Be Careful of What You Embed: Demystifying OLE Vulnerabilities

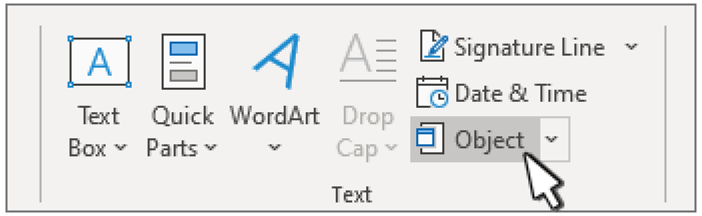
Yunpeng Tian[1] , Feng Dong[1], Haoyi Liu[1], Meng Xu[2],
Zesen Ye[3], Zhiniang Peng[1,3], Shenghui Li[1], Xiapu Luo[4], Haoyu Wang[1]

[1]Huazhong University of Science and Technology
[2]University of Waterloo
[3]Sangfor Technologies Inc.
[4]The Hong Kong Polytechnic University

What is OLE?

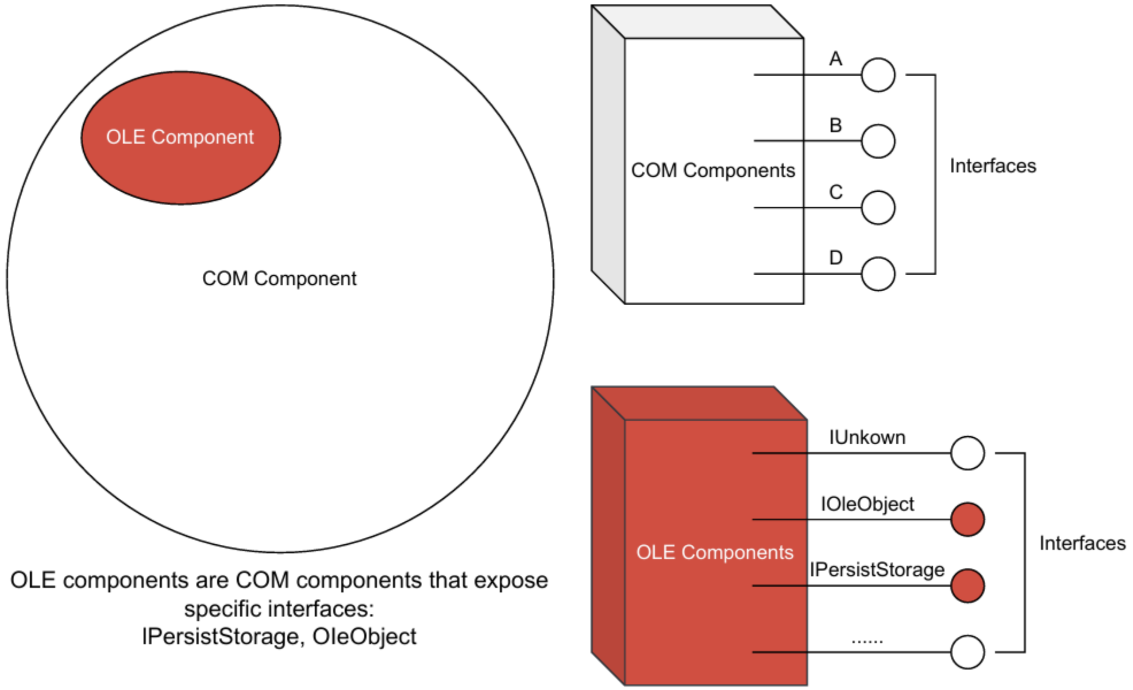
Introduction to OLE
• OLE(Object Linking and Embedding) is a technology that allows sharing of data and functionalities between Windows applications.
•Developed by Microsoft in the early 1990s.

Application Scenarios of OLE
•**Document Editing:** Embedding Excel spreadsheets in Word documents.
•**Presentations:** Embedding Visio diagrams in PowerPoint.
•**Data Sharing:** Sharing data between different software applications to maintain data consistency.



https://support.microsoft.com/en-us/office/embed-or-link-to-a-file-in-word-8d1a0ffd-956d-4368-887c-b374237b8d3a

Understanding OLE



OLE components are special COM components

Basics of OLE 2.0:

- **COM Interfaces:** All OLE components, as COM components, expose the ***IUnknown*** interface, allowing clients to discover other interfaces like ***IOleObject***, ***IOleLink***, and ***IViewObject2*** for specific OLE functionalities.

- **Structured Storage:** Since different OLE components have varying implementations of ***IPersistStorage***, the format of stored data can differ significantly.

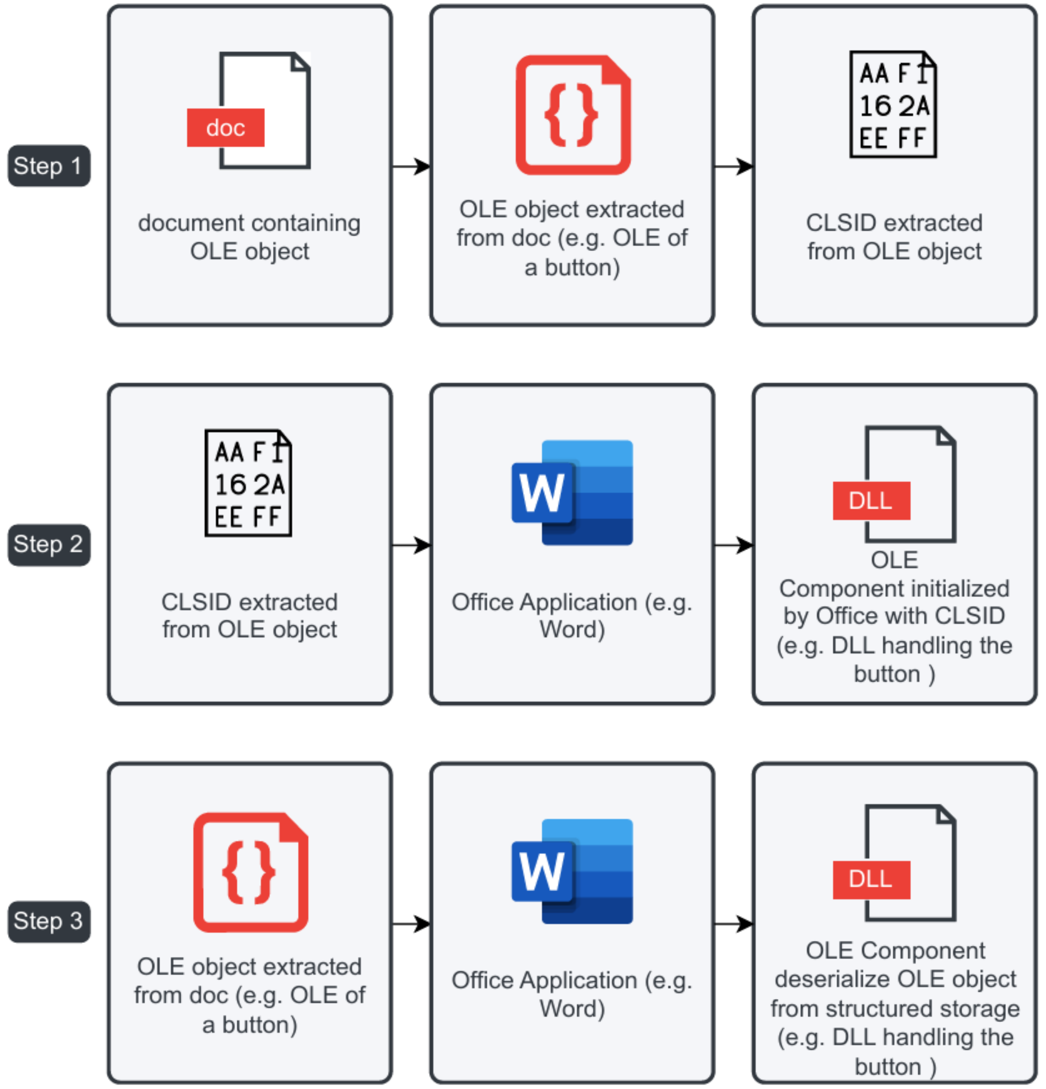
OLE *Object* Categories:

