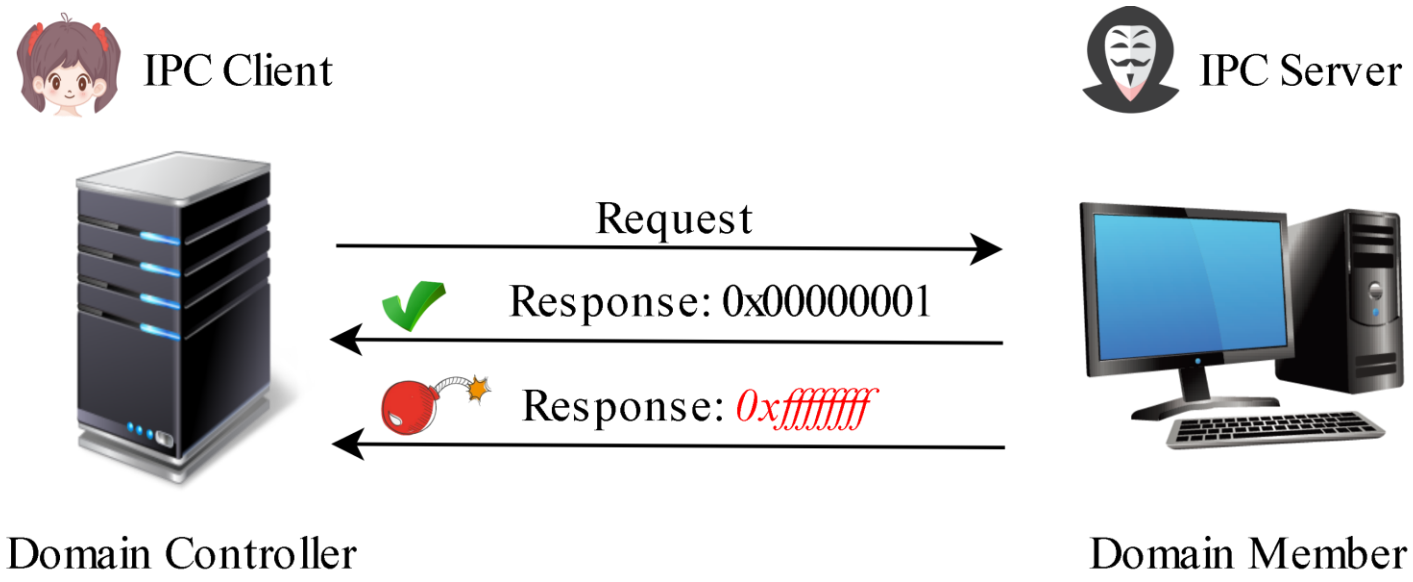
Sheep's Clothing, Wolf's Data: Detecting Server-Induced Client Vulnerabilities in Windows Remote IPC

Fangming Gu[1,2], Qingli Guo[1,2], Jie Lu[3], Qinghe Xie[1,2],
Beibei Zhao[1,2], Kangjie Lu[4], Hong Li[1,2] ,Xiaorui Gong[1,2]

[1]Institute of Information Engineering, Chinese Academy of Sciences
[2]School of Cyber Security, UCAS
[3]Institute of Computing Technology, Chinese Academy of Sciences
[4]University of Minnesota

Why research focus on Windows Remote IPC?



We comprehensively reveal the security threats of remote IPC clients, which have been overlooked in Windows.

Windows IPC – Communication Infrastructure

Windows IPC(Interprocess communications) contains a set of communication mechanisms which supports:

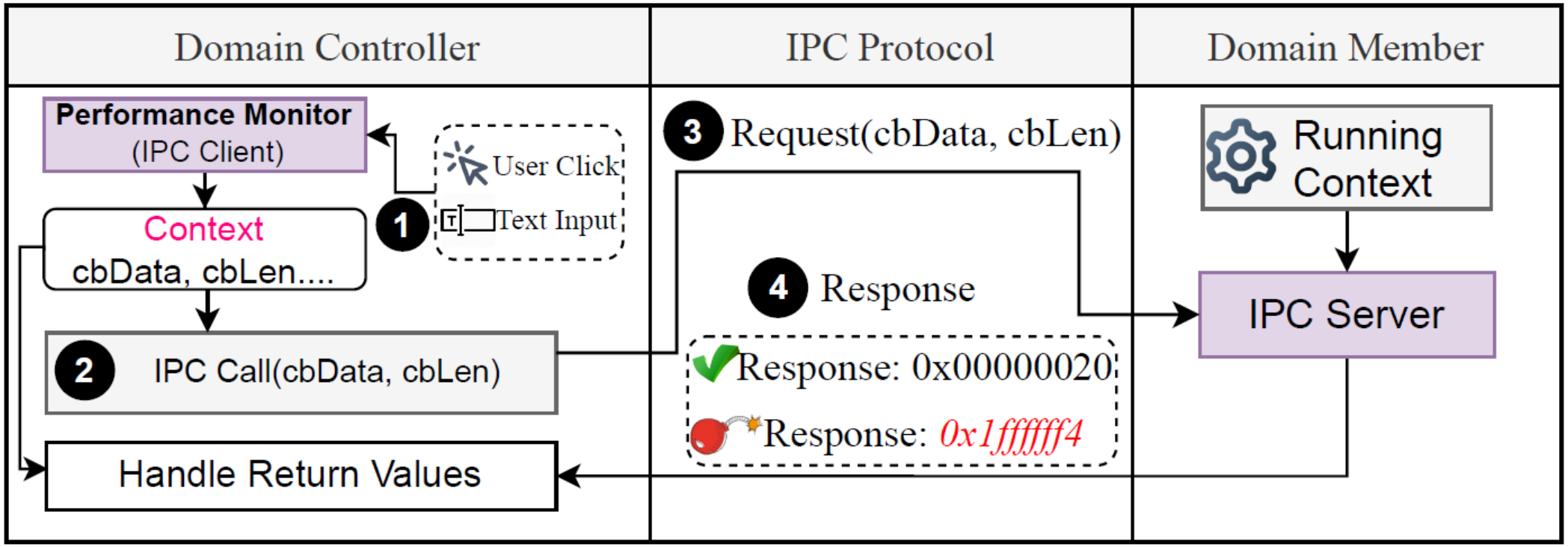
- Local and Remote Machine communication(Server&Client)
- Privilege Separation (Security boundaries)
- Flexibility of IPC Mechanisms(9 ways)
- Proxy-Based Remote Communication

