COMRACE: Detecting Data Race Vulnerabilities in COM Objects

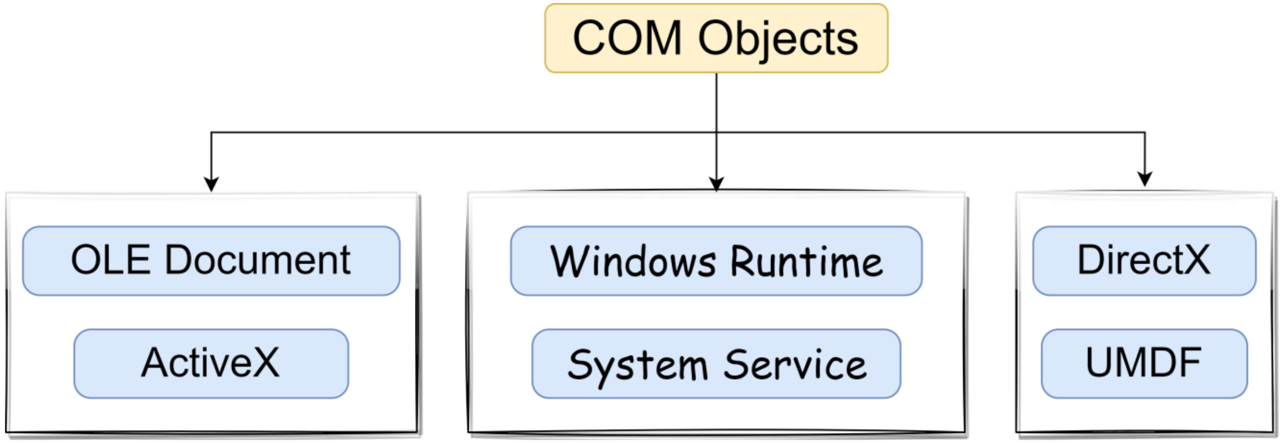
Fangming Gu[1,2], Qingli Guo[1,2], Lian Li[3,4], Zhiniang Peng[5,6], Wei Lin[1,2], Xiaobo Yang[1,2], Xiaorui Gong[1,2]

[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]School of Computer Science and Technology, UCAS
[5]Sangfor Technologies Inc
[6]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences

USENIX Security 2022

Introduction to COM Objects

Component Object Model (COM)

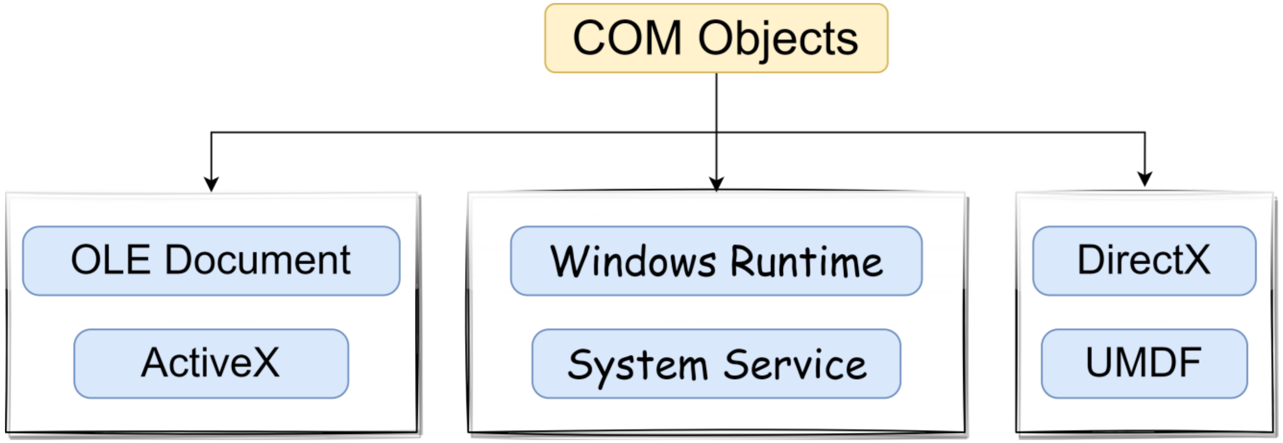


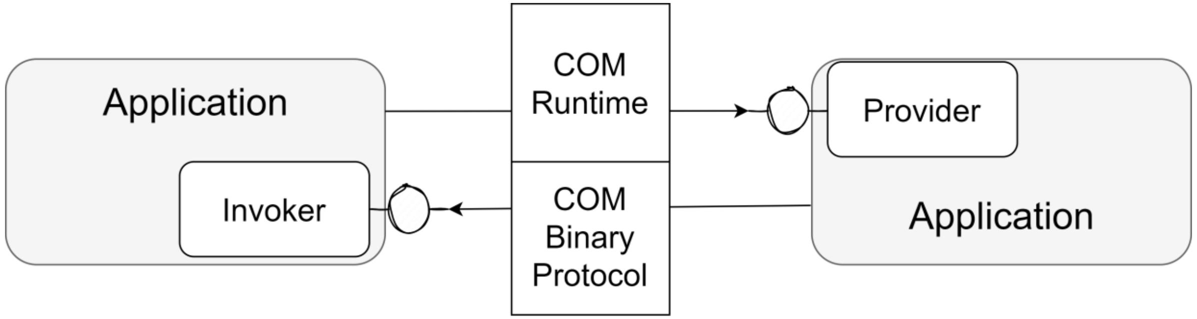
COM is a platform-independent, distributed, object-oriented system for creating binary software components that can interact.

Ref: https://docs.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal

Introduction to COM Objects

Component Object Model (COM)





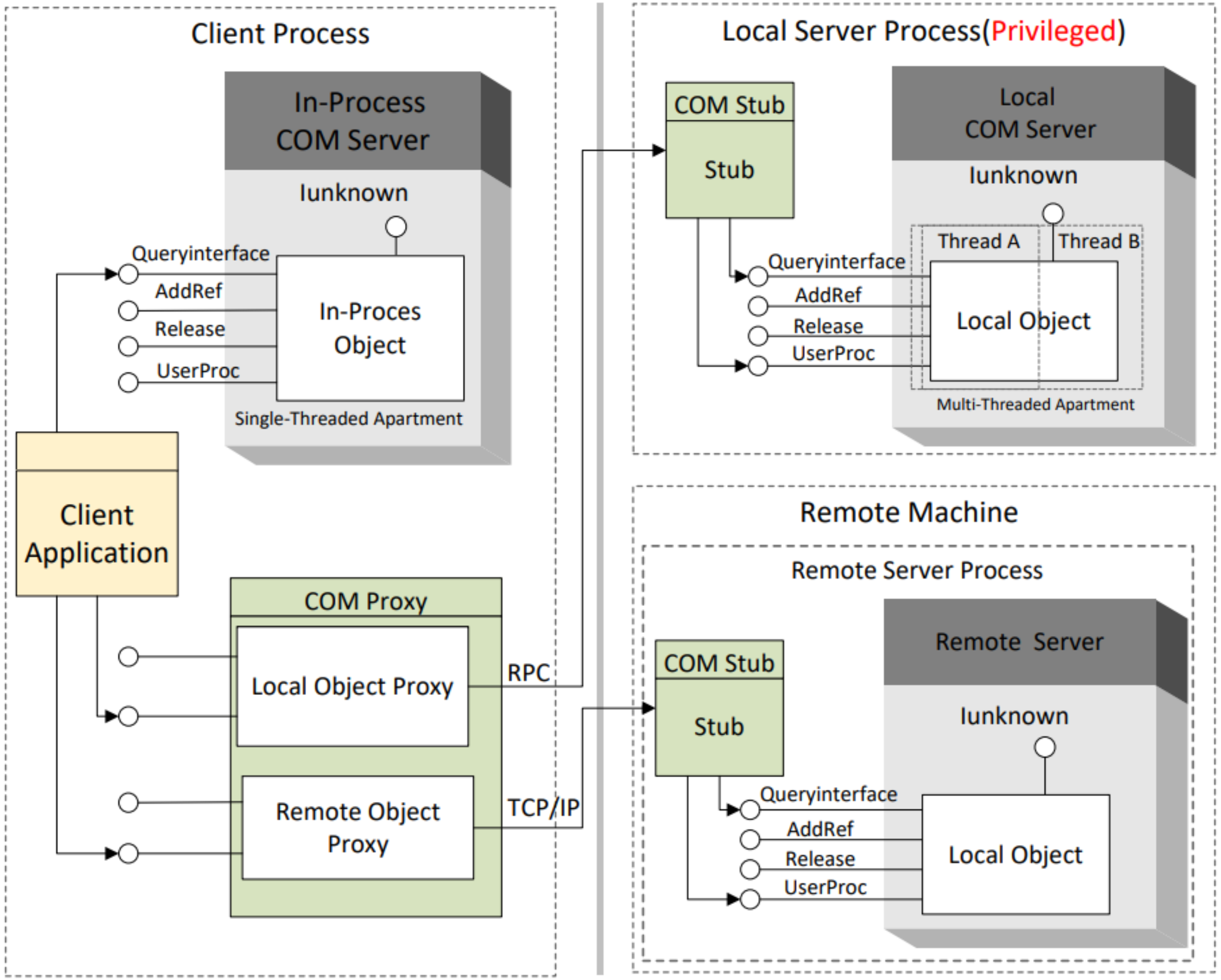
COM is a platform-independent, distributed, object-oriented system for creating binary software components that can interact.

The COM Communication Model

Ref: https://docs.microsoft.com/en-us/windows/win32/com/component-object-model--com--portal

