black hat EUROPE 2024

Enhancing Automatic Vulnerability Discovery for Windows RPC/COM in New Ways

Speakers:

R4nger @ Cyber-Kunlun
Fangming Gu @ institute of information and engineering
Dr. Zhiniang Peng @ HUST & Cyber-Kunlun

WhoAmI

Zhiniang Peng @edwardzpeng
Associate Professor @HUST
Security Researcher @Cyber-Kunlun

PhD in Cryptography, Work in Defensive & Offensive security

Published many research in both Industry & Academia

More about me:https://sites.google.com/site/zhiniangpeng

HUST：Huazhong University of Science and Technology

Cyber-Kunlun:World-Leading Vulnerability Research in China

Some of My Bugs

CVE-2018-20694,CVE-2018-20746,CVE-2018-20693,CVE-2018-20692,CVE-2018-20696,CVE-2018-20689,CVE-2018-20690,CVE-2018-10812,CVE-2019-6184,CVE-2019-6186,CVE-2019-6487,CVE-2019-1253,CVE-2019-1292,CVE-2019-1317,CVE-2019-1340,CVE-2019-1342,CVE-2019-1374,CVE-2019-8162,CVE-2019-1474,CVE-2019-18371,CVE-2019-18370,CVE-2020-0616,CVE-2020-0635,CVE-2020-0636,CVE-2020-0638,CVE-2020-0641,CVE-2020-0648,CVE-2020-0697,CVE-2020-0730,CVE-2020-3808,CVE-2020-0747,CVE-2020-0753,CVE-2020-0754,CVE-2020-0777,CVE-2020-0780,CVE-2020-0785,CVE-2020-0786,CVE-2020-0789,CVE-2020-0794,CVE-2020-0797,CVE-2020-0800,CVE-2020-0805,CVE-2020-0808,CVE-2020-0819,CVE-2020-0822,CVE-2020-0835,CVE-2020-0841,CVE-2020-0844,CVE-2020-0849,CVE-2020-0854,CVE-2020-0858,CVE-2020-0863,CVE-2020-0864,CVE-2020-0865,CVE-2020-0868,CVE-2020-0871,CVE-2020-0896,CVE-2020-0897,CVE-2020-0899,CVE-2020-0900,CVE-2020-0934,CVE-2020-0935,CVE-2020-0936,CVE-2020-0942,CVE-2020-0944,CVE-2020-0983,CVE-2020-0985,CVE-2020-0989,CVE-2020-1000,CVE-2020-1002,CVE-2020-1010,CVE-2020-1011,CVE-2020-1029,CVE-2020-1068,CVE-2020-1077,CVE-2020-1084,CVE-2020-1086,CVE-2020-1090,CVE-2020-1094,CVE-2020-1109,CVE-2020-1120,CVE-2020-1121,CVE-2020-1123,CVE-2020-1124,CVE-2020-1125,CVE-2020-1131,CVE-2020-1134,CVE-2020-1137,CVE-2020-1139,CVE-2020-1144,CVE-2020-1146,CVE-2020-1151,CVE-2020-1155,CVE-2020-1156,CVE-2020-1157,CVE-2020-1158,CVE-2020-1163,CVE-2020-1164,CVE-2020-1165,CVE-2020-1166,CVE-2020-1184,CVE-2020-1185,CVE-2020-1186,CVE-2020-1187,CVE-2020-1188,CVE-2020-1189,CVE-2020-1190,CVE-2020-1191,CVE-2020-1196,CVE-2020-1199,CVE-2020-1201,CVE-2020-1204,CVE-2020-1209,CVE-2020-1211,CVE-2020-1217,CVE-2020-1222,CVE-2020-1231,CVE-2020-1233,CVE-2020-1235,CVE-2020-1244,CVE-2020-1257,CVE-2020-1264,CVE-2020-1269,CVE-2020-1270,CVE-2020-1273,CVE-2020-1274,CVE-2020-1276,CVE-2020-1277,CVE-2020-1278,CVE-2020-1282,CVE-2020-1283,CVE-2020-1304,CVE-2020-1305,CVE-2020-1306,CVE-2020-1307,CVE-2020-1309,CVE-2020-1312,CVE-2020-1317,CVE-2020-1337,CVE-2020-1344,CVE-2020-1346,CVE-2020-1347,CVE-2020-1352,CVE-2020-1356,CVE-2020-1357,CVE-2020-1360,CVE-2020-1361,CVE-2020-1362,CVE-2020-1364,CVE-2020-5957,CVE-2020-1366,CVE-2020-1372,CVE-2020-1373,CVE-2020-1375,CVE-2020-1385,CVE-2020-1392,CVE-2020-1393,CVE-2020-1394,CVE-2020-1399,CVE-2020-1404,CVE-2020-1405,CVE-2020-1424,CVE-2020-1427,CVE-2020-1441,CVE-2020-0518,CVE-2020-1461,CVE-2020-1465,CVE-2020-1472,CVE-2020-1474,CVE-2020-1475,CVE-2020-1484,CVE-2020-1485,CVE-2020-1511,CVE-2020-1512,CVE-2020-0516,CVE-2020-1516,CVE-2020-1517,CVE-2020-1518,CVE-2020-1519,CVE-2020-1521,CVE-2020-1522,CVE-2020-1524,CVE-2020-1528,CVE-2020-1538,CVE-2020-8741,CVE-2020-1548,CVE-2020-1549,CVE-2020-1550,CVE-2020-1552,CVE-2020-1590,CVE-2020-1130,CVE-2020-16851,CVE-2020-16852,CVE-2020-1122,CVE-2020-1038,CVE-2020-17089,CVE-2020-16853,CVE-2020-16879,CVE-2020-16900,CVE-2020-16980,CVE-2020-17014,CVE-2020-17070,CVE-2020-17073,CVE-2020-17074,CVE-2020-17075,CVE-2020-17076,CVE-2020-17077,CVE-2020-17092,CVE-2020-17097,CVE-2020-17120,CVE-2021-1649,CVE-2021-1650,CVE-2021-1651,CVE-2021-1659,CVE-2021-1680,CVE-2021-1681,CVE-2021-1686,CVE-2021-1687,CVE-2021-1688,CVE-2021-1689,CVE-2021-1690,CVE-2021-1718,CVE-2021-1722,CVE-2021-24072,CVE-2021-24077,CVE-2021-3750,CVE-2021-24088,CVE-2021-26869,CVE-2021-26870,CVE-2021-26871,CVE-2021-26885,CVE-2021-28347,CVE-2021-28351,CVE-2021-28436,CVE-2021-28450,CVE-2021-31966,CVE-2021-34527,CVE-2021-42321,CVE-2021-36970,CVE-2021-38657,CVE-2021-40485,CVE-2021-41366,CVE-2021-42294,CVE-2021-42297,CVE-2021-43216,CVE-2021-43223,CVE-2021-43248,CVE-2022-21835,CVE-2022-21837,CVE-2022-21878,CVE-2022-21881,CVE-2022-21888,CVE-2022-21971,CVE-2022-21974,CVE-2022-21992,CVE-2022-23285,CVE-2022-23290,CVE-2022-24454,CVE-2022-29108,CVE-2022-24547,CVE-2022-23270,CVE-2022-26930,CVE-2022-29103,CVE-2022-29113,CVE-2022-38036,CVE-2022-35793,CVE-2022-35755,CVE-2022-35749,CVE-2022-35746,CVE-2022-34690,CVE-2022-21980,CVE-2022-22050,CVE-2022-22024,CVE-2022-22022,CVE-2022-30226,CVE-2022-30157,CVE-2022-29108,CVE-2022-21999,CVE-2023-21683,CVE-2023-21684,CVE-2023-21693,CVE-2023-21801,CVE-2023-23403,CVE-2023-23406,CVE-2023-23413,CVE-2023-24856,CVE-2023-24857,CVE-2023-24858,CVE-2023-24863,CVE-2023-24865,CVE-2023-24866,CVE-2023-24867,CVE-2023-24907,CVE-2023-24868,CVE-2023-24909,CVE-2023-24870,CVE-2023-24872,CVE-2023-24913,CVE-2023-24876,CVE-2023-24924,CVE-2023-24883,CVE-2023-24925,CVE-2023-24884,CVE-2023-24926,CVE-2023-24885,CVE-2023-24927,CVE-2023-24886,CVE-2023-24928,CVE-2023-24887,CVE-2023-24929,CVE-2023-28243,CVE-2023-28296,CVE-2023-29366,CVE-2023-29367,CVE-2023-32017,CVE-2023-32039,CVE-2023-32040,CVE-2023-32041,CVE-2023-32042,CVE-2023-32085,CVE-2023-35296,CVE-2023-35302,CVE-2023-35306,CVE-2023-35313,CVE-2023-35323,CVE-2023-35324,CVE-2023-36898,CVE-2023-36792,CVE-2023-36704,CVE-2023-36418,CVE-2023-36395,CVE-2023-36393,CVE-2023-35624,CVE-2023-21683,CVE-2023-29366,CVE-2023-46138,CVE-2023-42820,CVE-2023-42819,CVE-2024-21426,CVE-2024-29156,CVE-2024-26198,CVE-2024-21435,CVE-2024-21329,CVE-2024-21384,CVE-2024-20691,CVE-2024-21433,CVE-2024-20694,CVE-2024-0087,CVE-2024-0088,CVE-2024-30060,CVE-2024-29989,CVE-2024-38077,CVE-2024-38024,CVE-2024-38023,CVE-2024-38076,CVE-2024-38074,CVE-2024-38073,CVE-2024-35261,CVE-2024-38072,CVE-2024-38071,CVE-2024-38015,CVE-2024-43467,CVE-2024-43455,CVE-2024-38231,CVE-2024-38258,CVE-2024-43454,CVE-2024-38263,CVE-2024-38260,CVE-2024-38228,CVE-2024-43495,CVE-2024-43470,CVE-2024-38225,CVE-2024-43467,CVE-2024-38097,CVE-2024-38262,CVE-2024-43583

WhoAmI

Fangming Gu @afang5472

PhD Student at University of Chinese Academy and Sciences

Research on Windows Security and Reverse Engineering

Publishes on Usenix Security, NDSS and Black hat

Interested in automated bug finding ideas

WhoAmI

R4nger

Security Researcher @Cyber-Kunlun
Focus on Windows Security for several years
MSRC MVR

Cyber-Kunlun:World-Leading Vulnerability Research in China

Agenda

- ➢Introduction
- ➢ALPC Internals
- ➢XALPC Fuzz
- ➢XALPC Monitor
- ➢Summary

black hat EUROPE 2024

Introduction

Background

- RPC/COM is an important attack surface for Windows
  - RCE, LPE and Sandbox Escape
  - Many in-the-wild exploits in the past
- Previous vulnerability research focused on existing pattern
  - Race condition, File Redirection etc
  - Requiring significant time and effort investment

Motivation

Fuzzing RPC/COM Server in Windows

- Creating custom corpus and fuzzers for each interface
- Reverse engineering process proves inefficient and cumbersome

Our solution: XALPC

- A cutting-edge RPC/COM fuzzing and monitoring tool to hunting system-wide RPC/COM vulnerabilities

black hat EUROPE 2024

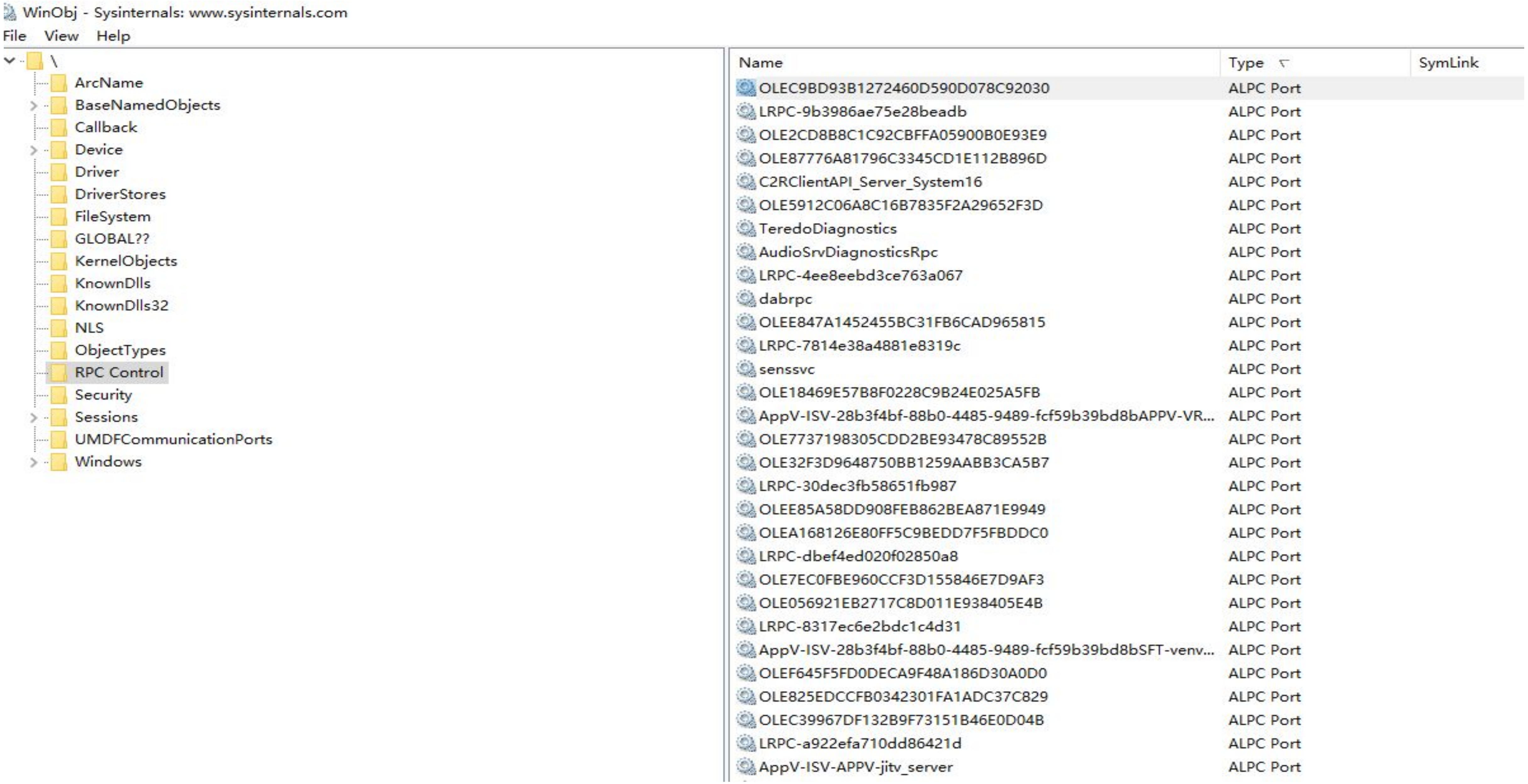
ALPC Internals

ALPC Internals

- ALPC (Advanced Local Procedure Call)
    - Inter-process communication on Windows
    - Server listening on an ALPC port
    - Client connecting to that port