- **Embedded Objects**: Self-contained, **stored in host document (e.g., Word in Excel)**.
- **Linked Objects**: References external files, updates reflect in the host document.

Focus: In-process embedded objects.

How OLE Works

- **Step 1**: Retrieve the CLSID from the document.
- **Step 2**: Invoke ***CoCreateInstance*** to load the module
- **Step 3**: Invoke ***IPersistStorage::Load*** to deserialize the OLE object


Step 1
doc
document containing OLE object
OLE object extracted from doc (e.g. OLE of a button)
AA F1 16 2A EE FF
CLSID extracted from OLE object
Step 2
AA F1 16 2A EE FF
CLSID extracted from OLE object
Office Application (e.g. Word)
DLL
OLE Component initialized by Office with CLSID (e.g. DLL handling the button)
Step 3
OLE object extracted from doc (e.g. OLE of a button)
Office Application (e.g. Word)
DLL
OLE Component deserialize OLE object from structured storage (e.g. DLL handling the button)

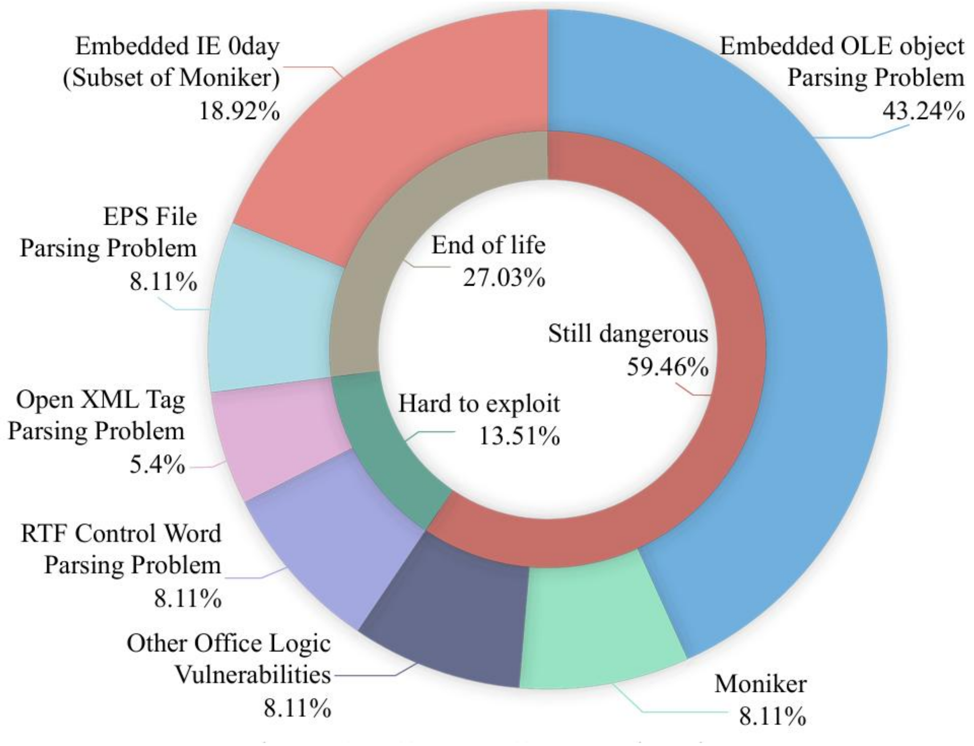
Summary of Known Office Vulnerabilities

Key Findings:

Embedded OLE Object Parsing Problems:
- Account for 43.24% of vulnerabilities.
- Pose significant current risk.

End-of-Life Applications:
- 27.03% of vulnerabilities are no longer applicable.

Difficult to Exploit:
- 13.51% are unlikely to be practically exploited (no incidents in the past five years).



OLE Attack Vectors

- **Type-1: Loading a COM Component Not Intended for OLE**
- **Type-2: DLL Preloading Attacks**
- **Type-3: OLE Data Parsing Error in *IPersistStorage***

OLE Attack Vectors

Type-1: Loading a COM Component Not Intended for OLE

- CLSID used to index and load components.
- With thousands of CLSIDs existing in the system, only a subset corresponds to actual OLE components
- **Triggered by loading COM components not intended for OLE.**
- Existing checks in Office are inadequate, risking vulnerabilities like uninitialized reads.

Type-1 CVEs account for 3 out of the 21 surveyed CVEs.

Example: CVE-2015-1770

- Caused by improper initialization of ***OSF.DLL***.
- Despite having a CLSID, ***OSF.DLL*** is not meant to be an OLE component.
- Crash traced to uninitialized fields during component loading.

```
CoCreateInstance (
  rclsid =
   ↪  {CDDBCC7C-BE18-4A58-9CBF-D62A012272CE},
  pUnkOuter = NULL,
  dwClsContext = CLSCTX_INPROC_SERVER |
   ↪  CLSCTX_INPROC_HANDLER,
  riid =
   ↪  {00000000-0000-0000-C000-000000000046},
  ppv = lpTargetPpv
)
```

In CVE-2015-1770, a COM instance is created by calling ***CoCreateInstance***

OLE Attack Vectors

Type-2: DLL Preloading Attacks

OLE components load unauthenticated DLLs.