Ref: Interprocess communications - Win32 apps | Microsoft Learn

The following IPC mechanisms are supported by Windows:

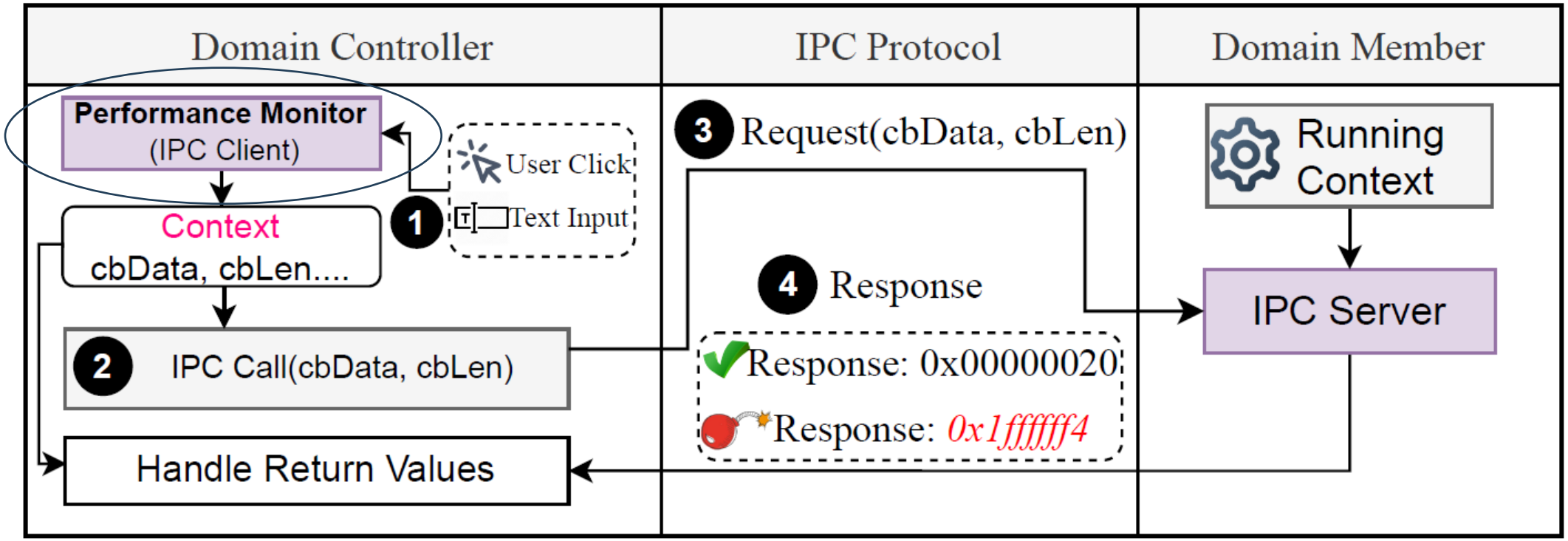
- Clipboard
- COM
- Data Copy
- DDE
- File Mapping
- Mailslots
- Pipes
- RPC
- Windows Sockets

Typical Remote IPC in Windows Domain



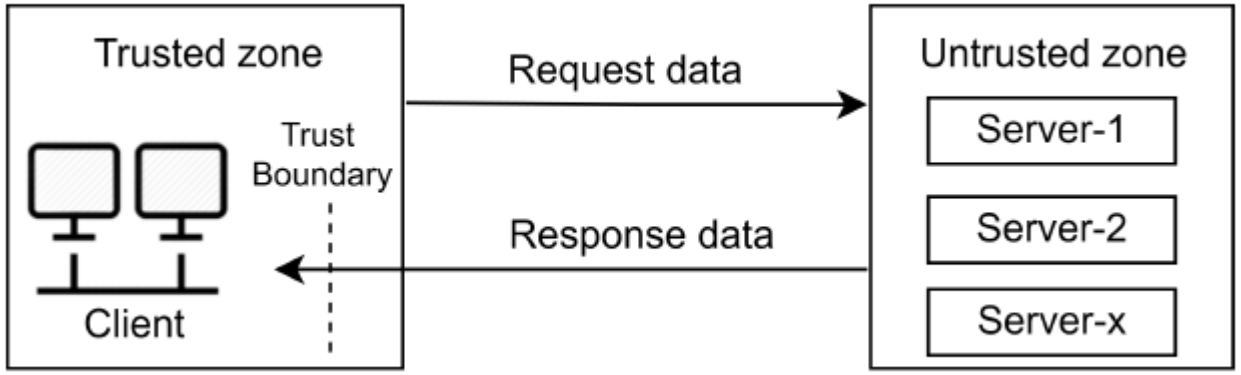
Typical IPC workflow of Performance Monitor in Windows Domain

Typical Remote IPC in Windows Domain



User Click/Input -> Client Generate IPC Context -> Send IPC Request -> Get Response -> Process Return values

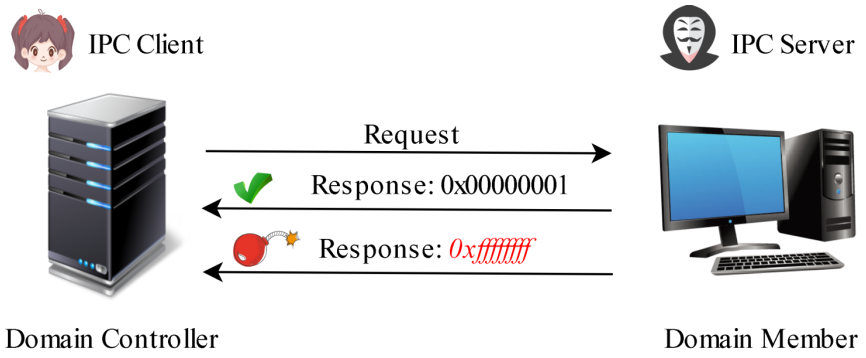
Threat Model in Windows IPC Clients



- In scenarios, we found lots of IPC Clients have higher privilege than the server
- The IPC Server is untrusted and low-privileged, the client is the testing target
- If the IPC Client is vulnerable, it will threaten the security of the server it runs on.