- Widely used in Windows
    - COM/RPC/ALPC Server depend on it for IPC

ALPC Port



WinObj - Sysinternals: www.sysinternals.com

- \
  - ArcName
  - BaseNamedObjects
  - Callback
  - Device
  - Driver
  - DriverStores
  - FileSystem
  - GLOBAL??
  - KernelObjects
  - KnownDlls
  - KnownDlls32
  - NLS
  - ObjectTypes
  - RPC Control
  - Security
  - Sessions
  - UMDFCommunicationPorts
  - Windows

Name	Type	SymLink
OLEC9BD93B1272460D590D078C92030	ALPC Port	
LRPC-9b3986ae75e28beadb	ALPC Port	
OLE2CD8B8C1C92CBFFA05900B0E93E9	ALPC Port	
OLE87776A81796C3345CD1E112B896D	ALPC Port	
C2RClientAPI_Server_System16	ALPC Port	
OLE5912C06A8C16B7835F2A29652F3D	ALPC Port	
TeredoDiagnostics	ALPC Port	
AudioSrvDiagnosticsRpc	ALPC Port	
LRPC-4ee8eebd3ce763a067	ALPC Port	
dabrpc	ALPC Port	
OLEE847A1452455BC31FB6CAD965815	ALPC Port	
LRPC-7814e38a4881e8319c	ALPC Port	
senssvc	ALPC Port	
OLE18469E57B8F0228C9B24E025A5FB	ALPC Port	
AppV-ISV-28b3f4bf-88b0-4485-9489-fcf59b39bd8bAPPV-VR...	ALPC Port	
OLE7737198305CDD2BE93478C89552B	ALPC Port	
OLE32F3D9648750BB1259AABB3CA5B7	ALPC Port	
LRPC-30dec3fb58651fb987	ALPC Port	
OLEE85A58DD908FEB862BEA871E9949	ALPC Port	
OLEA168126E80FF5C9BEDD7F5FBDDC0	ALPC Port	
LRPC-dbef4ed020f02850a8	ALPC Port	
OLE7EC0FBE960CCF3D155846E7D9AF3	ALPC Port	
OLE056921EB2717C8D011E938405E4B	ALPC Port	
LRPC-8317ec6e2bdc1c4d31	ALPC Port	
AppV-ISV-28b3f4bf-88b0-4485-9489-fcf59b39bd8bSFT-venv...	ALPC Port	
OLEF645F5FD0DECA9F48A186D30A0D0	ALPC Port	
OLE825EDCCFB0342301FA1ADC37C829	ALPC Port	
OLEC39967DF132B9F73151B46E0D04B	ALPC Port	
LRPC-a922efa710dd86421d	ALPC Port	
AppV-ISV-APPV-jitv_server	ALPC Port	

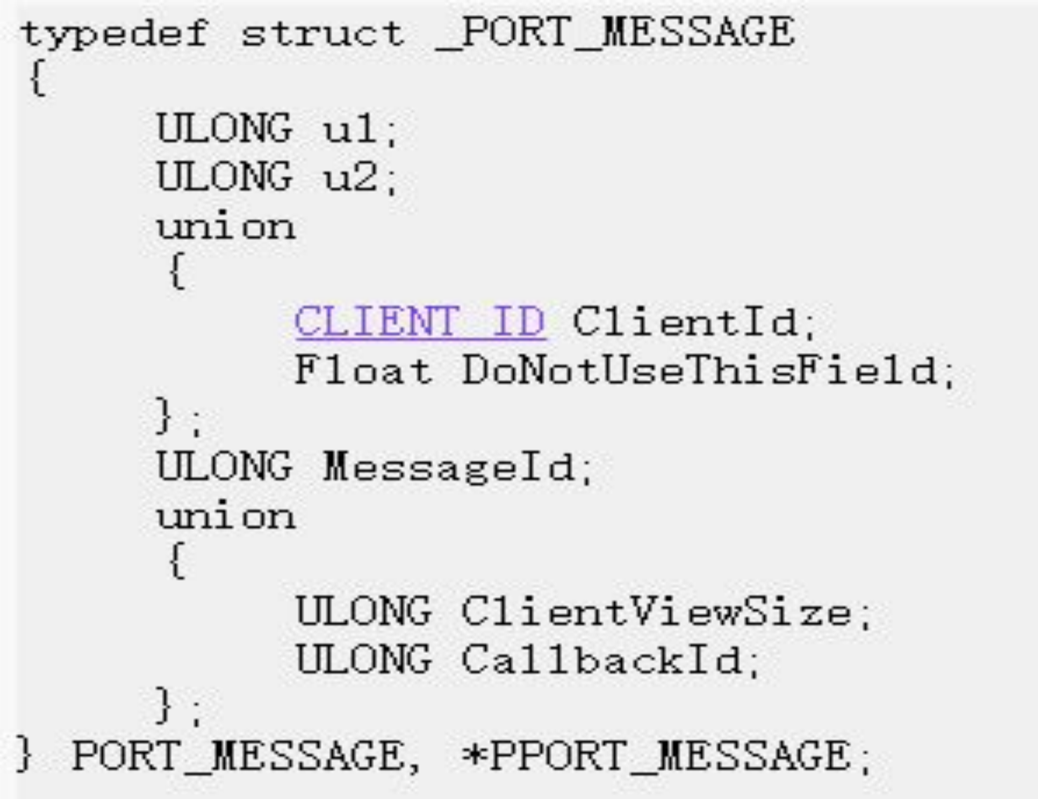
ALPC API

- ALPC Server
  - NtAlpcCreatePort
  - NtAlpcAcceptConnectPort
  - NtAlpcSendWaitReceivePort
- ALPC Client
  - NtAlpcConnectPort
  - NtAlpcDisconnectPort
  - NtAlpcSendWaitReceivePort

ALPC Message

ALPC Message include two parts

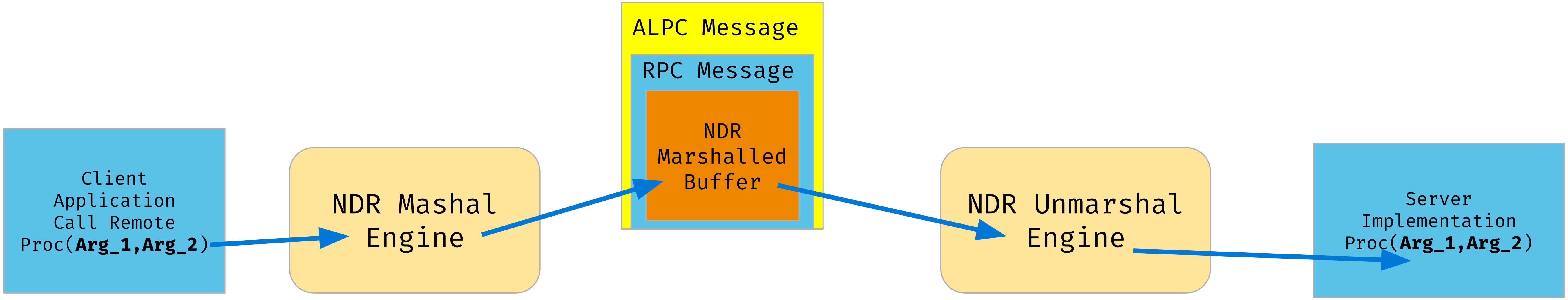
- PORT_MESSAGE: the header and data of the message
- ALPC_MESSAGE_ATTRIBUTES: Attributes header and data for advanced features



```
typedef struct _PORT_MESSAGE
{
        ULONG u1;
        ULONG u2;
        union
        {
                CLIENT_ID ClientId;
                Float DoNotUseThisField;
        };
        ULONG MessageId;
        union
        {
                ULONG ClientViewSize;
                ULONG CallbackId;
        };
} PORT_MESSAGE, *PPORT_MESSAGE;
```

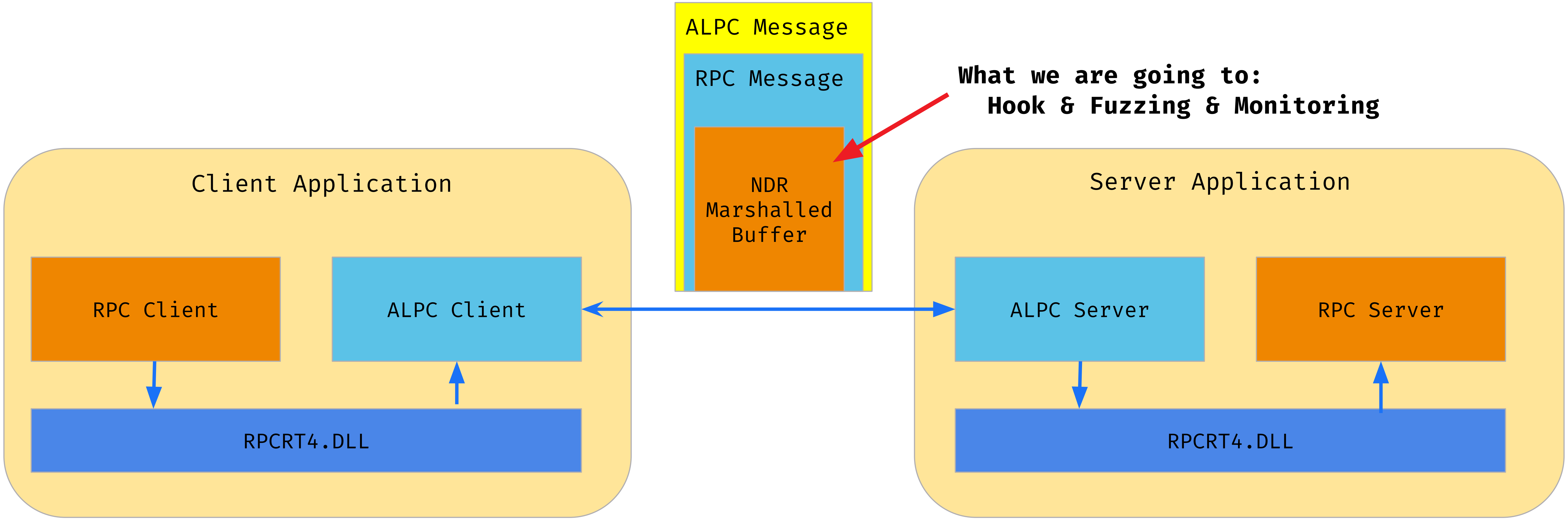
NDR Engine

- Network Data Representation (NDR) Engine
  - The marshaling engine of the RPC and DCOM components
  - Actual data in ALPC message is marshalled and unmarshalled by NDR


Client Application Call Remote Proc(**Arg_1,Arg_2**)
NDR Mashal Engine
ALPC Message
RPC Message
NDR Marshalled Buffer
NDR Unmarshal Engine
Server Implementation Proc(**Arg_1,Arg_2**)

- Online Document
  - https://learn.microsoft.com/en-us/windows/win32/rpc/rpc-ndr-engine

RPC over ALPC Architecture



What we are going to:
Hook & Fuzzing & Monitoring

Related Work：

A view into ALPC-RPC	Clément Rouault & Thomas Imbert
All about the ALPC, RPC, LPC, LRPC in your PC	Alex Ionescu
LPC & ALPC Interfaces	Thomas Garnier
ALPC Fuzzing Toolkit	Ben Nagy
Having FUN with COM	James Forshaw
COM in 60 Seconds	James Forshaw

black hat EUROPE 2024

XALPC Fuzz

How to fuzz ALPC effectively?

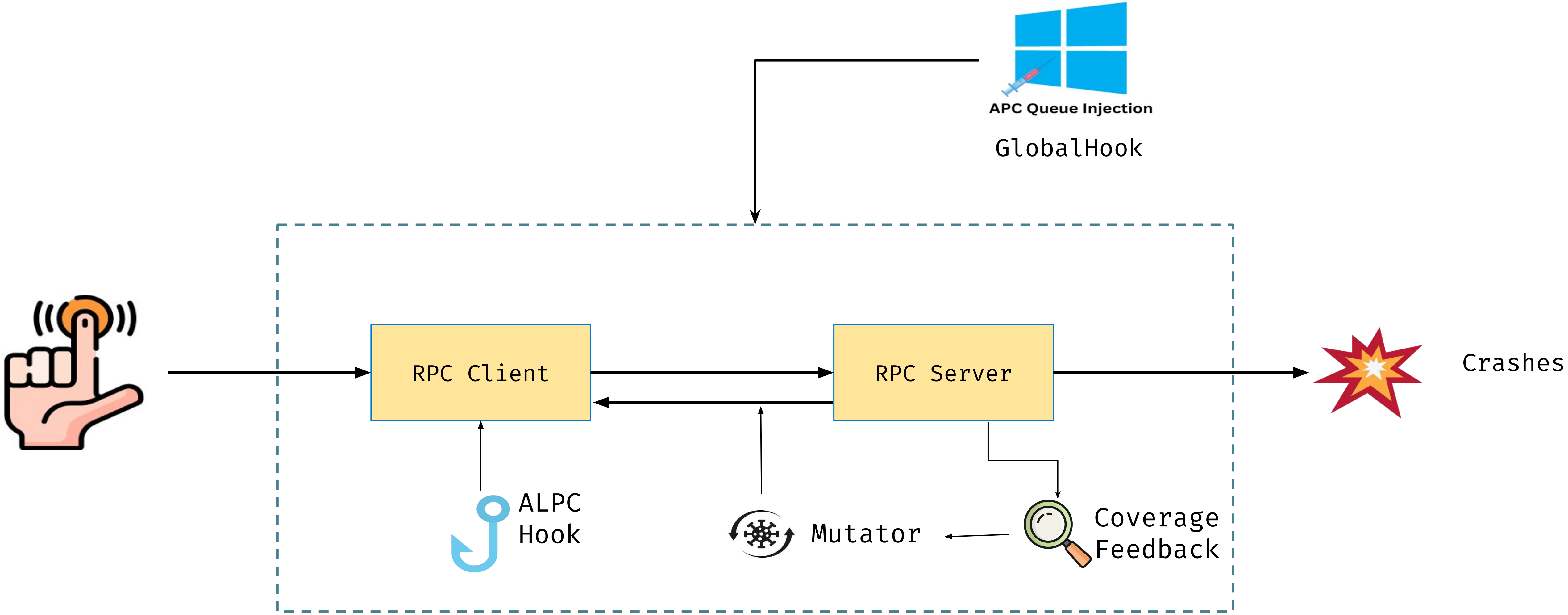
Challenges

1. ∞ Huge amounts of ALPC messages system-wide.
2. 🎯 How to mutate the message sent to ALPC Server?
3. ⌖ How to trigger more hidden ALPC messages?