- If the DLL is requested while the Registry does not hold a complete path for the DLL, Windows searches for the required library according to a predefined sequence of directories (i.e., DLL search order).

- Variations in Windows installations can lead to path searching for DLLs, allowing malicious insertion.

Type-2 CVEs account for 11 out of the 21 surveyed CVEs.

Example: CVE-2023-35343

- In ***CoCreateInstance*** at ***Windows Geolocation Service***, the ***GetFindMyDeviceEnabled*** method is invoked:

- LibraryW = LoadLibraryW(L"mdmcommon.dll");

- ***mdmcommon.dll*** does not exist in Windows Server.

If there is a malicious ***mdmcommon.dll*** in the current directory, it could lead to RCE.

OLE Attack Vectors

Type-3: OLE Data Parsing Error in *IPersistStorage*

Data parsing is prone to vulnerabilities due to untrusted input.

- ***IPersistStorage::Load*** is responsible for loading objects stored within the storage section in the input document
- Many OLE-related vulnerabilities stem from ***IPersistStorage::Load***.

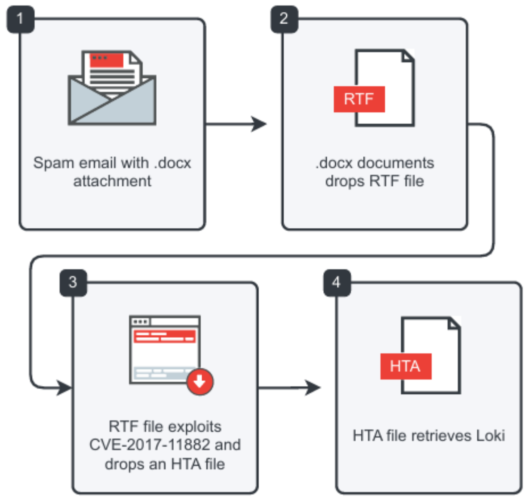
Security Implications:

- Systems must enforce strict data validation for storage data.
- Distinct methods of CLSID and storage data handling highlight differences between Type-1 and Type-3 vulnerabilities

Type-3 CVEs account for 7 out of the 21 surveyed CVEs.

Example: CVE-2017-11882

- Found in EQNEDT32.exe, used in Office for equation parsing.
- Vulnerability leads to stack overflow during font name parsing.
- Exploited via spear-phishing, allows execution of arbitrary code.

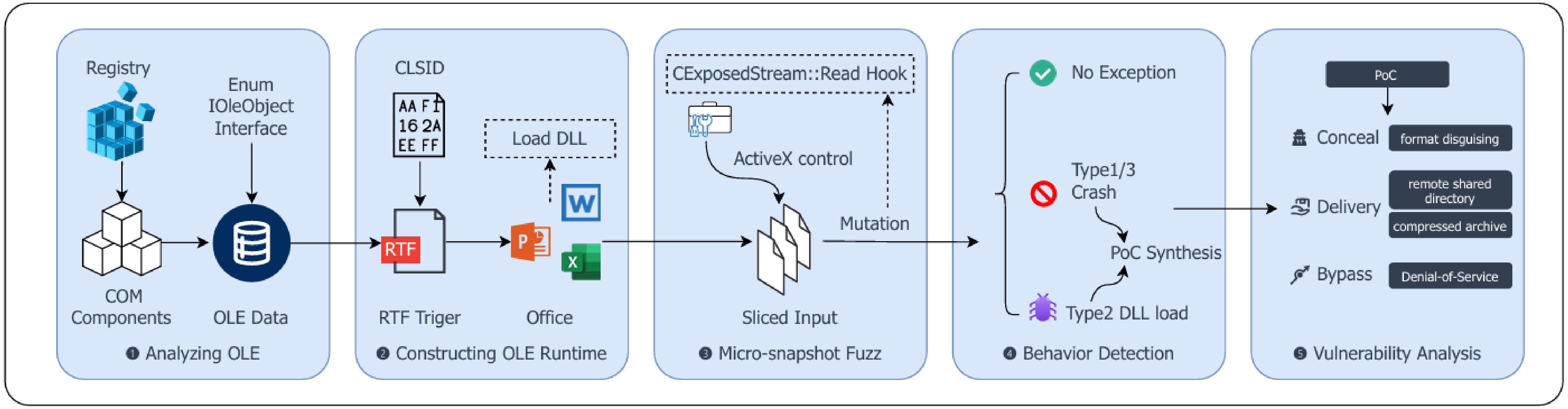

1
Spam email with .docx attachment
2
RTF
.docx documents drops RTF file
3
RTF file exploits CVE-2017-11882 and drops an HTA file
4
HTA
HTA file retrieves Loki

Overview

Facilitate the efficient and accurate detection of OLE-related vulnerabilities.

- Phase 1: Analyzing OLE components.
- Phase 2: Constructing an OLE runtime.
- Phase 3: Fuzz OLE-specific storage segment.
- Phase 4: Behavior Detection.
- Phase 5: Vulnerability Analysis.

OLEXPLORE: Auto Detection Framework



Analyzing OLE components

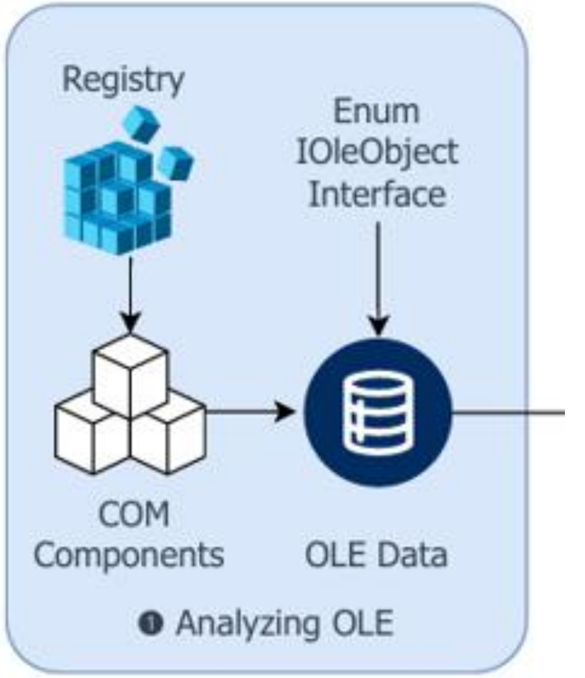
(for type-1)

To analyze OLE components, we first need to extract COM components and then analyze the OLE components from them.

1. **COM Component Collection:**
   Search Windows registry ***(HKEY_CLASSES_ROOT\CLSID)*** for CLSIDs.
   Get DLL locations from ***InprocServer32*** and ***LocalServer32*** sub-keys.