Example Clients:
Performance Monitor,
EventLog Viewer,
RemoteDesktop Client,
NetworkManger, …

Case Study: CVE-2024-38025



CVE-2024-38025: A client vulnerability in Windows Performance Monitor. A malicious Low-privileged(Monitored Machine) triggers a crash on the domain controller machine.

```
1  PERF_MACHINE::BuildNameTable(PERF_MACHINE *this,
   ↪  void *a2){                          Click and user input
2  v54 = RegConnectRegistryW(v53,
   ↪  HKEY_PERFORMANCE_NLSTEXT, &phkResult);
3  a2 = phkResult;
4  ...
5  cbData = 4;
6  if ( RegQueryValueExW((HKEY)a2, L"Last Help",
   ↪  0i64, &Type, (LPBYTE)&Data, &cbData) )
7    goto LABEL_121;
8  if ( RegQueryValueExW((HKEY)a2, L"Last Counter",
   ↪  0i64, &Type, v83, &cbData) )
9    goto LABEL_121;
10 v84 = cbData;
11 if (v84 > 0x40000) {goto FAILURE;}
12 if (v84 + *Data < 0) {goto FAILURE;}
13 if ( !*(_DWORD *)v83 < *Data && *Data < 0) {goto
   ↪  FAILURE;}
14 v89 = 8 * (v84 + 1);
15 v22 = operator new(v89 + 7);
16 // arbitrary memory write                Return Value
17 if ( !RegQueryValueExW(v13, ValueName, 0i64,
   ↪  &Type, (LPBYTE){(v89 + *v83), &cbData) )
18 }
```

Challenges in Vulnerability Discovery

Identification of IPC Clients

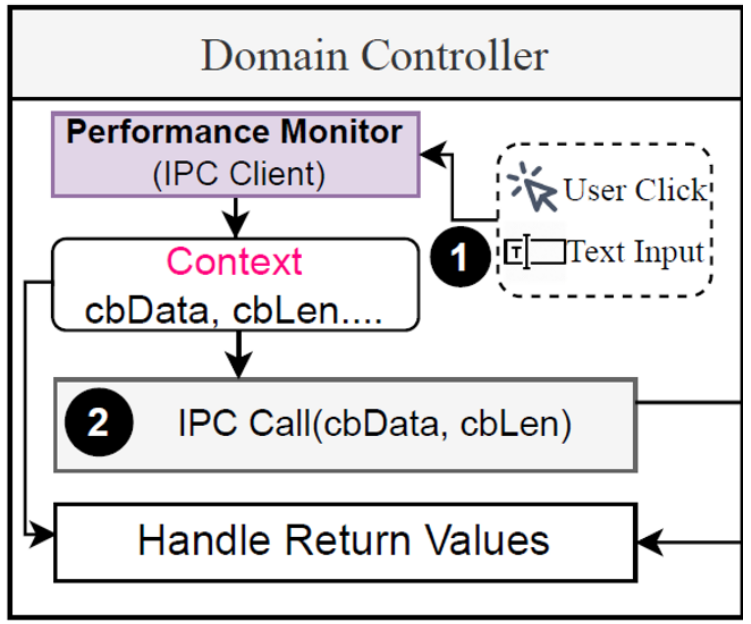
It is difficult to automatically identify IPC clients from large-scale binaries, especially in closed-source systems like Windows. Unlike server-side testing, where publicly available documentation helps identify remote APIs, client-side applications often lack such documentation, making it harder to pinpoint IPC calls.

Triggering IPC Calls

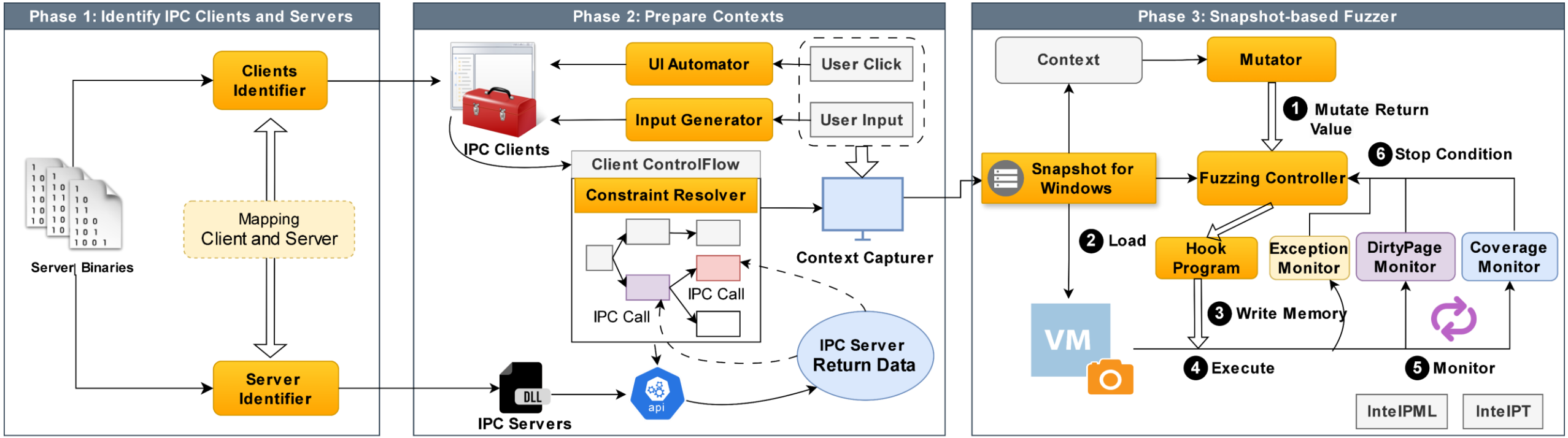
For effective fuzzing, it is essential to trigger as many IPC calls as possible within a client to test a wide range of return values.
Clients involve diverse environments (GUI, CLI, public APIs), and multiple IPC calls are interdependent, adding complexity to triggering the necessary calls for fuzzing.

Testing Efficiency

Context resuming and testing is often time-consuming, require better solution to Improve efficiency.