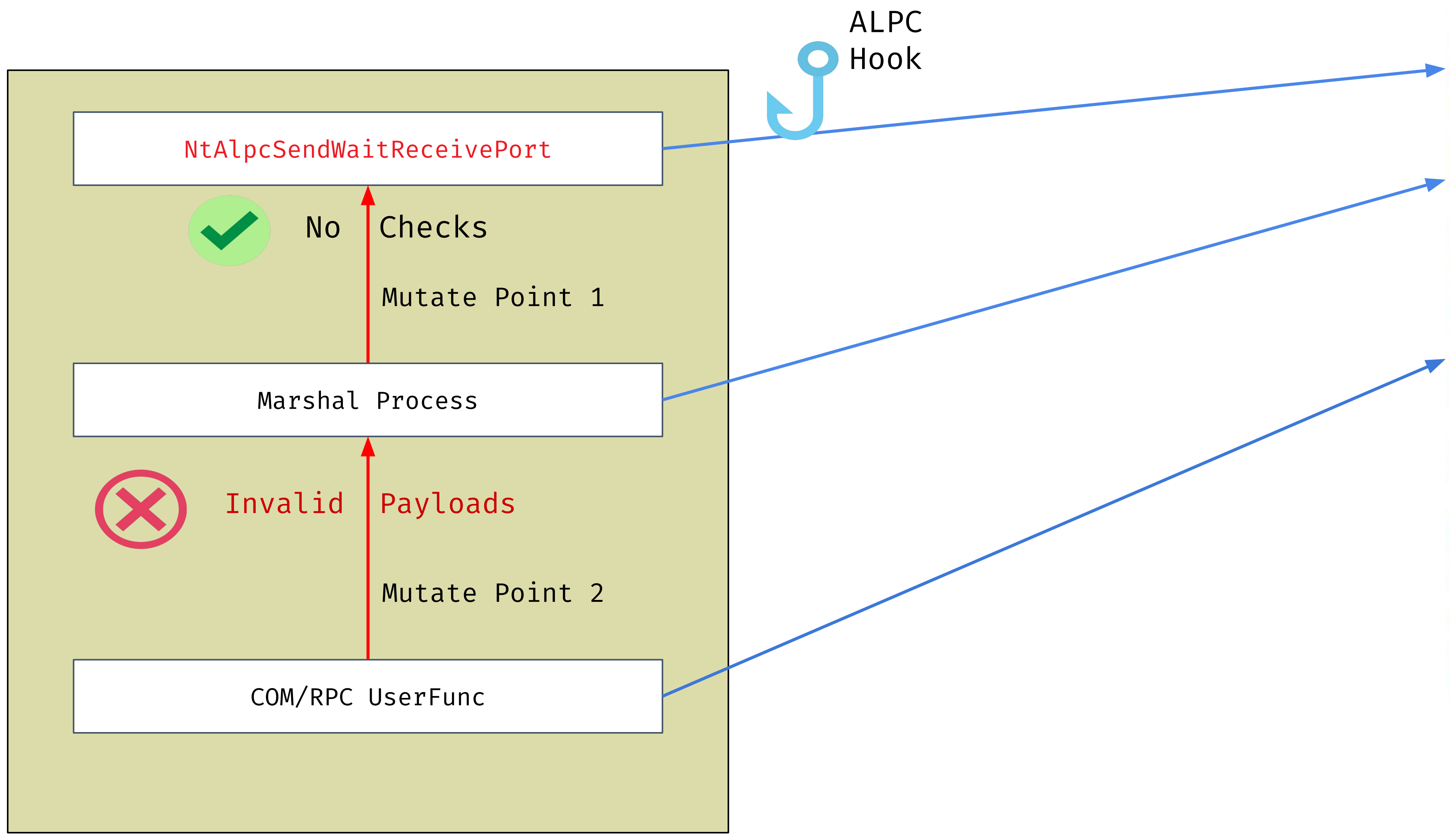
XALPCFuzz

Proposing a hook-based framework to fuzz Windows RPC/COM messages live & at scale.

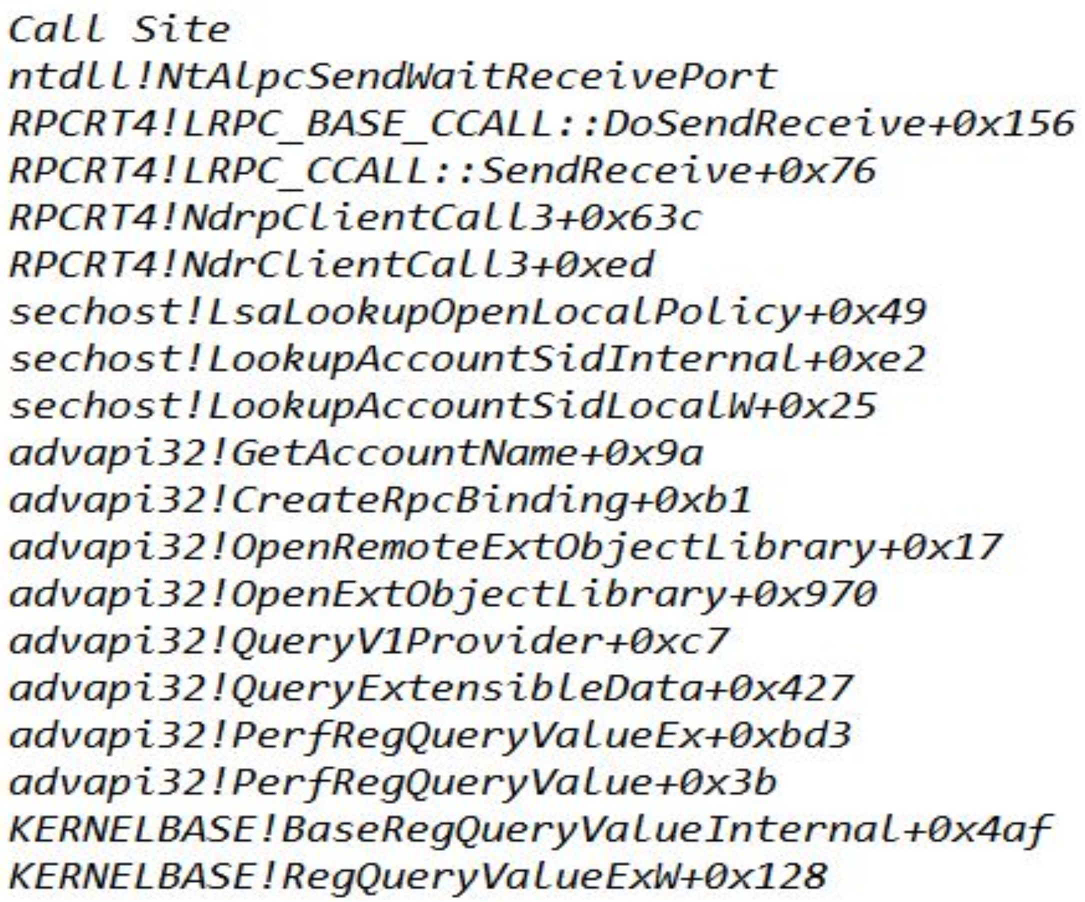
XALPCFuzz - Design



XALPCFuzz - Where to hook & mutate



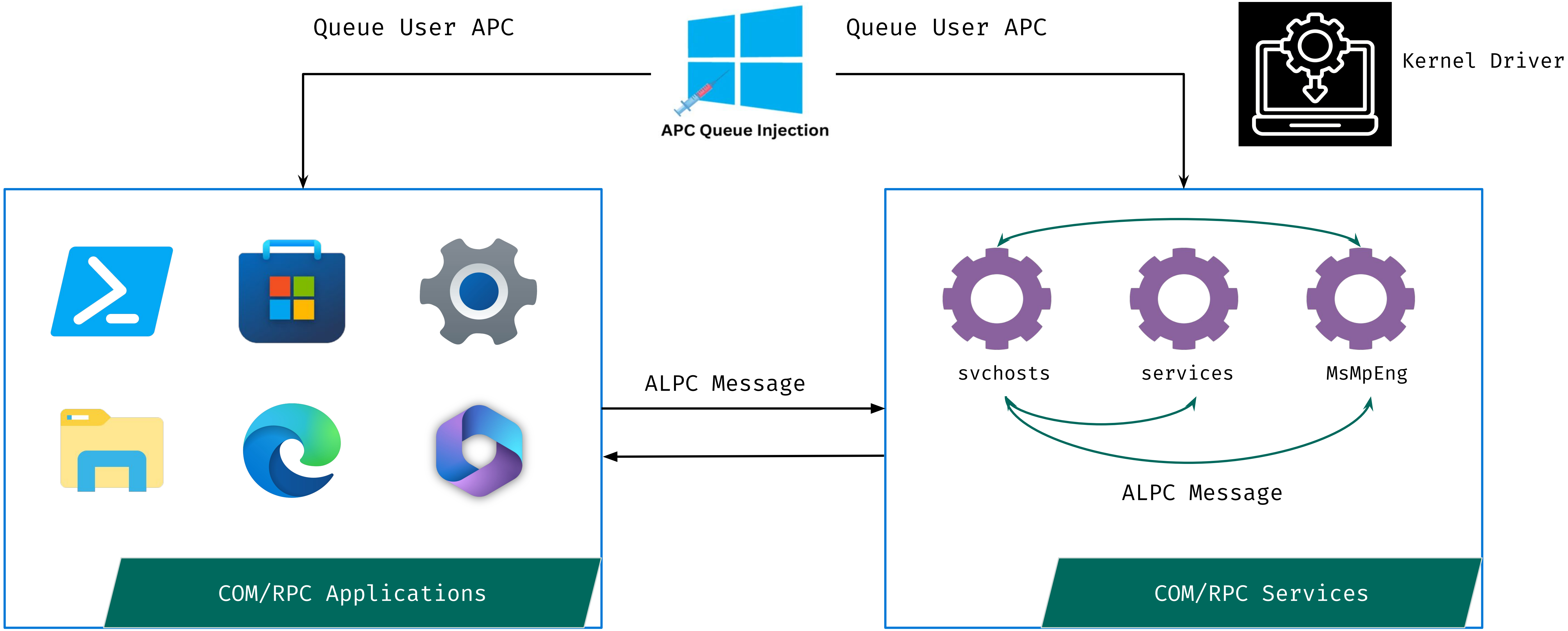
RPC Client



```
Call Site
ntdll!NtAlpcSendWaitReceivePort
RPCRT4!LRPC_BASE_CCALL::DoSendReceive+0x156
RPCRT4!LRPC_CCALL::SendReceive+0x76
RPCRT4!NdrpClientCall3+0x63c
RPCRT4!NdrClientCall3+0xed
sechost!LsaLookupOpenLocalPolicy+0x49
sechost!LookupAccountSidInternal+0xe2
sechost!LookupAccountSidLocalW+0x25
advapi32!GetAccountName+0x9a
advapi32!CreateRpcBinding+0xb1
advapi32!OpenRemoteExtObjectLibrary+0x17
advapi32!OpenExtObjectLibrary+0x970
advapi32!QueryV1Provider+0xc7
advapi32!QueryExtensibleData+0x427
advapi32!PerfRegQueryValueEx+0xbd3
advapi32!PerfRegQueryValue+0x3b
KERNELBASE!BaseRegQueryValueInternal+0x4af
KERNELBASE!RegQueryValueExW+0x128
```