2. **OLE Component Identification:**
   Create objects from CLSIDs and check for ***IOleObject*** interfaces**.**

3. **Interface Analysis:**
   Use PowerShell's Get-Member to list interfaces and properties.
   OleViewDotNet is used to decompile and export interface declarations.



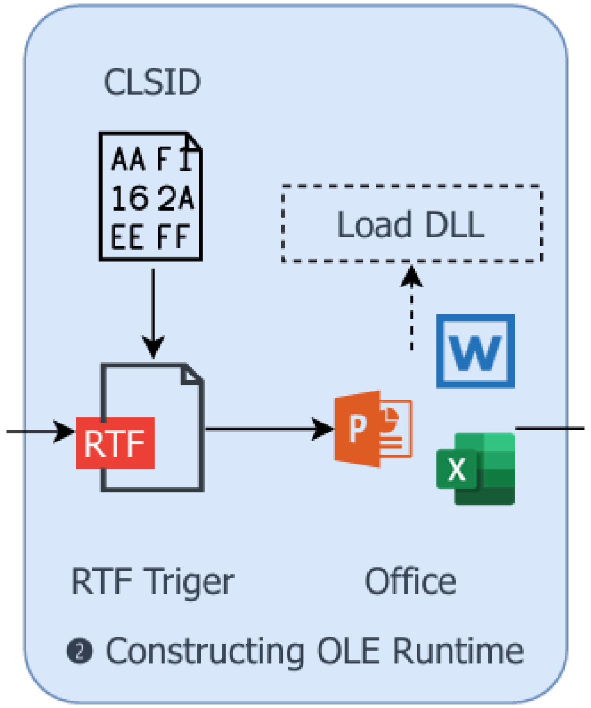
Constructing an OLE runtime

(for type-2)

We need to recreate the runtime environment for OLE components, but reverse engineering the GUI and Office applications themselves is challenging.

Solution:

- A ***misaligned feature*** enables loading and initializing OLE components without GUI interactions, eliminating the need for reverse engineering and simulating GUI interactions.
- Create RTFs for each OLE component using embedded CLSID.
- Simulate user actions to load OLE components and trigger initialization.



```
{\rtf1
{\object\objemb{\*\objclass None}
{\*\oleclsid\
'7b00000000-0000-0000-0000-000000000000\'7d}
{\*\objdata
 ↪  0105000001000000010000000000000000000000
0000000000000000000000000000
000000000000000000000}}}
```

An example of an RTF document that triggers the loading of an OLE object without requiring user interaction

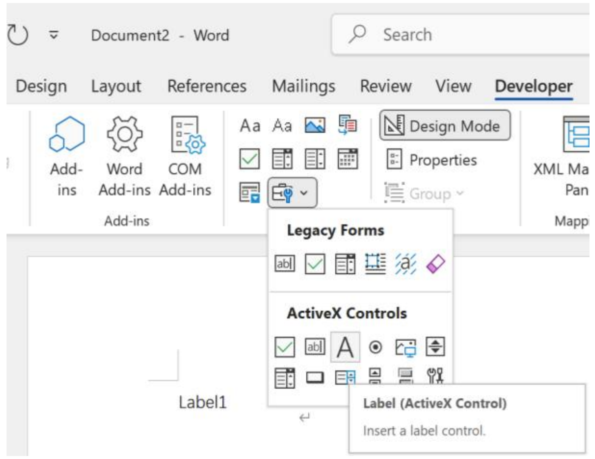
Fuzz OLE-specific storage segment

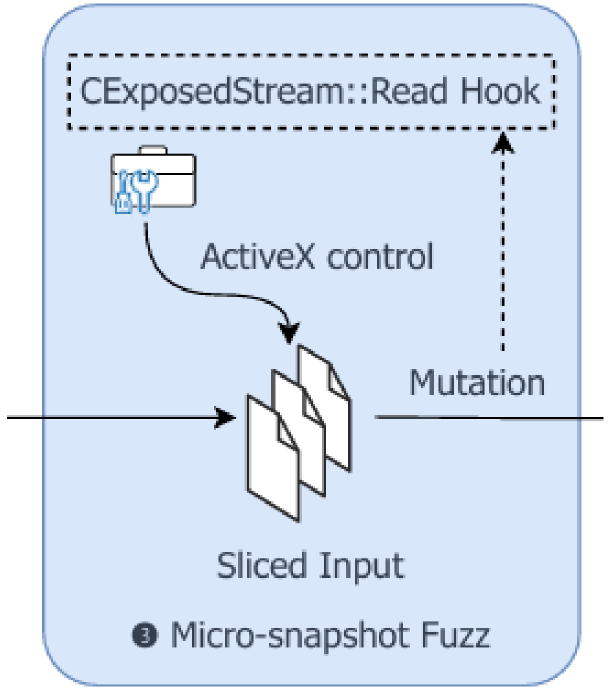
(for type-3)

We want to perform fuzz testing on OLE, but OLE components have a large number of structural validations, and ***each OLE component has its own format***. Randomly generated formats are difficult to pass these validations.

1) ActiveX-based input generation:
To build an initial corpus for ***known storage formats***, we developed 74 kinds of ***ActiveX controls*** . For example, the CheckBox Control with CLSID: 8BD21D40-EC42-11CE-9E0D-00AA006002F3 is related to the module ***FM20.DLL***. From the checkbox control, we can extract the binary file format required by the FM20.DLL component.

ActiveX controls, based on OLE's lower-level objects and interfaces, are embeddable OLE objects in Word or Excel files, offering interactive features.





Fuzz OLE-specific storage segment

2) Micro-snapshot fuzzing:
Fuzzing OLE components with ***unknown storage formats*** and no ActiveX support:

•**Naive Approach**: Treat input as a blob and apply random mutations, but this is often ineffective for structured formats.

•**Ideal Approach**: Reverse engineer input formats and use grammar-based generation for accurate mutations, though not practical due to many closed-source OLE components with little documentation.