Overview of GLEIPNIR

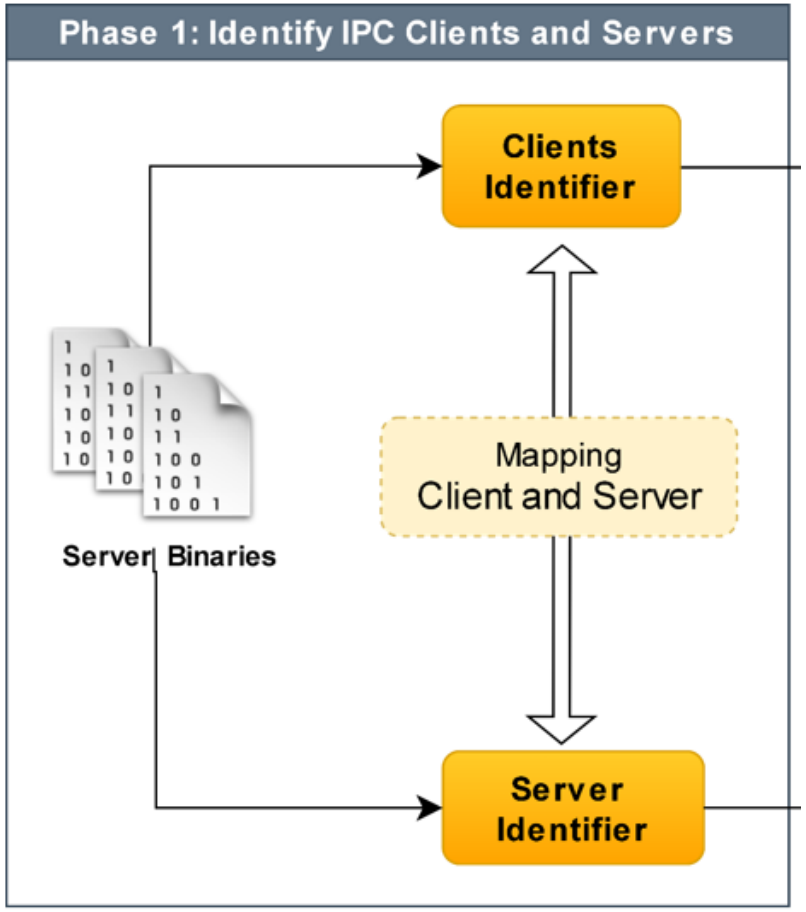


Overview of GLEIPNIR. The orange boxes highlight the functional component of GLEIPNIR, the Client Identifier, Server Identifier, UI Automator, Input Generator, Constraint Resolver, Mutator, Hook, Fuzzing Controller, coverage Monitor, Dirtypage Monitor, and Exception Monitor

Identify IPC Clients and Servers(Phase 1)

Identify IPC Clients and Servers contain 4 Steps:

- Identify IPC Clients
- Identify IPC Servers
- Map Client and Server
- Idenitfy SIDs and MIDs on Client



Identify IPC Clients and Servers(SID&MIDs)

TABLE I
CLIENT-SIDE IPC APIS USED IN RPC, COM AND WINSOCK.

IPC Stage	RPC API	COM API	WinSock API
Initialization	`RpcStringBindingComposeW`, `RpcBindingFromStringBindingW`	`CoCreateInstanceEx`	`WSAStartup`, `connect`
Data Access	`NdrClientCall(V1~V4)`, `NdrAsyncClientCall(V1~V2)`, `Ndr64AsyncClientCall`	`ObjectStublessClient`(Proxy Methods)	`getaddrinfo`, `send`, `recv`
Finalization	`RpcBindingFree`	`NdrCStdStubBuffer_Release`	`closesocket`, `WSACleanup`

Heuristic based algorithm to analyze the IPC client APIs, including call sites, parameters, callers of the APIs, bottom up

Analyze proxy stub and create mappings

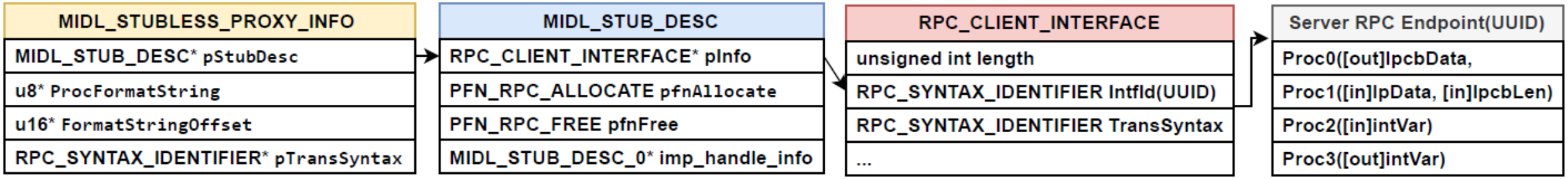
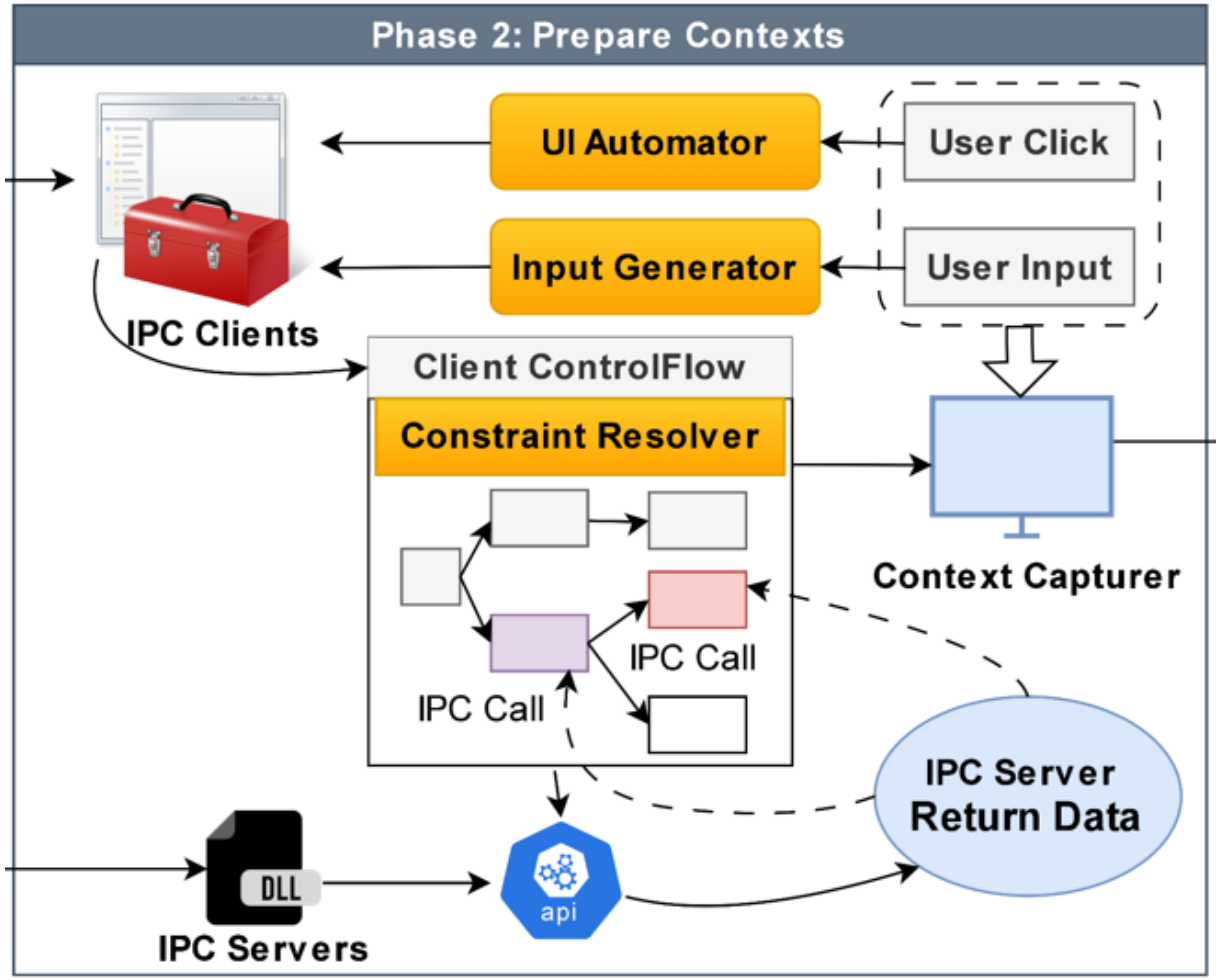