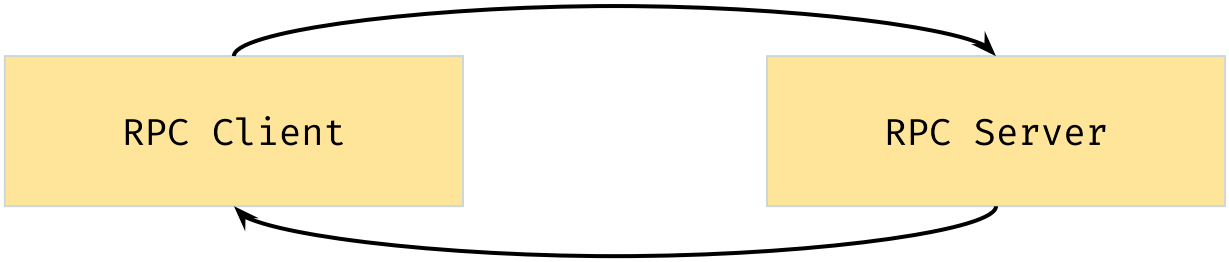
Local RPC Call Stack

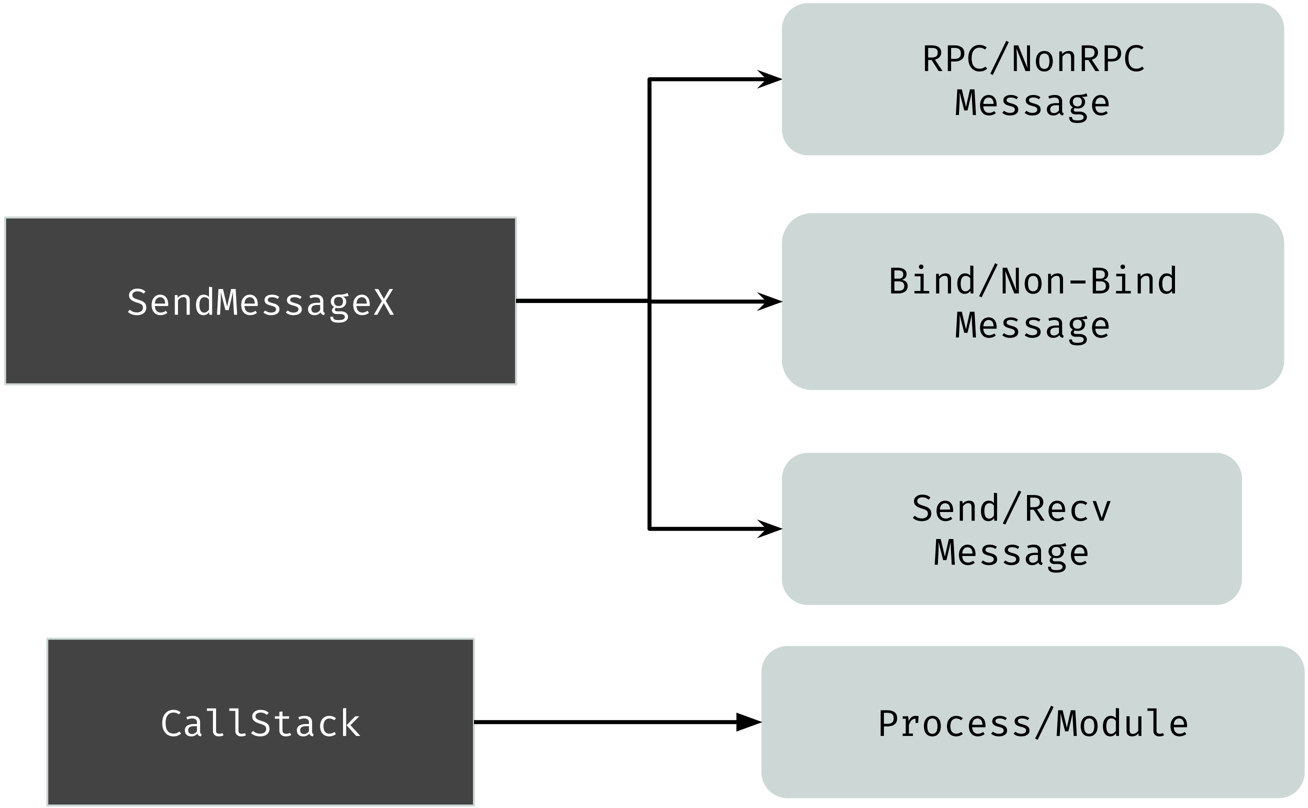
XALPCFuzz - Deploy globally



XALPCFuzz - Filter messages & processes

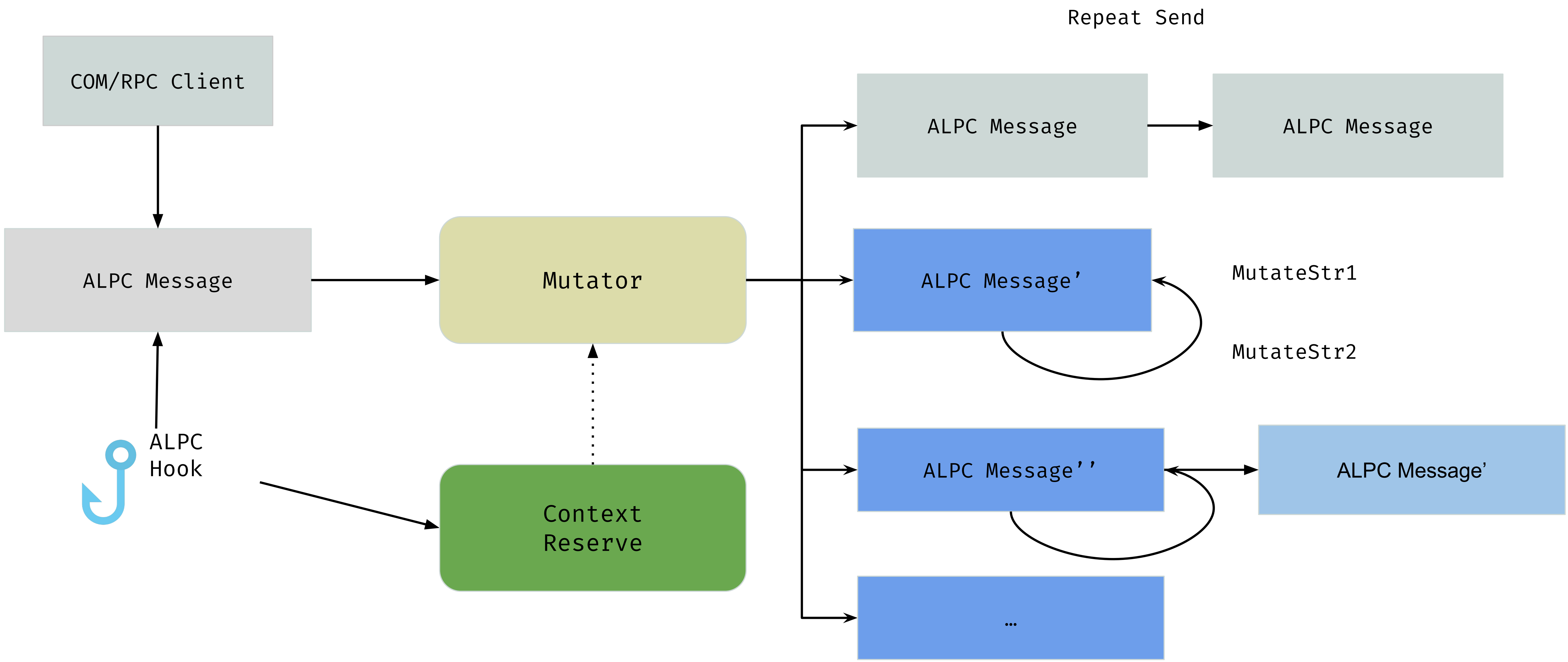
ALPC MessageType is PPORT_MESSAGE.
When the ALPC call contains an RPC message, SendMessageX contains marshalled RPC body.





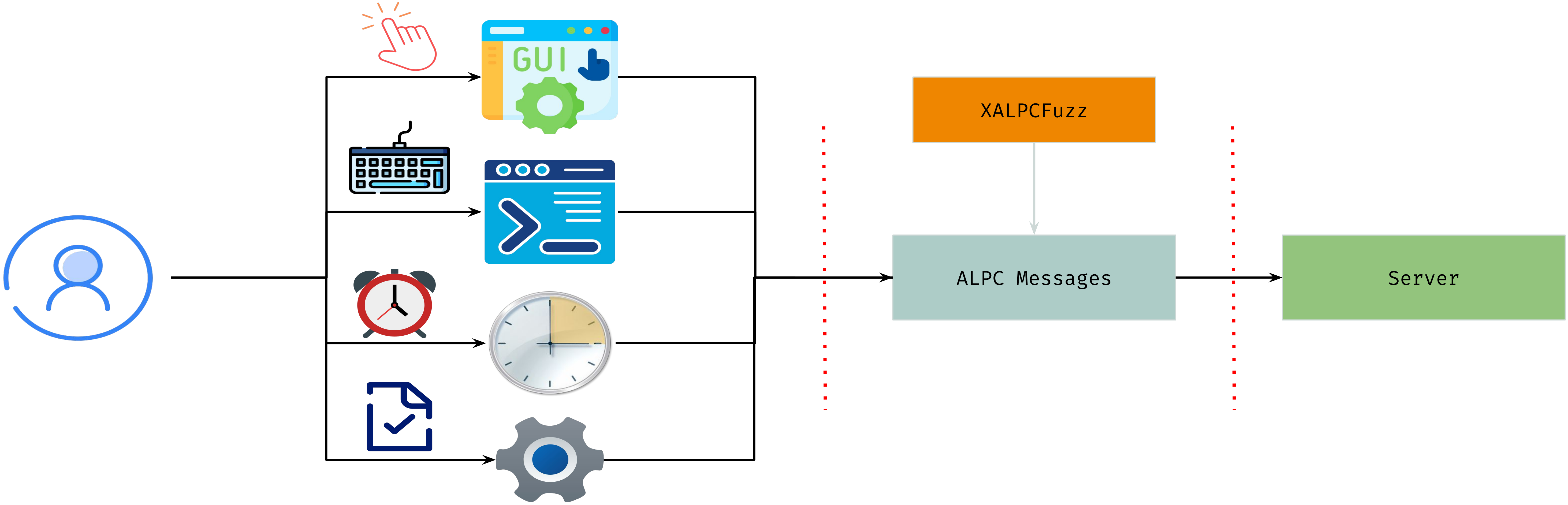
```
NTSTATUS NtAlpcSendWaitReceivePortFilter(
    HANDLE PortHandle,
    ULONG Flags,
    PPORT_MESSAGE SendMessageX,
    PALPC_MESSAGE_ATTRIBUTES SendMessageAttributes,
    PPORT_MESSAGE ReceiveMessage,
    PSIZE_T BufferLength,
    PALPC_MESSAGE_ATTRIBUTES ReceiveMessageAttributes,
    PLARGE_INTEGER Timeout
)
```

XALPCFuzz - Mutation Strategy



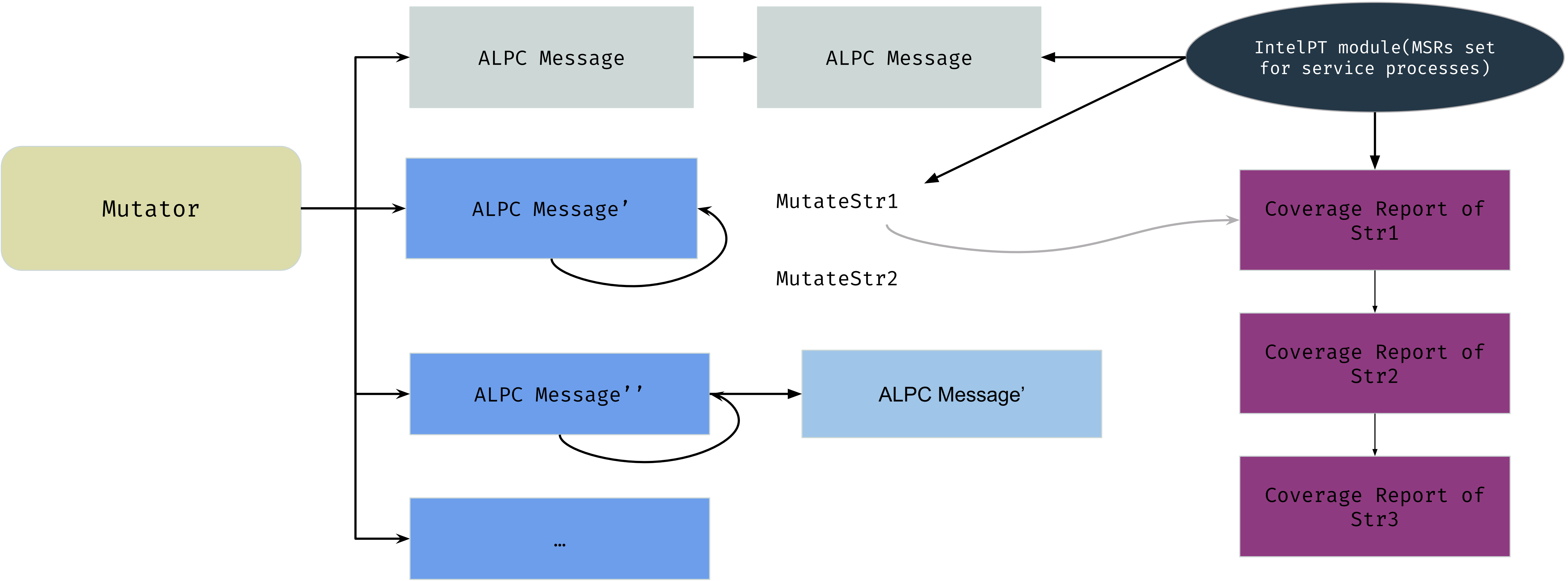
XALPCFuzz - Trigger

Trigger is an important component in this scenario.
We use it to generate more ALPC messages as mutation seeds.



XALPCFuzz - Coverage Monitor

Coverage guided fuzzing has been proved to be efficient in fuzzing.
We use IntelPT as a module to guide selecting the mutation strategy with more coverages come.

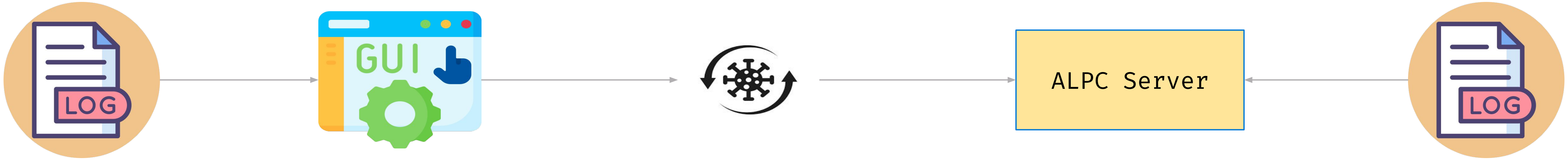


XALPCFuzz - Reproducibility

How to ensure reproducibility when XALPC report a crash?

- The calling stack can be deep.
- The crash may happens far away from the original `NtAlpcSendWaitReceivePortFilter` call.
- The ALPC call invoked by the client may have complex contexts constraints.

How to reproduce specified crashes?
By logging everything during the fuzzing procedure we could!
Sometimes, still need some RCA efforts, to figure out specific COM/RPC invocation details.