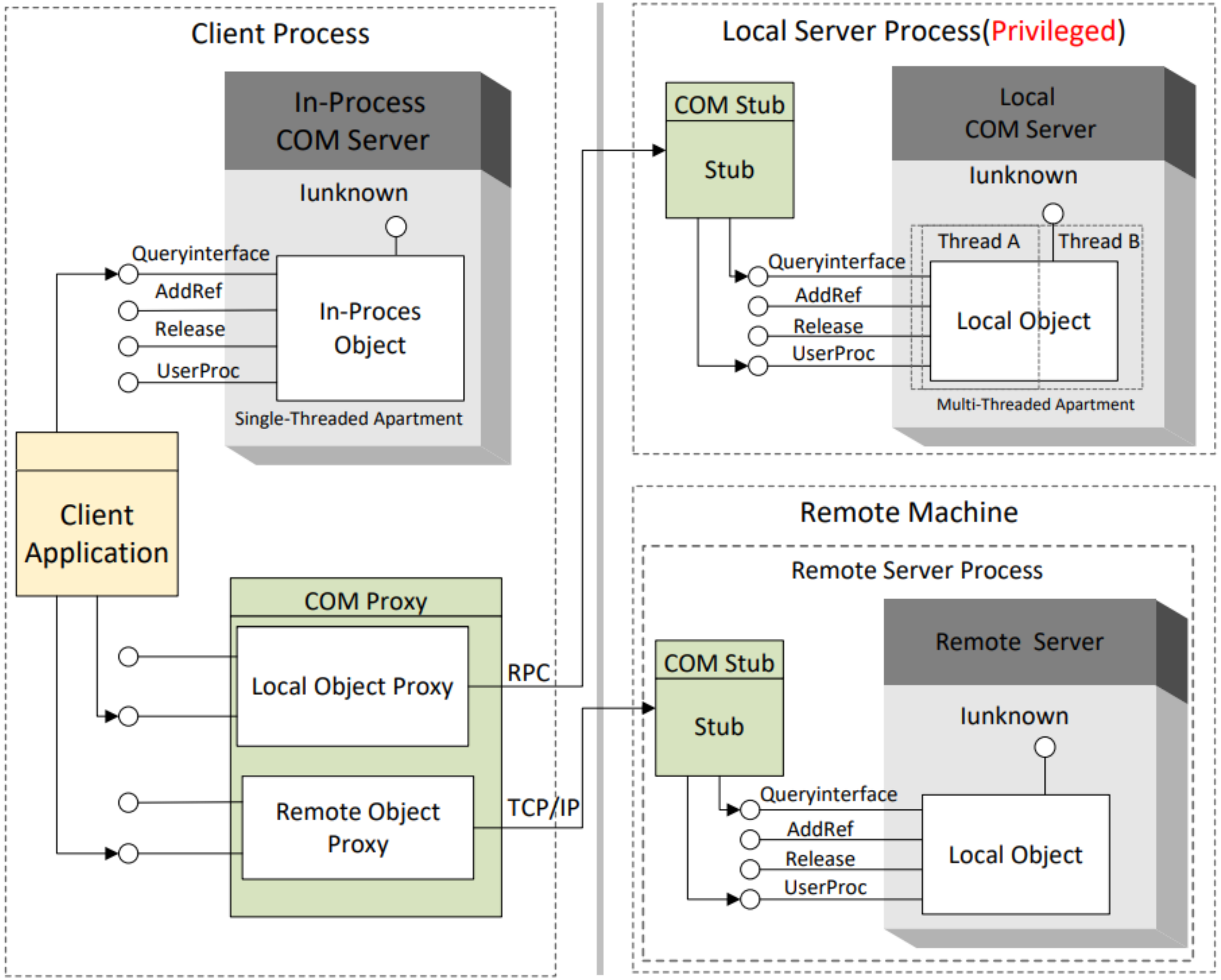
The COM Threading Model

The COM Threading Model



In-process and cross-process COM objects

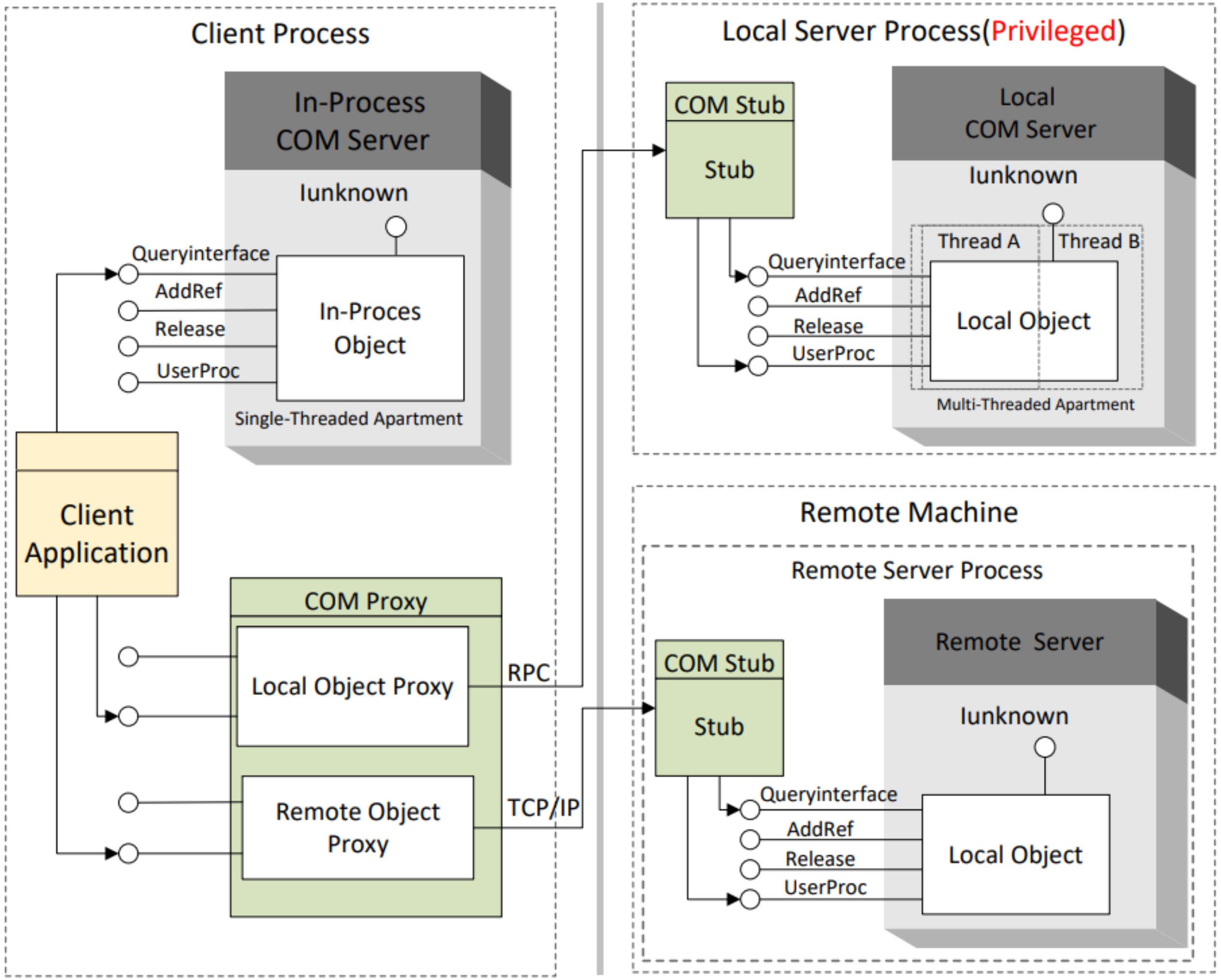
The COM Threading Model



In-process and cross-process COM objects

Single-threaded Apartment:
A single-threaded apartment (STA) consists of exactly one thread.

The COM Threading Model

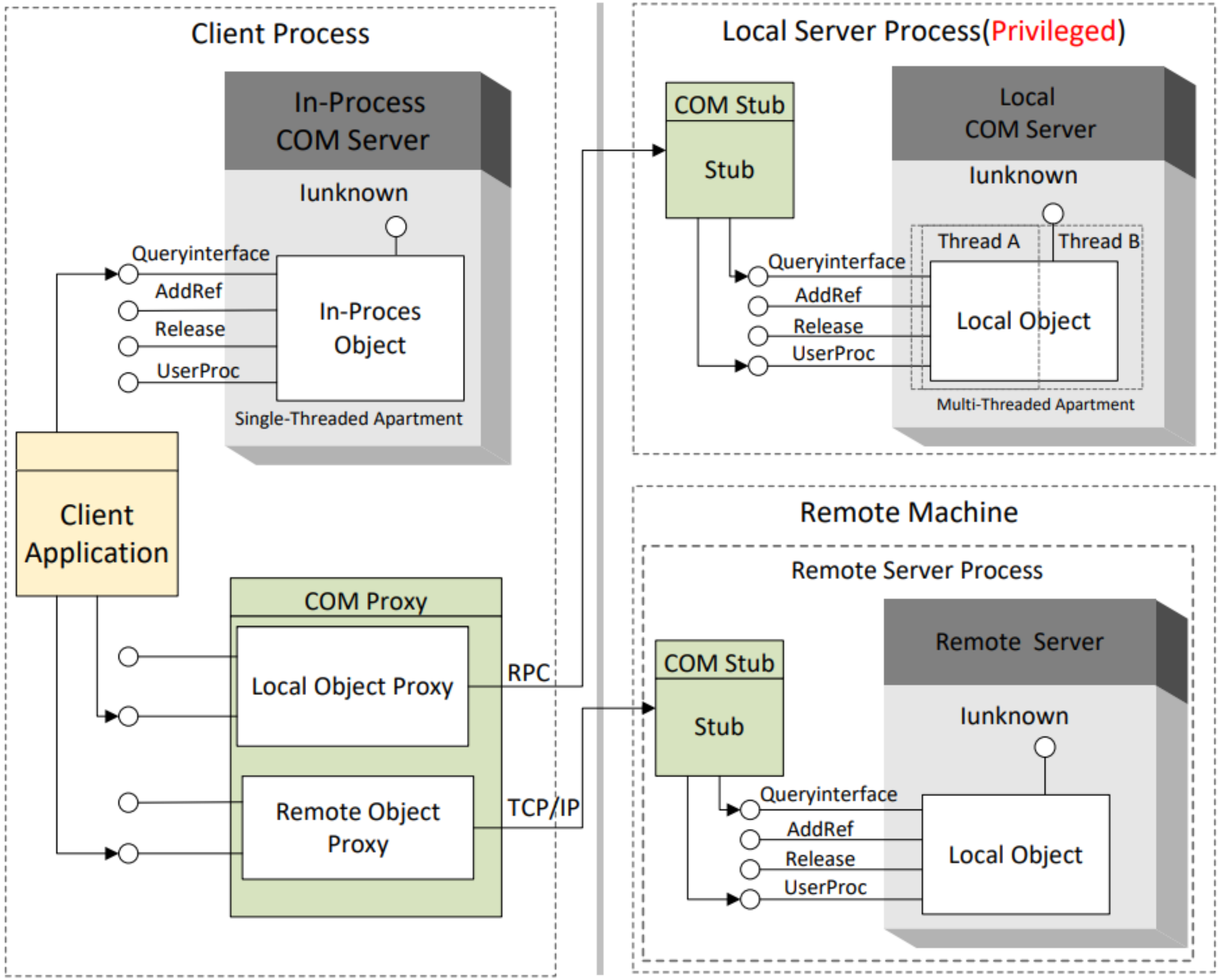


In-process and cross-process COM objects

Single-threaded Apartment:
A single-threaded apartment (STA) consists of exactly one thread.

Multi-threaded Apartment:
A multi-threaded apartment (MTA) consists of one or more threads.

The COM Threading Model



In-process and cross-process COM objects

Single-threaded Apartment:

A single-threaded apartment (STA) consists of exactly one thread.