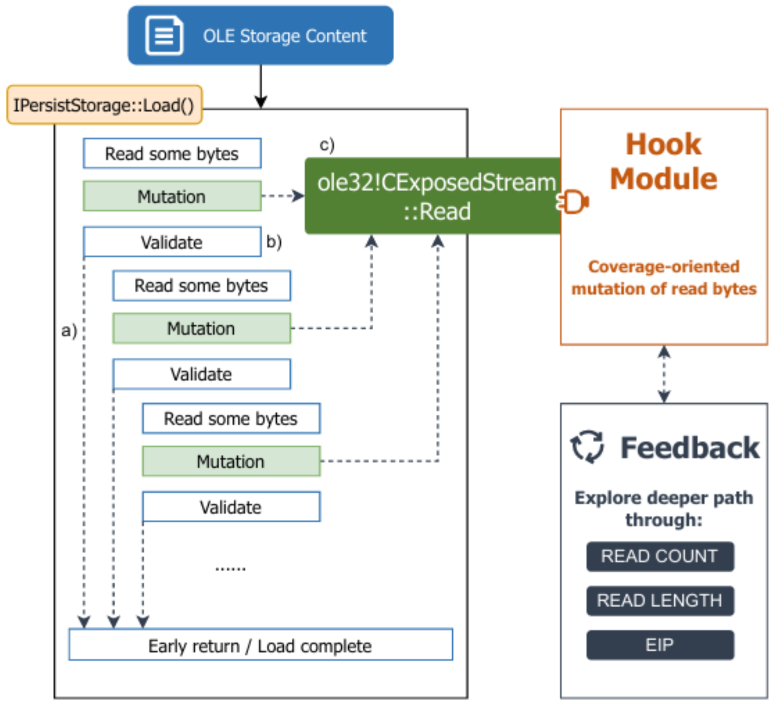
•**Our Solution**: For unknown formats, use micro-snapshot fuzzing, which divides the input blob into chunks, each corresponding to an event in snapshot fuzzing.

Fuzz OLE-specific storage segment

2) Micro-snapshot fuzzing:
Input Processing: Inputs are read chunk-by-chunk via the standard interface ***CExposedStream::Read*** and are not re-read. The second parameter of ***CExposedStream::Read*** is the buffer for storing data, the third is the amount to read, and the last is an integer pointer for the actual bytes read.

Snapshot Mechanism: If a chunk of input yields new coverage, OLEXPLORE takes a snapshot of the current process, which is restored in subsequent fuzzing to mutate the next chunk.

Systematic Exploration:
Base Case: Probe for one-chunk snapshots by mutating the chunk marked by the first ***CExposedStream::Read*** .
Inductive Case: For each k-chunk snapshot, resume the snapshot, generate, and check the (k + 1)-th chunk for new coverage. If new coverage is found, save it as a (k + 1)-chunk snapshot.



a) Repeated reading and checking during OLE storage retrieval.
b) Terminate reading if byte check fails.
c) Retrieving storage for the OLE component ultimately entails calling ole32!CExposedStream::Read.

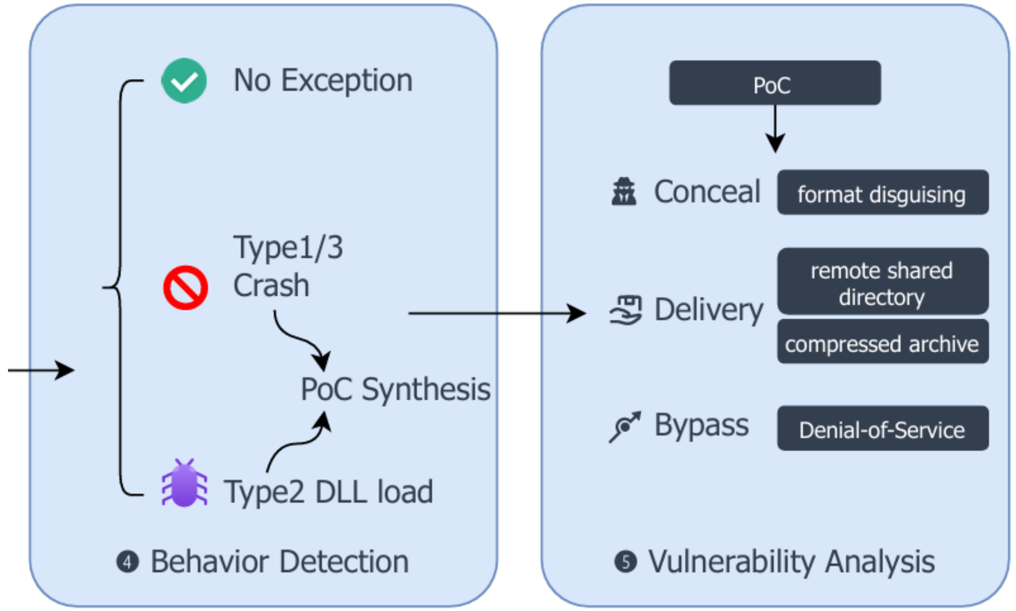
Behavior Detection & Vulnerability Analysis

Vulnerability Detection:

- Create mock DLLs to test priority in loading sequence.
- Use ***Process Monitor*** to track DLLs loaded by ***CoCreateInstance***.
- Detect unintended DLL loading using the ***LoadLibraryEx*** API.
- Enable ***PageHeap*** to detect memory errors and vulnerabilities.

Vulnerability Analysis :

- Use ***WinDbg*** to analyze crash dumps and identify vulnerabilities.
- Conduct tests in Protected View Mode to check security measures. Exploitation involves disguising RTFs, delivering DLLs, and bypassing Protected View.



Evaluation

Evaluation Setup:

- Tested on Windows 10, Server 2019, 2022, 2023, and Windows 11.
- Default OS settings with applications like Visual Studio and Microsoft Exchange.
- Desktop system: Intel i9-13900H, 32GB RAM.

Research Questions:

- RQ1: How effective are the most important components of OLEXPLORE (i.e., OLE identification and storage fuzzing)?
- RQ2: How effective is OLEXPLORE on detecting vulnerabilities within OLE components specific to Office?
- RQ3: How precise is OLEXPLORE in detecting unsafe OLE components?

Evaluation

RQ1: Evaluation of OLExplore’s Components

Identifying OLE components from COM components:

- Analyzed 7,361/7369 COM components on Windows 10/11.
- Prioritized Microsoft COM components for their widespread use.
- Examined 257 OLE objects.