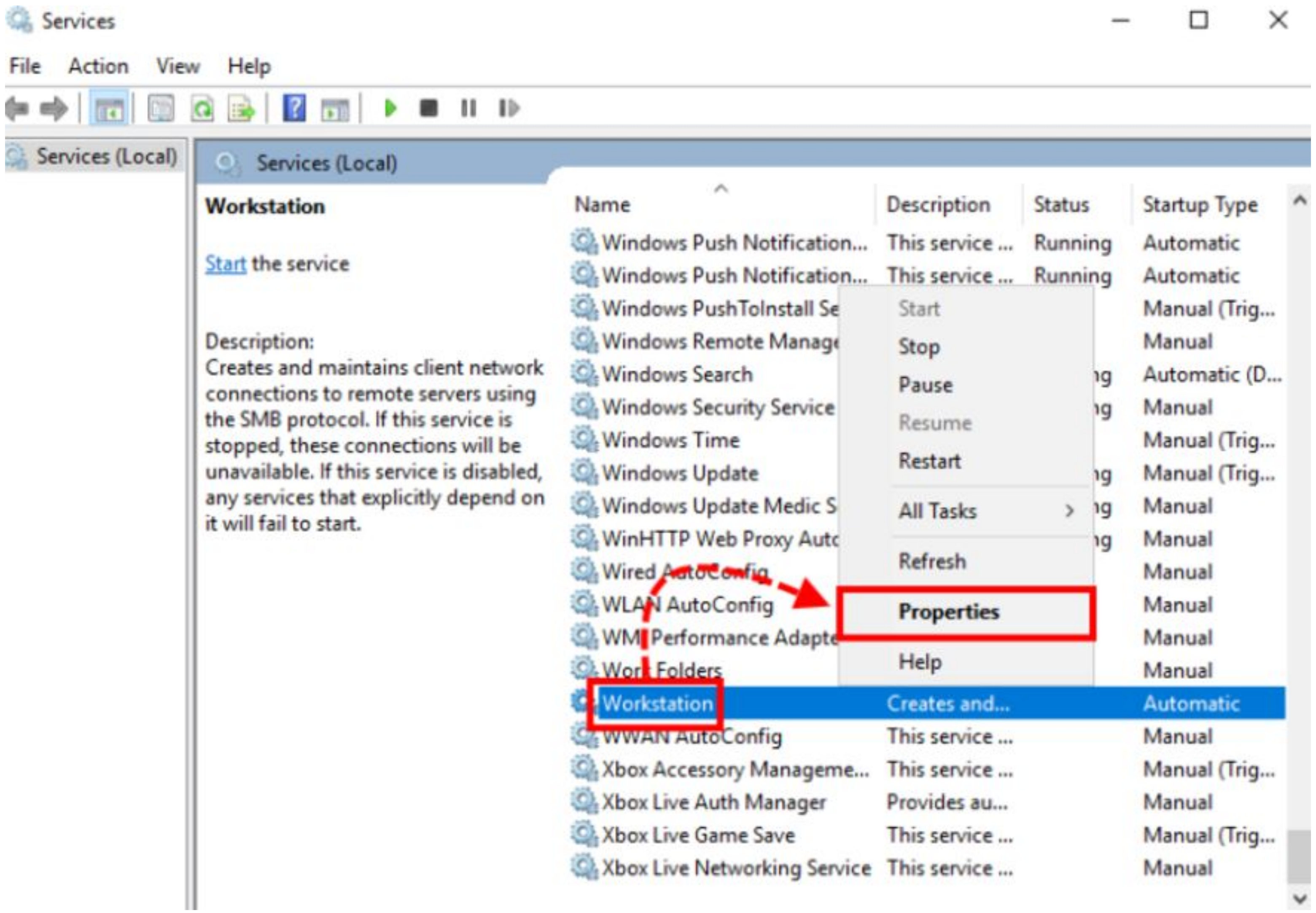
User Clicks, Inputs, schedule tasks, etc.

ALPC mutation Strategies, generated inputs

Call Stacks of the crash, register and memory info

Case Study

CVE-2024-38050 - Windows Workstation Service Elevation of Privilege Vulnerability



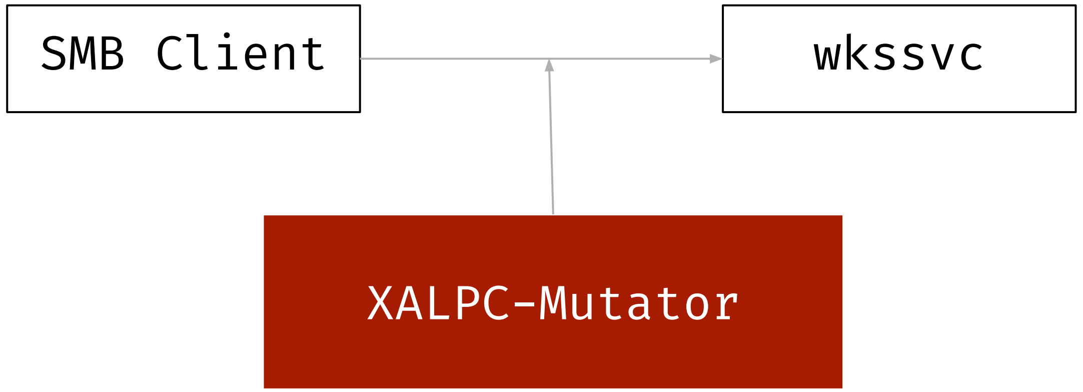
Workstation

Start the service

Description:
Creates and maintains client network connections to remote servers using the SMB protocol. If this service is stopped, these connections will be unavailable. If this service is disabled, any services that explicitly depend on it will fail to start.

Name	Description	Status	Startup Type
Windows Push Notification...	This service ...	Running	Automatic
Windows Push Notification...	This service ...	Running	Automatic
Windows PushToInstall Se			Manual (Trig...
Windows Remote Manage			Manual
Windows Search		ng	Automatic (D...
Windows Security Service		ng	Manual
Windows Time			Manual (Trig...
Windows Update		ng	Manual (Trig...
Windows Update Medic S		ng	Manual
WinHTTP Web Proxy Autc		ng	Manual
Wired AutoConfig			Manual
WLAN AutoConfig			Manual
WMI Performance Adapte			Manual
Work Folders			Manual
Workstation	Creates and...		Automatic
WWAN AutoConfig	This service ...		Manual
Xbox Accessory Manageme...	This service ...		Manual (Trig...
Xbox Live Auth Manager	Provides au...		Manual
Xbox Live Game Save	This service ...		Manual (Trig...
Xbox Live Networking Service	This service ...		Manual

The Workstation Service(wkssvc.dll) is an essential and important RPC Service,funcs including: Configuring properties and behavior of a Server Message Block network redirector (SMB network redirector), managing domain membership and computer names, gathering information.

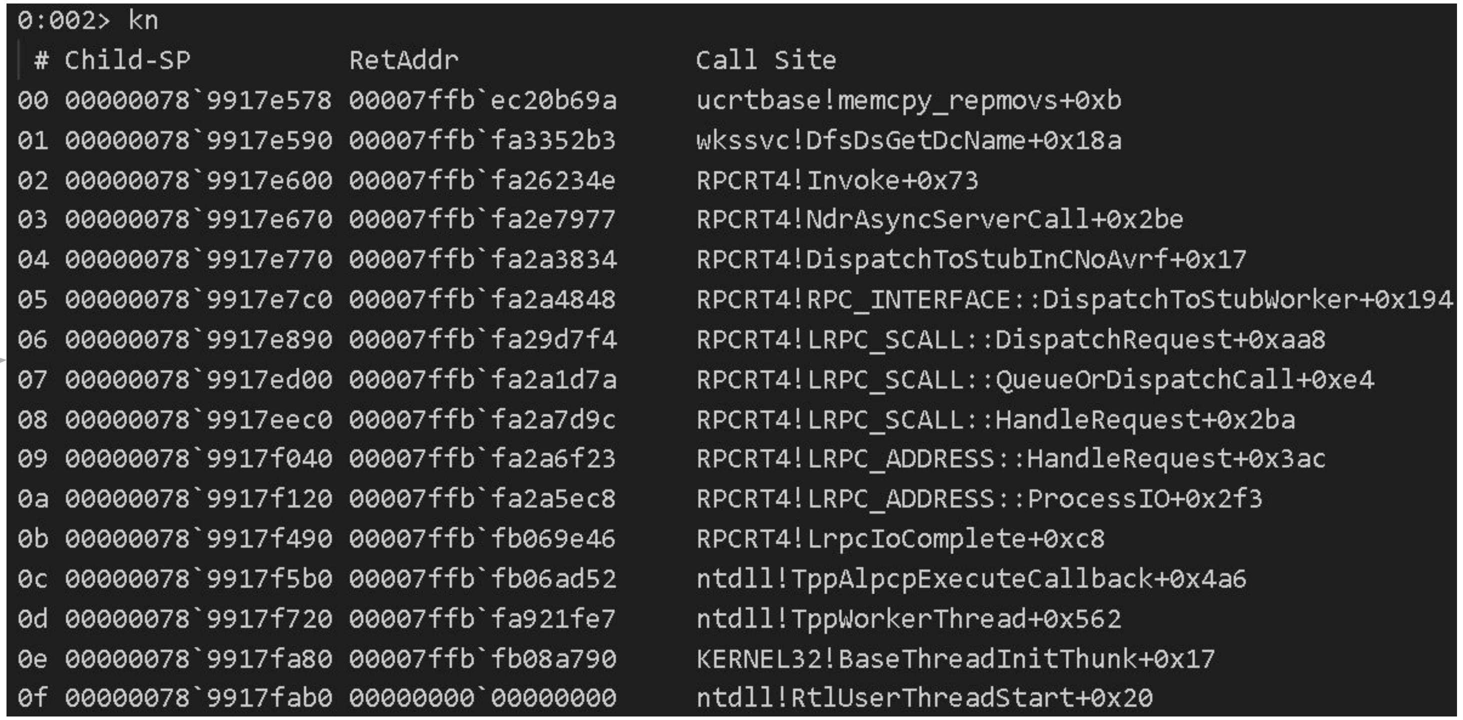


Case Study

CVE-2024-38050 - Windows Workstation Service Elevation of Privilege Vulnerability

```
typedef struct _RPC_Canonical
 {
     unsigned short var1;
     unsigned short var2;
     …
     char* DstBuffer;
} RPC_Canonical,  *PRPC_Canonical;
```

RPC related Structure(Corrupted with XALPC mutation)



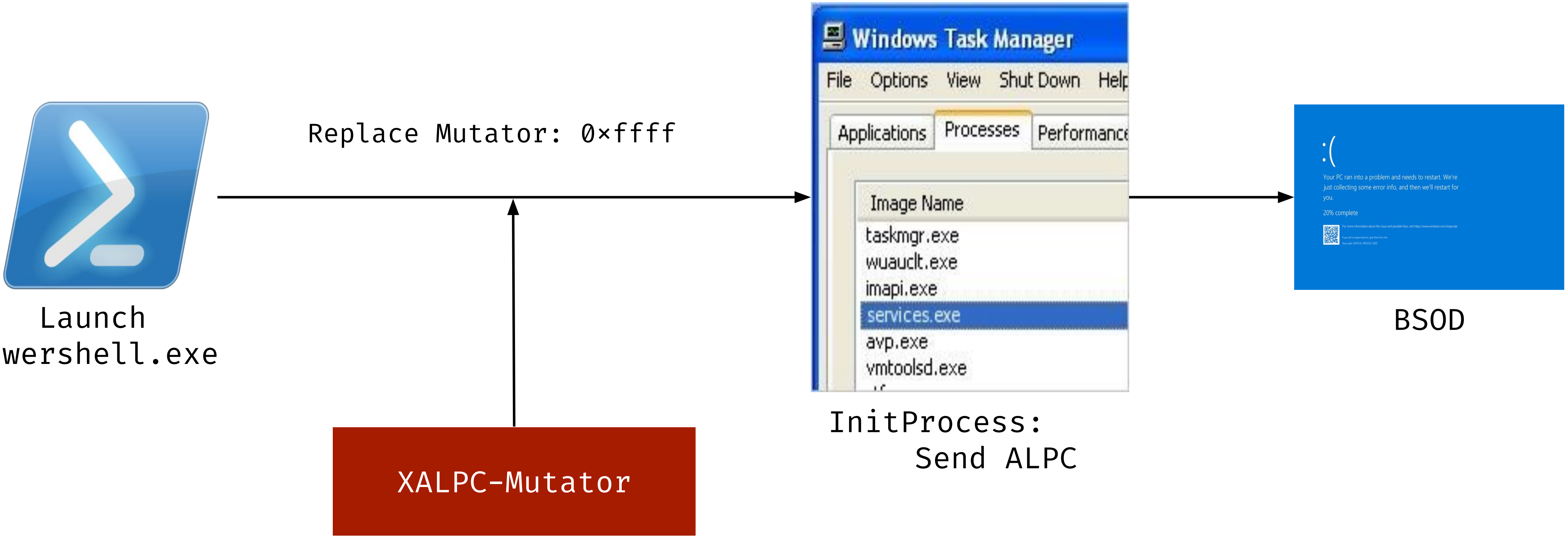
```
0:002> kn
 # Child-SP          RetAddr               Call Site
00 00000078`9917e578 00007ffb`ec20b69a     ucrtbase!memcpy_repmovs+0xb
01 00000078`9917e590 00007ffb`fa3352b3     wkssvc!DfsDsGetDcName+0x18a
02 00000078`9917e600 00007ffb`fa26234e     RPCRT4!Invoke+0x73
03 00000078`9917e670 00007ffb`fa2e7977     RPCRT4!NdrAsyncServerCall+0x2be
04 00000078`9917e770 00007ffb`fa2a3834     RPCRT4!DispatchToStubInCNoAvrf+0x17
05 00000078`9917e7c0 00007ffb`fa2a4848     RPCRT4!RPC_INTERFACE::DispatchToStubWorker+0x194
06 00000078`9917e890 00007ffb`fa29d7f4     RPCRT4!LRPC_SCALL::DispatchRequest+0xaa8
07 00000078`9917ed00 00007ffb`fa2a1d7a     RPCRT4!LRPC_SCALL::QueueOrDispatchCall+0xe4
08 00000078`9917eec0 00007ffb`fa2a7d9c     RPCRT4!LRPC_SCALL::HandleRequest+0x2ba
09 00000078`9917f040 00007ffb`fa2a6f23     RPCRT4!LRPC_ADDRESS::HandleRequest+0x3ac
0a 00000078`9917f120 00007ffb`fa2a5ec8     RPCRT4!LRPC_ADDRESS::ProcessIO+0x2f3
0b 00000078`9917f490 00007ffb`fb069e46     RPCRT4!LrpcIoComplete+0xc8
0c 00000078`9917f5b0 00007ffb`fb06ad52     ntdll!TppAlpcpExecuteCallback+0x4a6
0d 00000078`9917f720 00007ffb`fa921fe7     ntdll!TppWorkerThread+0x562
0e 00000078`9917fa80 00007ffb`fb08a790     KERNEL32!BaseThreadInitThunk+0x17
0f 00000078`9917fab0 00000000`00000000     ntdll!RtlUserThreadStart+0x20
```

Crash Call Stack of wkssvc

XALPC-Mutator

Case Study

Crash in Services.exe triggered in the initialization procedure of powershell.exe which causes a BSOD in the end.