Fig. 4. Parsing Procedure of MIDL_STUBLESS_PROXY_INFO.

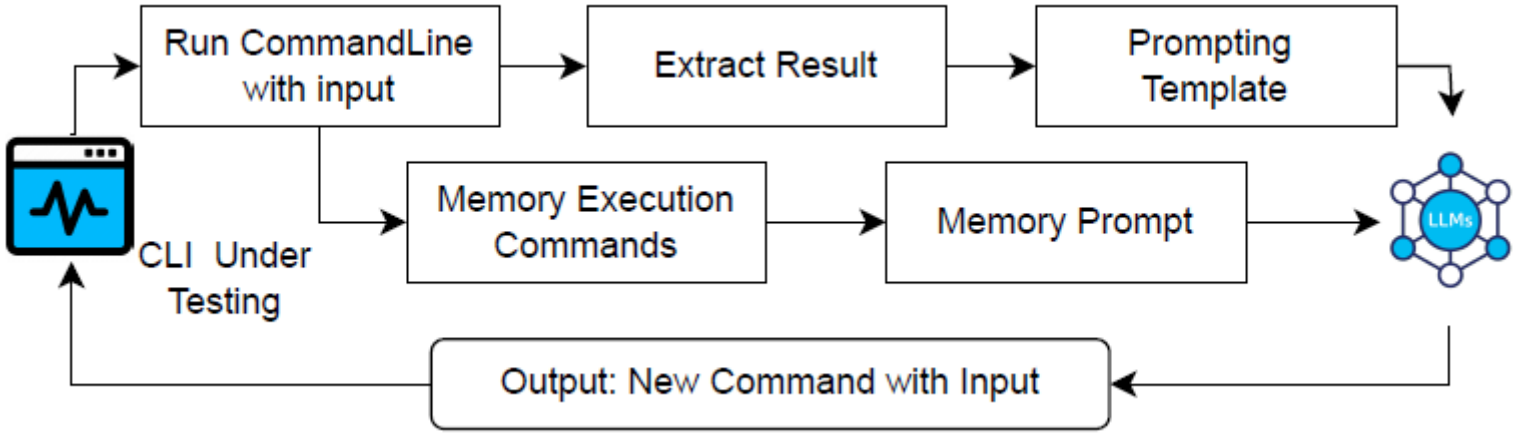
Prepare Contexts(Phase 2)

Prepare Contexts for three types of target Clients:

- UI Applications (User Inputs and Clicking)
- CLI Applications (User Inputs)
- Public APIs



Prepare Contexts(Phase 2)



CLI Apps: Generate Input by designing LLM based workflow

GUI Apps: Generate Input/Click using proposed UI Automator

Prepare Contexts(Phase 2)

