UNRAVELING AN RPC THRAED

Unraveling an RPC Thread

Abusing RPC server calls for code execution

Whoami

WHOAMI

Alessandro Magnosi

KlezVirus

Managing Consultant – R&D Lead

Red Teaming, Medical Device Security, Code Review

Bug Bounty Hunter

Hunting bugs for fun and a little profit

FOSS Developer

Not very active maintainer of several tools

AGENDA

Agenda

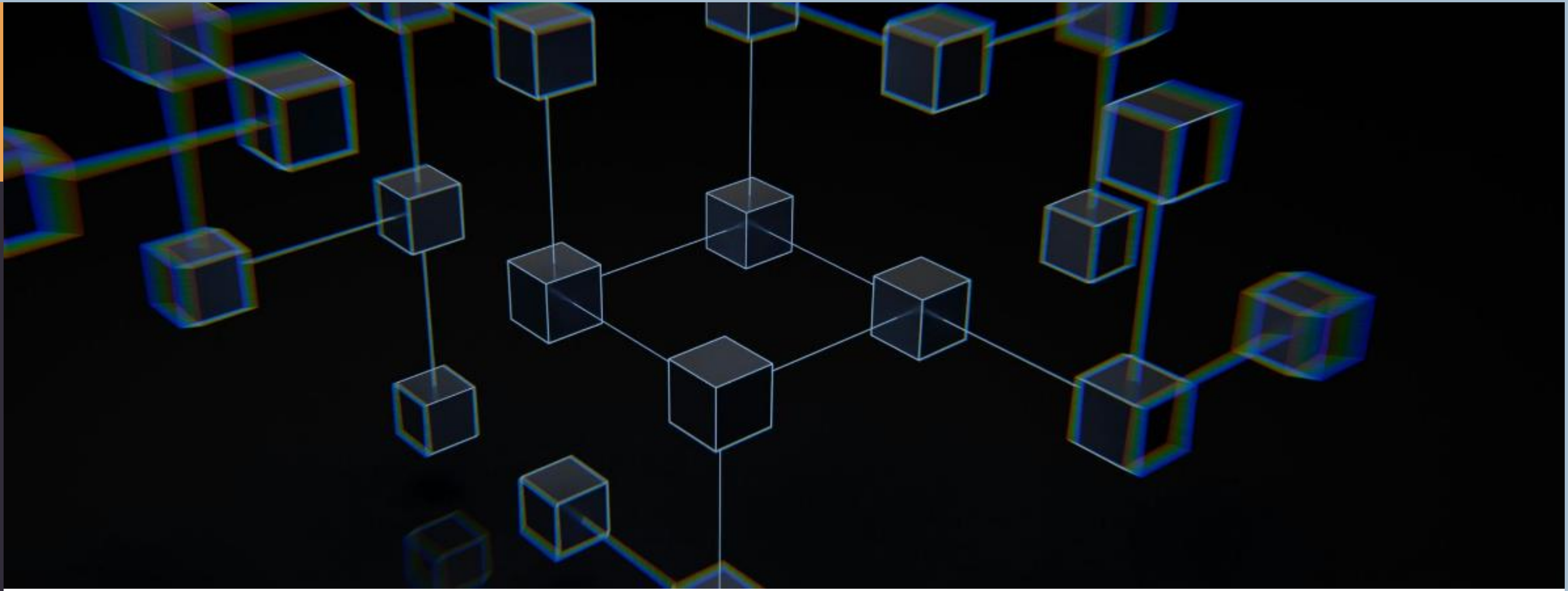
Intro
RPC Overview

RPC Server Calls
Uses and Abuses of RPC server calls

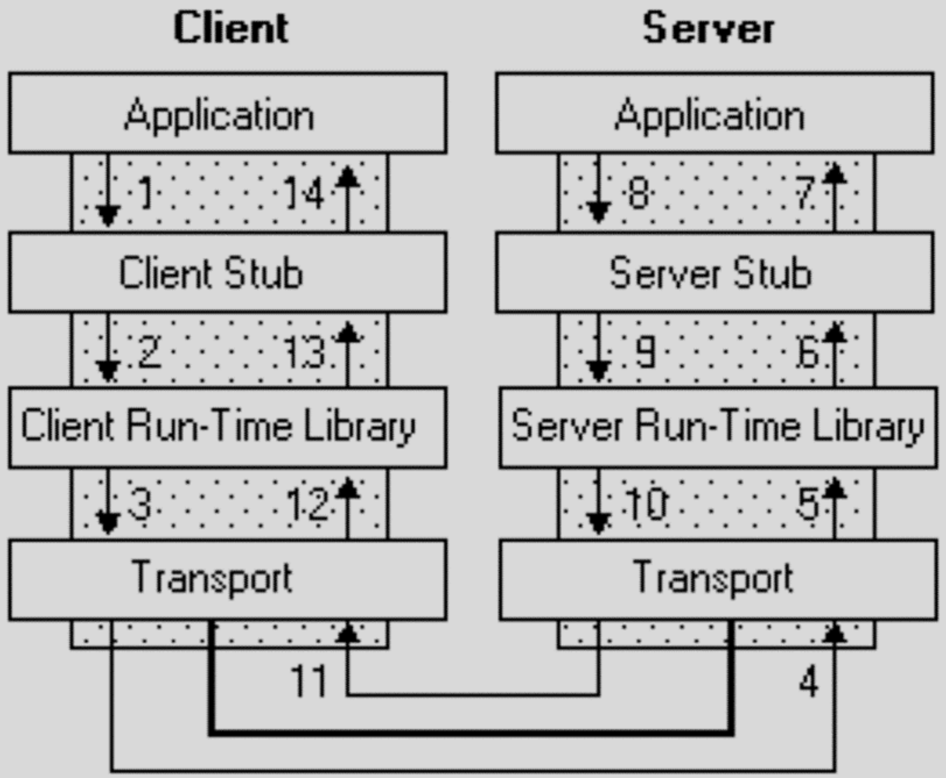
RpcCraft and RpcExec
Local/Remote Code Execution Libraries

Caveats and Limitations
When arbitrary execution is not enough

Conclusion / Demo



Introduction



RPC Infrastructure

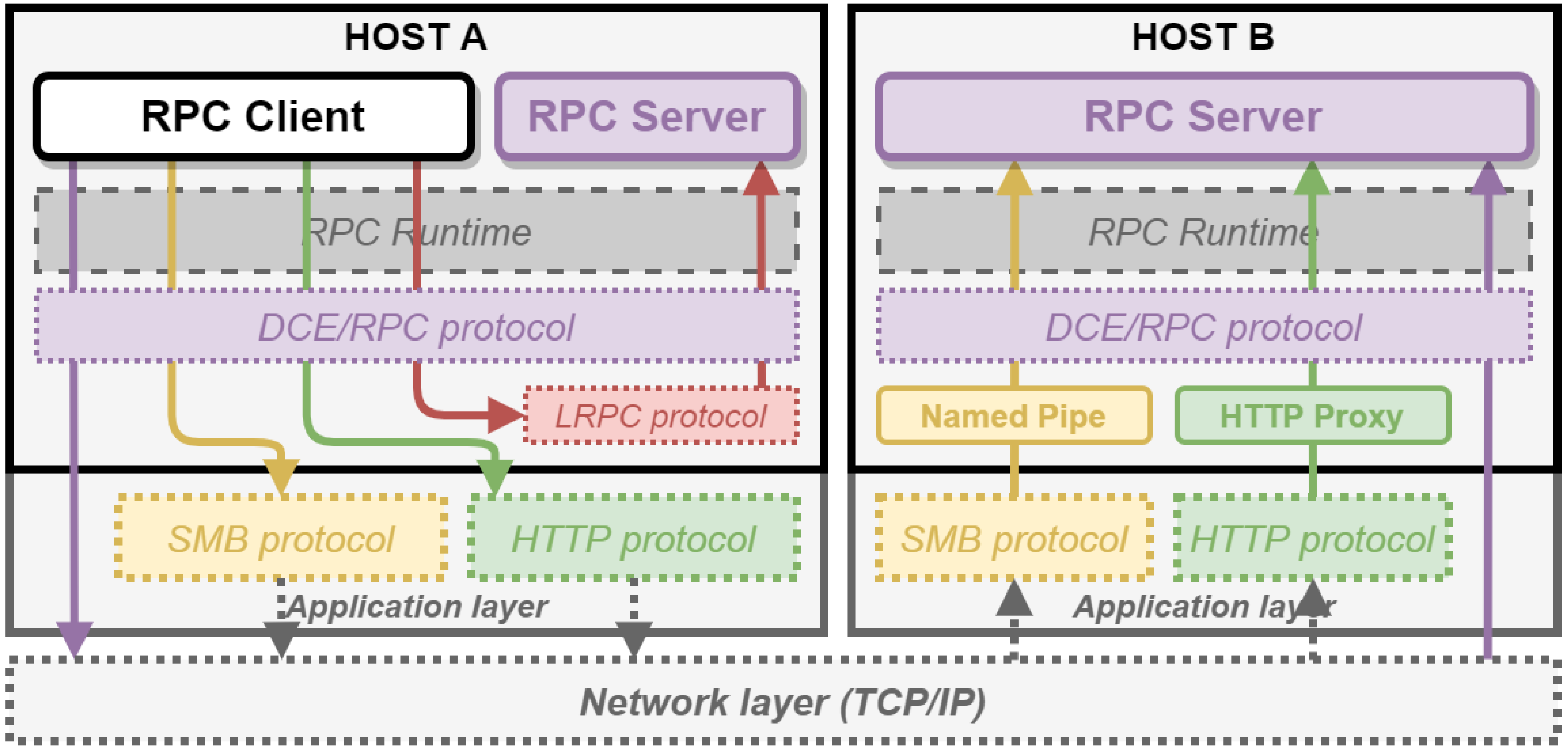
Windows RPC (Remote Procedure Call) facilitates the execution of distributed client/server function calls. With Windows RPC, a client can invoke server functions just as if they were local function calls.

Unmarshalling – The function reads the incoming data packet and converts the serialized parameters into their native in-memory representation

Dispatching – It then calls the server-side function with these parameters

Marshalling – Converts the function's return values and output parameters back into a network-friendly format to send back to the client.

RPC Protocol Sequences



https://itm4n.github.io/from-rpcview-to-petitpotam/

RPC Protocol Sequences

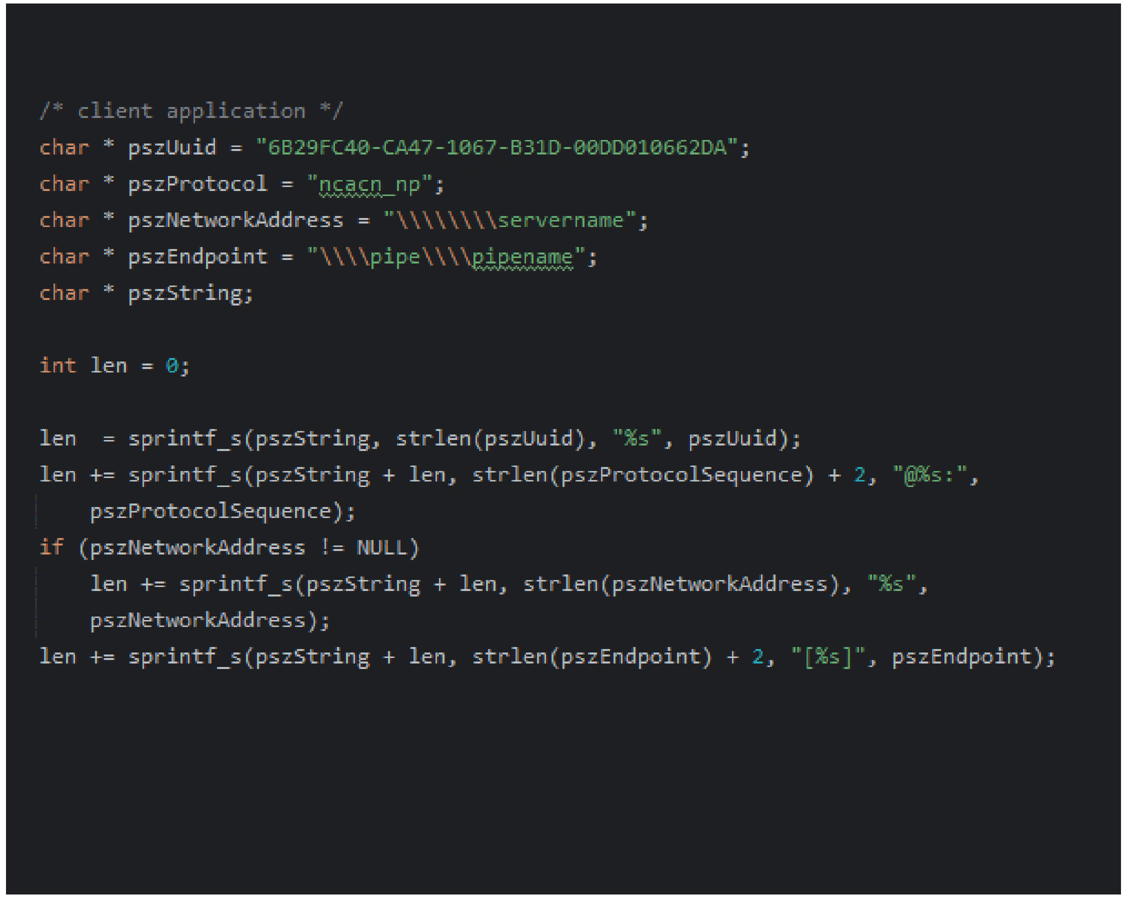
The RPC Protocol Sequence is a predefined string that specifies the protocol the RPC runtime will use to transfer messages, including the transport and network protocol.

Microsoft supports several RPC protocols, such as:

- Network Computing Architecture connection-oriented protocol (NCACN)
- Network Computing Architecture datagram protocol (NCADG)
- Network Computing Architecture local remote procedure call (NCALRPC)

Common protocol sequences include:

- **ncacn_ip_tcp**: Connection-oriented TCP/IP
- **ncacn_http**: Connection-oriented TCP/IP using HTTP proxy
- **ncacn_np**: Connection-oriented named pipes
- **ncadg_ip_udp**: Datagram-based UDP/IP
- **ncalrpc**: Local Procedure Calls



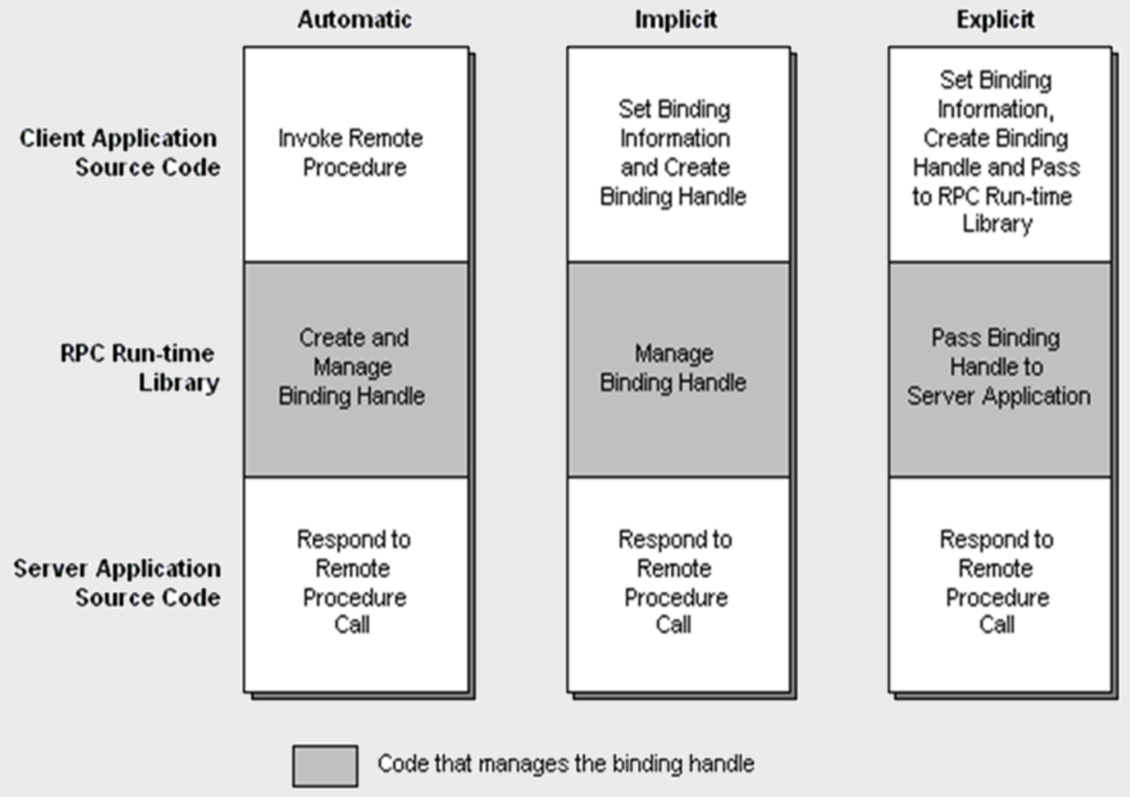
```
/* client application */
char * pszUuid = "6B29FC40-CA47-1067-B31D-00DD010662DA";
char * pszProtocol = "ncacn_np";
char * pszNetworkAddress = "\\\\\\\\servername";
char * pszEndpoint = "\\\\pipe\\\\pipename";
char * pszString;

int len = 0;

len  = sprintf_s(pszString, strlen(pszUuid), "%s", pszUuid);
len += sprintf_s(pszString + len, strlen(pszProtocolSequence) + 2, "@%s:",
    pszProtocolSequence);
if (pszNetworkAddress != NULL)
    len += sprintf_s(pszString + len, strlen(pszNetworkAddress), "%s",
    pszNetworkAddress);
len += sprintf_s(pszString + len, strlen(pszEndpoint) + 2, "[%s]", pszEndpoint);
```