Effectiveness of micro-snapshot fuzzing:

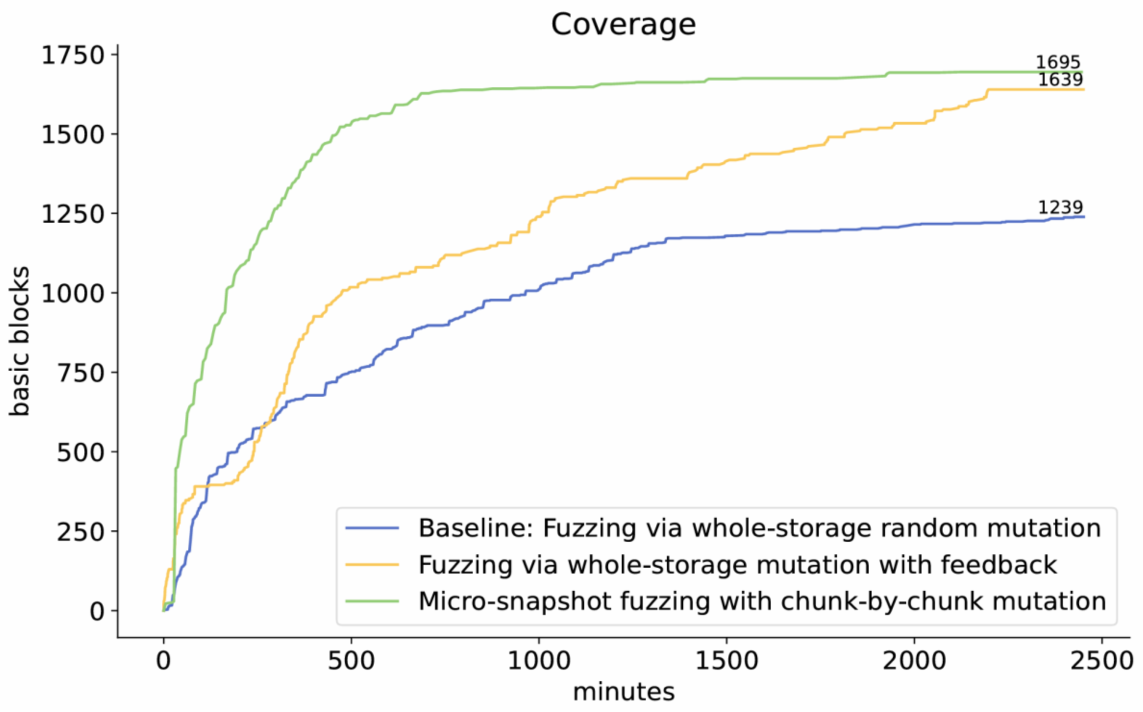
- With Micro-Snapshot Fuzzing
- Found 4 out of 7 identified Type-3 bugs in non-ActiveX items.
- Without Micro-Snapshot

Evaluation

RQ1: Evaluation of OLExplore's Components

Performance of micro-snapshot fuzzing:

- Micro-snapshot improves early-stage coverage via snapshotting.
- Execution Rate in ***inkobj.dll***:
  With snapshotting: 107 execs/s.
  Without snapshotting : 375 execs/s.
- Lower speed but higher coverage enhances vulnerability detection.

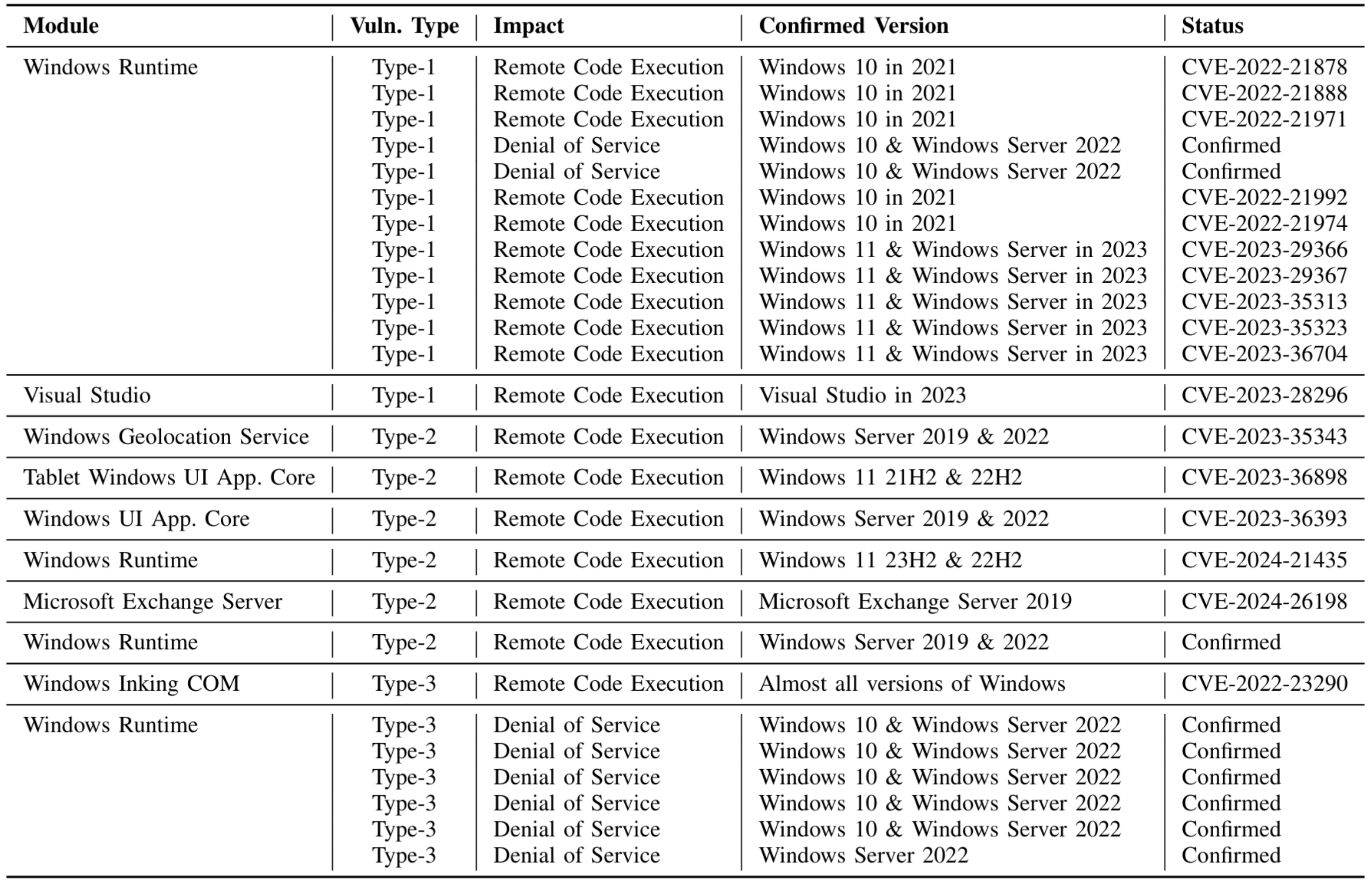


A 2500-minute testing period on inkobj.dll

Evaluation

RQ2: How effective is OLEXPLORE on detecting vulnerabilities within OLE components specific to Office?

- 26 vulnerabilities found, 17 of which have been assigned CVE IDs.
- 18 vulnerabilities listed are capable of being exploited for remote code execution.