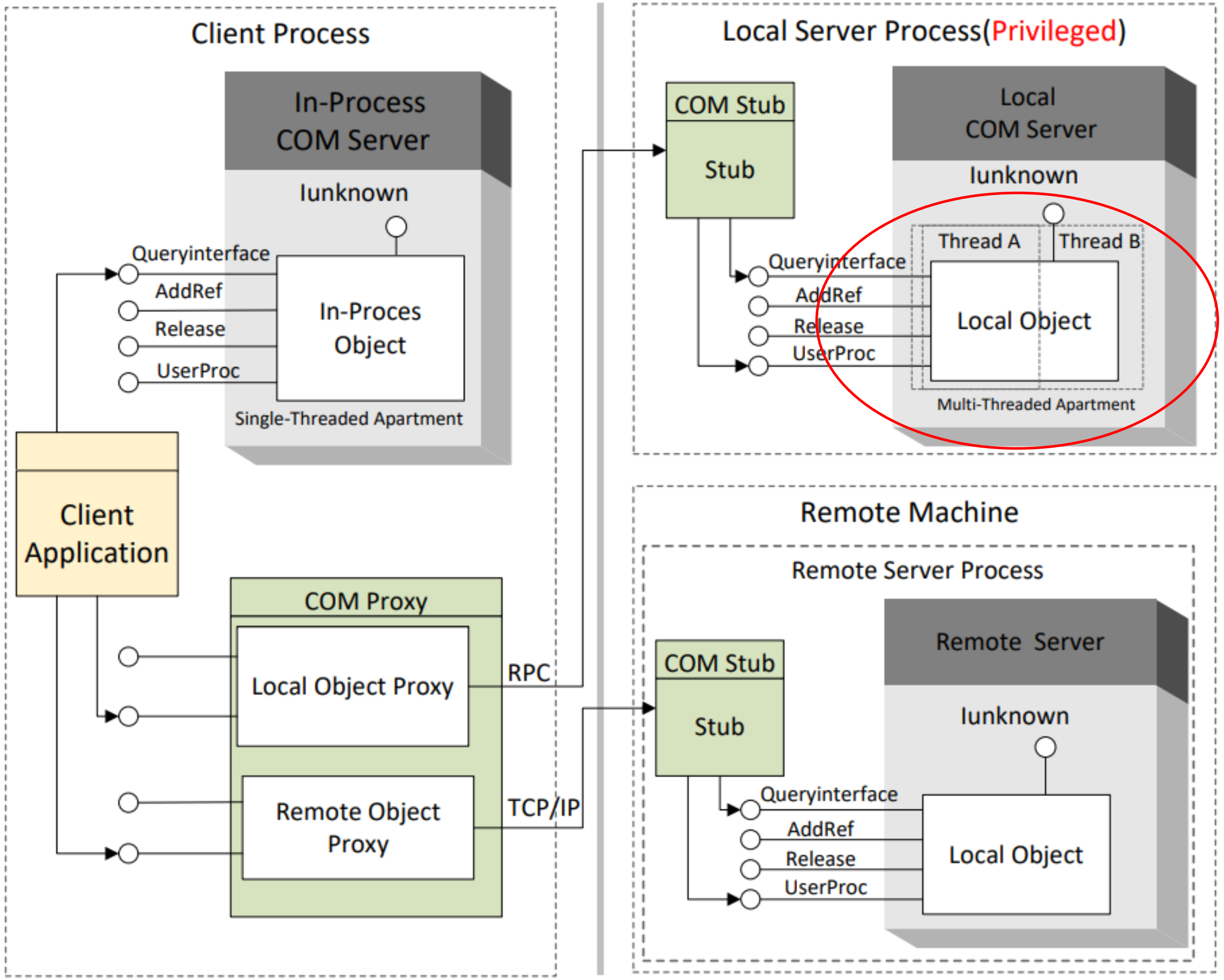
Multi-threaded Apartment:

A multi-threaded apartment (MTA) consists of one or more threads.

Neutral-threaded Apartment:

The neutral-threaded apartment (NTA) is introduced for more efficient cross apartment calls. Similar to MTA, COM objects in an NTA need to guarantee thread safety by themselves.

The COM Threading Model



In-process and cross-process COM objects

Single-threaded Apartment:

A single-threaded apartment (STA) consists of exactly one thread.

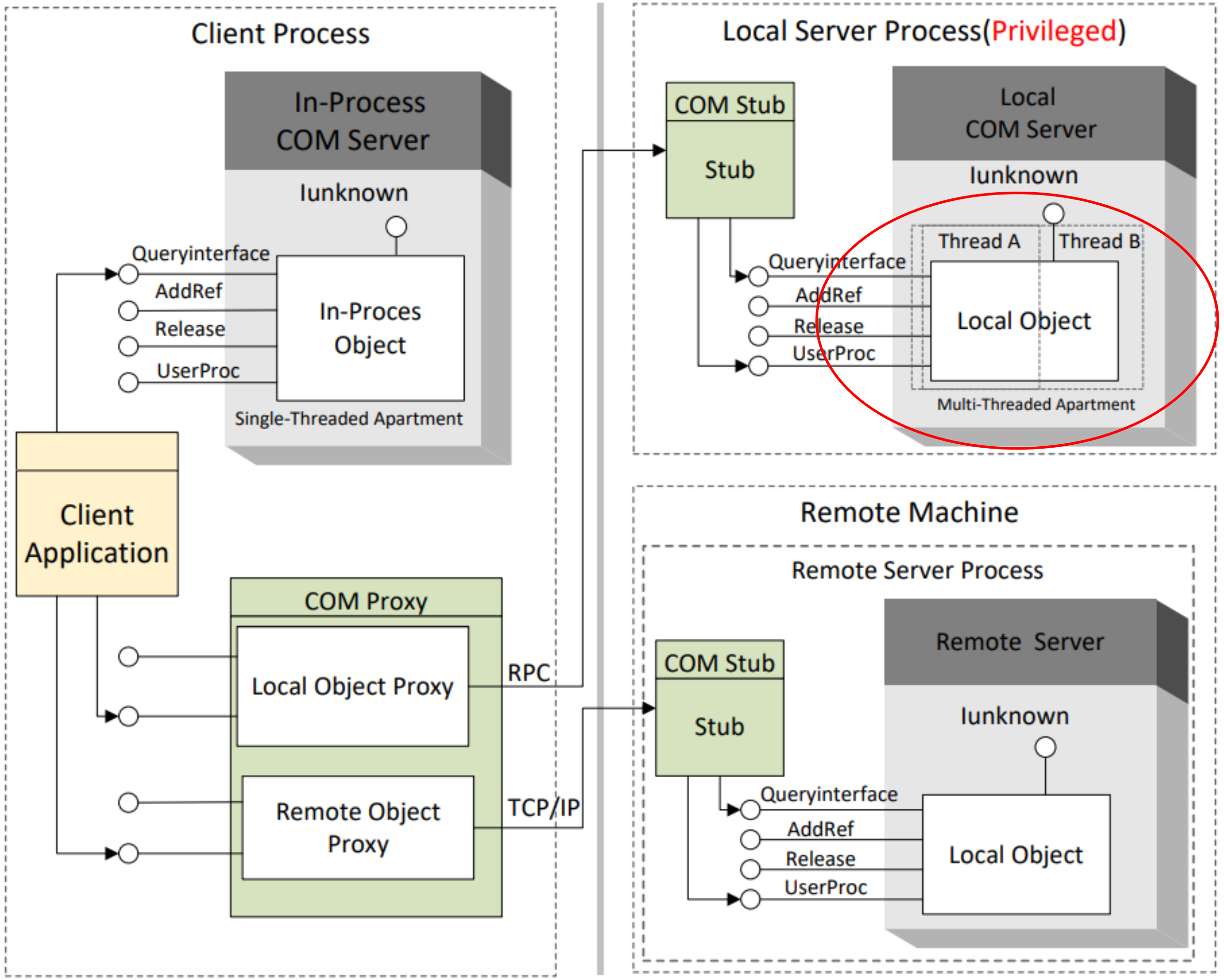
Multi-threaded Apartment:

A multi-threaded apartment (MTA) consists of one or more threads.

Neutral-threaded Apartment:

The neutral-threaded apartment (NTA) is introduced for more efficient cross apartment calls. Similar to MTA, COM objects in an NTA need to guarantee thread safety by themselves.

The COM Threading Model



In-process and cross-process COM objects

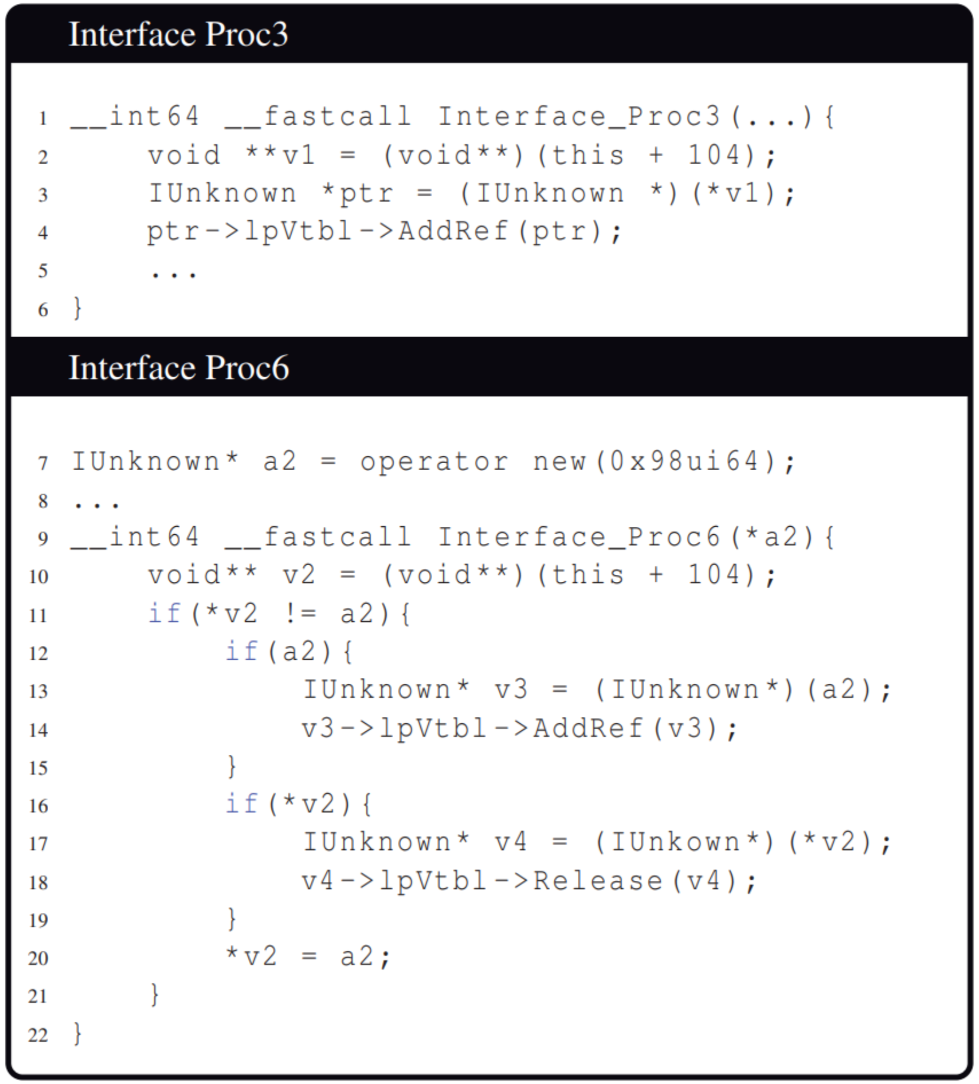
Single-threaded Apartment:
A single-threaded apartment (STA) consists of exactly one thread.

Multi-threaded Apartment:
A multi-threaded apartment (MTA) consists of one or more threads.

Neutral-threaded Apartment:
The neutral-threaded apartment (NTA) is introduced for more efficient cross apartment calls. Similar to MTA, COM objects in an NTA need to guarantee thread safety by themselves.