Binding Handles

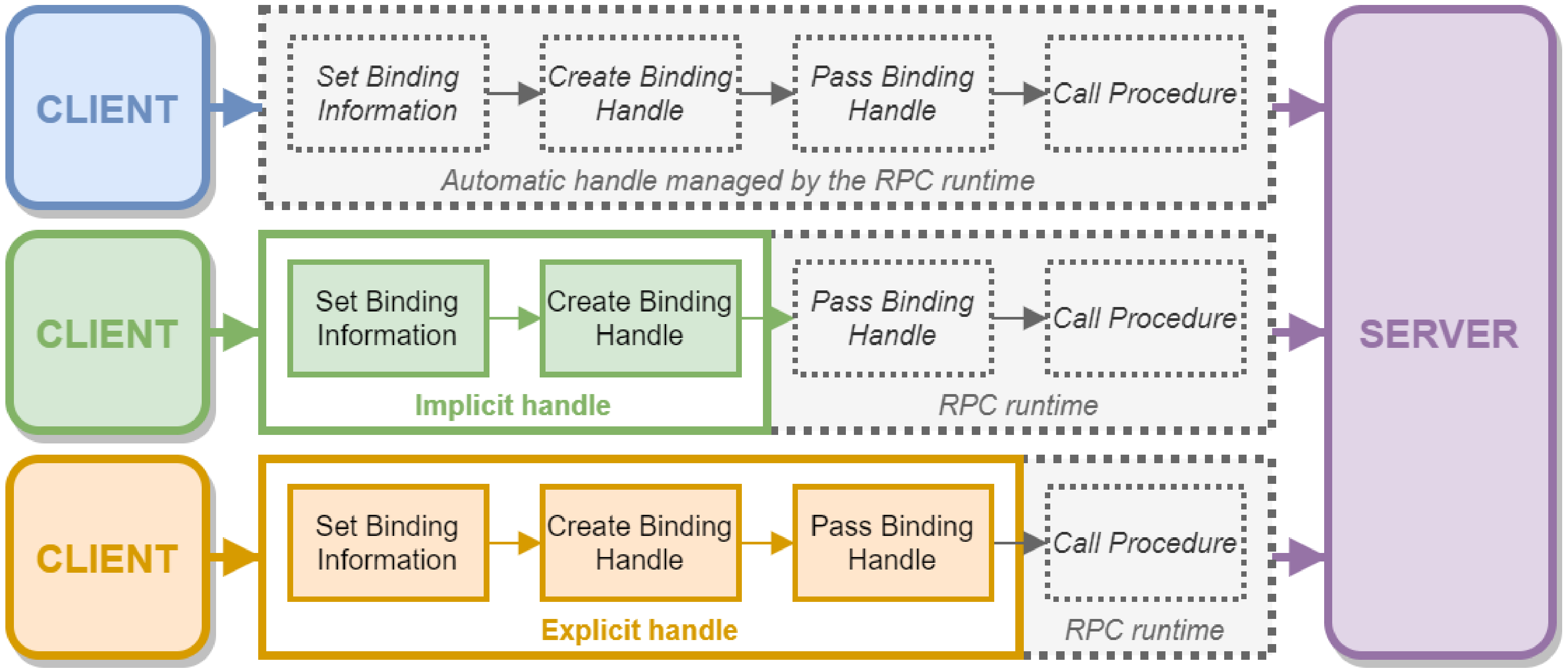
Automatic: Simplest. The server exports its binding information to a namespace, and the client stub handles the binding management automatically.

Implicit: The client application retrieves the server's binding information and assigns a server binding handle to a global variable before making any remote procedure calls.

Explicit: The client application supplies the binding handle as a parameter to each remote procedure call, enabling clients to manage bindings on a per-call basis to meet specialized requirements.


Automatic
Implicit
Explicit
Client Application Source Code
Invoke Remote Procedure
Set Binding Information and Create Binding Handle
Set Binding Information, Create Binding Handle and Pass to RPC Run-time Library
RPC Run-time Library
Create and Manage Binding Handle
Manage Binding Handle
Pass Binding Handle to Server Application
Server Application Source Code
Respond to Remote Procedure Call
Respond to Remote Procedure Call
Respond to Remote Procedure Call
Code that manages the binding handle

RPC Bindings



https://itm4n.github.io/from-rpcview-to-petitpotam/

```c
// Registration Flags Example
RPC_STATUS status;
status = RpcServerRegisterIf2(
    Iface_spec_s,                   // Interface to register.
    NULL,                           // NULL type UUID
    NULL,                           // Use the MIDL generated entry-point vector.
    RPC_IF_ALLOW_LOCAL_ONLY,        // Only allow local connections.
    RPC_C_LISTEN_MAX_CALLS_DEFAULT, // Use default number of concurrent calls.
    (unsigned)-1,                   // Infinite max size of incoming data blocks.
    NULL                            // No security callback.
);

// Security Callback Example
RPC_STATUS CALLBACK XSecurityCallback(RPC_IF_HANDLE hInterface, void* pBindingHandle) {
    return RPC_S_OK; // In this case, allows anyone.
}

status = RpcServerRegisterIf2(
    Iface_spec_s,                   // Interface to register.
    NULL,                           // NULL type UUID.
    NULL,                           // Use the MIDL generated entry-point vector.
    RPC_IF_ALLOW_LOCAL_ONLY,        // Only allow local connections.
    RPC_C_LISTEN_MAX_CALLS_DEFAULT, // Use default number of concurrent calls.
    (unsigned)-1,                   // Infinite max size of incoming data blocks.
    XSecurityCallback               // Security callback function.
);

// Server-side Authentication Example
RPC_STATUS serverStatus;
serverStatus = RpcServerRegisterAuthInfo(
    pszServerPrincipalName, // Server principal name.
    RPC_C_AUTHN_WINNT,      // Using NTLM as authentication service provider.
    NULL,                   // Use default key function, which is ignored for NTLM SSP.
    NULL                    // No arg for key function.
);
```

Protect RPC Endpoints

Registration Flags: These flags can be specified when registering the server interface to control access. RPC_IF_ALLOW_LOCAL_ONLY, as example, restricts connections to local clients only.

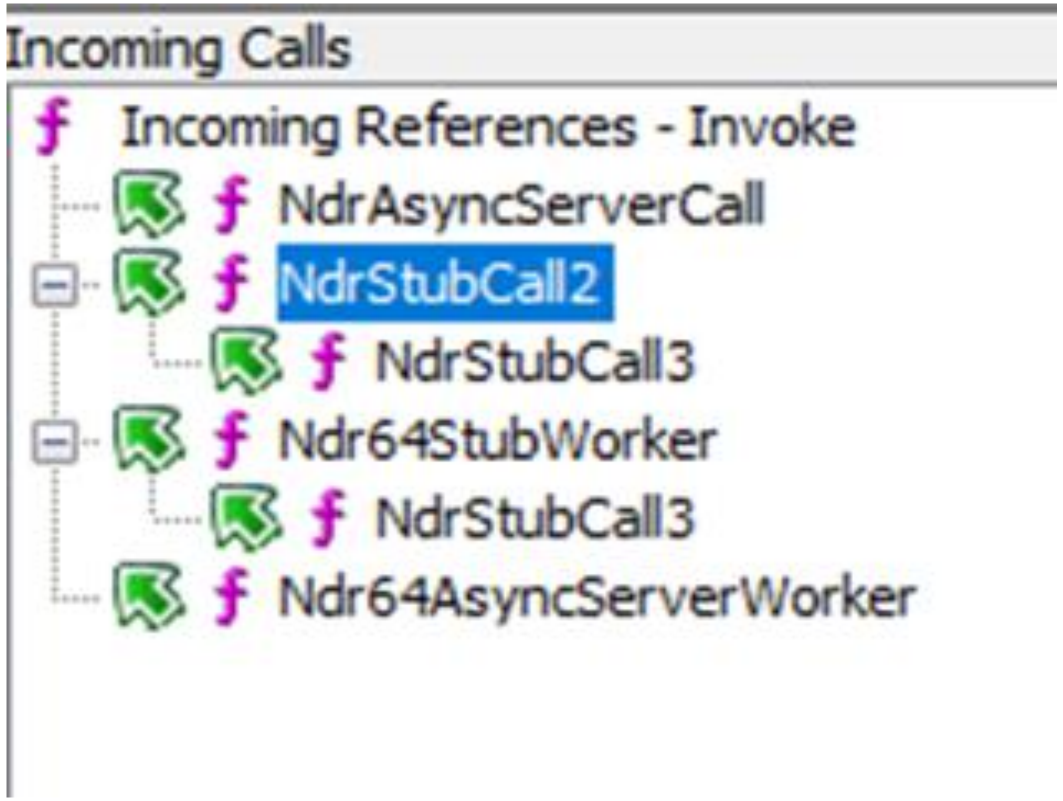
Security Callbacks: It is possible to implement custom security callback to determine whether a requesting client should be allowed or denied. This callback can be included as a parameter in RpcServerRegisterIf2.

Authenticated Bindings: Authenticate bindings on both the server and client sides to ensure secure communication. RpcServerRegisterAuthInfo registers authentication details server-side. On the client side, RpcBindingSetAuthInfoEx provides the binding handle and authentication information.

RPC Server Calls

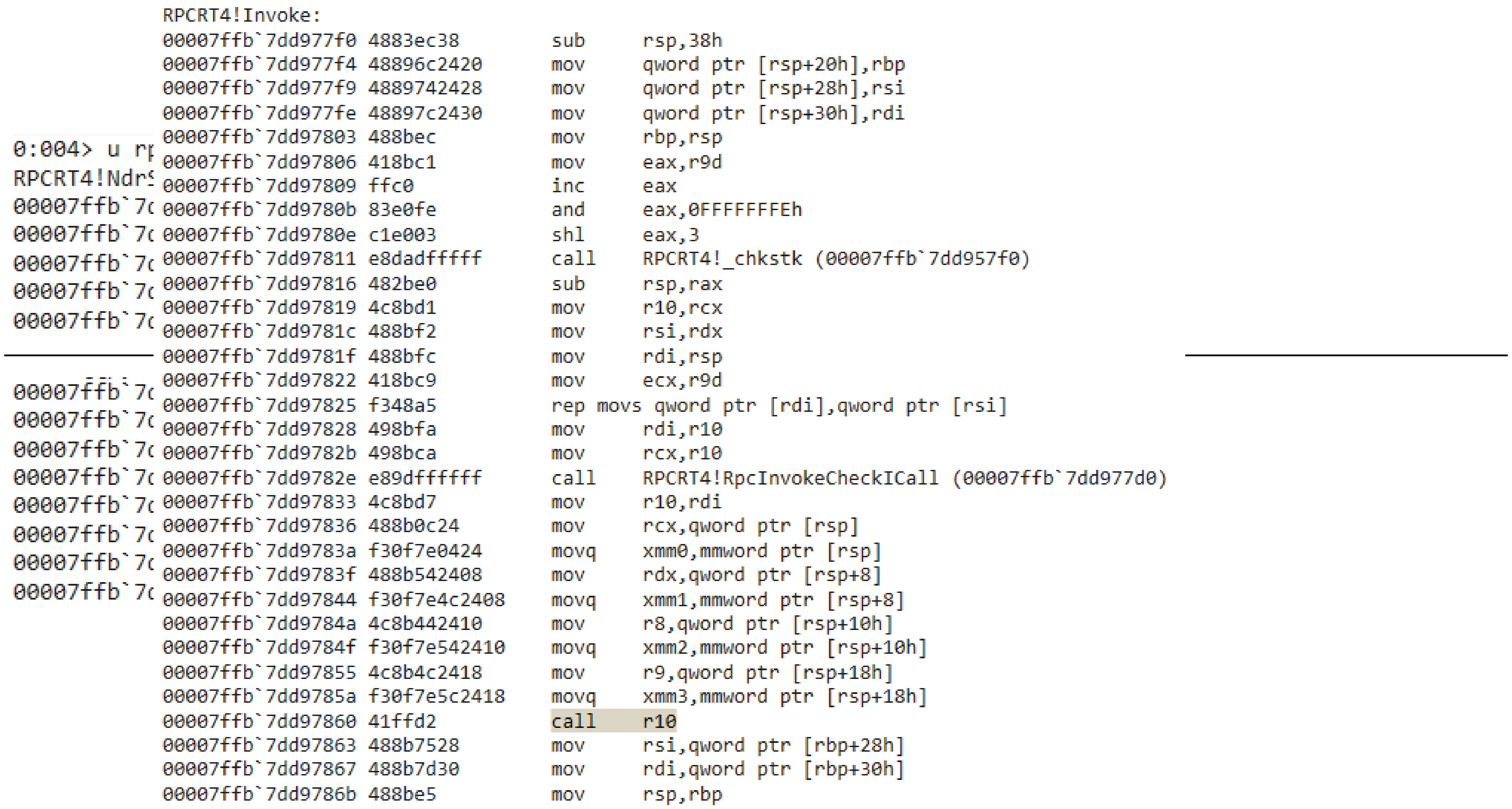
RPC Server Calls

- RPCRT4.dll implements numerous RPC infrastructure functions as wrappers to dynamically invoke server functionalities.
- Many functions ends up in calling the Invoke function to execute a specific interface function.
- **NdrServerCall2** (synchronous), **NdrServerCallAll**, and **NdrServerCallNdr64** (alias of NdrServerCallAll, asynchronous).
- These functions take one argument, a pointer to an **RPC_MESSAGE** structure.



```
Incoming Calls
f Incoming References - Invoke
  NdrAsyncServerCall
  NdrStubCall2
    NdrStubCall3
  Ndr64StubWorker
    NdrStubCall3
  Ndr64AsyncServerWorker
```