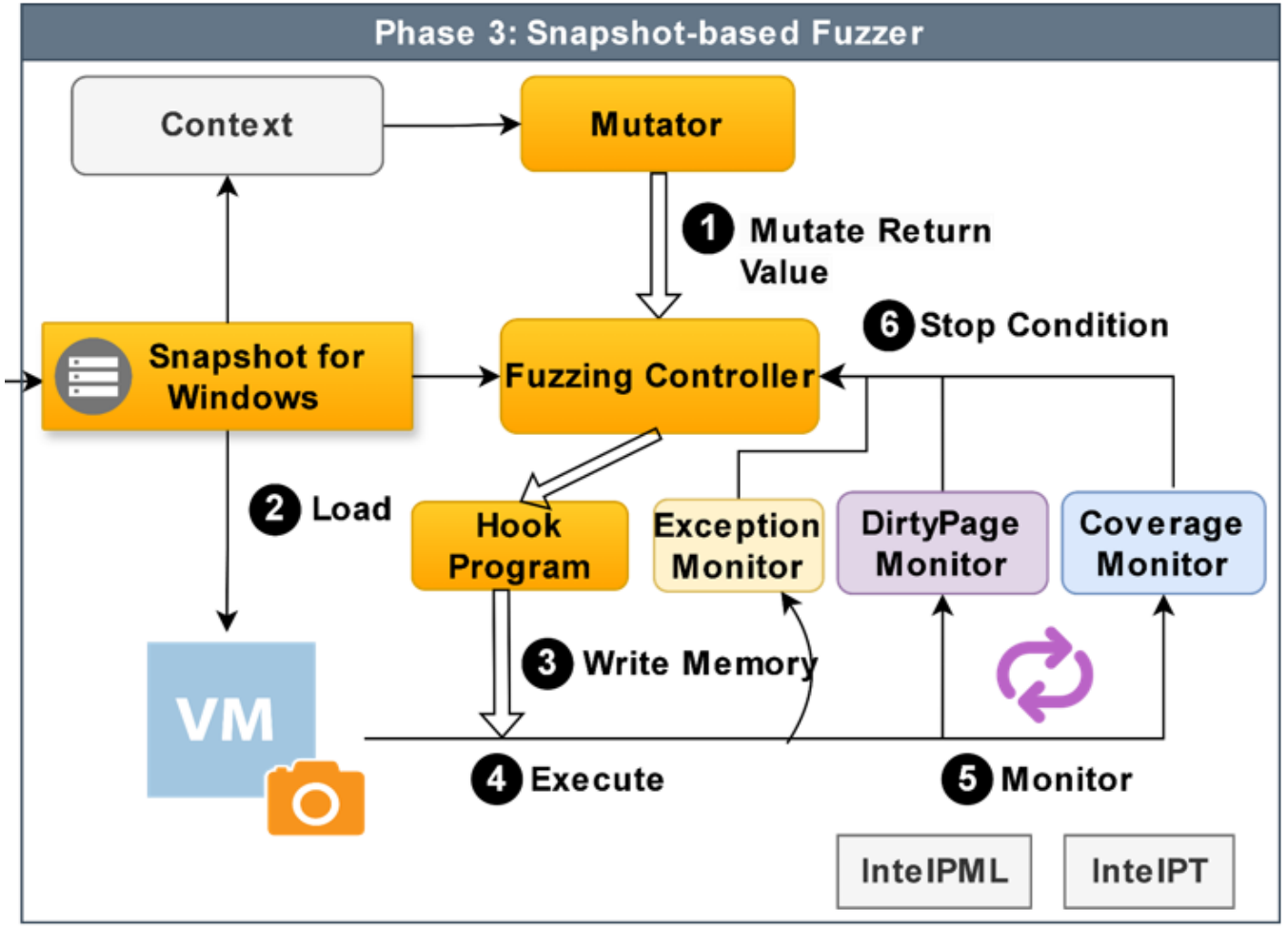
Prompt Type	Template	Instantiation
Interaction 1: CmdLine Arguments Generation		
CmdLine Context	**Start Prompt:** [As a professional security researcher, we want to test the CommandLine based <AppName> App. We need to pass proper arguments to make it functional. Here's the default output of the App: <Hint>]	As a professional security researcher, we want to test the CommandLine based "nfsadmin.exe" App. We need to pass proper arguments to make it functional. Here's the default output of the App: "Invalid option argument. Usage: nfsadmin [server | client | mapping] [\\ host]. For detailed help type nfsadmin [server|client|mapping] /?"
Command Memory	[**List of Tested Commands:** "listgroups, addmembers, ..., <Output>"	[nfsadmin.exe, {Help Text}]
Command Question	What is the command currently being tested? What is the execution result of the command? (<CommandLine>+<result>)	What is the Command currently being tested? What is the execution result of the Command? ("nfsadmin.exe", "Invalid option argument")
Input question	What is the next commandline to execute?	What is the next commandline to execute?
LLM Answer	Current command: "<Command>". Status: <result>. Operation: "<NextInput>"	Current Command: "List Help". Status: Yes. Operation: "nfsadmin mapping /?"
Interaction 2: CmdLine Arguments Generation		
CLI Context	The tested result is : <ReturnValue>	The tested result is: nfsadmin mapping [computer name] [common_options]...
Command Memory	[**List of Tested commands:** "listgroups, addmembers, ..., <Output>"	[nfsadmin.exe, "nfsadmin.exe mapping /?", {Help Text}]
Command Question	What is the command currently being tested? What is the execution result of the command? (<CommandLine>+<result>)	What is the command currently being tested? What is the execution result of the command? ("nfsadmin.exe mapping /?", "Yes")
Input Question	What is the next commandline to execute?	What is the next commandline to execute?
LLM Answer	Current command: "<command>". Status: result. Operation: "<NextInput>"	Current command: "List mapping's help". Status: Yes. Operation: "nfsadmin -u -p mapLookup"

Example of the LLM workflow for testing CLI Apps, more details please refer to the paper.

Prepare Contexts(Phase 3)

Based on the contexts acquired, need a solution to recover the contexts Efficiently each run:

- Snapshot based fuzzing
- Handle network packets
- Choose fuzzing strategy



Prepare Contexts(Phase 3)

- Mutation and Injection: The fuzzing controller retrieves mutated return values from the Mutator and injects them into the memory using a hook program after loading a snapshot.

- Test Execution and Monitoring: It resumes the client process and uses three monitors to collect data and decide whether to continue or stop the test.

- Iteration and Seed Management: Upon test completion, the controller either starts the next mutation or saves and restores seed queue values for further testing.

- Exception and dirtypage Monitoring: These Monitors tracks exceptions and memory changes for manual vulnerability logging and control when to stop the fuzzer.

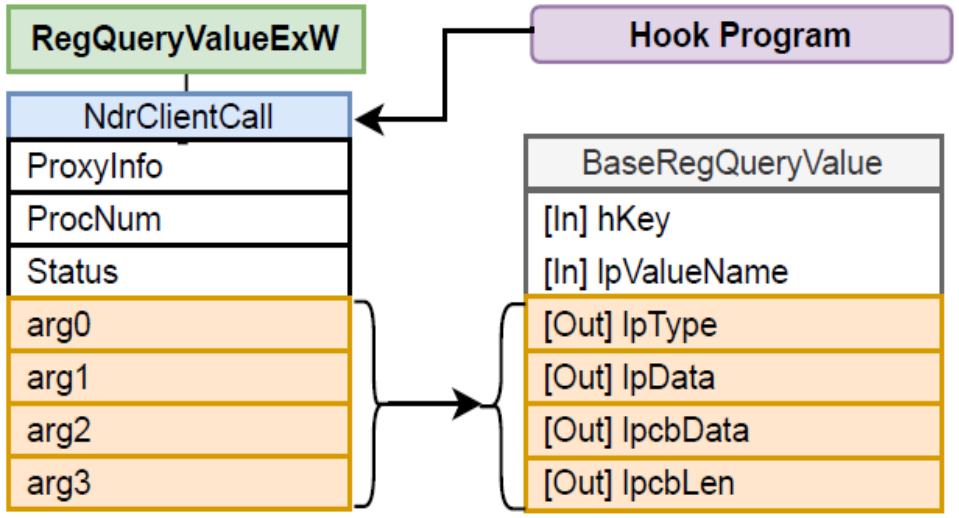


Fig. 9. The hook example for IPC call in Figure 8. `BaseRegQueryValue` is the server method. `NdrClientCall` is the IPC API in Tab I.