Enhancing the Effectiveness of XALPC Fuzzer

Further Enhancing XALPC Testing Capabilities need to improve：

- In Scenarios with Multiple Clients Under Testing, Windows May Freeze or Become Unstable, Likely Due to Critical ALPC Messages Being Corrupted.

- Coverage Cannot Differentiate Between Requests Generated by Our Payloads and Those from Normal Program Execution.

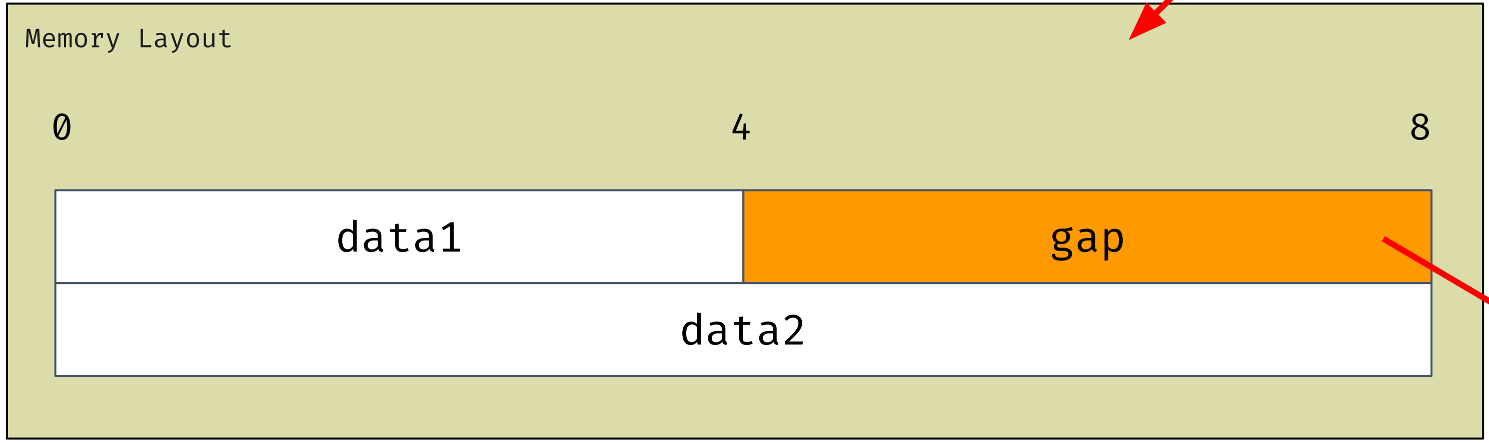
XALPC Monitor

Why Monitor

Typical uninitialized memory

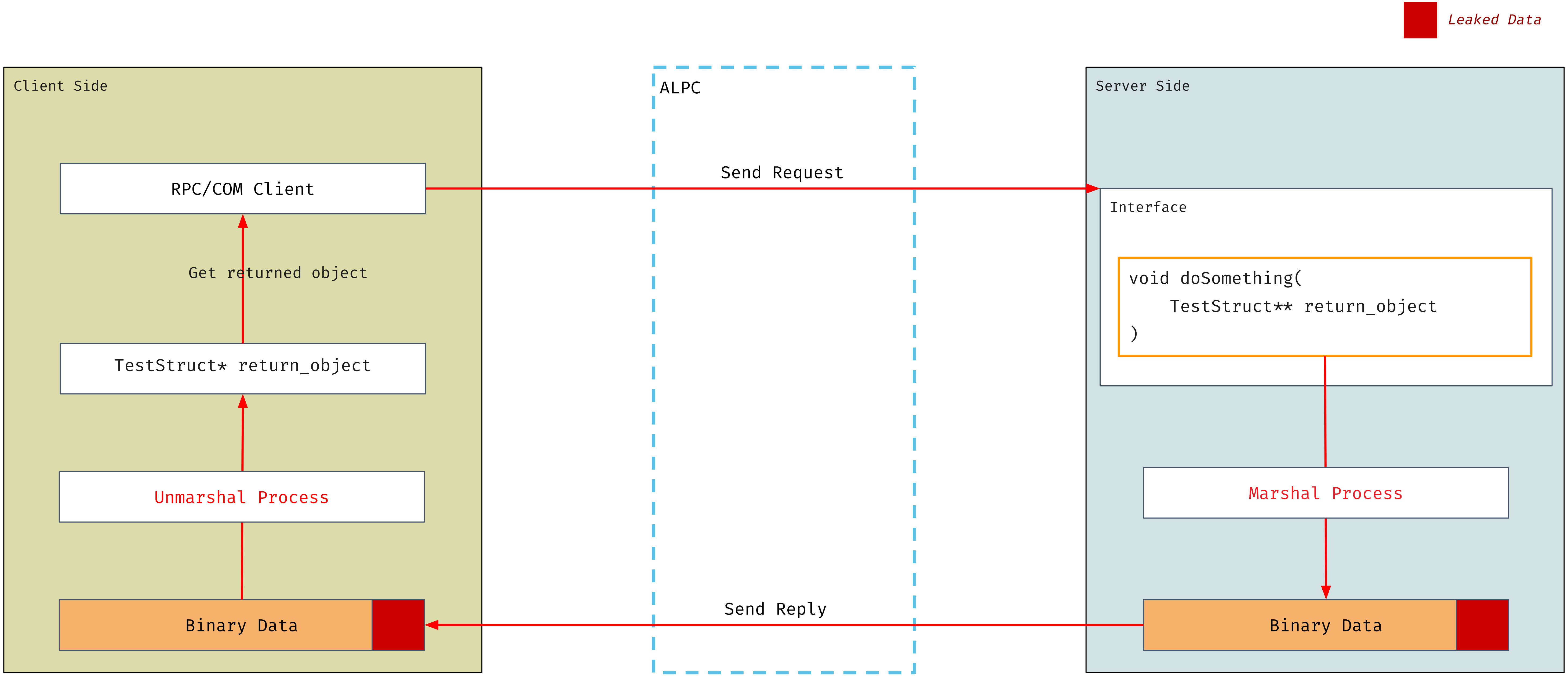
```
void doSomething(TestStruct** return_object){
     TestStruct* object = (TestStruct*)malloc( sizeof(TestStruct) );
     object->data1 = 0;
     object->data2 = 0;

     *return_object = object;  // return object to the caller
}
```

```
typedef struct TestStruct{
     DWORD data1;
     ULONG64 data2;
};
```



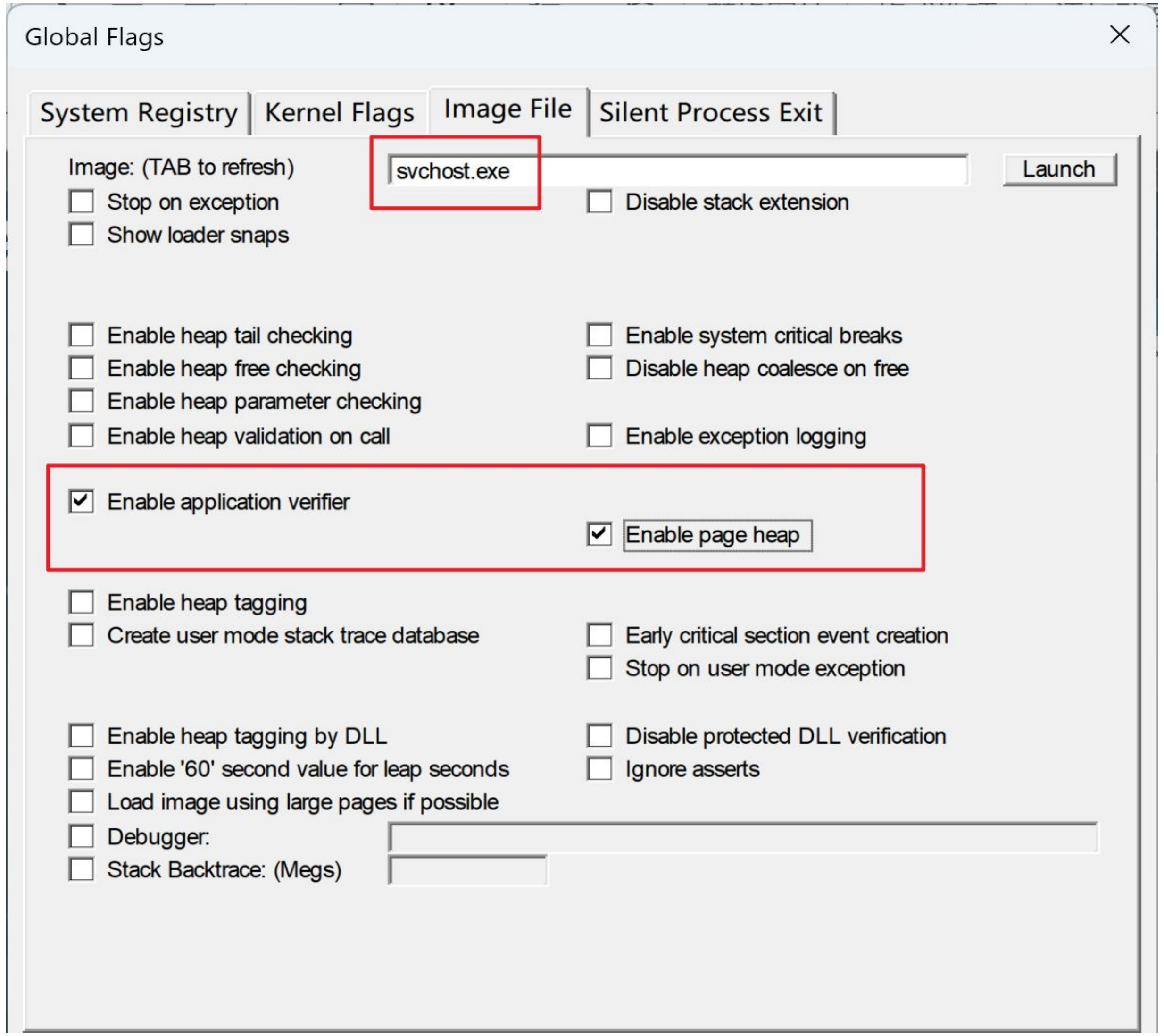
Uninitialized content due to alignment

Leak Uninitialized Data



```
void doSomething(
    TestStruct** return_object
)
```

Detect Leaked Memory



Global Flags

Image: (TAB to refresh) svchost.exe

□ Stop on exception
□ Show loader snaps
□ Disable stack extension

□ Enable heap tail checking
□ Enable heap free checking
□ Enable heap parameter checking
□ Enable heap validation on call
□ Enable system critical breaks
□ Disable heap coalesce on free
□ Enable exception logging

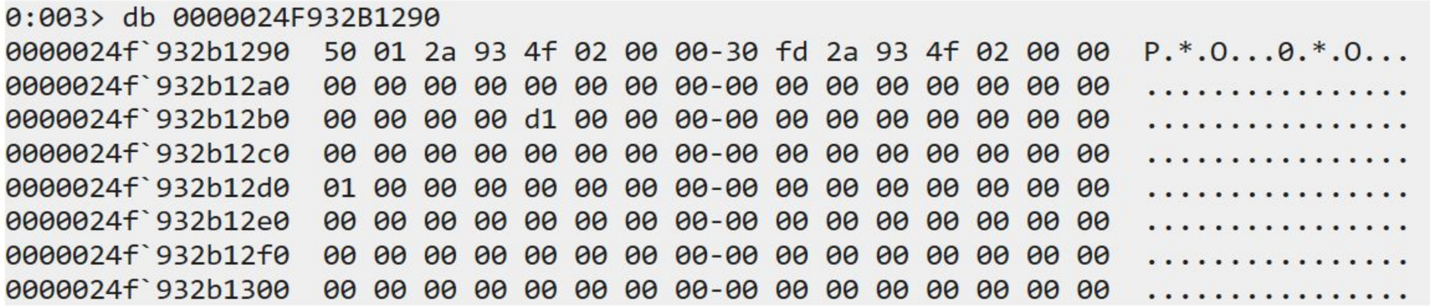
☑ Enable application verifier
☑ Enable page heap

□ Enable heap tagging
□ Create user mode stack trace database
□ Early critical section event creation
□ Stop on user mode exception

□ Enable heap tagging by DLL
□ Enable '60' second value for leap seconds
□ Load image using large pages if possible
□ Debugger:
□ Stack Backtrace: (Megs)
□ Disable protected DLL verification
□ Ignore asserts