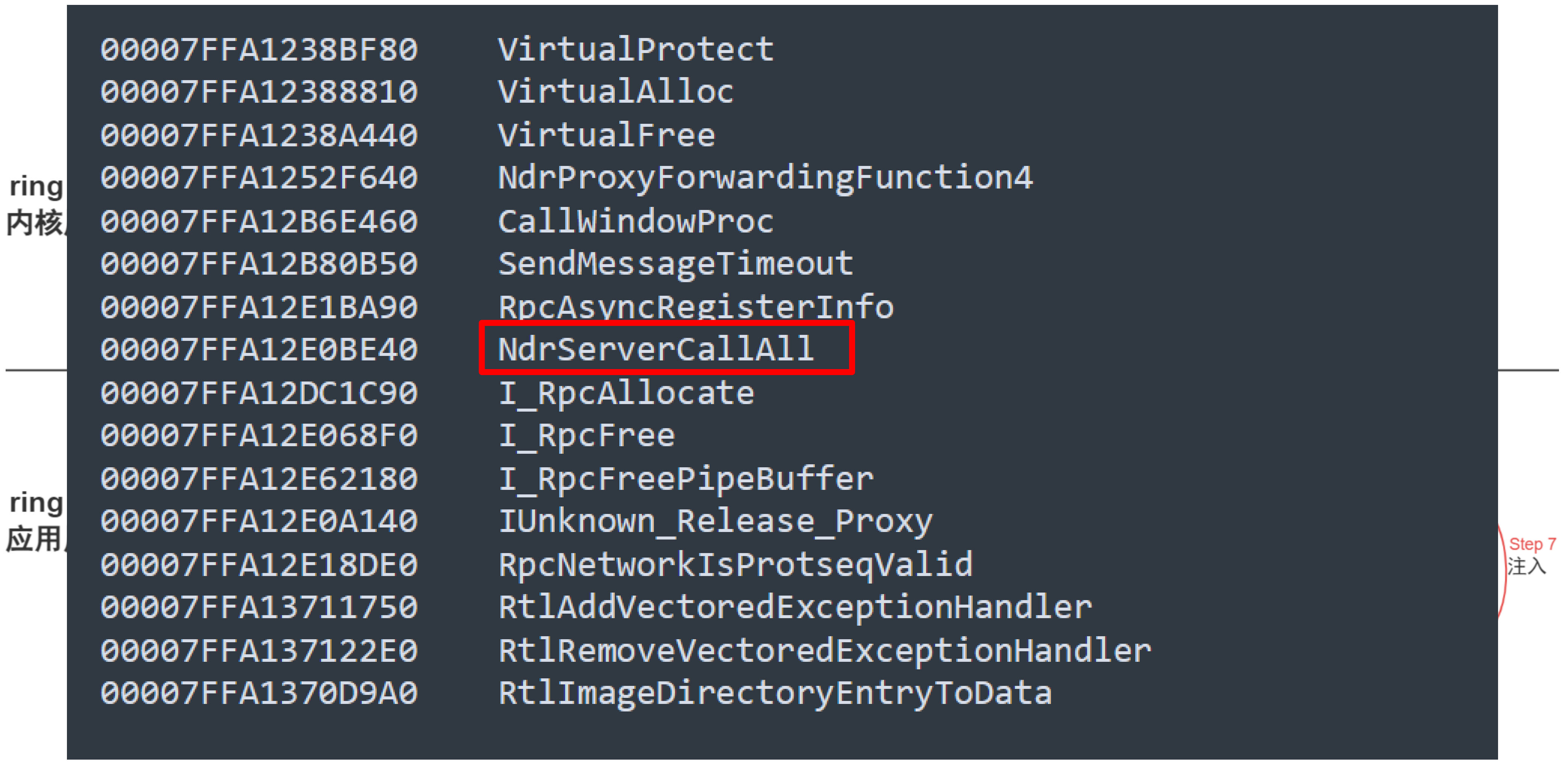
Execution Sequence



```
0:004> u rp
RPCRT4!Ndr
00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7

00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7
00007ffb`7
```

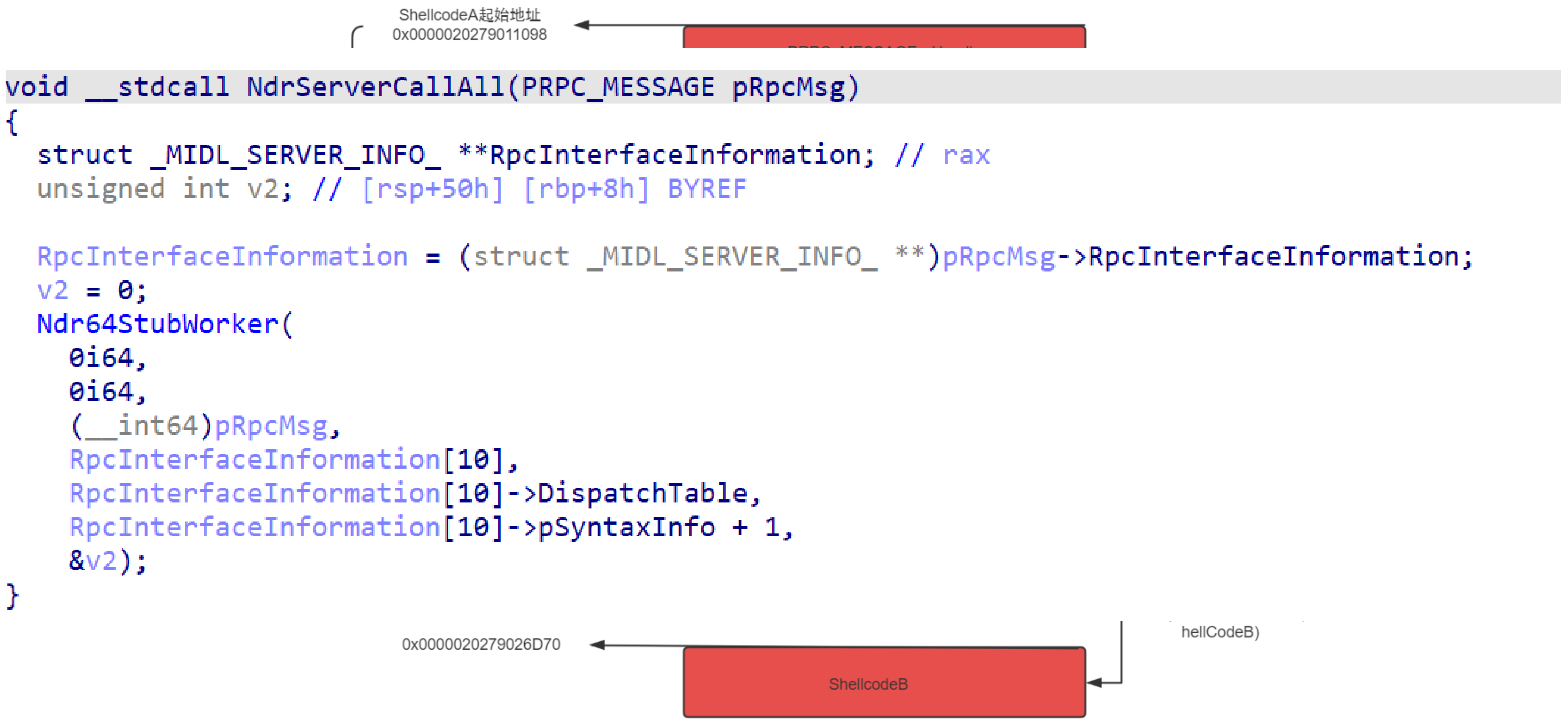
```
RPCRT4!Invoke:
00007ffb`7dd977f0 4883ec38        sub     rsp,38h
00007ffb`7dd977f4 48896c2420      mov     qword ptr [rsp+20h],rbp
00007ffb`7dd977f9 4889742428      mov     qword ptr [rsp+28h],rsi
00007ffb`7dd977fe 48897c2430      mov     qword ptr [rsp+30h],rdi
00007ffb`7dd97803 488bec          mov     rbp,rsp
00007ffb`7dd97806 418bc1          mov     eax,r9d
00007ffb`7dd97809 ffc0            inc     eax
00007ffb`7dd9780b 83e0fe          and     eax,0FFFFFFFEh
00007ffb`7dd9780e c1e003          shl     eax,3
00007ffb`7dd97811 e8dadfffff      call    RPCRT4!_chkstk (00007ffb`7dd957f0)
00007ffb`7dd97816 482be0          sub     rsp,rax
00007ffb`7dd97819 4c8bd1          mov     r10,rcx
00007ffb`7dd9781c 488bf2          mov     rsi,rdx
00007ffb`7dd9781f 488bfc          mov     rdi,rsp
00007ffb`7dd97822 418bc9          mov     ecx,r9d
00007ffb`7dd97825 f348a5          rep movs qword ptr [rdi],qword ptr [rsi]
00007ffb`7dd97828 498bfa          mov     rdi,r10
00007ffb`7dd9782b 498bca          mov     rcx,r10
00007ffb`7dd9782e e89dffffff      call    RPCRT4!RpcInvokeCheckICall (00007ffb`7dd977d0)
00007ffb`7dd97833 4c8bd7          mov     r10,rdi
00007ffb`7dd97836 488b0c24        mov     rcx,qword ptr [rsp]
00007ffb`7dd9783a f30f7e0424      movq    xmm0,mmword ptr [rsp]
00007ffb`7dd9783f 488b542408      mov     rdx,qword ptr [rsp+8]
00007ffb`7dd97844 f30f7e4c2408    movq    xmm1,mmword ptr [rsp+8]
00007ffb`7dd9784a 4c8b442410      mov     r8,qword ptr [rsp+10h]
00007ffb`7dd9784f f30f7e542410    movq    xmm2,mmword ptr [rsp+10h]
00007ffb`7dd97855 4c8b4c2418      mov     r9,qword ptr [rsp+18h]
00007ffb`7dd9785a f30f7e5c2418    movq    xmm3,mmword ptr [rsp+18h]
00007ffb`7dd97860 41ffd2          call    r10
00007ffb`7dd97863 488b7528        mov     rsi,qword ptr [rbp+28h]
00007ffb`7dd97867 488b7d30        mov     rdi,qword ptr [rbp+30h]
00007ffb`7dd9786b 488be5          mov     rsp,rbp
```

Abuse in the Wild



```
00007FFA1238BF80    VirtualProtect
00007FFA12388810    VirtualAlloc
00007FFA1238A440    VirtualFree
00007FFA1252F640    NdrProxyForwardingFunction4
00007FFA12B6E460    CallWindowProc
00007FFA12B80B50    SendMessageTimeout
00007FFA12E1BA90    RpcAsyncRegisterInfo
00007FFA12E0BE40    NdrServerCallAll
00007FFA12DC1C90    I_RpcAllocate
00007FFA12E068F0    I_RpcFree
00007FFA12E62180    I_RpcFreePipeBuffer
00007FFA12E0A140    IUnknown_Release_Proxy
00007FFA12E18DE0    RpcNetworkIsProtseqValid
00007FFA13711750    RtlAddVectoredExceptionHandler
00007FFA137122E0    RtlRemoveVectoredExceptionHandler
00007FFA1370D9A0    RtlImageDirectoryEntryToData
```

Abuse in the Wild



```
void __stdcall NdrServerCallAll(PRPC_MESSAGE pRpcMsg)
{
  struct _MIDL_SERVER_INFO_ **RpcInterfaceInformation; // rax
  unsigned int v2; // [rsp+50h] [rbp+8h] BYREF

  RpcInterfaceInformation = (struct _MIDL_SERVER_INFO_ **)pRpcMsg->RpcInterfaceInformation;
  v2 = 0;
  Ndr64StubWorker(
    0i64,
    0i64,
    (__int64)pRpcMsg,
    RpcInterfaceInformation[10],
    RpcInterfaceInformation[10]->DispatchTable,
    RpcInterfaceInformation[10]->pSyntaxInfo + 1,
    &v2);
}
```

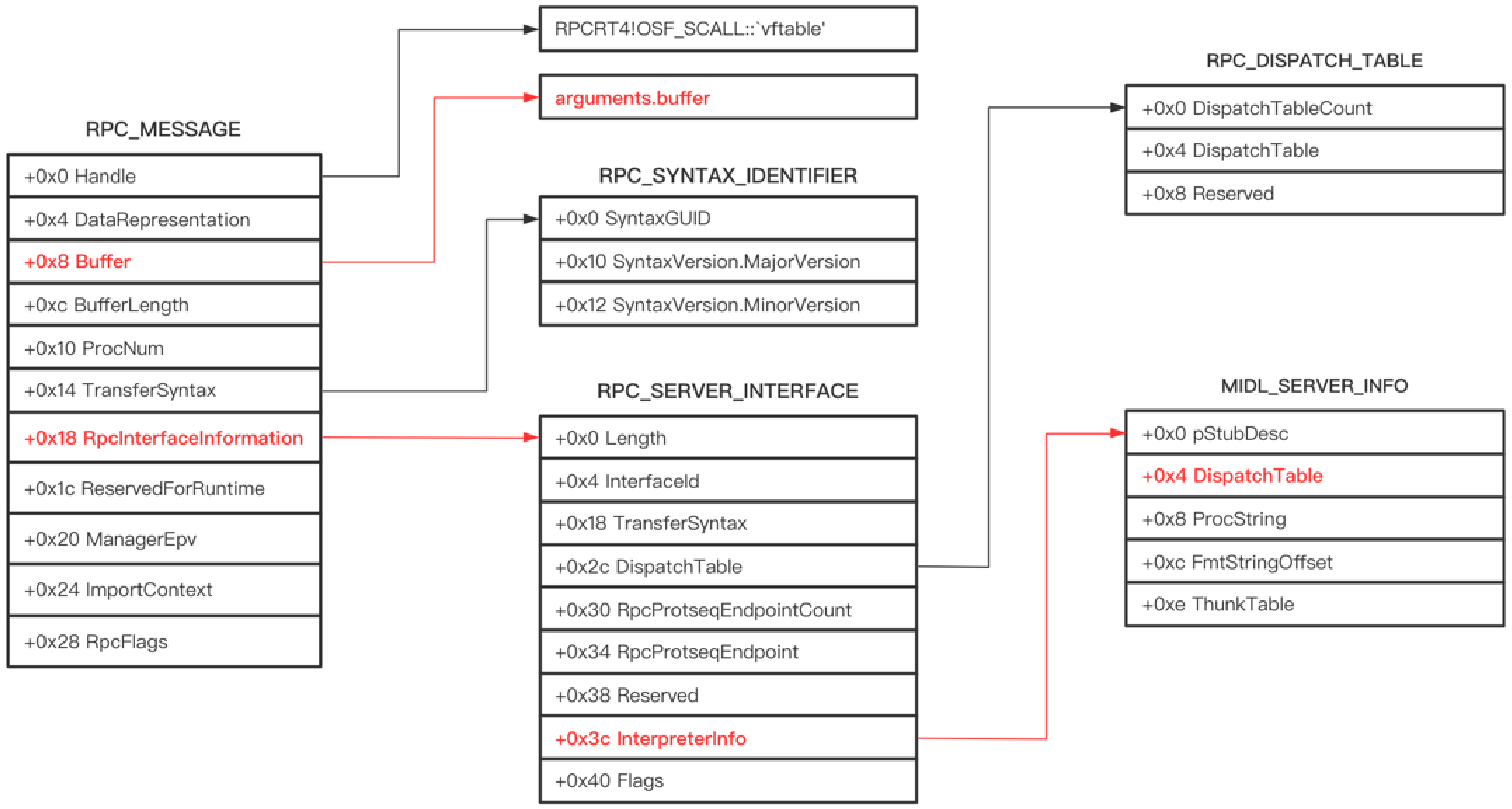
RpcCraft and RpcExec

All starts from a message

```
void NdrServerCall2(
  PRPC_MESSAGE pRpcMsg
);
```

```
typedef struct _RPC_MESSAGE {
  RPC_BINDING_HANDLE     Handle;
  unsigned long          DataRepresentation;
  void                   *Buffer;
  unsigned int           BufferLength;
  unsigned int           ProcNum;
  PRPC_SYNTAX_IDENTIFIER TransferSyntax;
  void                   *RpcInterfaceInformation;
  void                   *ReservedForRuntime;
  RPC_MGR_EPV            *ManagerEpv;
  void                   *ImportContext;
  unsigned long          RpcFlags;
} RPC_MESSAGE, *PRPC_MESSAGE;
```