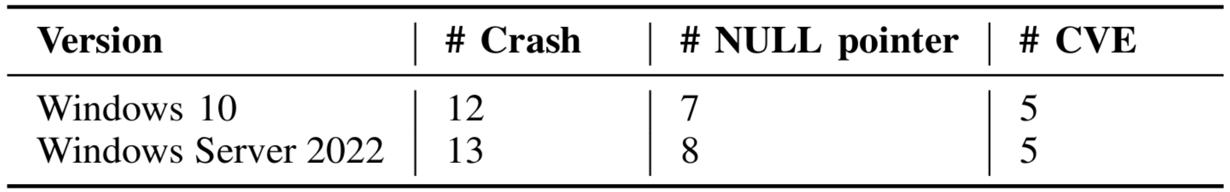


Module	Vuln. Type	Impact	Confirmed Version	Status
Windows Runtime	Type-1	Remote Code Execution	Windows 10 in 2021	CVE-2022-21878
	Type-1	Remote Code Execution	Windows 10 in 2021	CVE-2022-21888
	Type-1	Remote Code Execution	Windows 10 in 2021	CVE-2022-21971
	Type-1	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-1	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-1	Remote Code Execution	Windows 10 in 2021	CVE-2022-21992
	Type-1	Remote Code Execution	Windows 10 in 2021	CVE-2022-21974
	Type-1	Remote Code Execution	Windows 11 & Windows Server in 2023	CVE-2023-29366
	Type-1	Remote Code Execution	Windows 11 & Windows Server in 2023	CVE-2023-29367
	Type-1	Remote Code Execution	Windows 11 & Windows Server in 2023	CVE-2023-35313
	Type-1	Remote Code Execution	Windows 11 & Windows Server in 2023	CVE-2023-35323
	Type-1	Remote Code Execution	Windows 11 & Windows Server in 2023	CVE-2023-36704
Visual Studio	Type-1	Remote Code Execution	Visual Studio in 2023	CVE-2023-28296
Windows Geolocation Service	Type-2	Remote Code Execution	Windows Server 2019 & 2022	CVE-2023-35343
Tablet Windows UI App. Core	Type-2	Remote Code Execution	Windows 11 21H2 & 22H2	CVE-2023-36898
Windows UI App. Core	Type-2	Remote Code Execution	Windows Server 2019 & 2022	CVE-2023-36393
Windows Runtime	Type-2	Remote Code Execution	Windows 11 23H2 & 22H2	CVE-2024-21435
Microsoft Exchange Server	Type-2	Remote Code Execution	Microsoft Exchange Server 2019	CVE-2024-26198
Windows Runtime	Type-2	Remote Code Execution	Windows Server 2019 & 2022	Confirmed
Windows Inking COM	Type-3	Remote Code Execution	Almost all versions of Windows	CVE-2022-23290
Windows Runtime	Type-3	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-3	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-3	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-3	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-3	Denial of Service	Windows 10 & Windows Server 2022	Confirmed
	Type-3	Denial of Service	Windows Server 2022	Confirmed

Evaluation

RQ3: How precise is OLEXPLORE in detecting unsafe OLE components?

Type-1 and Type-3 vulnerabilities on Windows 10 version 10.0.19041.1237 yielded a total of 12 crash dump files. Out of these, 5 were confirmed as CVEs, while the remaining 7 were identified as null pointer issues upon further examination.



Version	# Crash	# NULL pointer	# CVE
Windows 10	12	7	5
Windows Server 2022	13	8	5

Similarly, on Windows Server 2022 version 10.0.20348.1487, the count of null pointer problems stands at 8.

The rationale for classifying these bugs as false positives rather than CVEs is rooted in their exploitability, or lack thereof, under modern Windows and Office mitigation mechanisms; such issues are substantially less likely to be leveraged for security attacks.

Conclusion

- We conducted an exhaustive survey on all publicly disclosed OLE-related vulnerabilities and delved into the fundamental causes of these flaws. Building on the outcomes of our investigation, we identified current attack surfaces for OLE and employed three vulnerability patterns to evaluate the exploitability of OLE.

- We developed OLEXPLORE, a pioneering tool for systematic vulnerability detection in OLE components. Novel techniques proposed in OLEXPLORE include: GUI-interaction bypassing, Micro-snapshot fuzzing, A vulnerability weaponization technique to bypass Office Protected View Mode.

- We systematically analyzed all registered COM components (a superset of OLE components) in popular Windows platforms, identified 257 OLE components, and reported 26 bugs. Out of those, 17 vulnerabilities have been assigned CVE numbers, each with the potential for Remote Code Execution.

Thank You ! Questions?

Backup Slides

Summary of Known Office Vulnerabilities

Exploited Office Vulnerabilities in the last 10 years

Observations:

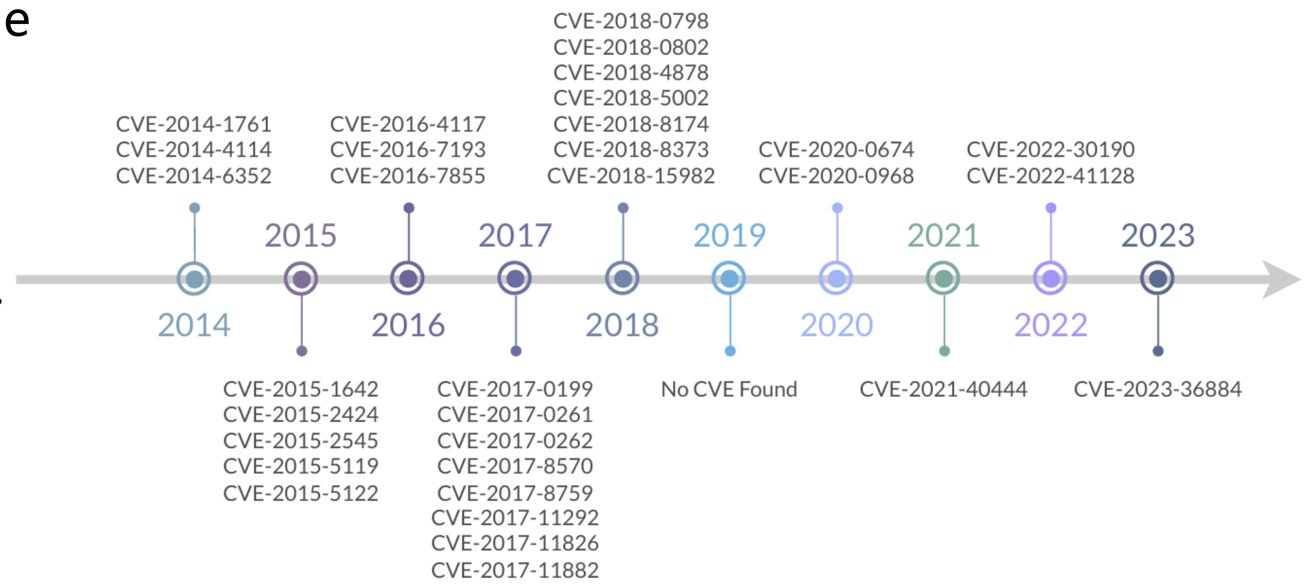
- Decrease in exploitable vulnerabilities over the last 10 years.

