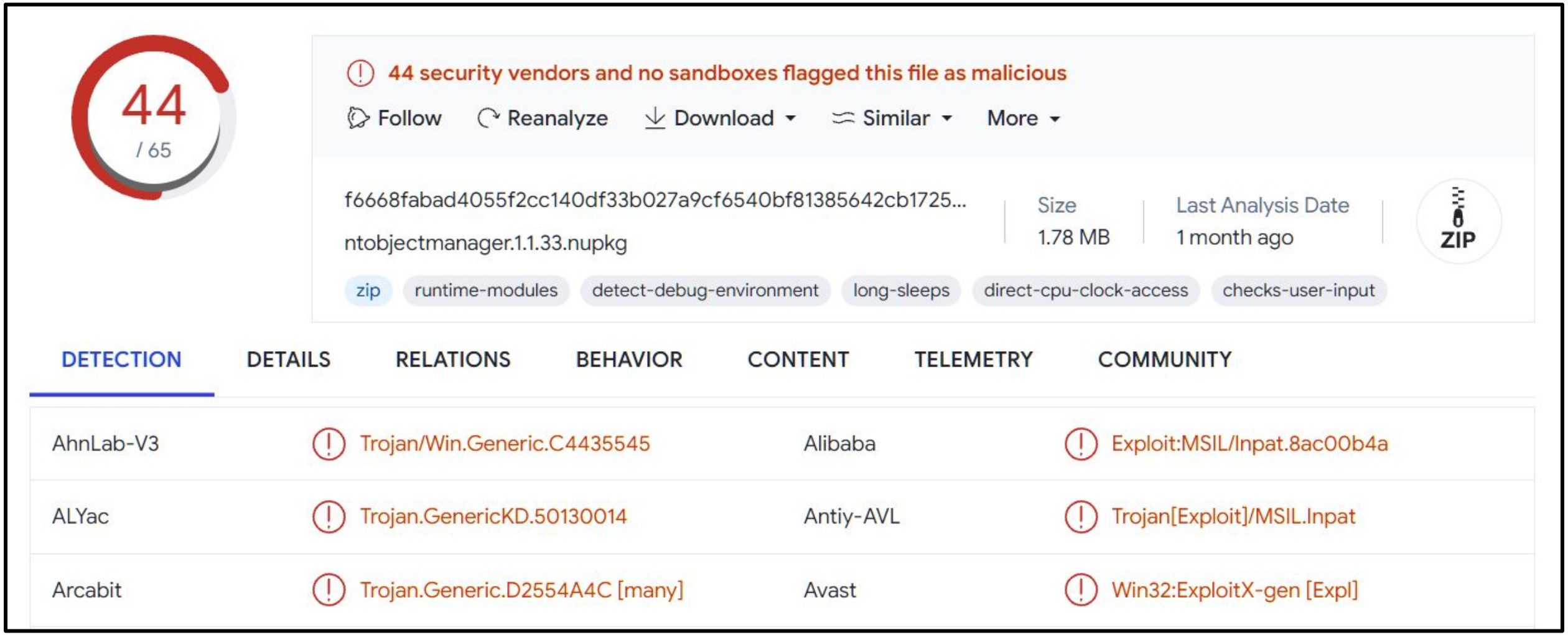
```
[uuid("4870536e-23fa-4cd5-9637-3f1a1699d3dc"), version(1.0)]
interface intf_4870536e_23fa_4cd5_9637_3f1a1699d3dc {
  HRESULT Test1([In] handle_t hBinding, [In] wchar_t* str);
  HRESULT Test2([In] handle_t hBinding, [Out] wchar_t** str);
  // ...
}
```

Old Style Pseudo C#

```
[uuid(4870536E-23FA-4CD5-9637-3F1A1699D3DC), version(1.0)]
interface intf_4870536e_23fa_4cd5_9637_3f1a1699d3dc {
  int Test1([in] handle_t hBinding, [in, string] wchar_t* str);
  int Test2([in] handle_t _hProcHandle, [out] wchar_t** str);
  // ...
}
```

New Style "Real" IDL

Another Reason to Generate IDL



DEMO

Thanks!
Questions?