Potential RPC_MESSAGE Structure

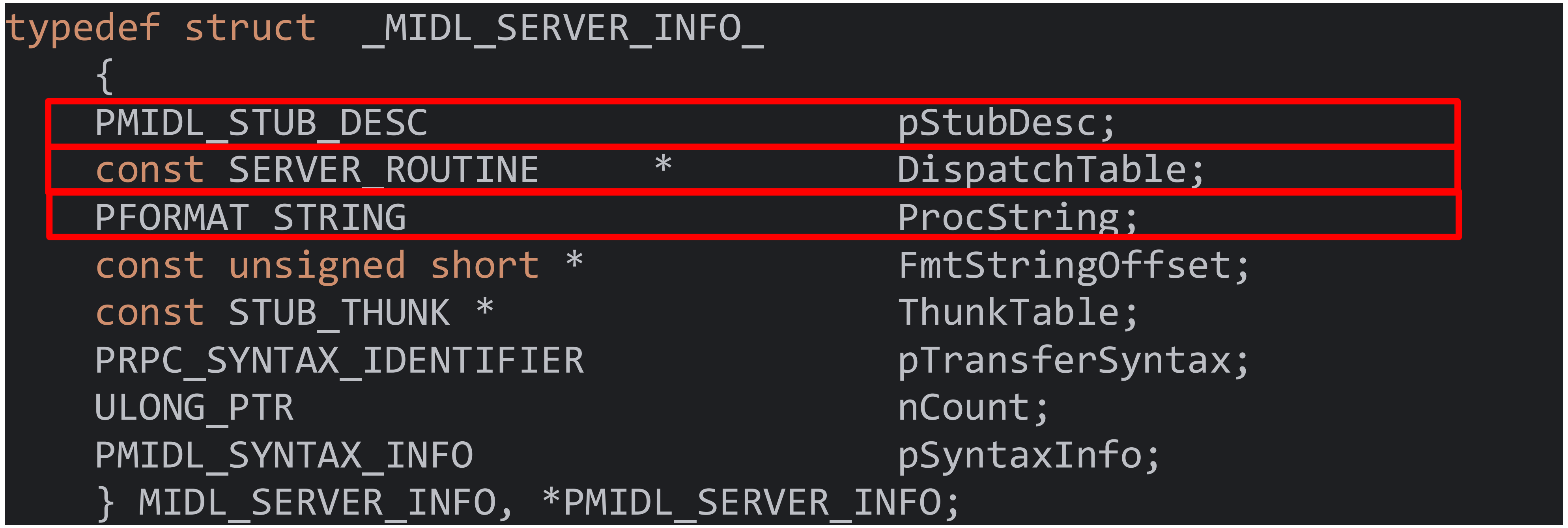


Dissecting the RPC_MESSAGE structure

```
void                                *RpcInterfaceInformation;
```

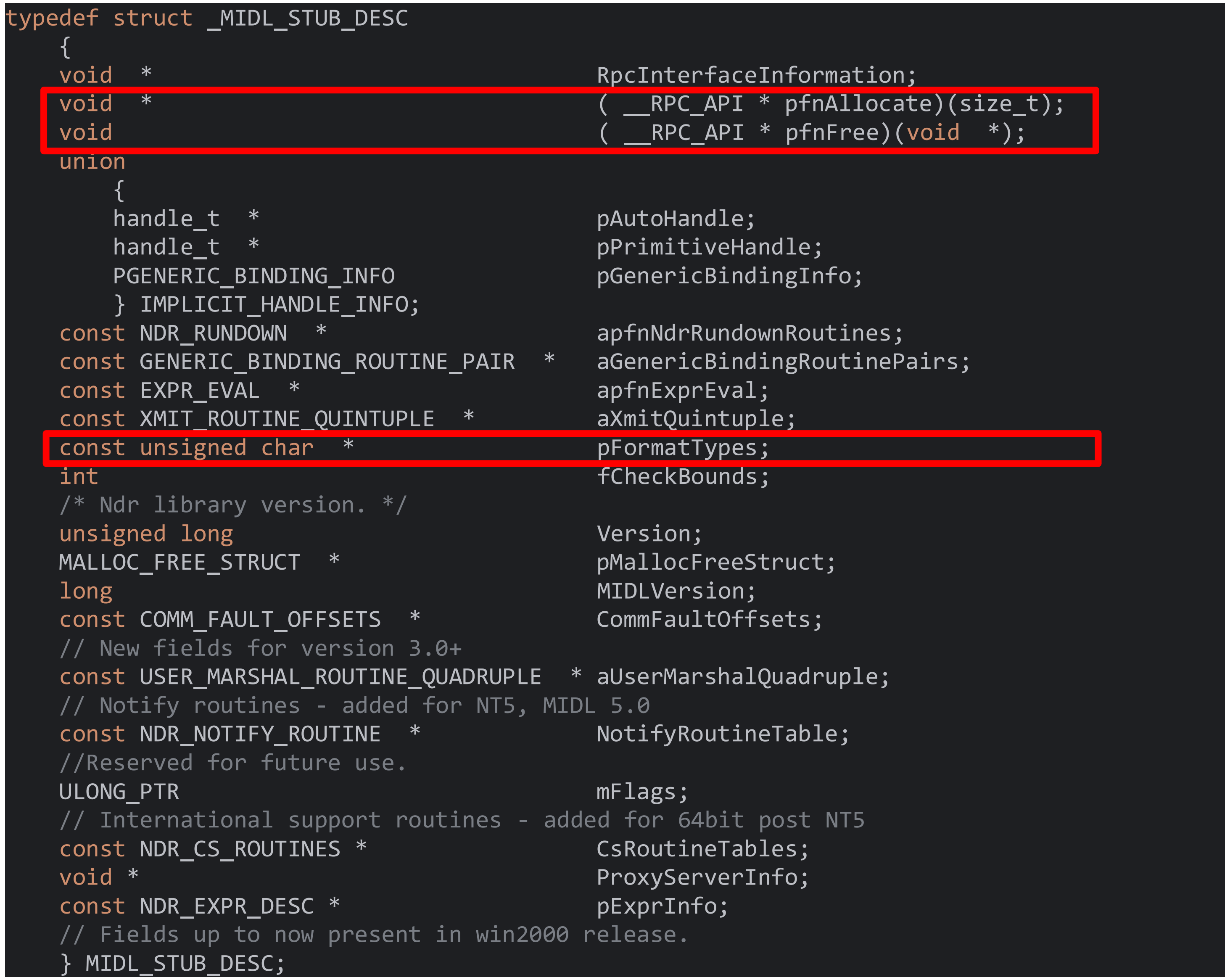
```
typedef struct _RPC_SERVER_INTERFACE
{
    unsigned int Length;
    RPC_SYNTAX_IDENTIFIER   InterfaceId;
    RPC_SYNTAX_IDENTIFIER   TransferSyntax;
    PRPC_DISPATCH_TABLE     DispatchTable;
    unsigned int            RpcProtseqEndpointCount;
    PRPC_PROTSEQ_ENDPOINT   RpcProtseqEndpoint;
    RPC_MGR_EPV __RPC_FAR * DefaultManagerEpv;
    void const __RPC_FAR  * InterpreterInfo;
    unsigned int Flags ;
} RPC_SERVER_INTERFACE, __RPC_FAR * PRPC_SERVER_INTERFACE;

typedef struct _RPC_CLIENT_INTERFACE
{
    unsigned int Length;
    RPC_SYNTAX_IDENTIFIER   InterfaceId;
    RPC_SYNTAX_IDENTIFIER   TransferSyntax;
    PRPC_DISPATCH_TABLE     DispatchTable;
    unsigned int            RpcProtseqEndpointCount;
    PRPC_PROTSEQ_ENDPOINT   RpcProtseqEndpoint;
    ULONG_PTR               Reserved;
    void const __RPC_FAR *  InterpreterInfo;
    unsigned int Flags ;
} RPC_CLIENT_INTERFACE, __RPC_FAR * PRPC_CLIENT_INTERFACE;
```

Dissecting the RPC_MESSAGE structure



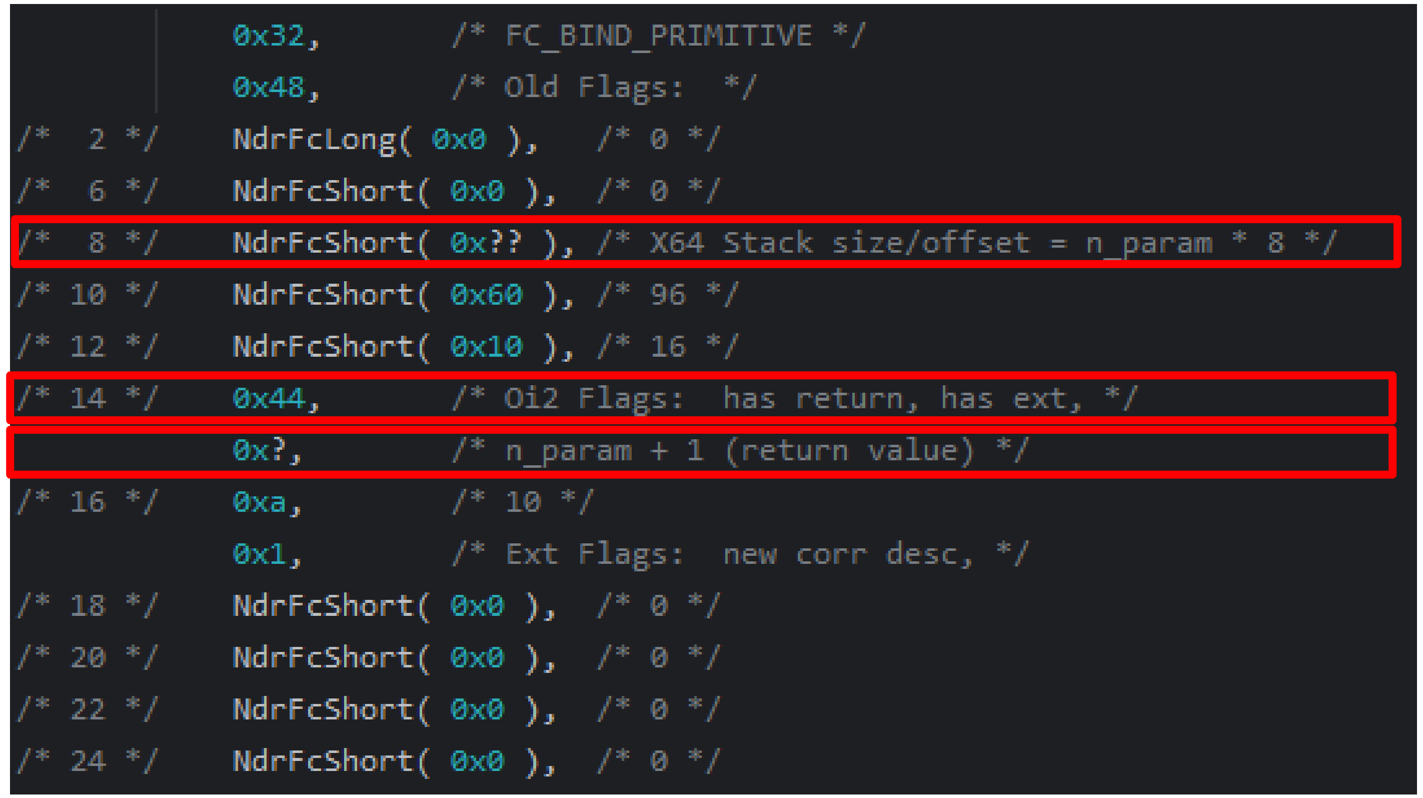
```
typedef struct  _MIDL_SERVER_INFO_
    {
    PMIDL_STUB_DESC                                    pStubDesc;
    const SERVER_ROUTINE     *                         DispatchTable;
    PFORMAT_STRING                                     ProcString;
    const unsigned short *                             FmtStringOffset;
    const STUB_THUNK *                                 ThunkTable;
    PRPC_SYNTAX_IDENTIFIER                             pTransferSyntax;
    ULONG_PTR                                          nCount;
    PMIDL_SYNTAX_INFO                                  pSyntaxInfo;
    } MIDL_SERVER_INFO, *PMIDL_SERVER_INFO;
```

Dissecting the RPC_MESSAGE structure



```
typedef struct _MIDL_STUB_DESC
    {
    void  *                                RpcInterfaceInformation;
    void  *                                ( __RPC_API * pfnAllocate)(size_t);
    void                                   ( __RPC_API * pfnFree)(void  *);
    union
        {
        handle_t  *                        pAutoHandle;
        handle_t  *                        pPrimitiveHandle;
        PGENERIC_BINDING_INFO              pGenericBindingInfo;
        } IMPLICIT_HANDLE_INFO;
    const NDR_RUNDOWN  *                   apfnNdrRundownRoutines;
    const GENERIC_BINDING_ROUTINE_PAIR  *  aGenericBindingRoutinePairs;
    const EXPR_EVAL  *                     apfnExprEval;
    const XMIT_ROUTINE_QUINTUPLE  *        aXmitQuintuple;
    const unsigned char  *                 pFormatTypes;
    int                                    fCheckBounds;
    /* Ndr library version. */
    unsigned long                          Version;
    MALLOC_FREE_STRUCT  *                  pMallocFreeStruct;
    long                                   MIDLVersion;
    const COMM_FAULT_OFFSETS  *            CommFaultOffsets;
    // New fields for version 3.0+
    const USER_MARSHAL_ROUTINE_QUADRUPLE  * aUserMarshalQuadruple;
    // Notify routines - added for NT5, MIDL 5.0
    const NDR_NOTIFY_ROUTINE  *            NotifyRoutineTable;
    //Reserved for future use.
    ULONG_PTR                              mFlags;
    // International support routines - added for 64bit post NT5
    const NDR_CS_ROUTINES *                CsRoutineTables;
    void *                                 ProxyServerInfo;
    const NDR_EXPR_DESC *                  pExprInfo;
    // Fields up to now present in win2000 release.
    } MIDL_STUB_DESC;
```

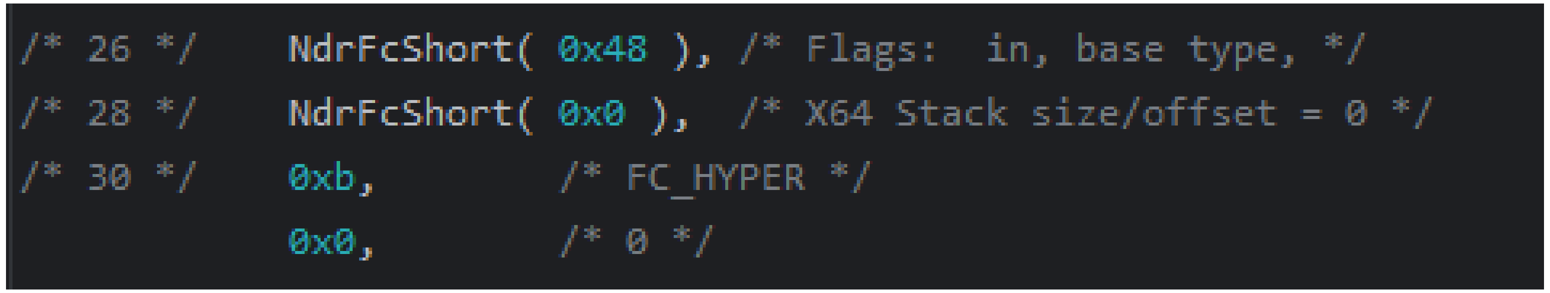
Dissecting the RPC_MESSAGE structure



```
            0x32,         /* FC_BIND_PRIMITIVE */
            0x48,         /* Old Flags:  */
/*  2 */    NdrFcLong( 0x0 ),   /* 0 */
/*  6 */    NdrFcShort( 0x0 ),  /* 0 */
/*  8 */    NdrFcShort( 0x?? ), /* X64 Stack size/offset = n_param * 8 */
/* 10 */    NdrFcShort( 0x60 ), /* 96 */
/* 12 */    NdrFcShort( 0x10 ), /* 16 */
/* 14 */    0x44,         /* Oi2 Flags:  has return, has ext, */
            0x?,          /* n_param + 1 (return value) */
/* 16 */    0xa,          /* 10 */
            0x1,          /* Ext Flags:  new corr desc, */
/* 18 */    NdrFcShort( 0x0 ),  /* 0 */
/* 20 */    NdrFcShort( 0x0 ),  /* 0 */
/* 22 */    NdrFcShort( 0x0 ),  /* 0 */
/* 24 */    NdrFcShort( 0x0 ),  /* 0 */
```