Reasons for Decline:

- Microsoft's effective patching.
- Many vulnerabilities reaching end of lifecycle.

Attackers need to find new attack surfaces.

- OLE vulnerabilities remain significant due to their high proportion.
- Essential to conduct security checks on OLE objects.



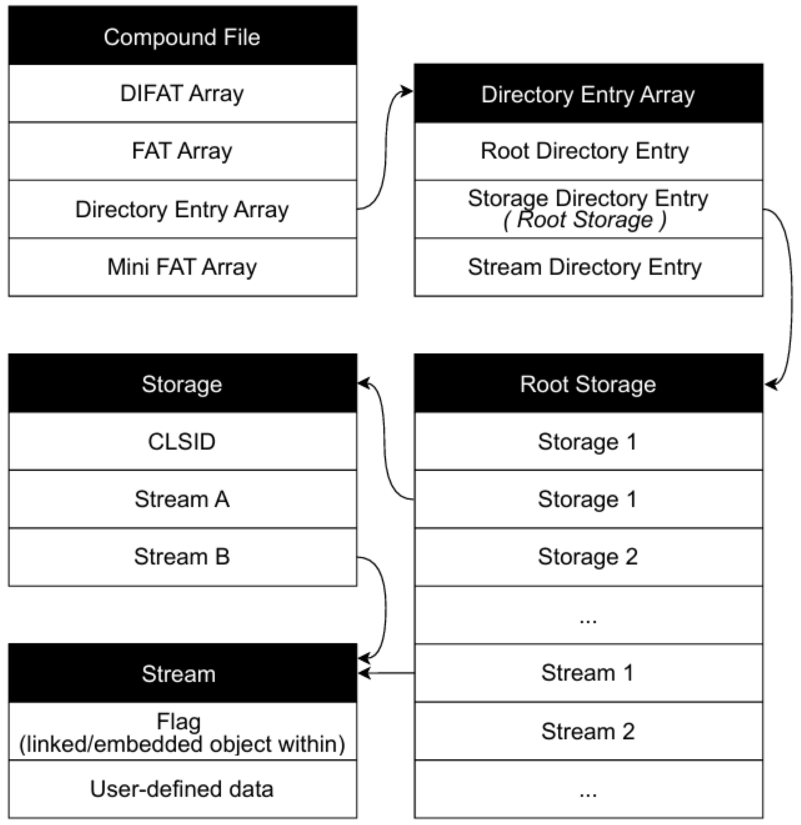
OLE Attack Vectors

Type-3: OLE Data Parsing Error in *IPersistStorage*

OLE Structured Storage:

- Stores heterogeneous data objects within a single file transparently.
- **Stream (IStream Interface):** Acts like traditional files with read/write methods.
- **Storage (IStorage Interface)**: Functions like directories, containing streams and other storages.
- In the past, data format variations required manual reverse engineering for fuzz testing.

Type-3 CVEs account for 7 out of the 21 surveyed CVEs.



A sketch of the format of a compound document with OLE objects embedded

Evaluation

RQ2: How effective is OLEXPLORE on detecting vulnerabilities within OLE components specific to Office?

Type-1 vulnerability example: CVE-2022-21971

```
void WapAuthProvider::~WapAuthProvider(__int64 this) {
  void *v2; // rcx
  void *v3; // rcx

  *(_QWORD *)this = &WapAuthProvider::`vftable';
  LocalFree(*(HLOCAL *)(this + 56));
  v2 = *(void **)(this + 64);
  *(_QWORD *)(this + 56) = 0i64;
  LocalFree(v2);
  v3 = *(void **)(this + 80);        // <-- [0] uninitialized
  *(_QWORD *)(this + 64) = 0i64;
  LocalFree(v3);                     // <-- [1] free
  *(_QWORD *)(this + 80) = 0i64;
}
```

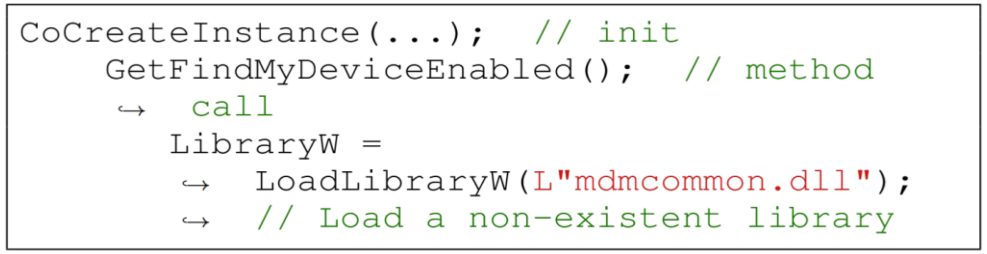
Uninitialized Pointer Dereference:

In the destructor **WapAuthProvider::~WapAuthProvider**, a pointer at offset 0x50 is freed without initialization, leading to potential remote arbitrary code execution.

Evaluation

RQ2: How effective is OLEXPLORE on detecting vulnerabilities within OLE components specific to Office?

Type-2 vulnerability example: CVE-2023-35343



```
CoCreateInstance(...);  // init
    GetFindMyDeviceEnabled();  // method
     ↪  call
       LibraryW =
        ↪  LoadLibraryW(L"mdmcommon.dll");
        ↪  // Load a non-existent library
```

Missing Library Loading:

The system attempts to load ***mdmcommon.dll*** via ***LoadLibraryW***, but the file is absent in Windows Server environments.

Attackers can exploit this by placing a malicious DLL in the target directory for remote code execution.

Evaluation

RQ2: How effective is OLEXPLORE on detecting vulnerabilities within OLE components specific to Office?

Type-3 vulnerability example: CVE-2022-23290

```
HeapAlloc(*(HANDLE *)Default, *((_DWORD *)Default + 2), 0x70);
...
v6 = *(void **)(this+0x10); // Uninitialized pointer
HeapFree(*(HANDLE *)Default, *((_DWORD *)Default + 2), v6);

mov rdi, qword ptr [rax+10h]
ds:00000158`42fcbfa0=c0c0c0c0c0c0c0c0
```

Memory Corruption Due to Partial Initialization:

In ***CSketchInk::FreeStrokeList***, an uninitialized pointer causes memory corruption when accessing allocated memory, indicated by abnormal values (c0c0c0c0c0c0c0c0).