Case Study: CVE-2020-1394

Case Study: CVE-2020-1394



Interface Proc3

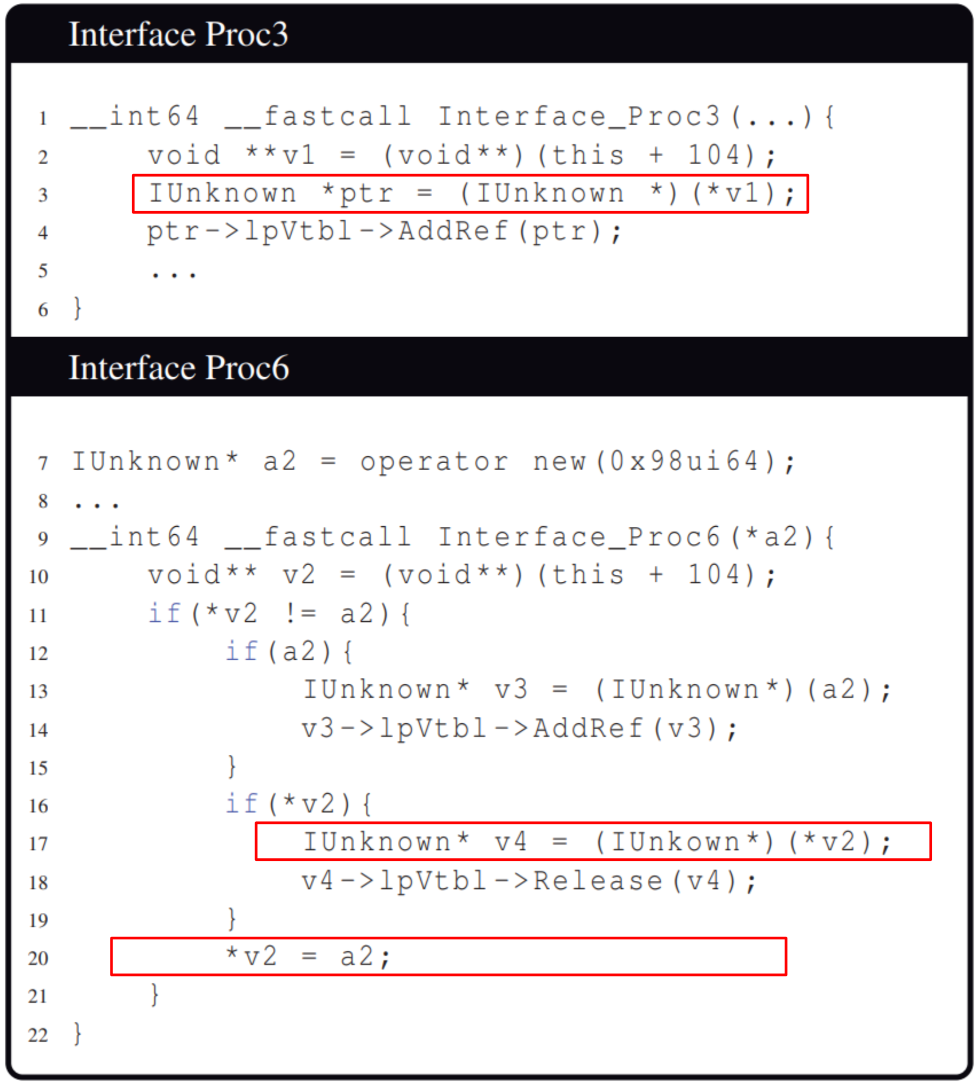
```
__int64 __fastcall Interface_Proc3(...){
    void **v1 = (void**)(this + 104);
    IUnknown *ptr = (IUnknown *)(*v1);
    ptr->lpVtbl->AddRef(ptr);
    ...
}
```

Interface Proc6

```
IUnknown* a2 = operator new(0x98ui64);
...
__int64 __fastcall Interface_Proc6(*a2){
    void** v2 = (void**)(this + 104);
    if(*v2 != a2){
        if(a2){
            IUnknown* v3 = (IUnknown*)(a2);
            v3->lpVtbl->AddRef(v3);
        }
        if(*v2){
            IUnknown* v4 = (IUnkown*)(*v2);
            v4->lpVtbl->Release(v4);
        }
        *v2 = a2;
    }
}
```

The GeoLocation COM Object

Case Study: CVE-2020-1394



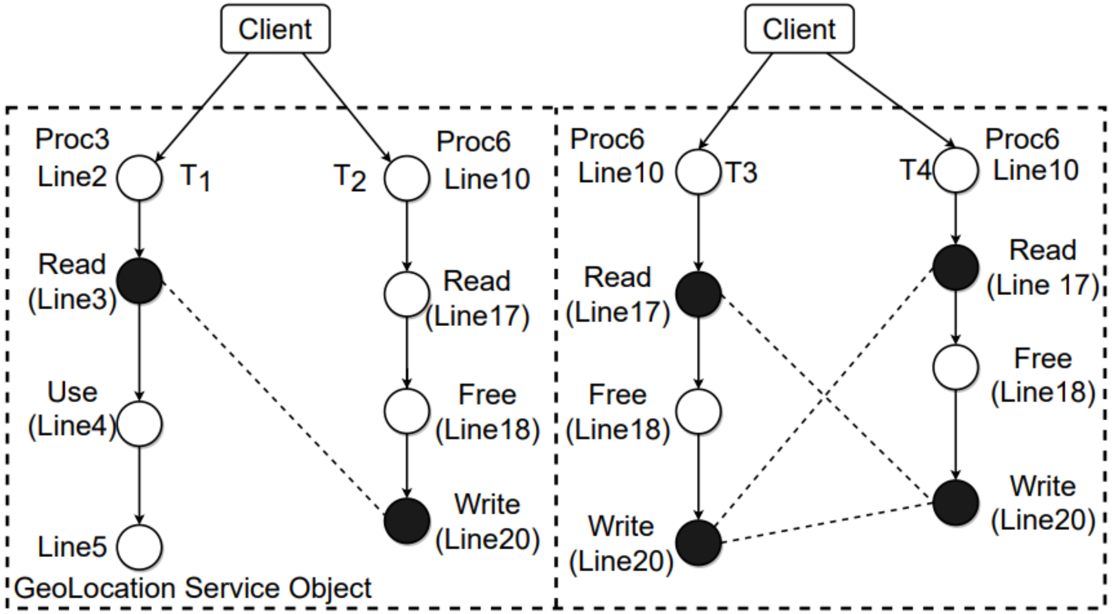
Interface Proc3

```
1  __int64 __fastcall Interface_Proc3(...){
2      void **v1 = (void**)(this + 104);
3      IUnknown *ptr = (IUnknown *)(*v1);
4      ptr->lpVtbl->AddRef(ptr);
5      ...
6  }
```

Interface Proc6

```
7  IUnknown* a2 = operator new(0x98ui64);
8  ...
9  __int64 __fastcall Interface_Proc6(*a2){
10     void** v2 = (void**)(this + 104);
11     if(*v2 != a2){
12         if(a2){
13             IUnknown* v3 = (IUnknown*)(a2);
14             v3->lpVtbl->AddRef(v3);
15         }
16         if(*v2){
17             IUnknown* v4 = (IUnkown*)(*v2);
18             v4->lpVtbl->Release(v4);
19         }
20         *v2 = a2;
21     }
22 }
```