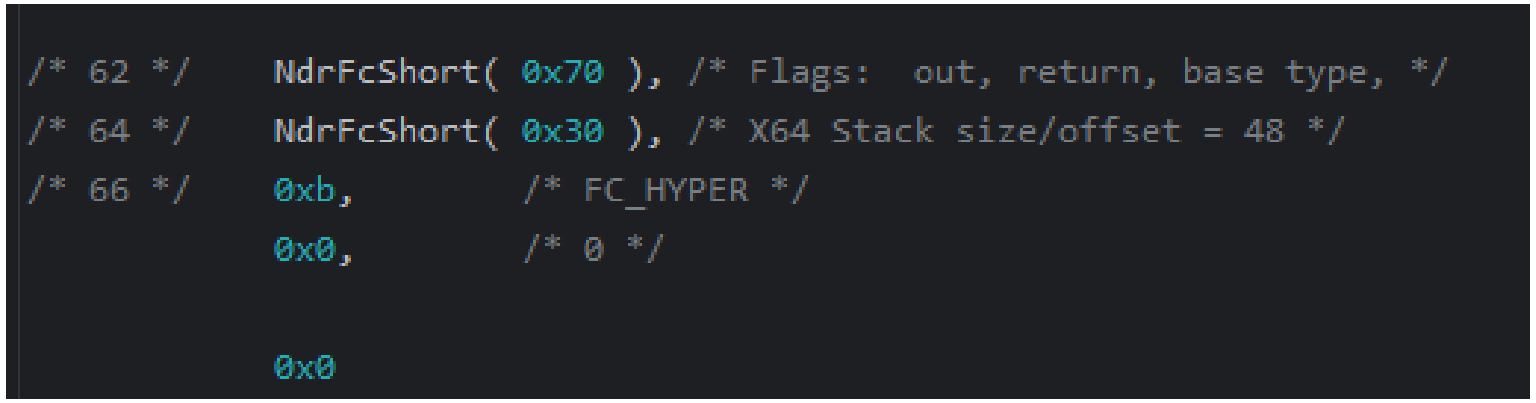
Dissecting the RPC_MESSAGE structure

Input parameter definition:



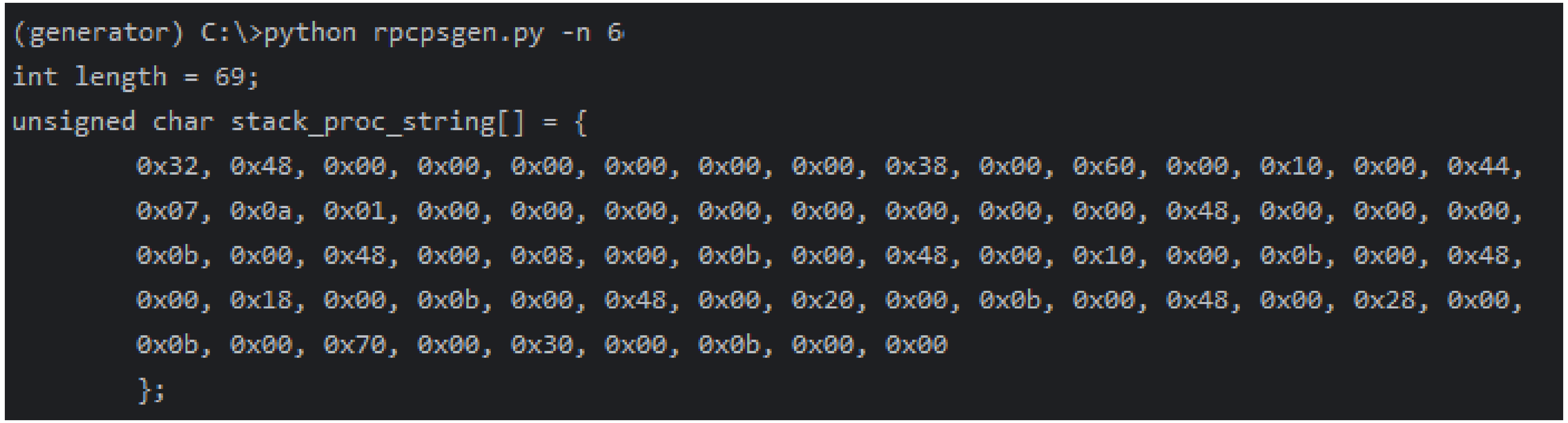
```
/* 26 */    NdrFcShort( 0x48 ), /* Flags:  in, base type, */
/* 28 */    NdrFcShort( 0x0 ),  /* X64 Stack size/offset = 0 */
/* 30 */    0xb,         /* FC_HYPER */
            0x0,         /* 0 */
```

Return value definition:



```
/* 62 */    NdrFcShort( 0x70 ), /* Flags:  out, return, base type, */
/* 64 */    NdrFcShort( 0x30 ), /* X64 Stack size/offset = 48 */
/* 66 */    0xb,         /* FC_HYPER */
            0x0,         /* 0 */

            0x0
```

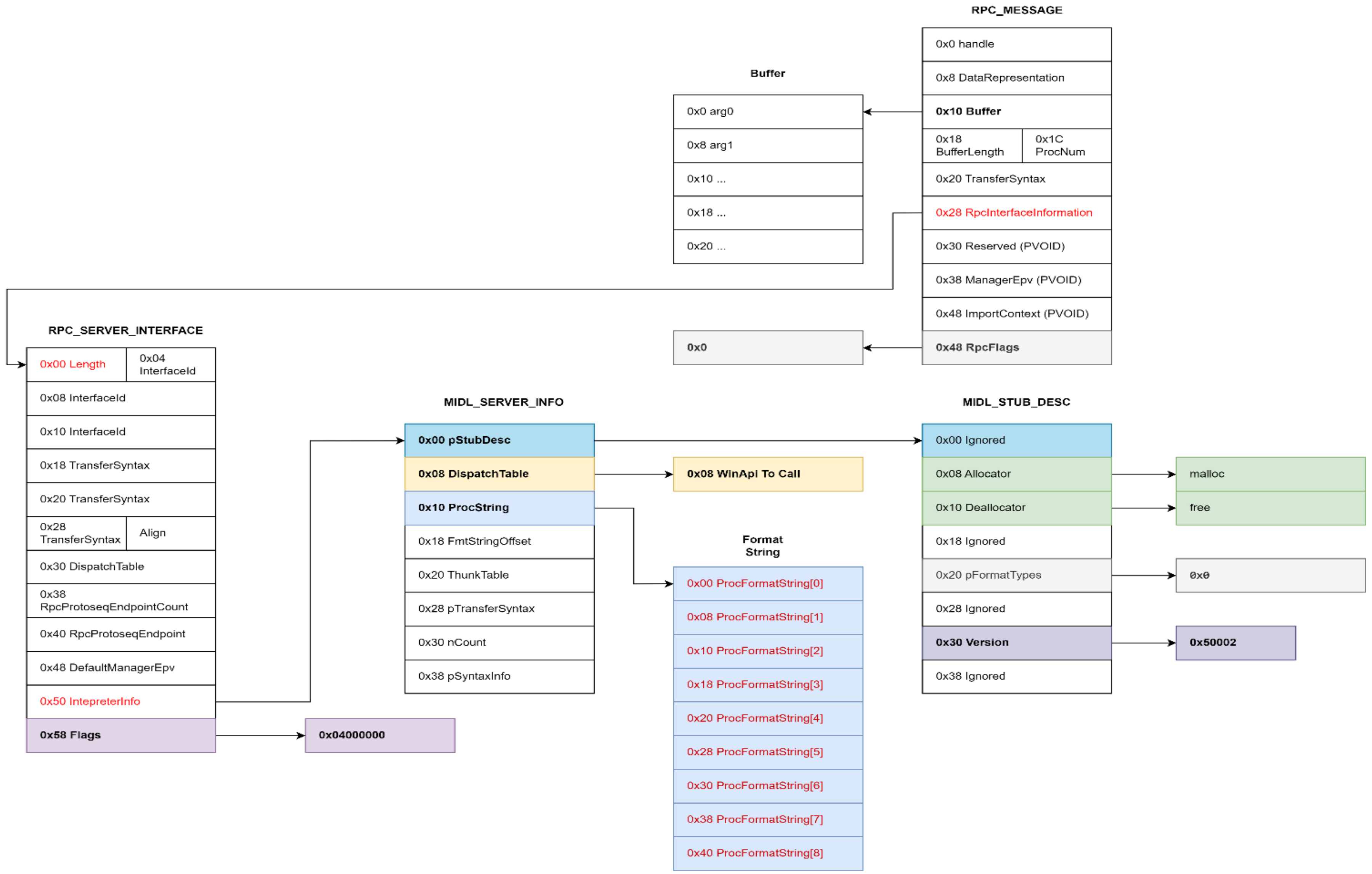
Dissecting the RPC_MESSAGE structure



```
(generator) C:\>python rpcpsgen.py -n 6
int length = 69;
unsigned char stack_proc_string[] = {
        0x32, 0x48, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x38, 0x00, 0x60, 0x00, 0x10, 0x00, 0x44,
        0x07, 0x0a, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x00, 0x00, 0x00,
        0x0b, 0x00, 0x48, 0x00, 0x08, 0x00, 0x0b, 0x00, 0x48, 0x00, 0x10, 0x00, 0x0b, 0x00, 0x48,
        0x00, 0x18, 0x00, 0x0b, 0x00, 0x48, 0x00, 0x20, 0x00, 0x0b, 0x00, 0x48, 0x00, 0x28, 0x00,
        0x0b, 0x00, 0x70, 0x00, 0x30, 0x00, 0x0b, 0x00, 0x00
        };
```

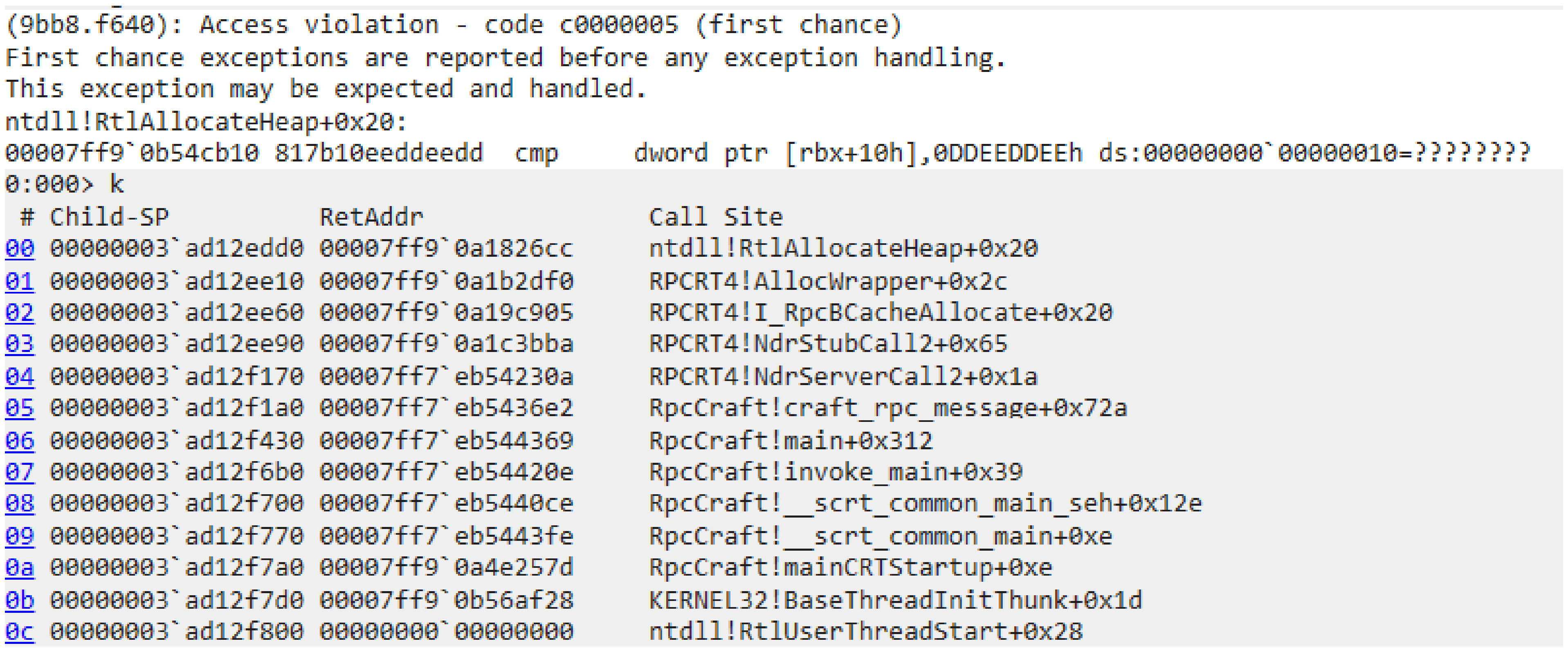
https://gist.github.com/klezVirus/cd1617904f96830f1cae65b350c8109b

Final Message Structure



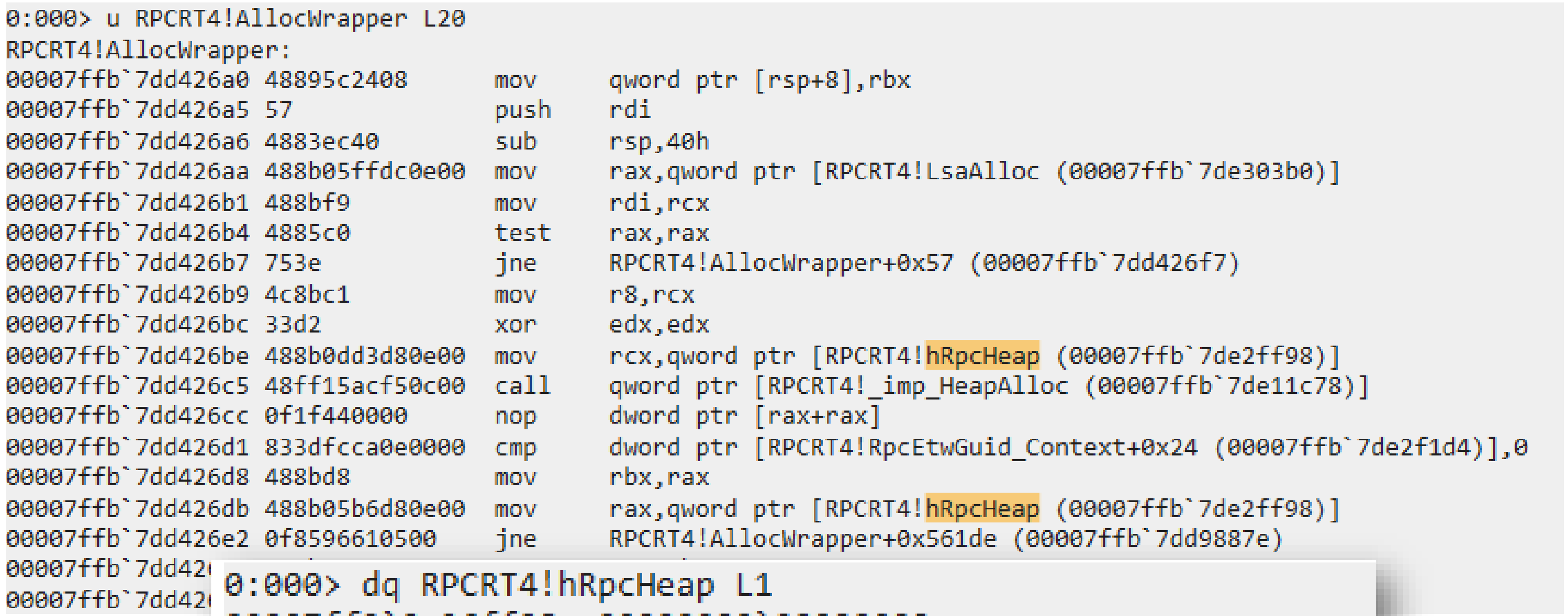
bsi

But we have a crash

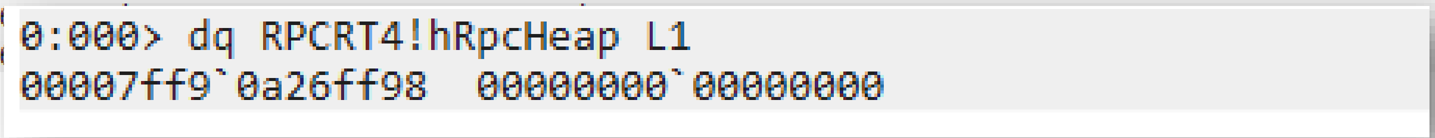


```
(9bb8.f640): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
ntdll!RtlAllocateHeap+0x20:
00007ff9`0b54cb10 817b10eeddeedd  cmp     dword ptr [rbx+10h],0DDEEDDEEh ds:00000000`00000010=????????
0:000> k
 # Child-SP          RetAddr               Call Site
00 00000003`ad12edd0 00007ff9`0a1826cc     ntdll!RtlAllocateHeap+0x20
01 00000003`ad12ee10 00007ff9`0a1b2df0     RPCRT4!AllocWrapper+0x2c
02 00000003`ad12ee60 00007ff9`0a19c905     RPCRT4!I_RpcBCacheAllocate+0x20
03 00000003`ad12ee90 00007ff9`0a1c3bba     RPCRT4!NdrStubCall2+0x65
04 00000003`ad12f170 00007ff7`eb54230a     RPCRT4!NdrServerCall2+0x1a
05 00000003`ad12f1a0 00007ff7`eb5436e2     RpcCraft!craft_rpc_message+0x72a
06 00000003`ad12f430 00007ff7`eb544369     RpcCraft!main+0x312
07 00000003`ad12f6b0 00007ff7`eb54420e     RpcCraft!invoke_main+0x39
08 00000003`ad12f700 00007ff7`eb5440ce     RpcCraft!__scrt_common_main_seh+0x12e
09 00000003`ad12f770 00007ff7`eb5443fe     RpcCraft!__scrt_common_main+0xe
0a 00000003`ad12f7a0 00007ff9`0a4e257d     RpcCraft!mainCRTStartup+0xe
0b 00000003`ad12f7d0 00007ff9`0b56af28     KERNEL32!BaseThreadInitThunk+0x1d
0c 00000003`ad12f800 00000000`00000000     ntdll!RtlUserThreadStart+0x28
```