Evaluations

- RQ1: How effective is GLEIPNIR in identifying clients and servers?
- RQ2: How effective is GLEIPNIR in preparing the context?
- RQ3: How effective is GLEIPNIR in detecting client vulnerabilities?
- RQ4: How efficient is GLEIPNIR in fuzzing clients?
- RQ5: How does GLEIPNIR compare with other fuzzing approaches?
- RQ6: How effective are the strategies adopted by GLEIPNIR?

Evaluations(1/6)

- **RQ1**: How effective is GLEIPNIR in identifying clients and servers?

NUMBER OF INFERRED IPC CLIENTS

Client	Number	Matched Server	Expected Server	IPC Call			
				RPC	COM	Winsock2	Total
Built-in App	18	73	84	1,182	324	180	1,686
Public API	145	56	64	874	212	62	1,148
Total	163	129	148	2,056	536	242	2,834

GLEIPNIR identifies 145 Public APIs, and 129 servers, missing 19 servers due to legacy clients with removed servers. After manual verification, no false positives were found. The 2,834 IPC calls identified include 72.54% RPC, 18.91% COM, and 8.54% Winsock, with RPC and COM being the primary IPC methods.

Evaluations(2/6)

- **RQ2**: How effective is GLEIPNIR in preparing the context?

GLEIPNIR prepared contexts for 2,834 IPC calls at 537 unique remote methods. It triggered 1,686 IPC calls from BuiltIn applications and 1,148 from Public APIs using user interactions and constraint solving. A total of 2,169 IPC requests were triggered, with constraint solving improving results by 11.28%.

RESULT OF PREPARE FUZZING CONTEXTS. RIC STANDS FOR RECOVERED IPC CALL, TIC STANDS FOR TRIGGERED IPC CALL, AND TW/O STANDS FOR TRIGGERED WITHOUT CONSTRAINTS RESOLVE.

ID	Application	RIC	TIC	Tw/oCS	Events	
					User Click	User Input
01	Performance Monitor	78	72	66	36	1
02	Eventlog Viewer	102	88	82	76	2
03	Device Manager	32	26	24	27	1
04	Windows Server Backup	60	45	42	32	1
05	Windows Disk Management	86	70	65	26	2
06	Service Management	70	59	52	68	1
07	Routing and Remote Access	116	92	86	77	6
08	Task Scheduler	80	64	59	84	12
09	Shared Folders	48	37	37	52	3
10	File Server Resource Manager	73	56	54	49	14
11	DFS Management	56	40	33	39	4
12	Group Policy Management	84	54	50	96	22
13	dfs replication	54	42	41	0	24
14	wmic	92	76	64	0	66
15	nfsadmin	60	44	40	0	32
16	mount	62	45	38	0	28
17	ftp	42	32	29	0	42
18	Windows Admin Center	491	365	300	184	77
	Builtin Total	1,686	1,307	1,162	846	338
	Public APIs	1,148	862	787	0	58
	Total	2,834	2,169	1,949	846	410

Evaluations(3/6)

- **RQ3**: How effective is GLEIPNIR in detecting client vulnerabilities?

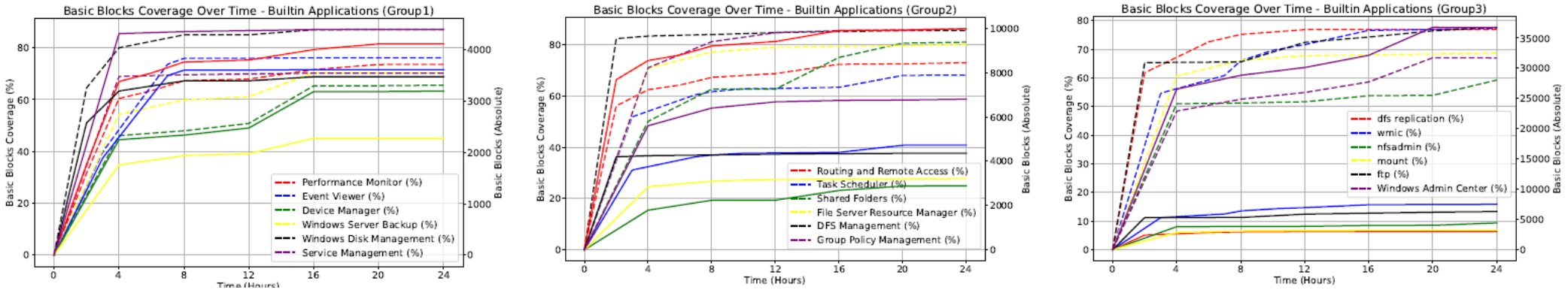
14 vulnerabilities have CVE numbers, and 19 have been confirmed by Microsoft. These vulnerabilities caused memory corruption, with 20 leading to remote code execution and 5 resulting in information leakage.

LIST OF VULNERABILITIES DISCOVERED BY GLEIPNIR (CONFIRMED BY MSRC).