The GeoLocation COM Object

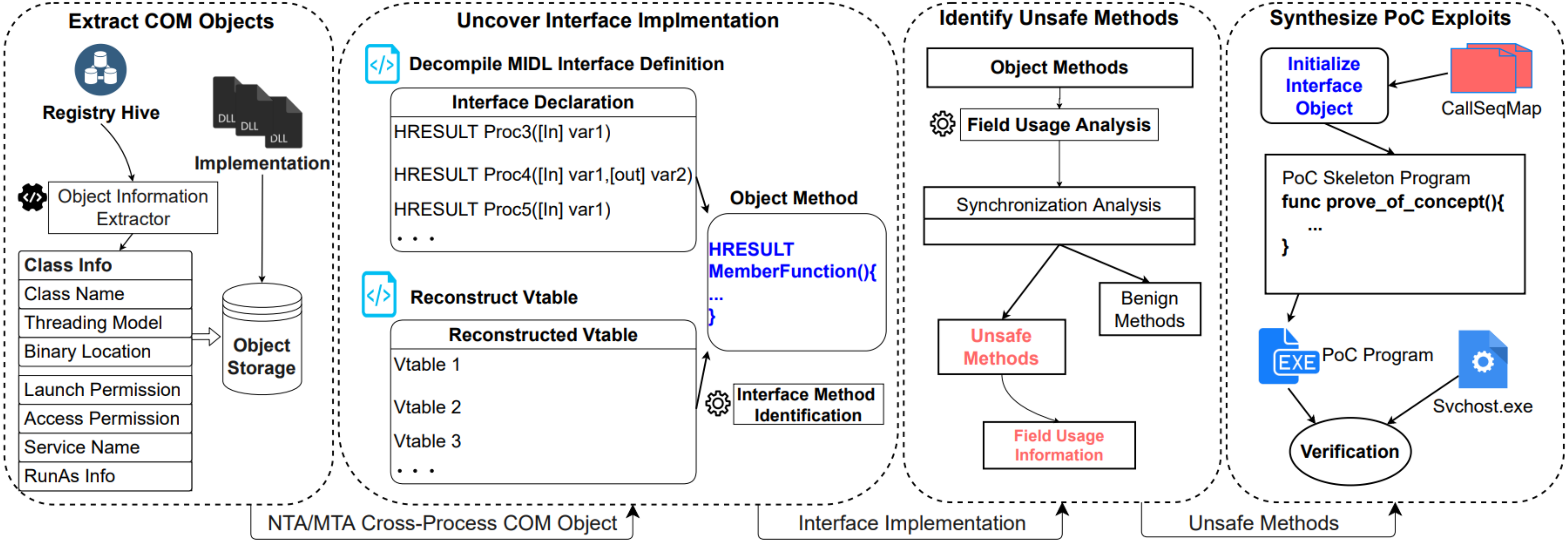


(a). Use-After-Free

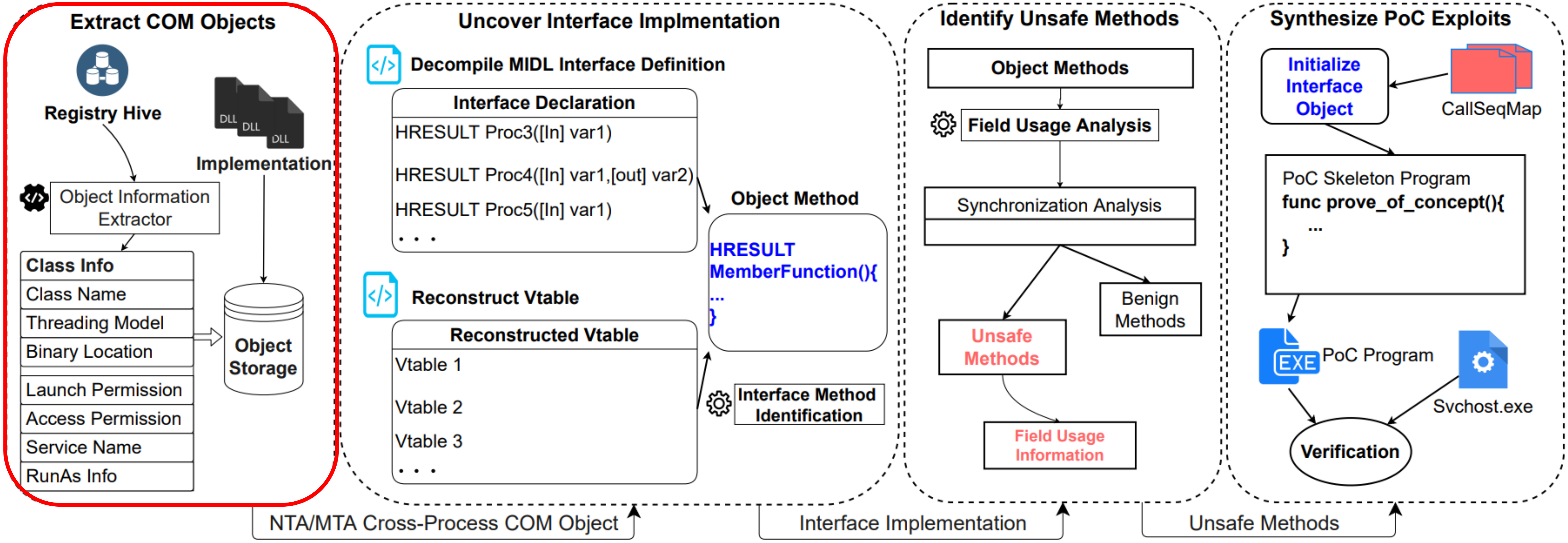
(b). Double-Free

Two Data Races in Object's Interface Methods

Overview of COMRACE

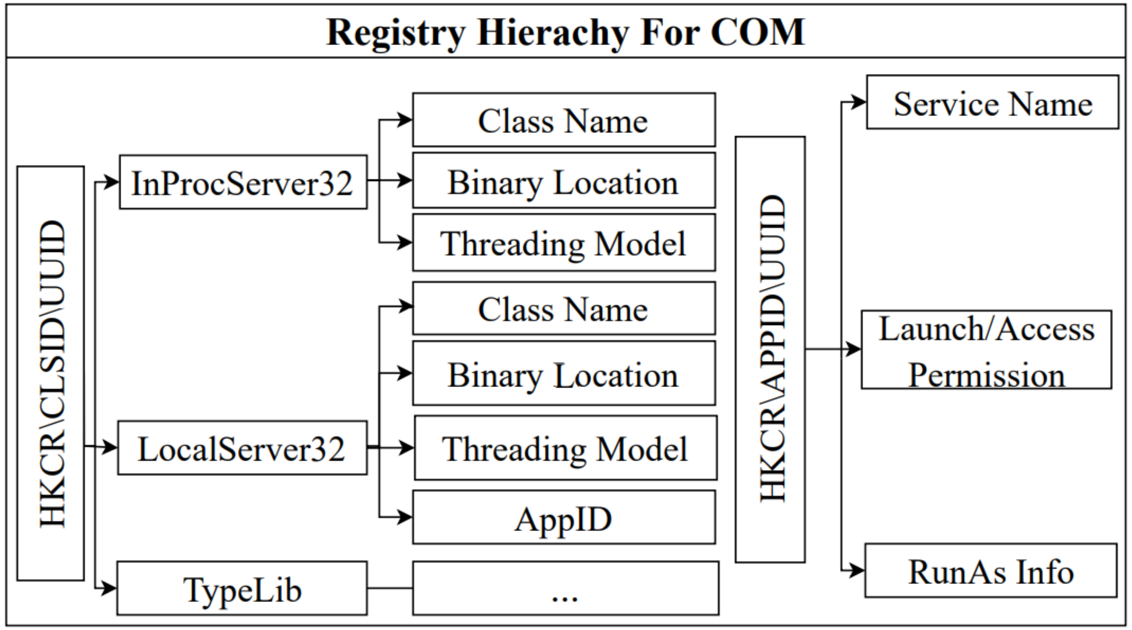


Extract COM Objects Information

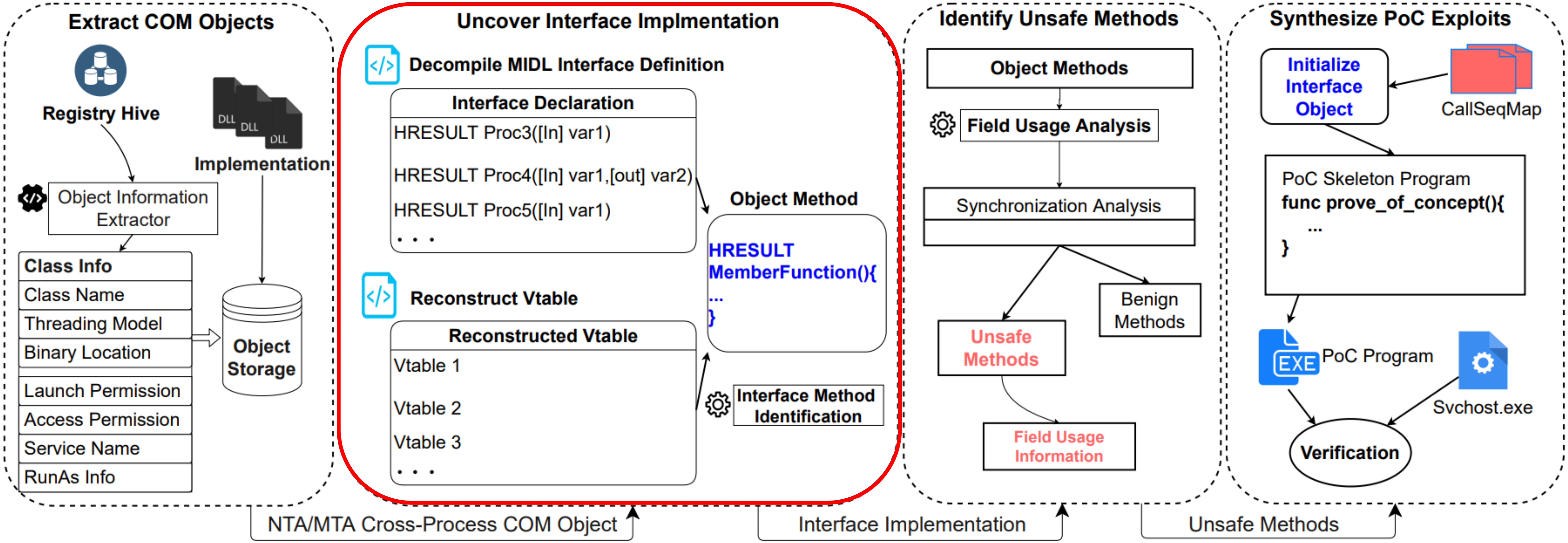


Extract COM Objects Information

- COM Basic Information Extraction(CLSID)
  - Traverse registry for Basic COM information
- COM Server Information Extraction(APPID)
  - Extract Class and Service relationship
  - Threading Model
  - Binary Location
  - AppID
- Service Information Extraction(CLSID)
  - Service Name
  - Launch Permission
  - RunAs Info



Uncover Interface Implementation



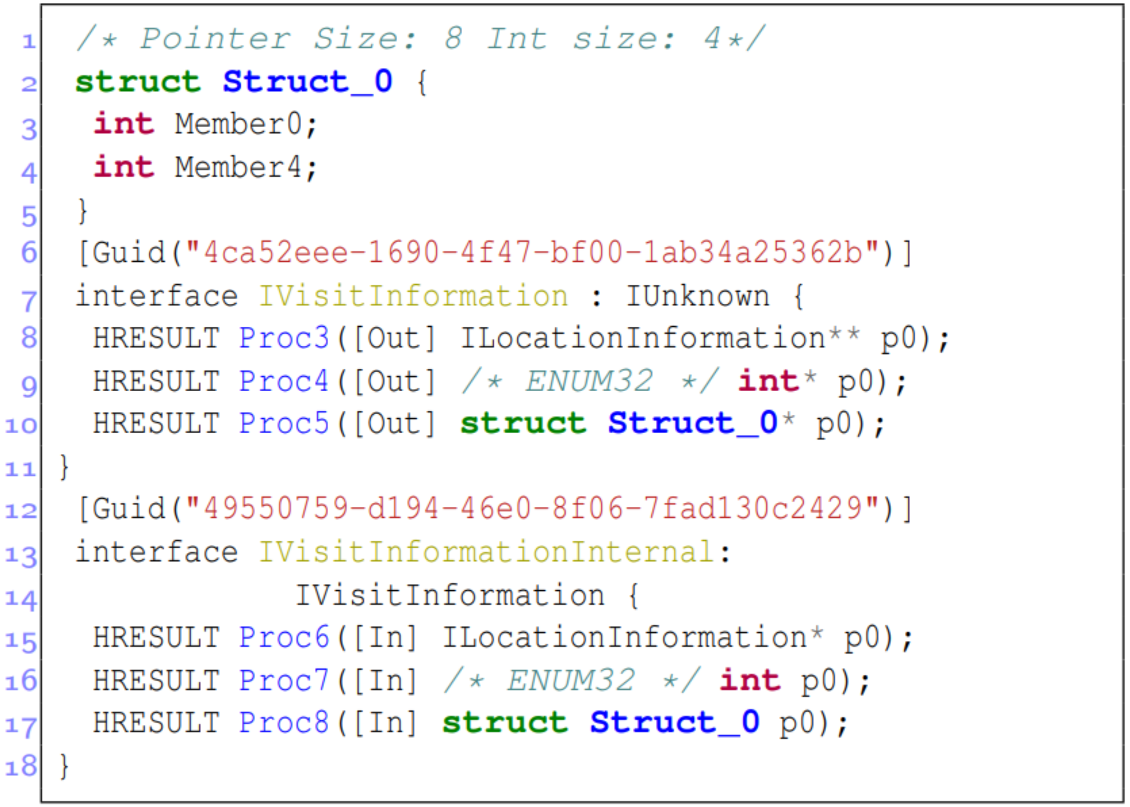
Uncover Interface Implementation

- Retrieve Interface Declaration
  - Use the tool OleViewDotNet[1] to decompile interface declaration from binary files
- Reconstruct Vtables
  - Heuristic based approach
  - Code pattern search
- Match Interface to Vtable
  - Parameter type and layout consistent check
  - Interface inheritance check

[1]. James Forshaw. Oleviewdotnet. https://github.com/tyranid/oleviewdotnet, 2020.