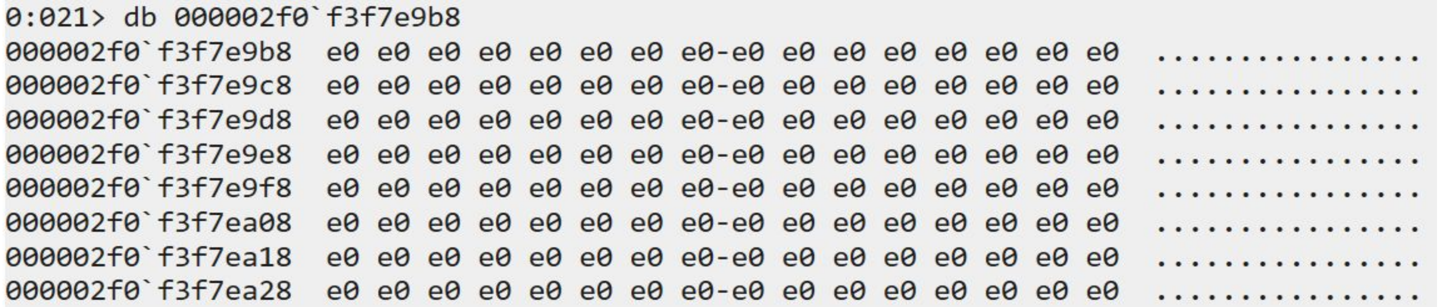
gflags.exe

Without Page Heap (filled with unpredictable value)



```
0:003> db 0000024F932B1290
0000024f`932b1290  50 01 2a 93 4f 02 00 00-30 fd 2a 93 4f 02 00 00  P.*.O...0.*.O...
0000024f`932b12a0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000024f`932b12b0  00 00 00 00 d1 00 00 00-00 00 00 00 00 00 00 00  ................
0000024f`932b12c0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000024f`932b12d0  01 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000024f`932b12e0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000024f`932b12f0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
0000024f`932b1300  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

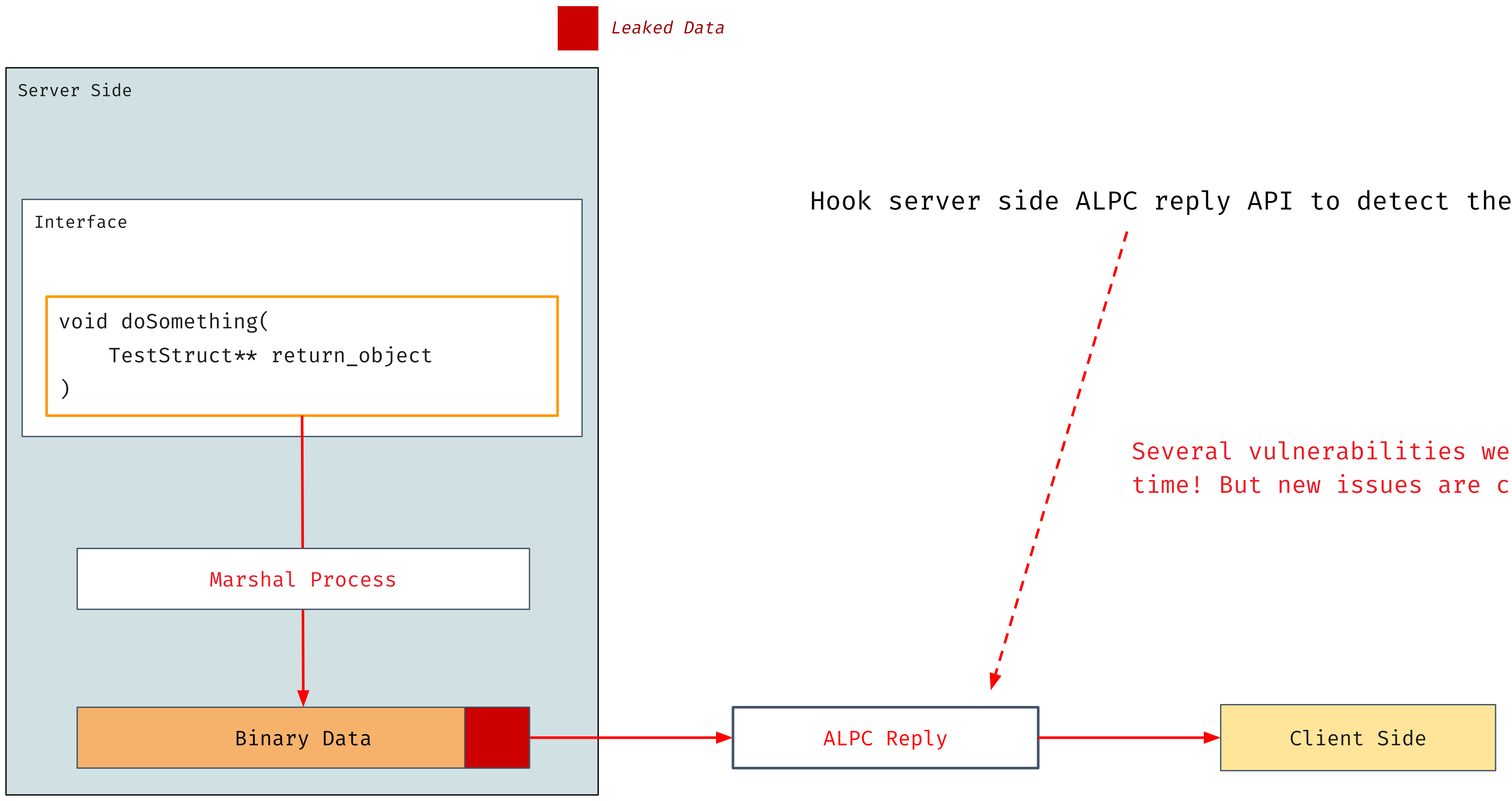
With Page Heap (filled with fixed value)



```
0:021> db 000002f0`f3f7e9b8
000002f0`f3f7e9b8  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7e9c8  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7e9d8  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7e9e8  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7e9f8  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7ea08  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7ea18  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
000002f0`f3f7ea28  e0 e0 e0 e0 e0 e0 e0 e0-e0 e0 e0 e0 e0 e0 e0 e0  ................
```

Treat 0×e0e0 as a signature!

Install Hook in Server Side



```
void doSomething(
    TestStruct** return_object
)
```

Hook server side ALPC reply API to detect the vulnerabilities

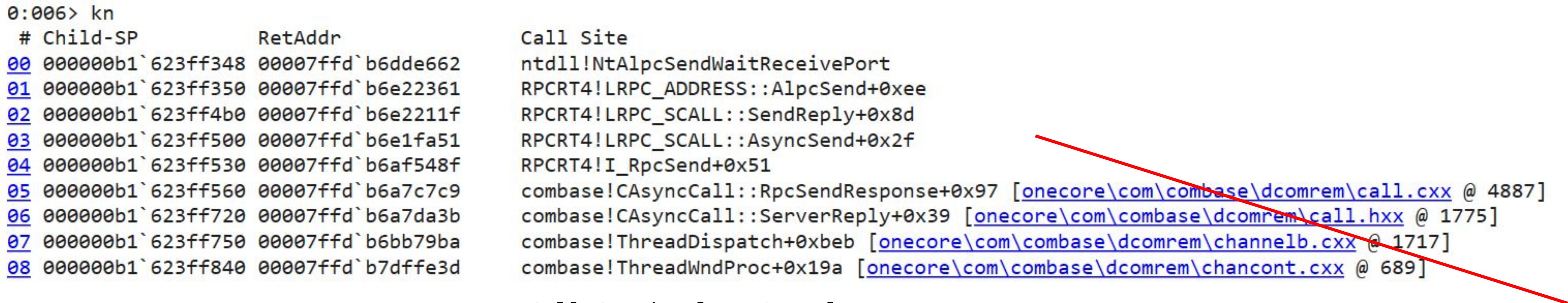
Several vulnerabilities were discovered in a short time! But new issues are coming.

Challenges

Several vulnerabilities were reported, but hard to reproduce

We don't have enough information related to the vulnerable function at ALPC level

- Hard to locate the vulnerable function
  - ALPC call stack is separated from the vulnerable function, we can’t get the function name from the stack



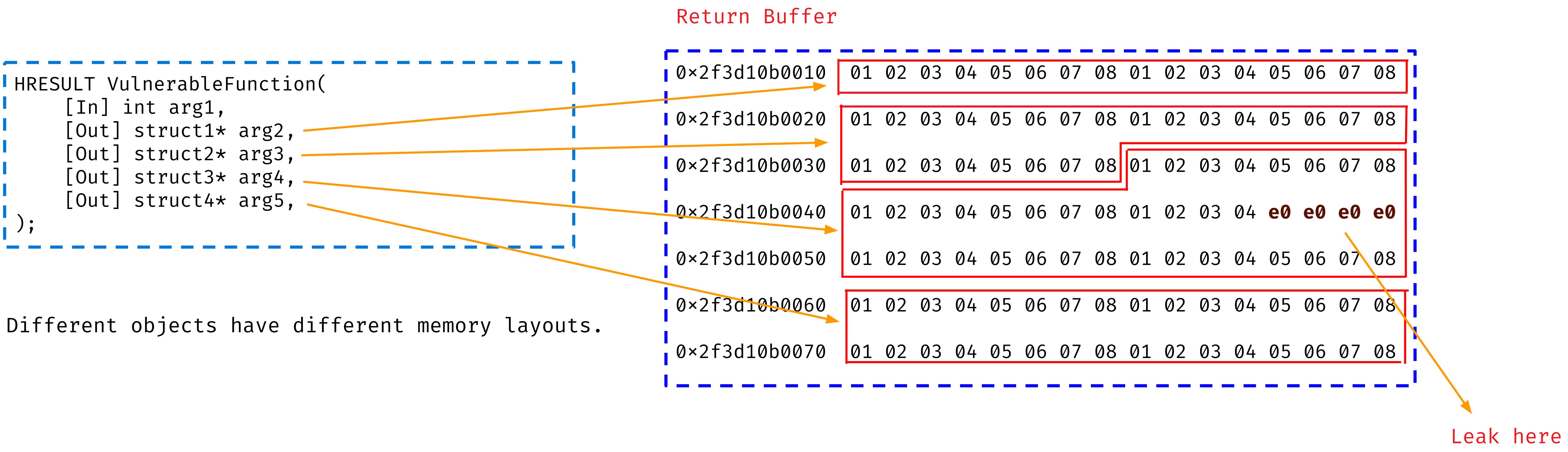
```
0:006> kn
 # Child-SP          RetAddr               Call Site
00 000000b1`623ff348 00007ffd`b6dde662     ntdll!NtAlpcSendWaitReceivePort
01 000000b1`623ff350 00007ffd`b6e22361     RPCRT4!LRPC_ADDRESS::AlpcSend+0xee
02 000000b1`623ff4b0 00007ffd`b6e2211f     RPCRT4!LRPC_SCALL::SendReply+0x8d
03 000000b1`623ff500 00007ffd`b6e1fa51     RPCRT4!LRPC_SCALL::AsyncSend+0x2f
04 000000b1`623ff530 00007ffd`b6af548f     RPCRT4!I_RpcSend+0x51
05 000000b1`623ff560 00007ffd`b6a7c7c9     combase!CAsyncCall::RpcSendResponse+0x97 [onecore\com\combase\dcomrem\call.cxx @ 4887]
06 000000b1`623ff720 00007ffd`b6a7da3b     combase!CAsyncCall::ServerReply+0x39 [onecore\com\combase\dcomrem\call.hxx @ 1775]
07 000000b1`623ff750 00007ffd`b6bb79ba     combase!ThreadDispatch+0xbeb [onecore\com\combase\dcomrem\channelb.cxx @ 1717]
08 000000b1`623ff840 00007ffd`b7dffe3d     combase!ThreadWndProc+0x19a [onecore\com\combase\dcomrem\chancont.cxx @ 689]
```

Call Stack of ALPC Reply

No information about the vulnerable function

Challenges

- Hard to locate the vulnerable parameters for complex interface
  - For interfaces with multiple Out parameters, we don't know which parameter caused the info leak.
  - Much time of Reverse Engineering is required.



Return Buffer

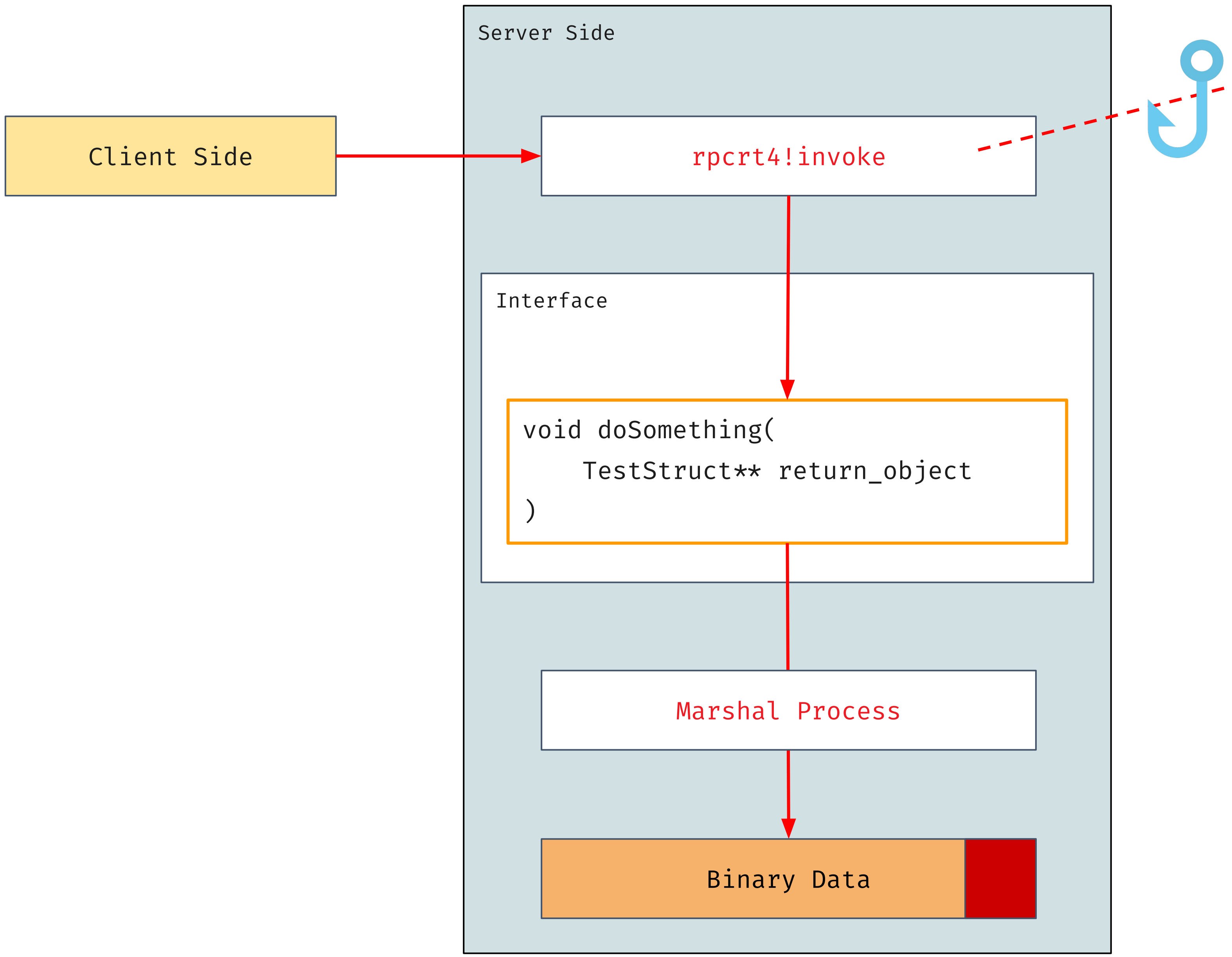
```
HRESULT VulnerableFunction(
      [In] int arg1,
      [Out] struct1* arg2,
      [Out] struct2* arg3,
      [Out] struct3* arg4,
      [Out] struct4* arg5,
);
```