But we have a crash



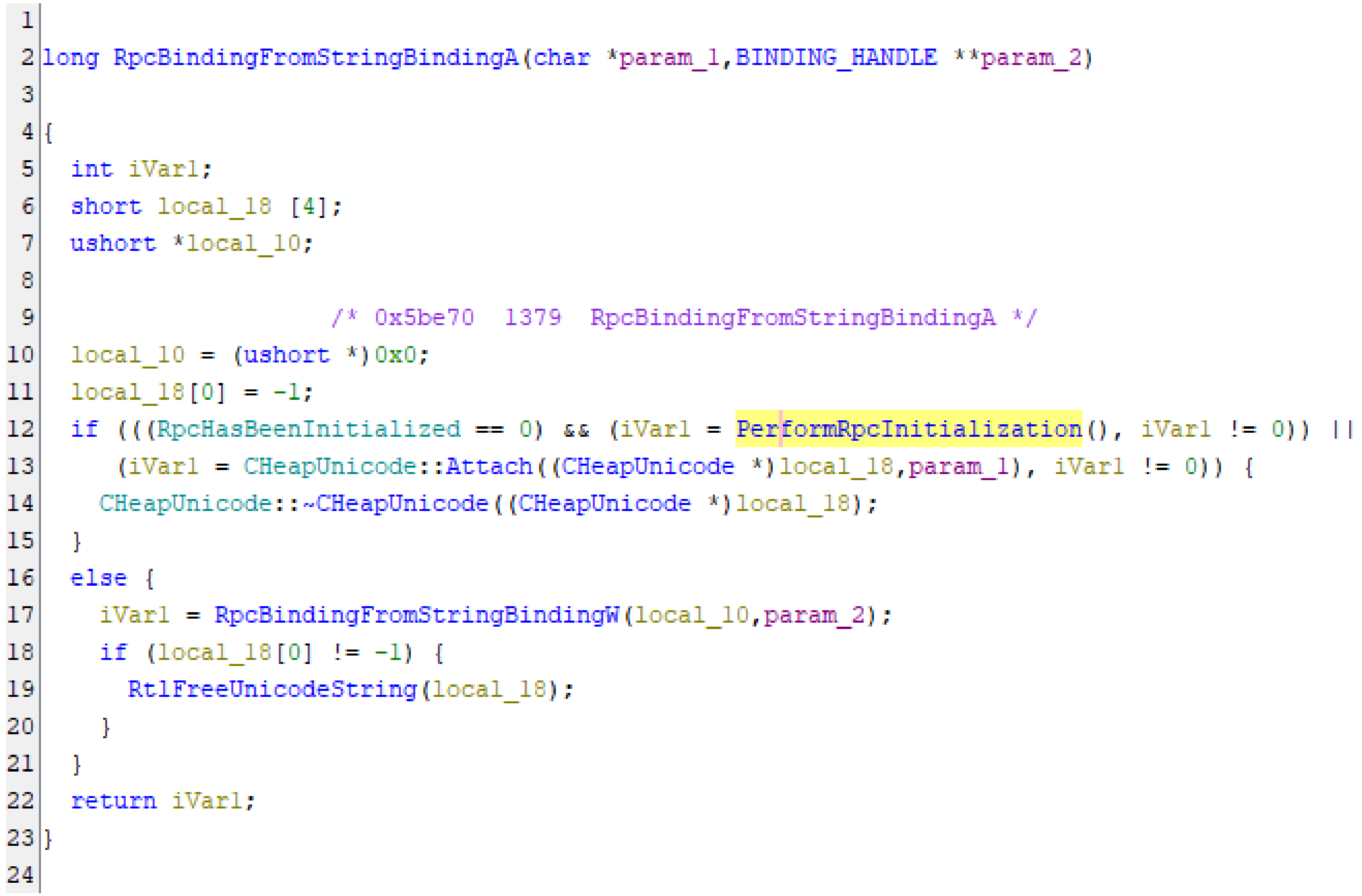
```
0:000> u RPCRT4!AllocWrapper L20
RPCRT4!AllocWrapper:
00007ffb`7dd426a0 48895c2408      mov     qword ptr [rsp+8],rbx
00007ffb`7dd426a5 57              push    rdi
00007ffb`7dd426a6 4883ec40        sub     rsp,40h
00007ffb`7dd426aa 488b05ffdc0e00  mov     rax,qword ptr [RPCRT4!LsaAlloc (00007ffb`7de303b0)]
00007ffb`7dd426b1 488bf9          mov     rdi,rcx
00007ffb`7dd426b4 4885c0          test    rax,rax
00007ffb`7dd426b7 753e            jne     RPCRT4!AllocWrapper+0x57 (00007ffb`7dd426f7)
00007ffb`7dd426b9 4c8bc1          mov     r8,rcx
00007ffb`7dd426bc 33d2            xor     edx,edx
00007ffb`7dd426be 488b0dd3d80e00  mov     rcx,qword ptr [RPCRT4!hRpcHeap (00007ffb`7de2ff98)]
00007ffb`7dd426c5 48ff15acf50c00  call    qword ptr [RPCRT4!_imp_HeapAlloc (00007ffb`7de11c78)]
00007ffb`7dd426cc 0f1f440000      nop     dword ptr [rax+rax]
00007ffb`7dd426d1 833dfcca0e0000  cmp     dword ptr [RPCRT4!RpcEtwGuid_Context+0x24 (00007ffb`7de2f1d4)],0
00007ffb`7dd426d8 488bd8          mov     rbx,rax
00007ffb`7dd426db 488b05b6d80e00  mov     rax,qword ptr [RPCRT4!hRpcHeap (00007ffb`7de2ff98)]
00007ffb`7dd426e2 0f8596610500    jne     RPCRT4!AllocWrapper+0x561de (00007ffb`7dd9887e)
00007ffb`7dd42
00007ffb`7dd42
```



```
0:000> dq RPCRT4!hRpcHeap L1
00007ff9`0a26ff98  00000000`00000000
```

We're missing initialization!

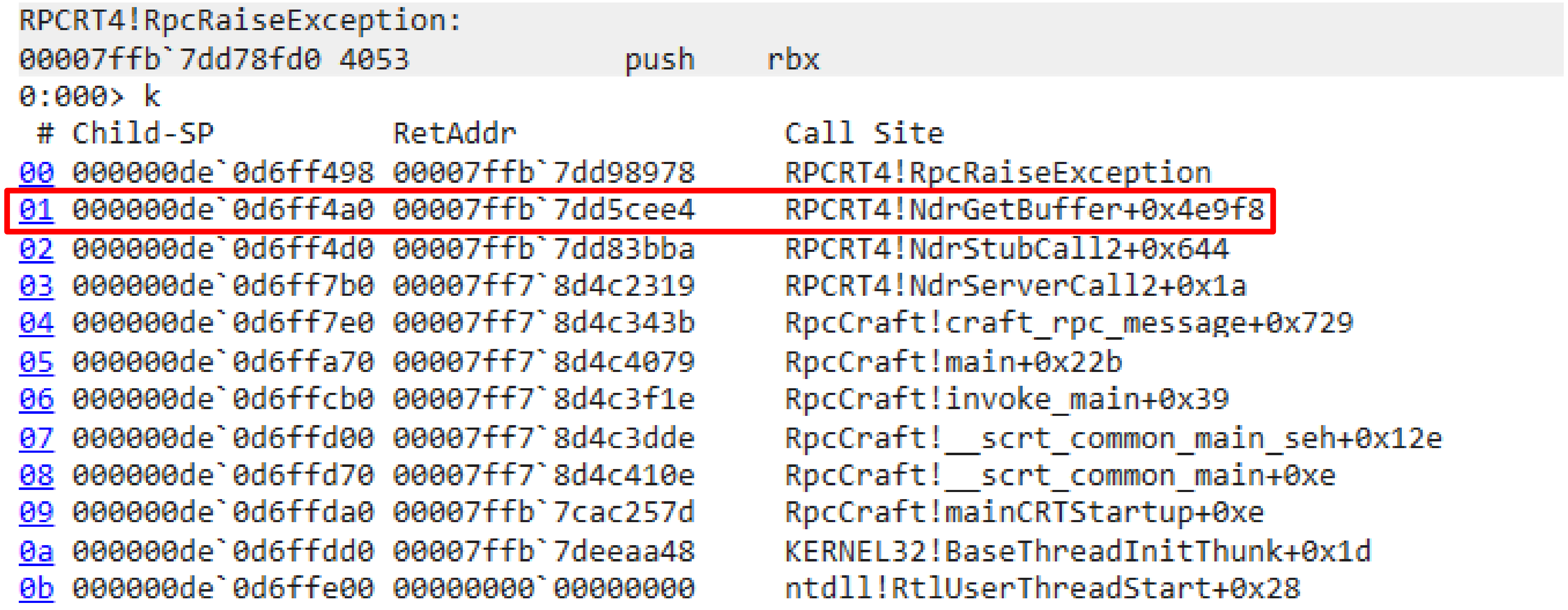
Finding the initialization routine



```

long RpcBindingFromStringBindingA(char *param_1,BINDING_HANDLE **param_2)

{
  int iVar1;
  short local_18 [4];
  ushort *local_10;

                    /* 0x5be70  1379  RpcBindingFromStringBindingA */
  local_10 = (ushort *)0x0;
  local_18[0] = -1;
  if (((RpcHasBeenInitialized == 0) && (iVar1 = PerformRpcInitialization(), iVar1 != 0)) ||
     (iVar1 = CHeapUnicode::Attach((CHeapUnicode *)local_18,param_1), iVar1 != 0)) {
    CHeapUnicode::~CHeapUnicode((CHeapUnicode *)local_18);
  }
  else {
    iVar1 = RpcBindingFromStringBindingW(local_10,param_2);
    if (local_18[0] != -1) {
      RtlFreeUnicodeString(local_18);
    }
  }
  return iVar1;
}

```

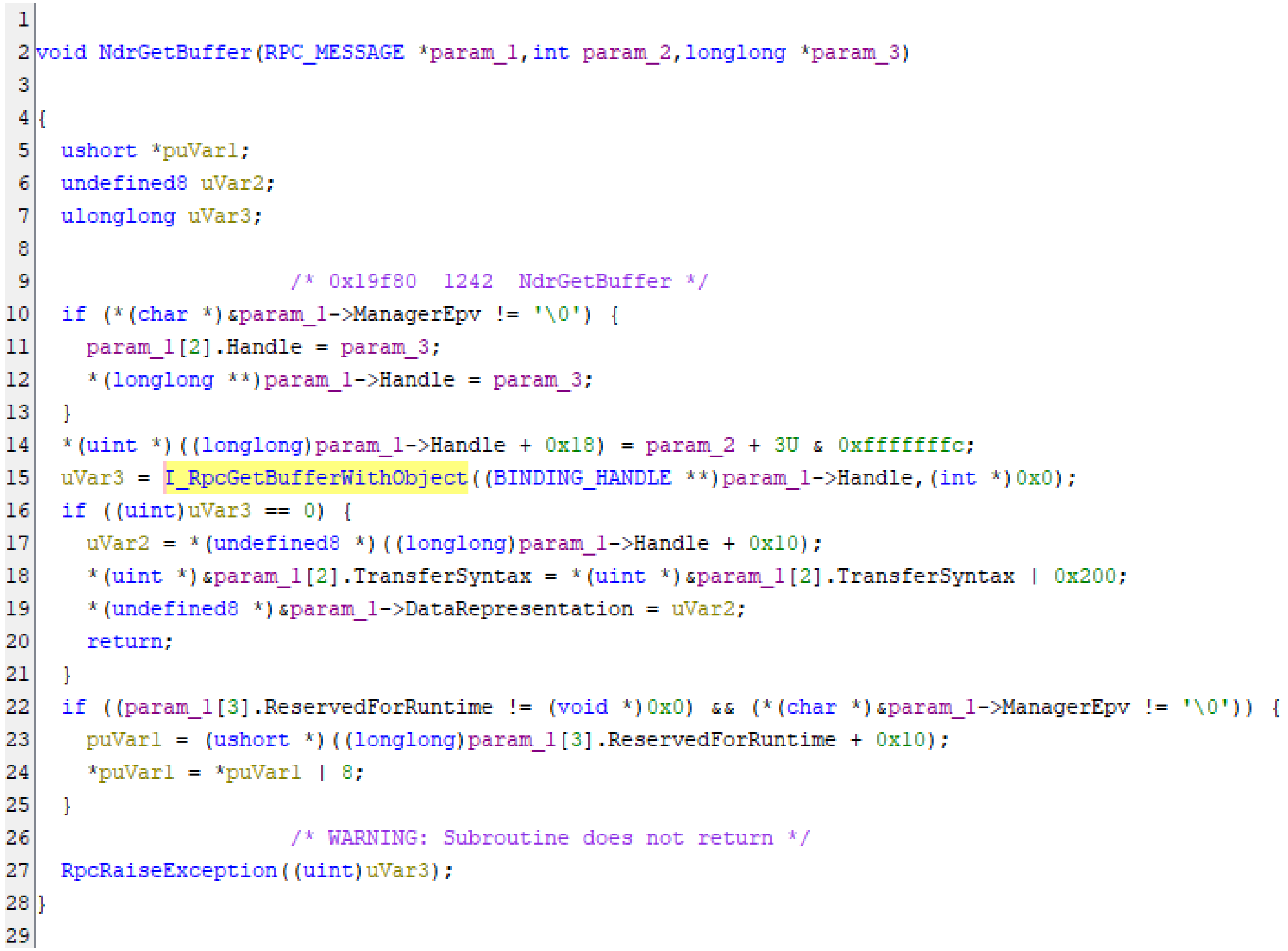
Caveats and Limitations

Missing Binding Handle



```
RPCRT4!RpcRaiseException:
00007ffb`7dd78fd0 4053            push    rbx
0:000> k
 # Child-SP          RetAddr               Call Site
00 000000de`0d6ff498 00007ffb`7dd98978     RPCRT4!RpcRaiseException
01 000000de`0d6ff4a0 00007ffb`7dd5cee4     RPCRT4!NdrGetBuffer+0x4e9f8
02 000000de`0d6ff4d0 00007ffb`7dd83bba     RPCRT4!NdrStubCall2+0x644
03 000000de`0d6ff7b0 00007ff7`8d4c2319     RPCRT4!NdrServerCall2+0x1a
04 000000de`0d6ff7e0 00007ff7`8d4c343b     RpcCraft!craft_rpc_message+0x729
05 000000de`0d6ffa70 00007ff7`8d4c4079     RpcCraft!main+0x22b
06 000000de`0d6ffcb0 00007ff7`8d4c3f1e     RpcCraft!invoke_main+0x39
07 000000de`0d6ffd00 00007ff7`8d4c3dde     RpcCraft!__scrt_common_main_seh+0x12e
08 000000de`0d6ffd70 00007ff7`8d4c410e     RpcCraft!__scrt_common_main+0xe
09 000000de`0d6ffda0 00007ffb`7cac257d     RpcCraft!mainCRTStartup+0xe
0a 000000de`0d6ffdd0 00007ffb`7deeaa48     KERNEL32!BaseThreadInitThunk+0x1d
0b 000000de`0d6ffe00 00000000`00000000     ntdll!RtlUserThreadStart+0x28
```