Uncover Interface Implementation

- ➢ Retrieve Interface Declaration
  - ➢ Use the tool OleViewDotNet[1] to decompile interface declaration from binary files
- ➢ Reconstruct Vtables
  - ➢ Heuristic based approach
  - ➢ Code pattern search
- ➢ Match Interface to Vtable
  - ➢ Parameter type and layout consistent check
  - ➢ Interface inheritance check



```
/* Pointer Size: 8 Int size: 4*/
struct Struct_0 {
 int Member0;
 int Member4;
}
[Guid("4ca52eee-1690-4f47-bf00-1ab34a25362b")]
interface IVisitInformation : IUnknown {
 HRESULT Proc3([Out] ILocationInformation** p0);
 HRESULT Proc4([Out] /* ENUM32 */ int* p0);
 HRESULT Proc5([Out] struct Struct_0* p0);
}
[Guid("49550759-d194-46e0-8f06-7fad130c2429")]
interface IVisitInformationInternal:
             IVisitInformation {
 HRESULT Proc6([In] ILocationInformation* p0);
 HRESULT Proc7([In] /* ENUM32 */ int p0);
 HRESULT Proc8([In] struct Struct_0 p0);
}
```

Decompiled interface declaration from the binary file LocationFramework.dll of GeoLocation Object.

[1]. James Forshaw. Oleviewdotnet. https://github.com/tyranid/oleviewdotnet, 2020.

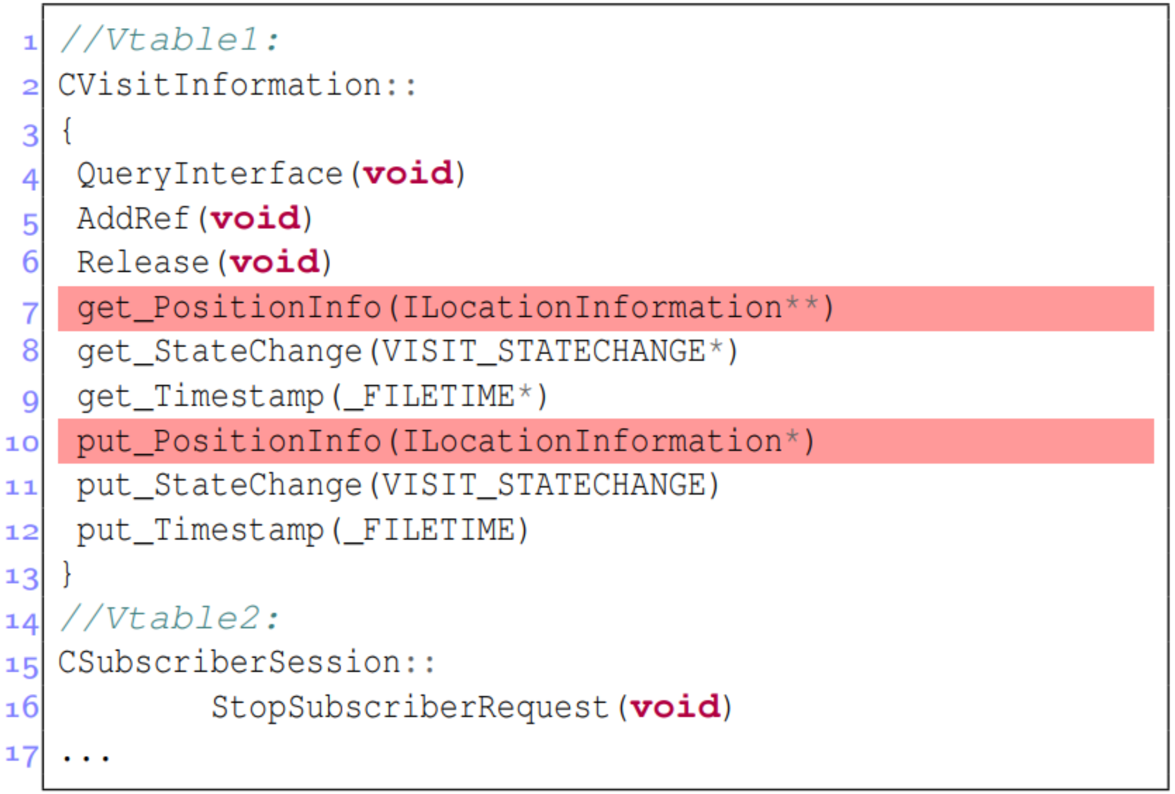
Uncover Interface Implementation

- Retrieve Interface Declaration
    - Use the tool OleViewDotNet[1] to decompile interface declaration from binary files
- Reconstruct Vtables
    - Heuristic based approach
    - Code pattern search
- Match Interface to Vtable
    - Parameter type and layout consistent check
    - Interface inheritance check

[1]. James Forshaw. Oleviewdotnet. https://github.com/tyranid/oleviewdotnet, 2020.

Uncover Interface Implementation

- ➢ Retrieve Interface Declaration
  - ➢ Use the tool OleViewDotNet[1] to decompile interface declaration from binary files
- ➢ Reconstruct Vtables
  - ➢ Heuristic based approach
  - ➢ Code pattern search
- ➢ Match Interface to Vtable
  - ➢ Parameter type and layout consistent check
  - ➢ Interface inheritance check

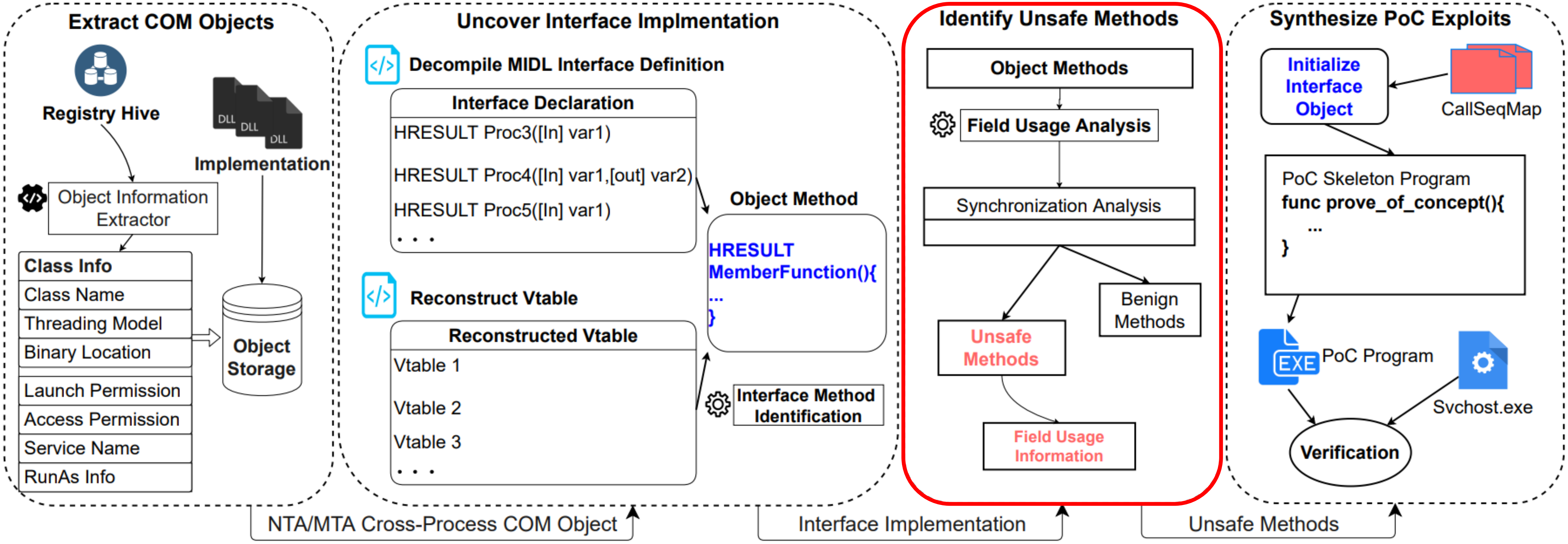


```
//Vtable1:
CVisitInformation::
{
 QueryInterface(void)
 AddRef(void)
 Release(void)
 get_PositionInfo(ILocationInformation**)
 get_StateChange(VISIT_STATECHANGE*)
 get_Timestamp(_FILETIME*)
 put_PositionInfo(ILocationInformation*)
 put_StateChange(VISIT_STATECHANGE)
 put_Timestamp(_FILETIME)
}
//Vtable2:
CSubscriberSession::
        StopSubscriberRequest(void)
...
```

Reconstructed Vtables of COM object GeoLocation.

[1]. James Forshaw. Oleviewdotnet. https://github.com/tyranid/oleviewdotnet, 2020.

Identify Unsafe Methods



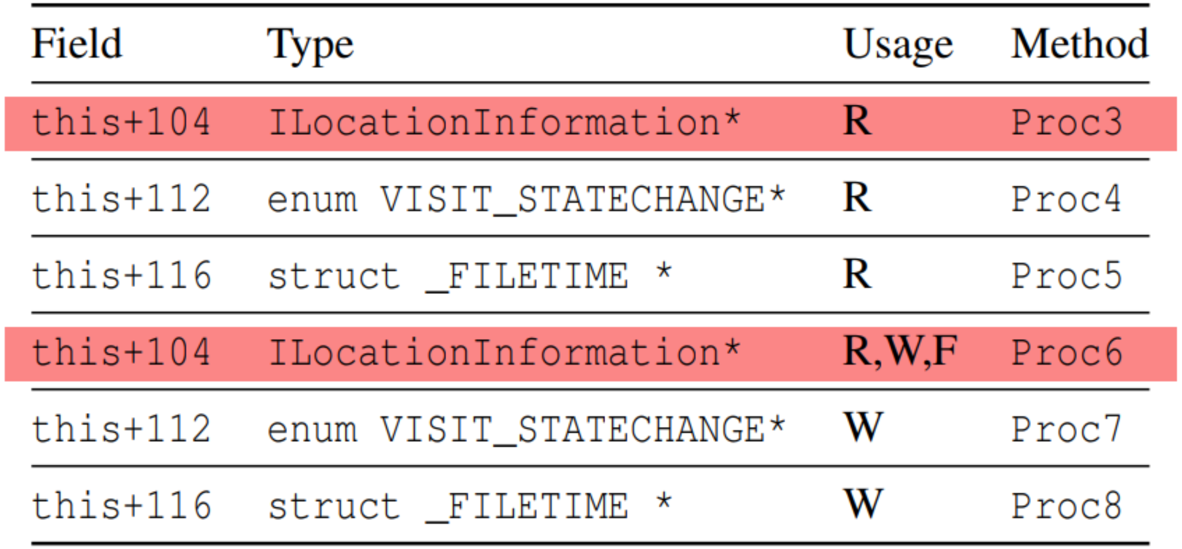
Identify Unsafe Methods

- Type propagation and track field usage
  - Resolve a virtual call target given the type of a member field
- Conduct a case analysis for each instruction
  - Sync: count number of Synchronization operations
  - Lock and unlock balance
- Predefined free and synchronization APIs
  - To track the sensitive free and lock/unlock ops

For more details, please refer to the paper.