```
0×2f3d10b0010  01 02 03 04 05 06 07 08 01 02 03 04 05 06 07 08
0×2f3d10b0020  01 02 03 04 05 06 07 08 01 02 03 04 05 06 07 08
0×2f3d10b0030  01 02 03 04 05 06 07 08 01 02 03 04 05 06 07 08
0×2f3d10b0040  01 02 03 04 05 06 07 08 01 02 03 04 e0 e0 e0 e0
0×2f3d10b0050  01 02 03 04 05 06 07 08 01 02 03 04 05 06 07 08
0×2f3d10b0060  01 02 03 04 05 06 07 08 01 02 03 04 05 06 07 08
0×2f3d10b0070  01 02 03 04 05 06 07 08 01 02 03 04 05 06 07 08
```

Different objects have different memory layouts.

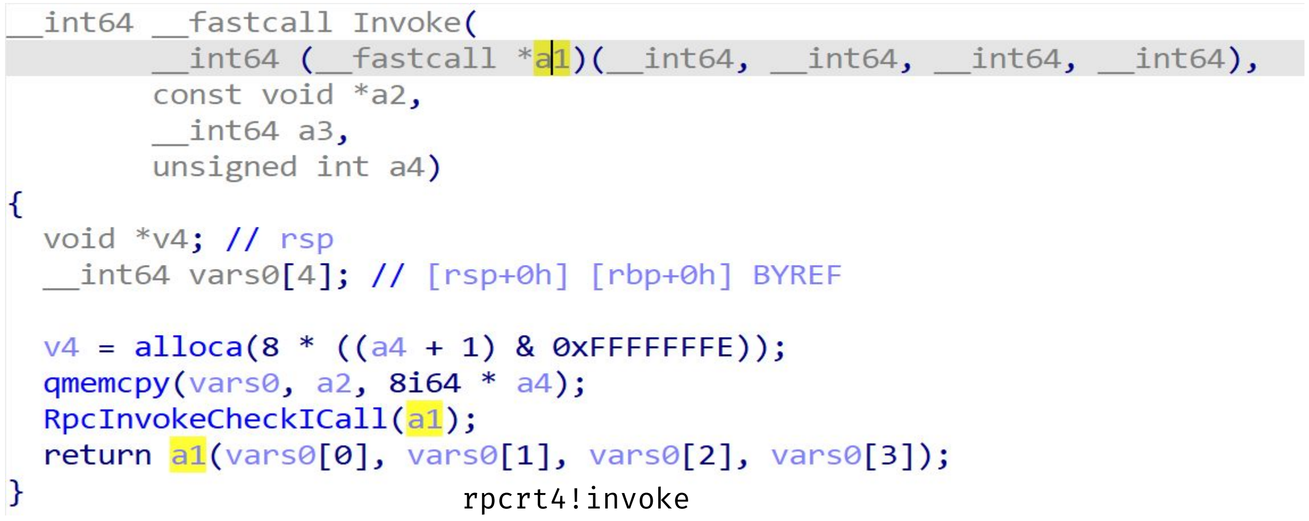
Leak here

Solution



- For challenge 1

  Hook the entry point to get the function address, store the value at global position



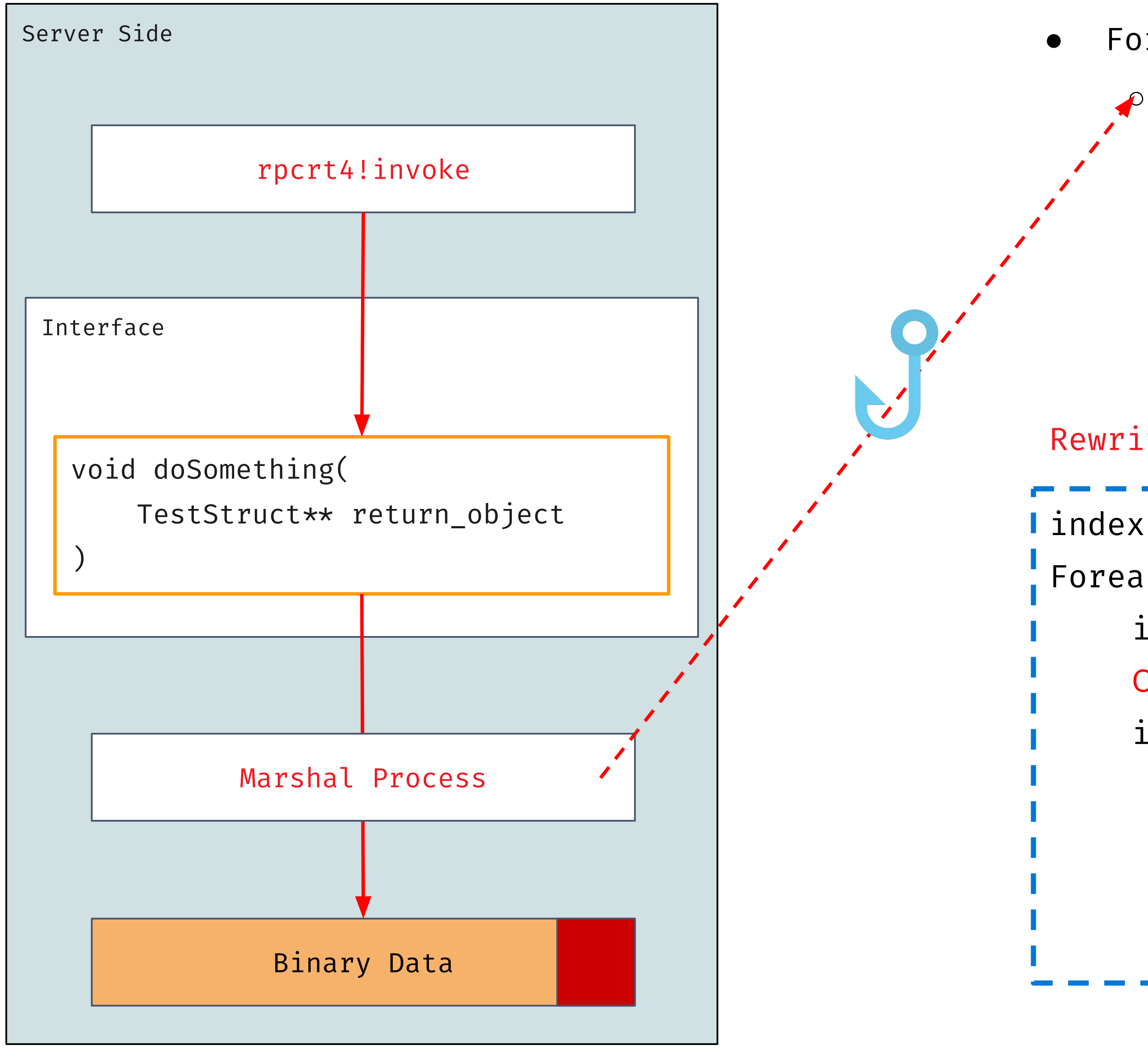
```
__int64 __fastcall Invoke(
        __int64 (__fastcall *a1)(__int64, __int64, __int64, __int64),
        const void *a2,
        __int64 a3,
        unsigned int a4)
{
  void *v4; // rsp
  __int64 vars0[4]; // [rsp+0h] [rbp+0h] BYREF

  v4 = alloca(8 * ((a4 + 1) & 0xFFFFFFFE));
  qmemcpy(vars0, a2, 8i64 * a4);
  RpcInvokeCheckICall(a1);
  return a1(vars0[0], vars0[1], vars0[2], vars0[3]);
}
```

rpcrt4!invoke

Store the function address

```
global_func_addr = a1;
```

Solution



- For challenge 2
  - Hook the Marshal Process
    - Detect the leaked data
    - Identify which parameters cause the info leak
    - Read the function address from the global position

Rewrite Marshal Process

```
index = 0;
Foreach param in out_params:
    index++;
    CallMarshalHandler( param, return_buffer );
    if DetectUninitializedMemory( return_buffer ) == True:
        // bingo, we find the vul
        vul_func_addr = global_func_addr;
        vul_param_index = index
        ReportVul( vul_func_addr, vul_param_index, … );
```

pseudocode

Case Study

CVE-2023-35325 - Windows Print Spooler Information Disclosure Vulnerability

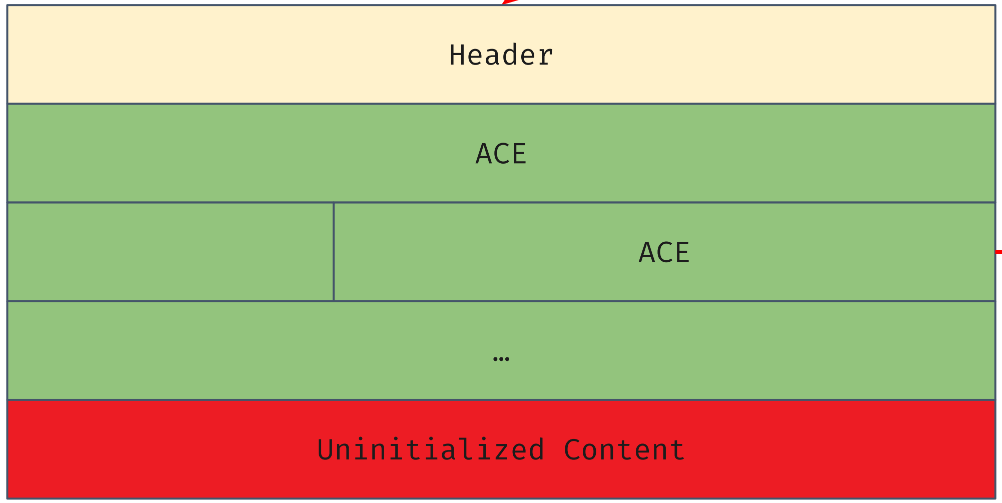
```
BOOL GetPrinter(
  _In_   HANDLE   hPrinter,
  _In_   DWORD    Level,
  _Out_  LPBYTE   pPrinter,
  _In_   DWORD    cbBuf,
  _Out_  LPDWORD  pcbNeeded
);
```

RPC Interface

Forget to empty!

```
acl_buffer = operator new(total_acl_size);

InitializeAcl(acl_buffer, total_acl_size, 2u)

[ … ]

for (int idx=0; idx< ace_count; idx++){
    AddAce(acl_buffer, 2u, 0xFFFFFFFF, ace[idx], ace_size[idx]);
}
```

localspl!DuplicateAclWithPermission



Return back to caller

Case Study

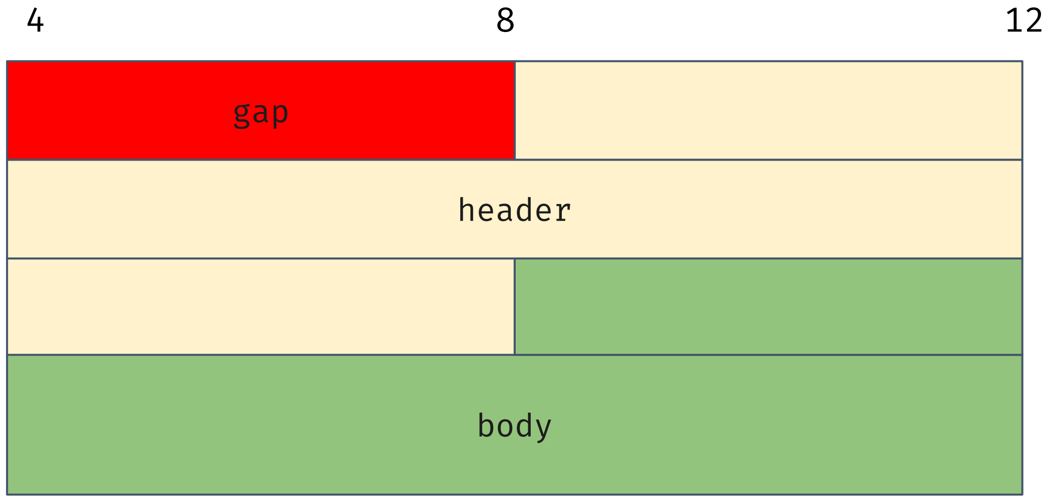
CVE-2023-32042 - OLE Automation Information Disclosure Vulnerability

```
unsigned char * WINAPI BSTR_UserMarshal64(ULONG *pFlags, unsigned char
*buffer, BSTR *pstr)
{

    DWORD len = SysStringByteLen(*pstr);

    ALIGN_POINTER(&buffer, 7); // Align the buffer to 8 bytes

     *(ULONG64*) buffer = (len + 1) >> 1;
     *(DWORD*) (buffer + 8) = len;
     *(DWORD*) (buffer + 12) = (len + 1) >> 1;

    memcpy(buffer + 0x10, *pstr, len+1);

     [ ... ]
}
```

header : 0x10 bytes

body

If buffer is not aligned to 8 bytes:
    Gap Memory will be skipped without initialization!



BSTR is used almost everywhere in COM, but no one noticed this bug!

Abuse Uninitialized Memory Leakage

- Through appropriate heap allocation and deallocation, leak the heap addresses within the RPC/COM service to bypass ASLR.

- For user-mode memory corruption exploitation, Information Leakage is often important.

black hat EUROPE 2024

Summary

XALPC

XALPC Fuzz

Hook-based framework to fuzz Windows RPC/COM messages

Client-side hook

Automatically mutating ALPC messages based on the existing messages to discover vulnerabilities

XALPC Monitor

Hook-based framework to monitor Windows RPC/COM messages

Server-side hook

Monitor and identify leaked memory information in ALPC messages

10+ CVEs found.

Future work

Trigger as many RPC/COM calls as possible:

- Run Windows client applications / play various features provided by Windows as much as possible

- Generate client code for RPC/COM to interact with Windows services, and cover as many functionalities as possible.

Better mutation:

- Mutation based on NDR format

- Coverage guided mutation

black hat
EUROPE 2024

Thanks!