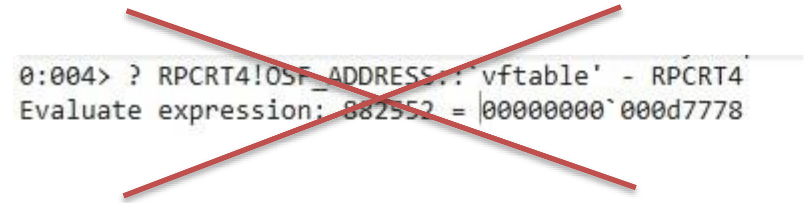
Missing Binding Handle



```

void NdrGetBuffer(RPC_MESSAGE *param_1,int param_2,longlong *param_3)

{
  ushort *puVar1;
  undefined8 uVar2;
  ulonglong uVar3;

                    /* 0x19f80  1242  NdrGetBuffer */
  if (*(char *)&param_1->ManagerEpv != '\0') {
    param_1[2].Handle = param_3;
    *(longlong **)param_1->Handle = param_3;
  }
  *(uint *)((longlong)param_1->Handle + 0x18) = param_2 + 3U & 0xfffffffc;
  uVar3 = I_RpcGetBufferWithObject((BINDING_HANDLE **)param_1->Handle,(int *)0x0);
  if ((uint)uVar3 == 0) {
    uVar2 = *(undefined8 *)((longlong)param_1->Handle + 0x10);
    *(uint *)&param_1[2].TransferSyntax = *(uint *)&param_1[2].TransferSyntax | 0x200;
    *(undefined8 *)&param_1->DataRepresentation = uVar2;
    return;
  }
  if ((param_1[3].ReservedForRuntime != (void *)0x0) && (*(char *)&param_1->ManagerEpv != '\0')) {
    puVar1 = (ushort *)((longlong)param_1[3].ReservedForRuntime + 0x10);
    *puVar1 = *puVar1 | 8;
  }
                    /* WARNING: Subroutine does not return */
  RpcRaiseException((uint)uVar3);
}

```

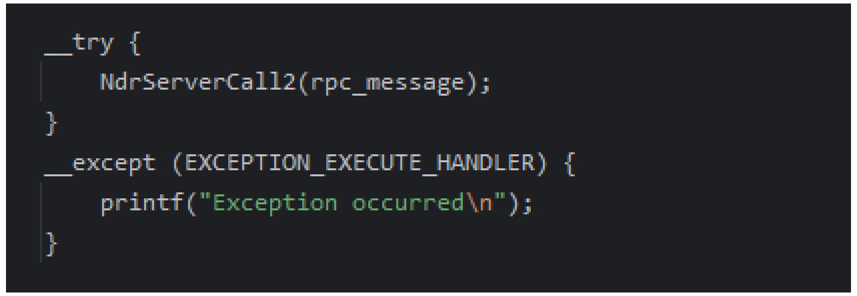
Possible Solutions

RPC_BINDING_HANDLE: The ideal solution would be to craft or reuse a valid handle, capable of passing all the checks performed on it.



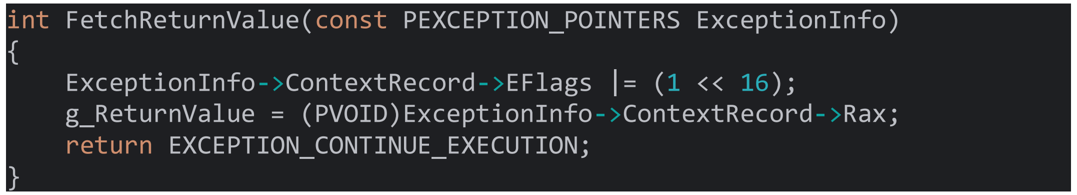
```
0:004> ? RPCRT4!OSF_ADDRESS::`vftable' - RPCRT4
Evaluate expression: 882552 = 00000000`000d7778
```

C++ Style Exception: Surrounding the faulting call within a __try/__except block is enough to prevent a crash, but usless to recover the return value.



```
__try {
    NdrServerCall2(rpc_message);
}
__except (EXCEPTION_EXECUTE_HANDLER) {
    printf("Exception occurred\n");
}
```

C++ Exception + VEH: This is the common ground, where we are both able to recover the value and prevent crashes.



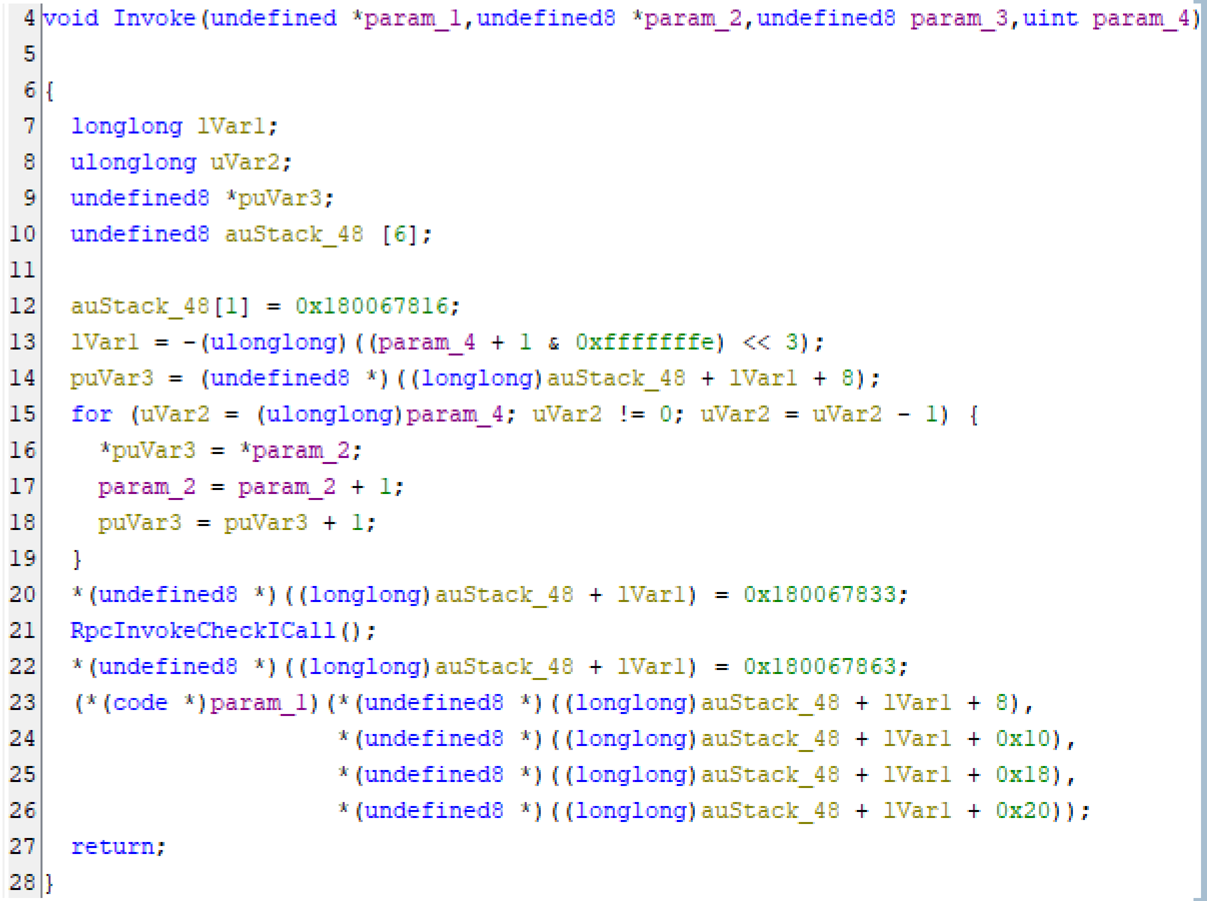
```
int FetchReturnValue(const PEXCEPTION_POINTERS ExceptionInfo)
{
    ExceptionInfo->ContextRecord->EFlags |= (1 << 16);
    g_ReturnValue = (PVOID)ExceptionInfo->ContextRecord->Rax;
    return EXCEPTION_CONTINUE_EXECUTION;
}
```

And Remotely?

Patching CFG: RPC calls are subject to CFG control checks. From Win11 the check is performed by RpcInvokeCheckICall.

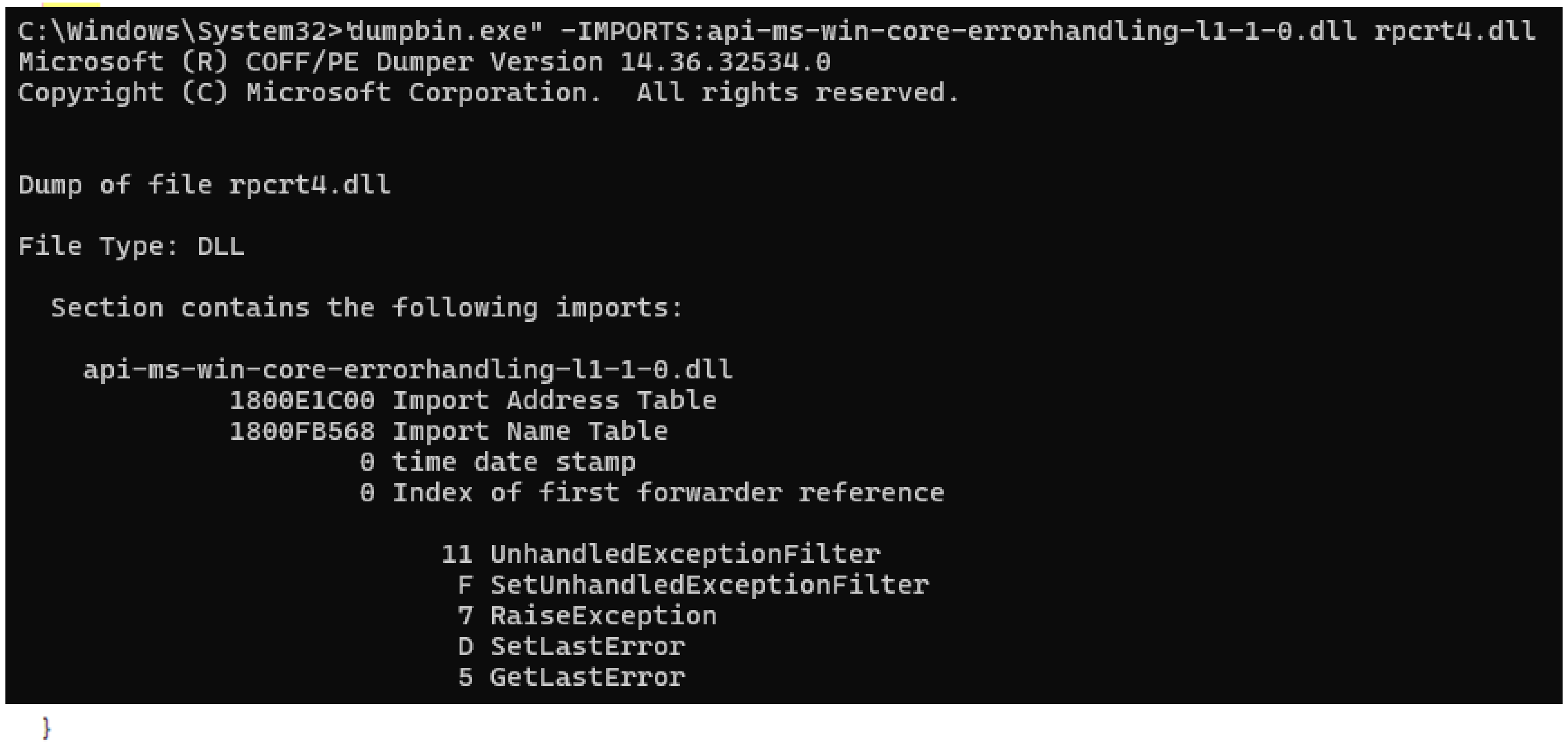
Remote Initialization: The RPC initialization needs to be performed remotely. As the function doesn't take parameters, it is simply invoked.

Redirect Exception to Thread Exit: As the call is invoked as a remote thread, redirecting the exception to the thread exit will prevent crashes.



```
void Invoke(undefined *param_1,undefined8 *param_2,undefined8 param_3,uint param_4)

{
  longlong lVar1;
  ulonglong uVar2;
  undefined8 *puVar3;
  undefined8 auStack_48 [6];

  auStack_48[1] = 0x180067816;
  lVar1 = -(ulonglong)((param_4 + 1 & 0xfffffffe) << 3);
  puVar3 = (undefined8 *)((longlong)auStack_48 + lVar1 + 8);
  for (uVar2 = (ulonglong)param_4; uVar2 != 0; uVar2 = uVar2 - 1) {
    *puVar3 = *param_2;
    param_2 = param_2 + 1;
    puVar3 = puVar3 + 1;
  }
  *(undefined8 *)((longlong)auStack_48 + lVar1) = 0x180067833;
  RpcInvokeCheckICall();
  *(undefined8 *)((longlong)auStack_48 + lVar1) = 0x180067863;
  (*(code *)param_1)(*(undefined8 *)((longlong)auStack_48 + lVar1 + 8),
                     *(undefined8 *)((longlong)auStack_48 + lVar1 + 0x10),
                     *(undefined8 *)((longlong)auStack_48 + lVar1 + 0x18),
                     *(undefined8 *)((longlong)auStack_48 + lVar1 + 0x20));
  return;
}
```

Patching – Thread Exit



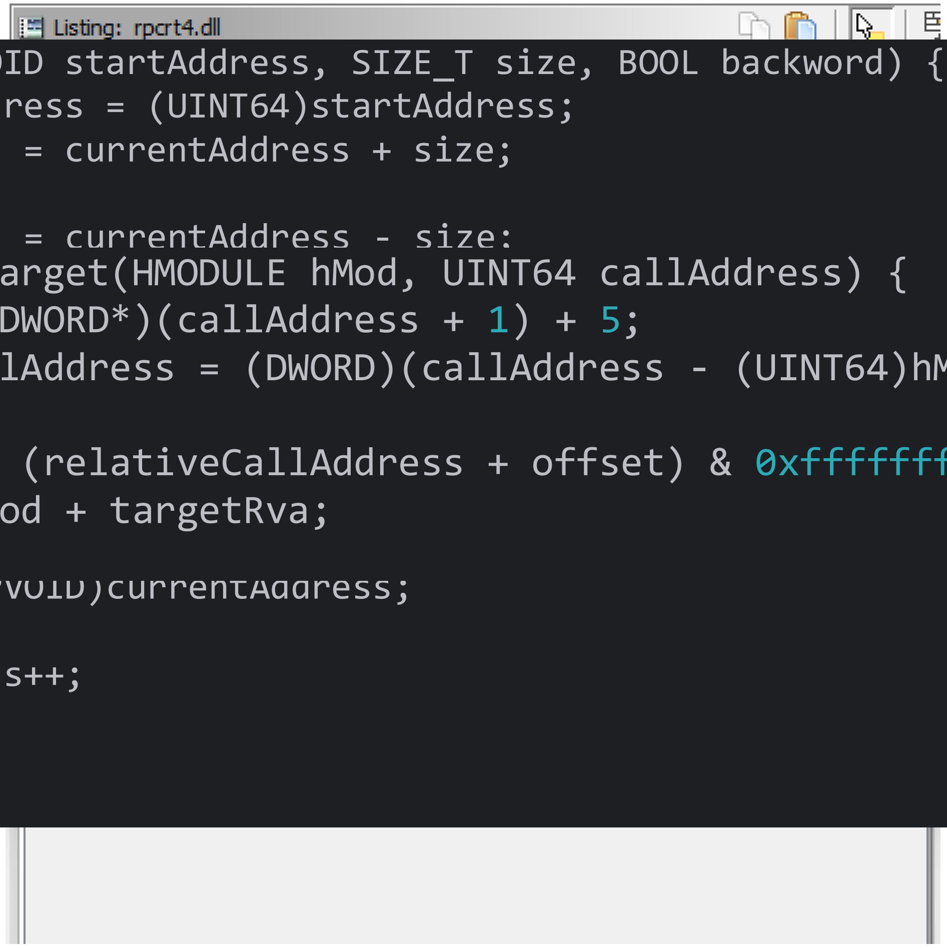
```
C:\Windows\System32>"dumpbin.exe" -IMPORTS:api-ms-win-core-errorhandling-l1-1-0.dll rpcrt4.dll
Microsoft (R) COFF/PE Dumper Version 14.36.32534.0
Copyright (C) Microsoft Corporation.  All rights reserved.


Dump of file rpcrt4.dll

File Type: DLL

  Section contains the following imports:

    api-ms-win-core-errorhandling-l1-1-0.dll
             1800E1C00 Import Address Table
             1800FB568 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

                          11 UnhandledExceptionFilter
                           F SetUnhandledExceptionFilter
                           7 RaiseException
                           D SetLastError
                           5 GetLastError
```