ID	IPC Client Name	Function Name	Windows Version	Status	Security Impact
1	Routing and Remote Access	DeleteProtocolFromRouterConfig	Windows 11 Insider Build 26063.1	CVE-2024-30014	Remote code execution
2	Routing and Remote Access	CDhcpRelayComponent::QueryDataObject	Windows 11 Insider Build 26063.1	CVE-2024-30015	Remote code execution
3	Routing and Remote Access	TFSComponent::Construct	Windows 11 Insider Build 26063.1	CVE-2024-30022	Remote code execution
4	Windows Performance Monitor	GetSystemPerfData	Windows 11 Insider Build 26040.1	CVE-2024-38019	Remote code execution
5	Windows Performance Monitor	PERF_MACHINE::BuildNameTable	Windows 11 Insider Build 26040.1	CVE-2024-38025	Remote code execution
6	Windows Performance Monitor	UpdateMultiCounterV2CounterValue	Windows 11 Insider Build 26040.1	CVE-2024-38028	Remote code execution
7	Windows Performance Monitor	CollectServerQueueObjectData	Windows 11 Insider Build 26040.1	confirmed	Remote code execution
8	Windows Performance Monitor	PerflibV2QueryCounterData	Windows 11 Insider Build 26040.1	confirmed	Remote code execution
9	Windows Eventlog Viewer	Event::SetData	Windows 11 Insider Build 26040.1	confirmed	Information Disclosure
10	Windows Eventlog Viewer	Event::SetDataEx	Windows 11 Insider Build 26040.1	confirmed	Information Disclosure
11	Windows Eventlog Viewer	Event::ProcessData	Windows 11 Insider Build 26040.1	pending	Information Disclosure
12	Windows Admin Center	ApplicationServer	Windows 11 Insider Build 26040.1	confirmed	Remote code execution
13	Windows Admin Center	CategorySample	Windows 11 Insider Build 26040.1	CVE-2024-43475	Information Disclosure
14	Windows Disk Management	ActivationUser	Windows 11 Insider Build 26040.1	pending	Information Disclosure
15	Windows Disk Management	AppExtension	Windows 11 Insider Build 26040.1	pending	Remote code execution
16	Windows Task scheduler	TaskSchedulerProcess	Windows 11 Insider Build 26040.1	pending	Remote code execution
17	Windows DFS Replicator	BundlePackage	Windows 11 Insider Build 26040.1	pending	Remote code execution
18	PublicAPI	MSMQManagement.BytesInQueue	Windows 11 Insider Build 26040.1	CVE-2024-20680	Remote code execution
19	PublicAPI	CollectDiskObjectData	Windows 11 Insider Build 26040.1	pending	Remote code execution
20	PublicAPI	MprAdminPortEnum	Windows 11 Insider Build 26080.1	CVE-2024-38114	Remote code execution
21	PublicAPI	MprAdminConnectionEnum	Windows 11 Insider Build 26080.1	CVE-2024-38115	Remote code execution
22	PublicAPI	MprAdminDeviceEnum	Windows 11 Insider Build 26080.1	CVE-2024-38116	Remote code execution
23	PublicAPI	MprConfigTransportEnum	Windows 11 Insider Build 26080.1	CVE-2024-30023	Remote code execution
24	PublicAPI	MprAdminInterfaceEnum	Windows 11 Insider Build 26080.1	CVE-2024-30024	Remote code execution
25	PublicAPI	MprConfigInterfaceEnum	Windows 11 Insider Build 26080.1	CVE-2024-30029	Remote code execution

Evaluations(4/6)

➢ **RQ4**: How efficient is GLEIPNIR in fuzzing clients?



(a) Performance of fuzzing app-group1 (b) Performance of fuzzing app-group2 (c) Performance of fuzzing app-group3

Fig. 11. Performance of testing built-in applications. The left y-axis represents the percentage of basic block coverage relative to IPC-related code, while the right y-axis displays the absolute values of covered basic blocks. Absolute values are shown with solid lines, and percentages are depicted with dashed lines.

Figure shows the testing efficiency of GLEIPNIR across 18 built-in applications. Within four hours, the test coverage increased rapidly, with four applications stabilizing. After 24 hours, all applications' coverage stabilized, detecting 14 vulnerabilities.

Evaluations(5/6)

- **RQ5**: How does GLEIPNIR compare with other fuzzing approaches?

COMPARISON OF GLEIPNIR SNAPSHOT FUZZING AGAINST WINAFL AND WINNIE. FOR "SPEED" AND "COVERAGE," THE LAST ROW IN THE TABLE INDICATES THE AVERAGE VALUES ACROSS ALL APPLICATIONS. FOR "BUGS/VULNERABILITIES FOUND," THE LAST ROW DENOTES THE TOTAL NUMBER ACROSS ALL APPLICATIONS.

Application	Speed(exec/sec)			Coverage(# of new BBs)			Bug/Vuln Found		
	WinAFL	WINNIE	GLEIPNIR	WinAFL	WINNIE	GLEIPNIR	WinAFL	WINNIE	GLEIPNIR
RemoteQMStartReceive2	1.8	2.2	238	320	365	1,026	1/0	1/1	3/1
QuerySnapshotsByVolume	2.1	2.3	215	336	352	2,072	2/0	2/0	5/2
QMMgmgGetInfo	1.8	2.4	176	291	310	1,872	2/1	2/1	6/2
CollectDiskObjectData	3.5	3.6	252	190	252	2,417	1/1	2/1	4/1
MprAdminPortEnum	3.2	3.1	222	224	265	1,644	3/1	3/1	5/1
MprAdminConnectionEnum	3.5	4.2	275	230	280	1,571	1/0	2/0	3/0
MprAdminDeviceEnum	3.1	3.8	185	298	365	1,820	1/0	1/0	2/0
MprConfigTransportEnum	3.0	3.3	218	318	327	1,440	4/1	2/1	5/1
MprAdminInterfaceEnum	3.5	3.6	289	320	330	1,520	2/0	2/0	3/0
MprConfigInterfaceEnum	3.3	3.9	275	370	389	1,798	3/0	3/0	4/0
Average/Total	2.88	3.24	234.5	289.7	323.5	1,718	20/4	20/5	40/8

Evaluations(6/6)

- **RQ6**: How effective are the strategies adopted by GLEIPNIR?

During GUI testing, replacing our depth-first and IP-based input strategies with random strategies resulted in zero IPC calls and no vulnerability detections. The random click strategy triggered 665 IPC calls and detected 8 vulnerabilities, which is 403 fewer calls and 8 fewer vulnerabilities than the original strategy. For CLI applications, the random input strategy failed to trigger any IPC calls or detect vulnerabilities.

TESTING STRATEGIES AND THEIR RESULTS. CS STANDS FOR CONSTRAINT SOLVING, WHILE PUB REPRESENTS PUBLIC API.

Strategies	TIPC	Vulns	AvgSpeed (exec/sec)
GUI-RandomInput	0	0	-
GUI-RandomClick	665	8	-
GUI-W/O CS	950	13	-
GUI-GLEIPNIR	1068	16	-
CLI-RandomInput	0	0	-
CLI-W/O CS	220	2	-
CLI-GLEIPNIR	239	4	-
PUB-W/O CS	750	32	-
PUB-GLEIPNIR	862	40	
W/O Snapshot	-	11	5.3
Stop-500	-	20	282
Stop-1000	-	25	240
Stop-2000	-	22	186
Stop-3000	-	16	120

Conclusion

- We present GLEIPNIR, focusing on client-side vulnerabilities often overlooked in previous research targeting server-side issues.

- The Solution applies static analyses to identify IPC clients and servers to establish mappings between them, using LLM and UI automation techniques to prepare contexts for further testing.

- In testing 76 client applications over 7 days, GLEIPNIR identified 25 vulnerabilities. The identified vulnerabilities led to 14 CVEs and a total bounty reward of $36,000

Thanks for listening!

Q&A

Contact: gufangming@iie.ac.cn