Identify Unsafe Methods

- Type propagation and track field usage
  - Resolve a virtual call target given the type of a member field
- Conduct a case analysis for each instruction
  - Sync: count number of Synchronization operations
  - Lock and unlock balance
- Predefined free and synchronization APIs
  - To track the sensitive free and lock/unlock ops

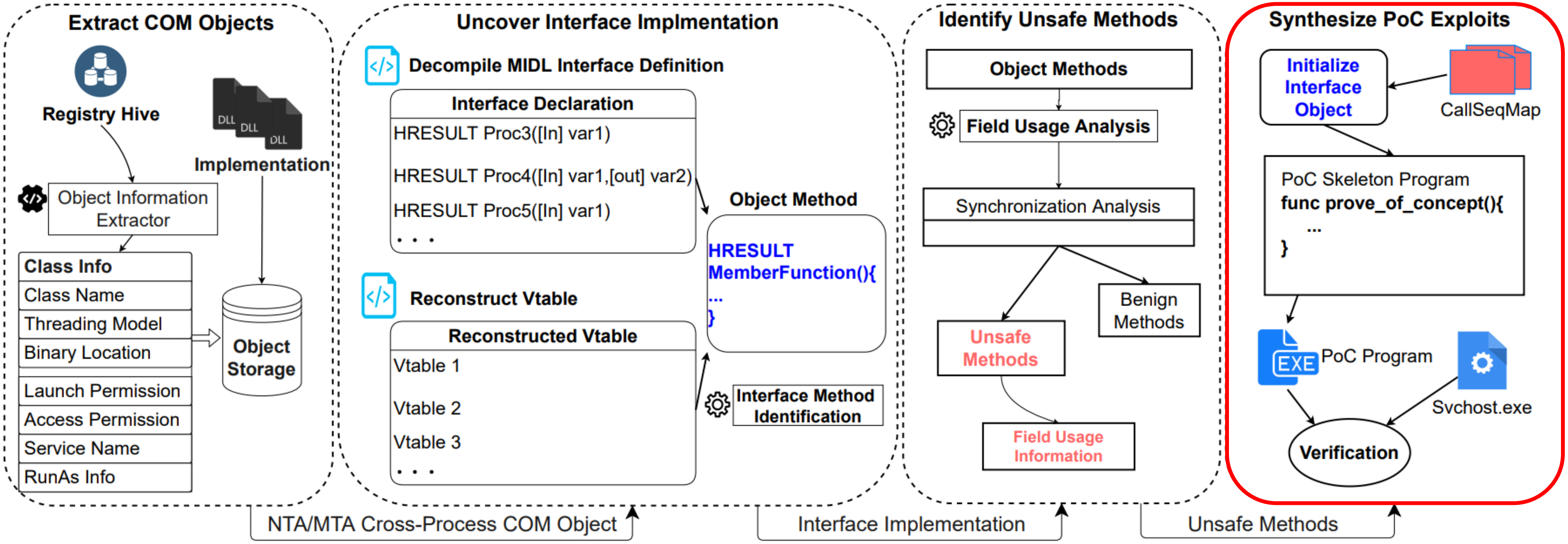


Field	Type	Usage	Method
this+104	ILocationInformation*	R	Proc3
this+112	enum VISIT_STATECHANGE*	R	Proc4
this+116	struct _FILETIME *	R	Proc5
this+104	ILocationInformation*	R,W,F	Proc6
this+112	enum VISIT_STATECHANGE*	W	Proc7
this+116	struct _FILETIME *	W	Proc8

Field usages and field types for interface methods of GeoLocation. R, W, and F stand for Read, Write, and Free, respectively.

For more details, please refer to the paper.

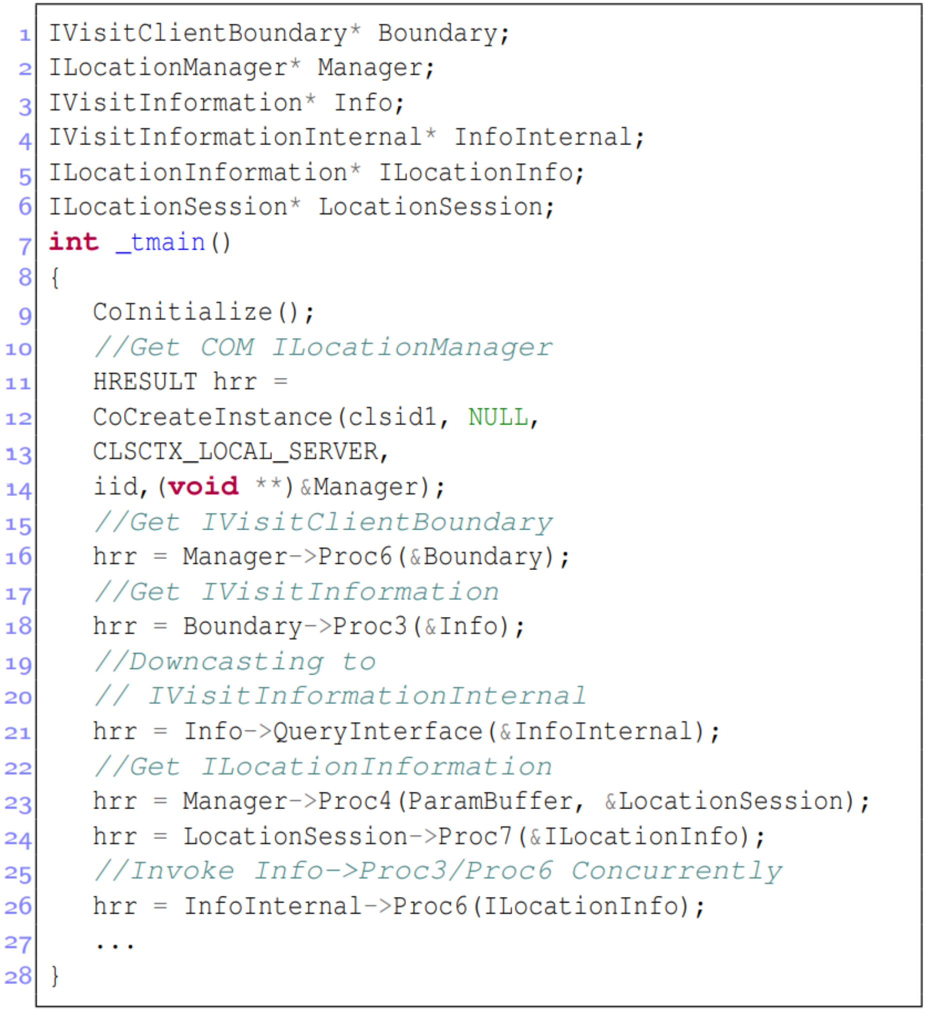
Synthesize PoC Exploits



Synthesize PoC Exploits

- Construct PoC skeleton program
  - Pre-generated header file with all recovered interface declarations
  - Standard program entry and exiting procedure
- Method Invocation preparation
  - Primitive-typed value set
  - Interface acquisition
  - Interface-typed argument set
- Running Concurrently
  - Running with [2]PageHeap enabled
  - Collect runtime information

For more details, please refer to the paper.



```
IVisitClientBoundary* Boundary;
ILocationManager* Manager;
IVisitInformation* Info;
IVisitInformationInternal* InfoInternal;
ILocationInformation* ILocationInfo;
ILocationSession* LocationSession;
int _tmain()
{
   CoInitialize();
   //Get COM ILocationManager
   HRESULT hrr =
   CoCreateInstance(clsid1, NULL,
   CLSCTX_LOCAL_SERVER,
   iid,(void **)&Manager);
   //Get IVisitClientBoundary
   hrr = Manager->Proc6(&Boundary);
   //Get IVisitInformation
   hrr = Boundary->Proc3(&Info);
   //Downcasting to
   // IVisitInformationInternal
   hrr = Info->QueryInterface(&InfoInternal);
   //Get ILocationInformation
   hrr = Manager->Proc4(ParamBuffer, &LocationSession);
   hrr = LocationSession->Proc7(&ILocationInfo);
   //Invoke Info->Proc3/Proc6 Concurrently
   hrr = InfoInternal->Proc6(ILocationInfo);
   ...
}
```

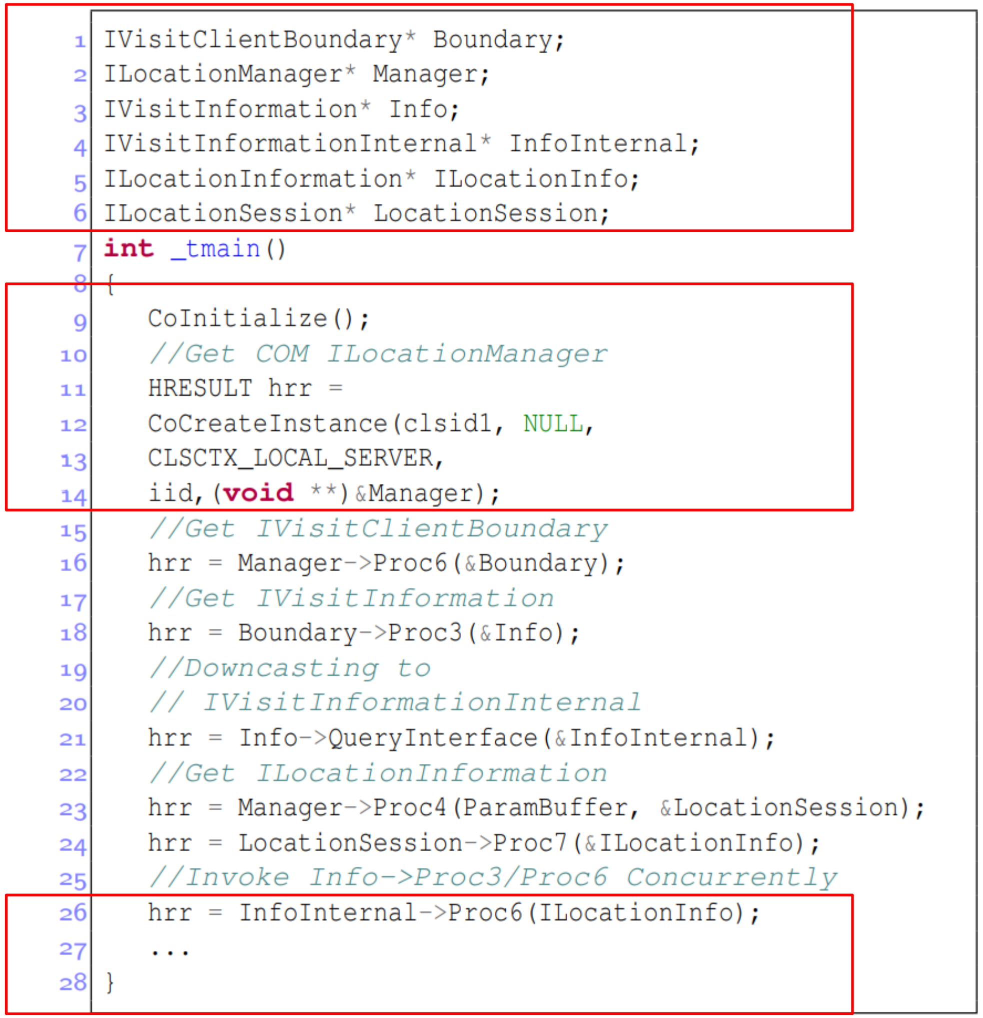
Manually constructed PoC for CVE-2020-1394

[2]Gflags and pageheap. https://docs.microsoft.com/en-us /windows-hardware/drivers/debugger /gflags-and-pageheap, 2017.