}

CAVEATS AND LIMITATIONS

CAVEATS AND LIMITATIONS

Patching – CFG Check



```
PVOID HuntForCall(PVOID startAddress, SIZE_T size, BOOL backword) {
    UINT64 currentAddress = (UINT64)startAddress;
    UINT64 endAddress = currentAddress + size;
    if (backword) {
       currentAddress = currentAddress - size;
```

```
UINT64 CalculateCallTarget(HMODULE hMod, UINT64 callAddress) {
    DWORD offset = *(DWORD*)(callAddress + 1) + 5;
    DWORD relativeCallAddress = (DWORD)(callAddress - (UINT64)hMod);

    DWORD targetRva = (relativeCallAddress + offset) & 0xffffffff;
    return (UINT64)hMod + targetRva;
}
```

```
            return (PVOID)currentAddress;
        }
        currentAddress++;
    }
    return NULL;
}
```

Operation Sequence

Find RpcInitialization Routine: Find and execute the RPC initialization routine to populate RPC Runtime global variables.

Find and Patch RpcRaiseException: Search and patch this function to avoid exception-derived crashes in remote processes.

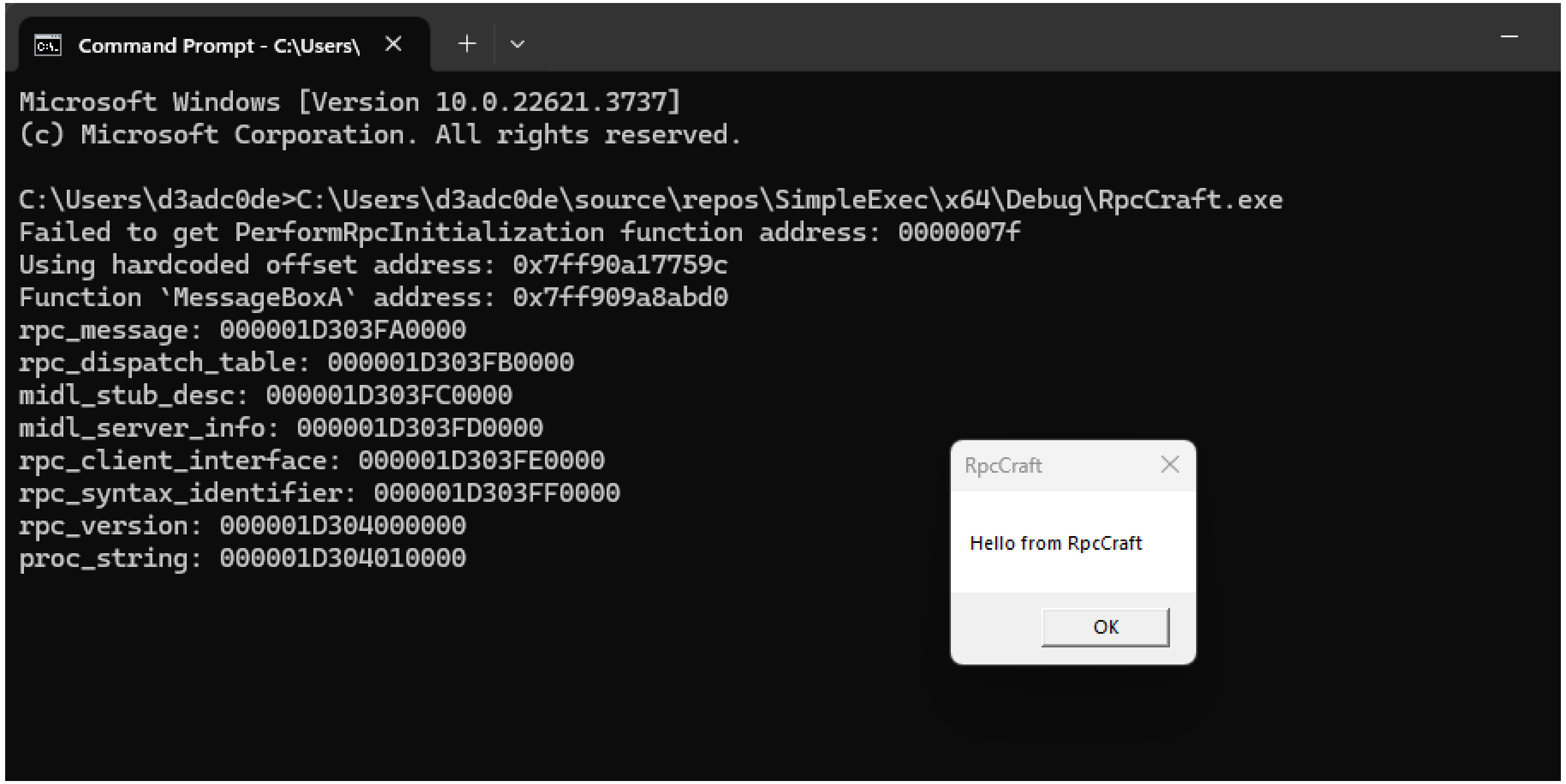
Find and Patch RpcInvokeCheckICall: Use COP hunting strategies to locate this call and patch it to defeat CFG.

Craft RPC_MESSAGE: Generate a valid RPC message to be passed to the RPC server call.

Profit: Simply invoke the call directly or via Thread Creation.

DEMO

Finally, try it - Local



Command Prompt - C:\Users\

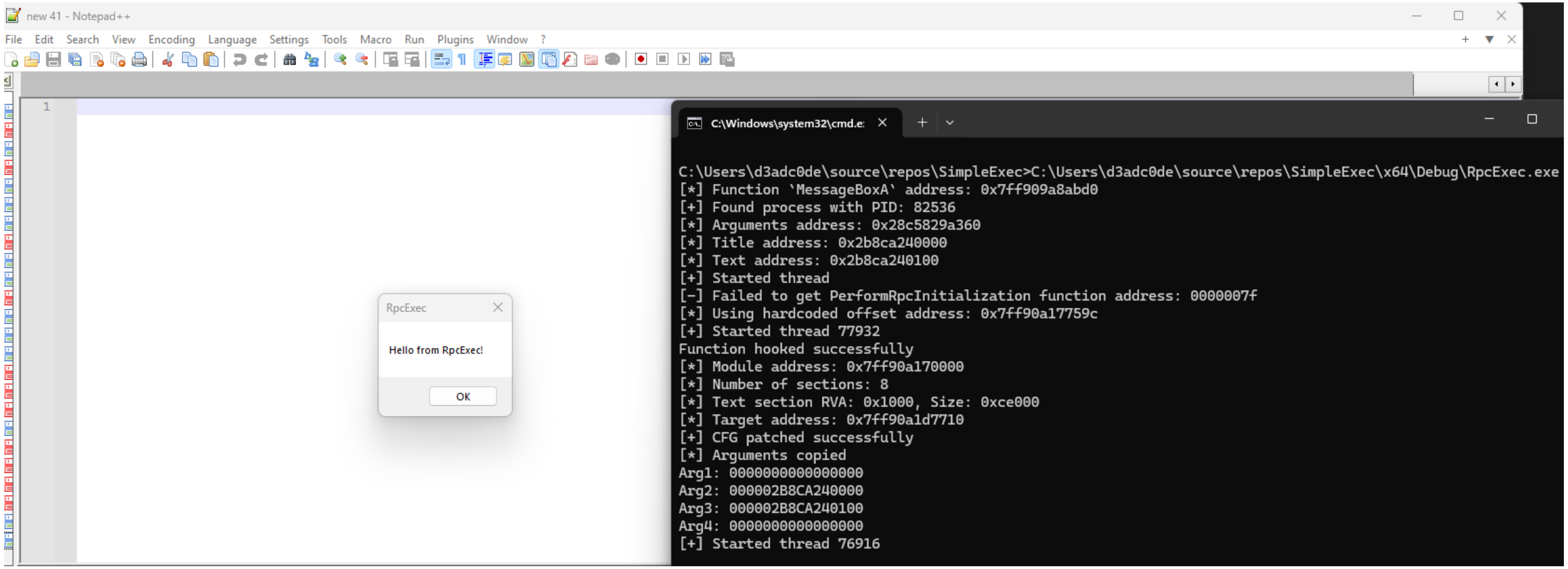
```
Microsoft Windows [Version 10.0.22621.3737]
(c) Microsoft Corporation. All rights reserved.

C:\Users\d3adc0de>C:\Users\d3adc0de\source\repos\SimpleExec\x64\Debug\RpcCraft.exe
Failed to get PerformRpcInitialization function address: 0000007f
Using hardcoded offset address: 0x7ff90a17759c
Function `MessageBoxA` address: 0x7ff909a8abd0
rpc_message: 000001D303FA0000
rpc_dispatch_table: 000001D303FB0000
midl_stub_desc: 000001D303FC0000
midl_server_info: 000001D303FD0000
rpc_client_interface: 000001D303FE0000
rpc_syntax_identifier: 000001D303FF0000
rpc_version: 000001D304000000
proc_string: 000001D304010000
```

RpcCraft

Hello from RpcCraft

DEMO

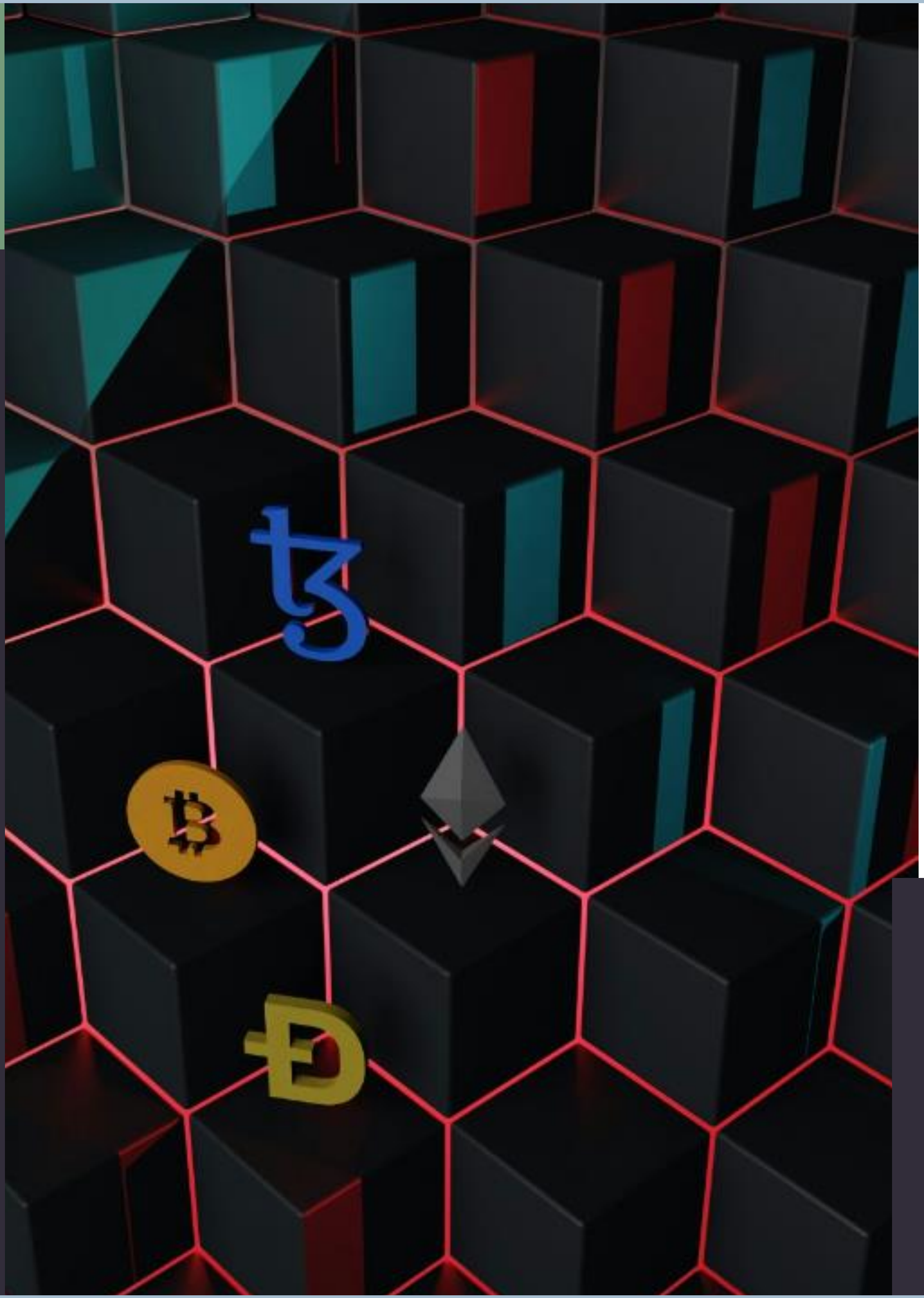
Finally, try it - Remote


new 41 - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
C:\Windows\system32\cmd.e:
OK

RpcExec

Hello from RpcExec!

```
C:\Users\d3adc0de\source\repos\SimpleExec>C:\Users\d3adc0de\source\repos\SimpleExec\x64\Debug\RpcExec.exe
[*] Function `MessageBoxA` address: 0x7ff909a8abd0
[+] Found process with PID: 82536
[*] Arguments address: 0x28c5829a360
[*] Title address: 0x2b8ca240000
[*] Text address: 0x2b8ca240100
[+] Started thread
[-] Failed to get PerformRpcInitialization function address: 0000007f
[*] Using hardcoded offset address: 0x7ff90a17759c
[+] Started thread 77932
Function hooked successfully
[*] Module address: 0x7ff90a170000
[*] Number of sections: 8
[*] Text section RVA: 0x1000, Size: 0xce000
[*] Target address: 0x7ff90a1d7710
[+] CFG patched successfully
[*] Arguments copied
Arg1: 0000000000000000
Arg2: 000002B8CA240000
Arg3: 000002B8CA240100
Arg4: 0000000000000000
[+] Started thread 76916
```



Conclusion

Key Takeaways

Callstack: The RPC fake call invocation can be used as a proxy to masquerade the callstack of calls that are originating from a new thread.

Remote Threads: This system allows to execute calls within a remote thread with arbitrary parameters, without requiring custom structures or handlers, offering an option to the widely used/abused NtContinue.

Railgun: This implementation can ultimately be extended to create an alternative version of the popular Railgun library by Metasploit.

Detection: The library, as of now, still suffers from the need of remotely patching RPCRT4 for stability, which offers a chance for detection by security solutions.

THANK YOU

Thank you!