Synthesize PoC Exploits

- Construct PoC skeleton program
  - Pre-generated header file with all recovered interface declarations
  - Standard program entry and exiting procedure
- Method Invocation preparation
  - Primitive-typed value set
  - Interface acquisition
  - Interface-typed argument set
- Running Concurrently
  - Running with [2]PageHeap enabled
  - Collect runtime information

For more details, please refer to the paper.



```
IVisitClientBoundary* Boundary;
ILocationManager* Manager;
IVisitInformation* Info;
IVisitInformationInternal* InfoInternal;
ILocationInformation* ILocationInfo;
ILocationSession* LocationSession;
int _tmain()
{
    CoInitialize();
    //Get COM ILocationManager
    HRESULT hrr =
    CoCreateInstance(clsid1, NULL,
    CLSCTX_LOCAL_SERVER,
    iid,(void **)&Manager);
    //Get IVisitClientBoundary
    hrr = Manager->Proc6(&Boundary);
    //Get IVisitInformation
    hrr = Boundary->Proc3(&Info);
    //Downcasting to
    // IVisitInformationInternal
    hrr = Info->QueryInterface(&InfoInternal);
    //Get ILocationInformation
    hrr = Manager->Proc4(ParamBuffer, &LocationSession);
    hrr = LocationSession->Proc7(&ILocationInfo);
    //Invoke Info->Proc3/Proc6 Concurrently
    hrr = InfoInternal->Proc6(ILocationInfo);
    ...
}
```

Manually constructed PoC for CVE-2020-1394

[2]Gflags and pageheap. https://docs.microsoft.com/en-us /windows-hardware/drivers/debugger /gflags-and-pageheap, 2017.

Evaluations

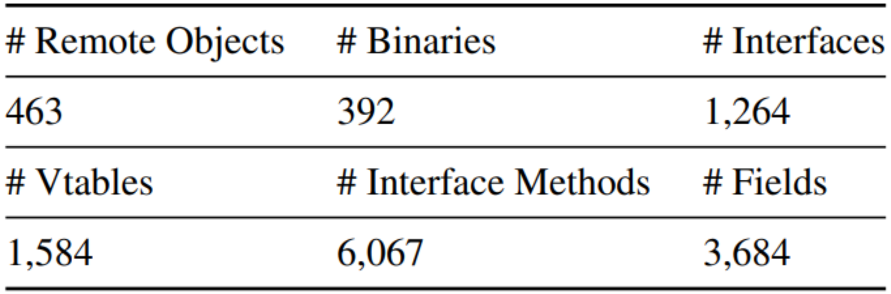
- RQ1: How effective can COMRACE analyze commercial off-the-shelf COM binaries?
- RQ2: How effective can COMRACE detect unsafe interface methods in COM binaries, and are they prevalent on the windows platform?
- RQ3: How dangerous are those data race bugs and can they cause severe damages?
- RQ4: How precise is COMRACE in detecting unsafe interface methods.

Evaluations(1/4)

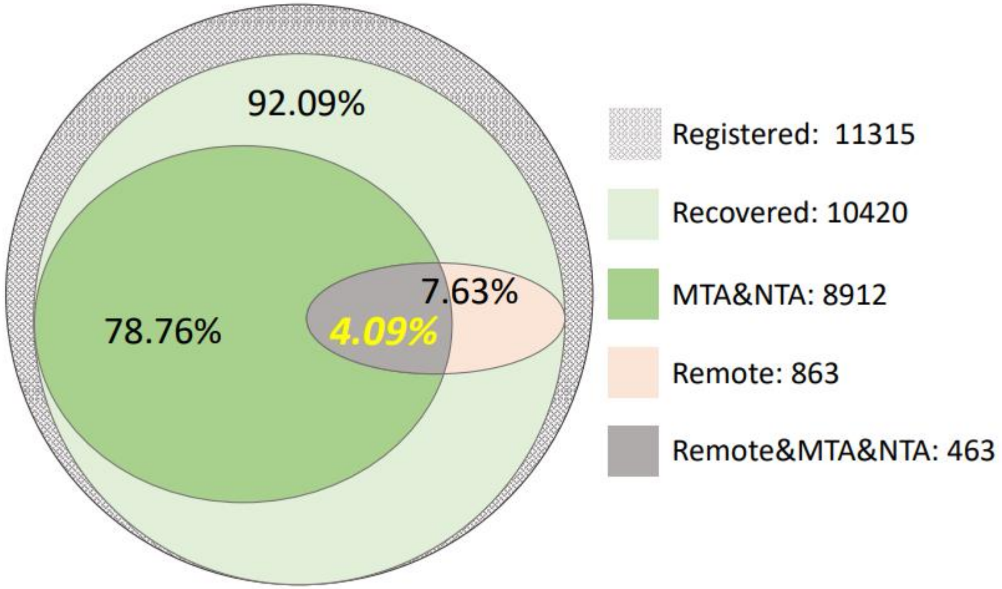
- **RQ1**: How effective can COMRACE analyze commercial off-the-shelf COM binaries?

- Among the total 11,315 COM objects on the Windows 10 (build 10.0.18363.657) platform, COMRACE successfully analyze 10,420 of them, with a success rate of 92.1%.

- 8,912 of the analyzed COM objects support MTA or NTA threading model, 463 among them are cross-process COM objects, which are prone to data race attacks. Each COM class consists of 8 member fields.



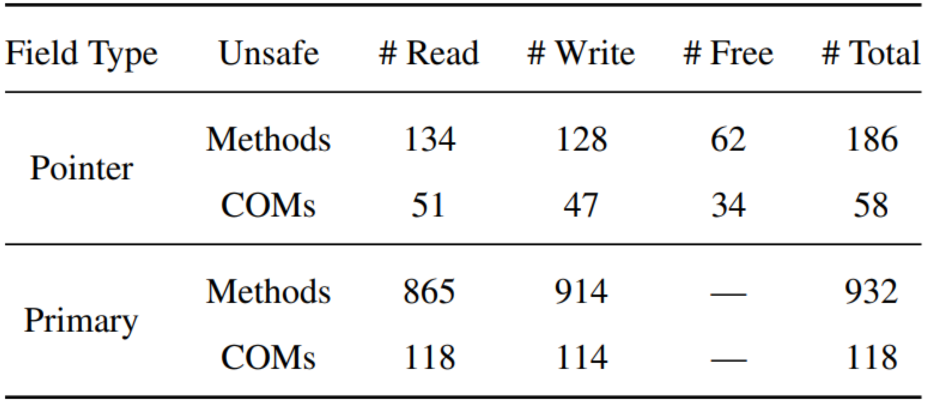
# Remote Objects	# Binaries	# Interfaces
463	392	1,264
# Vtables	# Interface Methods	# Fields
1,584	6,067	3,684



Statistics of total and analyzed COM objects on Windows 10(build 10.0.18363.657)

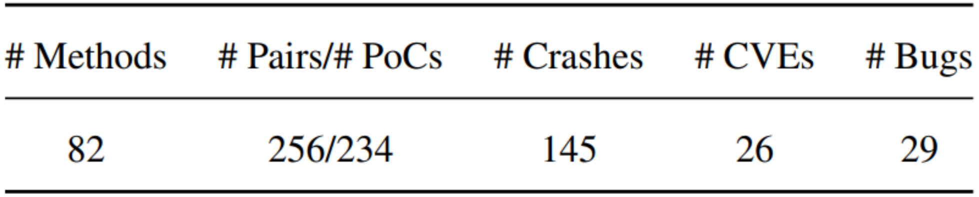
Evaluations(2/4)

- **RQ2**: How effective can COMRACE detect unsafe interface methods in COM binaries, and are they prevalent on the windows platform?
  - 62% of valid PoC programs can trigger memory corruption bugs.
  - Unsafe methods and unsafe COM objects are prevalent (18.4% of total methods, and 38.0% of total objects), suggesting wildly existing data race bugs.
  - Our experiments demonstrate that those unsafe methods are highly possible to trigger run-time bugs, and some can result in serious security violations (26 confirmed CVEs).



Field Type	Unsafe	# Read	# Write	# Free	# Total
Pointer	Methods	134	128	62	186
	COMs	51	47	34	58
Primary	Methods	865	914	—	932
	COMs	118	114	—	118

Number of unsafe methods and unsafe COM objects reported by COMRACE.



# Methods	# Pairs/# PoCs	# Crashes	# CVEs	# Bugs
82	256/234	145	26	29

Statistics of constructed PoC Programs.

Evaluations(3/4)

- **RQ3**: How dangerous are those data race bugs and can they cause severe damages?

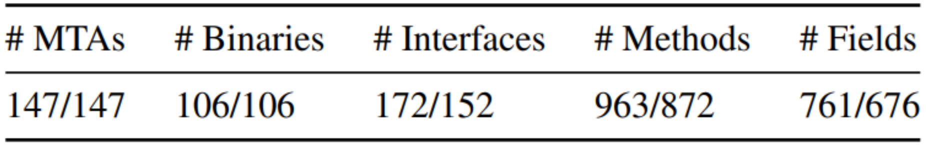
  - All the 26 confirmed vulnerabilities can lead to privilege escalation.

  - 23 of them can be exploited to escape the sandboxed security boundary. (imposed by the Windows Application container)

  - More importantly, in 20 vulnerabilities, the sandboxed privilege can be escalated to NT AUTHORITY/SYSTEM. This suggests that an attacker can gain unlimited privileges from those PoC exploits, posing serious security threats.

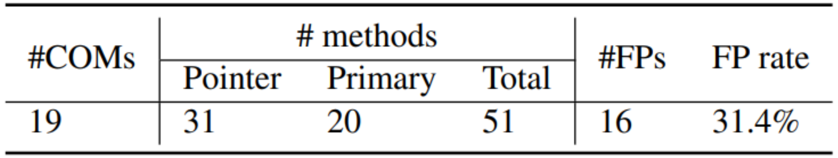
Evaluations(4/4)

- **RQ4**: How precise is COMRACE in detecting unsafe interface methods.



# MTAs	# Binaries	# Interfaces	# Methods	# Fields
147/147	106/106	172/152	963/872	761/676

- We evaluate the precision of COMRACE on the open-source ReactOS platform, result show that COMRACE can successfully extract all 147 MTA COM objects (out of 434 total COM objects) from 106 binary files and recover 152 out of 172 interfaces.



#COMs	# methods			#FPs	FP rate
	Pointer	Primary	Total		
19	31	20	51	16	31.4%

Number of unsafe methods and unsafe COM objects on ReactOS

- We fail to recover 20 interfaces because COMRACE cannot locate the binary files implementing those interfaces, although they are declared in the IDL source files. Manual inspection indicates that they may not be publicly accessible.

- COMRACE reports 19 unsafe COM objects with 51 unsafe interface methods, There are 16 false positives (Column 5), with a false positive rate of 31.4%. 10 false positives are due to incorrect alias. 6 false positives come from locking/unlocking primitives unmatched due to control flows.

Conclusion

- We present COMRACE, the first data race vulnerability detection tool for COM objects.

- The Solution applies static binary analyses to detect unsafe interface methods from off-the-shelf COM binaries, then verifies static analysis results with synthesized PoCs.

- Experiments show unsafe methods and unsafe COM objects are prevalent on Windows.

- COMRACE automatically synthesized 234 PoCs from 82 unsafe methods, 145 PoCs lead to critical memory corruption, exposing 26 CVEs.

Thanks for listening!

Q&A

Contant: Fangming Gu, gufangming@iie.ac.cn