Pig in a Poke: Automatically Detecting and Exploiting Link Following Vulnerabilities in Windows File Operations

Bocheng Xiang, Yuan Zhang, Fengyu Liu, Hao Huang, Zihan Lin, Min Yang

Fudan University

Background

- Symbolic links are widely used in file operations on the Windows system.
  - e.g., Shortcuts, Directory junctions etc.
- Two primary types of non-privileged symbolic links exist in Windows
  - Directory Junctions[1] (aka. Mount Points): enable the linking of one directory to another target directory
  - Object Manager[2] Symlinks: operate at the kernel object level and can redirect access to devices, named pipes
- Opportunistic Lock[3] (Oplocks)
  - Oplocks are synchronization primitives in Windows that delay file operations for performance

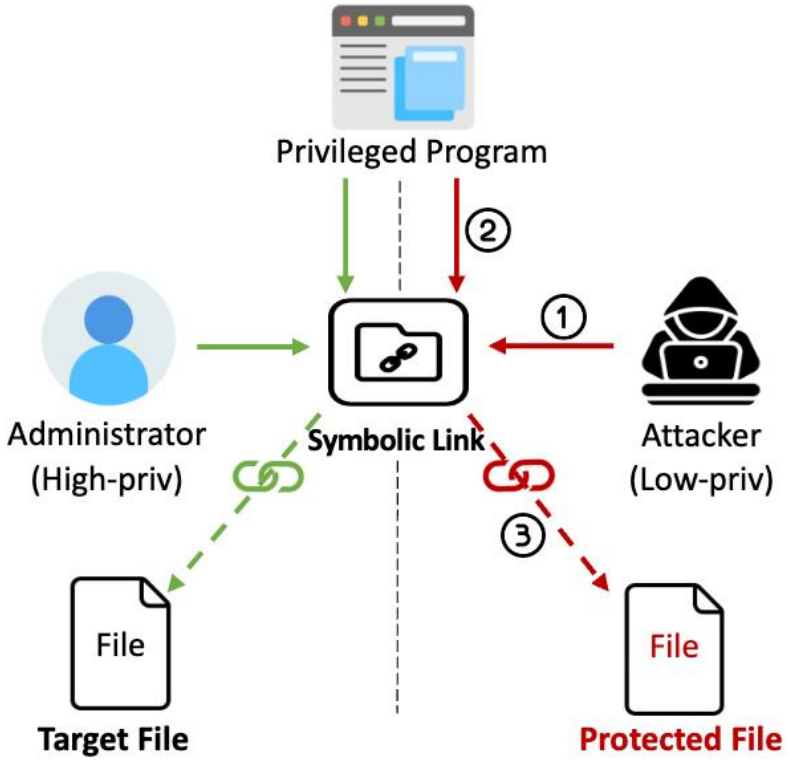
[1] https://learn.microsoft.com/en-us/windows/win32/fileio/hard-links-and-junctions

[2] https://en.wikipedia.org/wiki/Object_Manager

[3] https://learn.microsoft.com/en-us/windows/win32/fileio/opportunistic-locks

Link-Following Vulnerability (LFVuln)

Low-privileged attackers can exploit crafted symbolic links to redirect high-privileged programs' file operations to protected files



Attack Workflow

① Attacker creates malicious symbolic links to a protected file

② Privileged program follows the link without validation

③ Protected file is accessed or modified with elevated privileges

A Real-World LFVuln in Detection and Exploitation

Privileged Program Vulnerable Code

```
1  void IPC_CleanTempFile() {
     // Operates on an attacker-controlled directory
2    wchar_t * dirPath = GetTempFilePath();
3    sprintf(searchPath, L"%s\\*.tmp", dirPath);
     // Locate .tmp files in the specified directory
4    hFind = FindFirstFile(searchPath, &fdata);
5    do {
6      wchar_t filePath[MAX_PATH];
7      PathCombine(filePath, dirPath, fdata.cFileName);
       /** Remove located temporary files */
8      DeleteFile(filePath);
9    } while (FindNextFile(hFind, &fdata));
10 }
```

Attacker Exploit Code

```
1  void Exploit() {
2    HANDLE hFile = CreateFile(L"attack.tmp");
3    Oplock oplock = new Oplock(hFile);
     // Use Oplock to create a time window
4    oplock->SetOplock(Trigger_Callback, hFile);
5    oplock->Release();
6  }
7  void Trigger_Callback(HANDLE hFile){
8    wchar_t* dirPath = GetFileDir(hFile);
9    wchar_t* symPath = L"\\RPC Control\\attack.tmp"
10   CreateJunction(dirPath, L"\\RPC Control");
11   CreateObjSymlink(symPath, victim_file);
12 }
```

- **Discription:**

The privileged program deletes files without strict validation, enabling arbitrary file deletion and local privilege escalation[1]

- **Detection:**

Detecting the vulnerability requires the presence of a *.tmp* file in a designated directory

- **Exploitation:**

Successful Exploitation depends on a precise file operation order, ensuring symbolic links is in place exactly between file enumeration and deletion

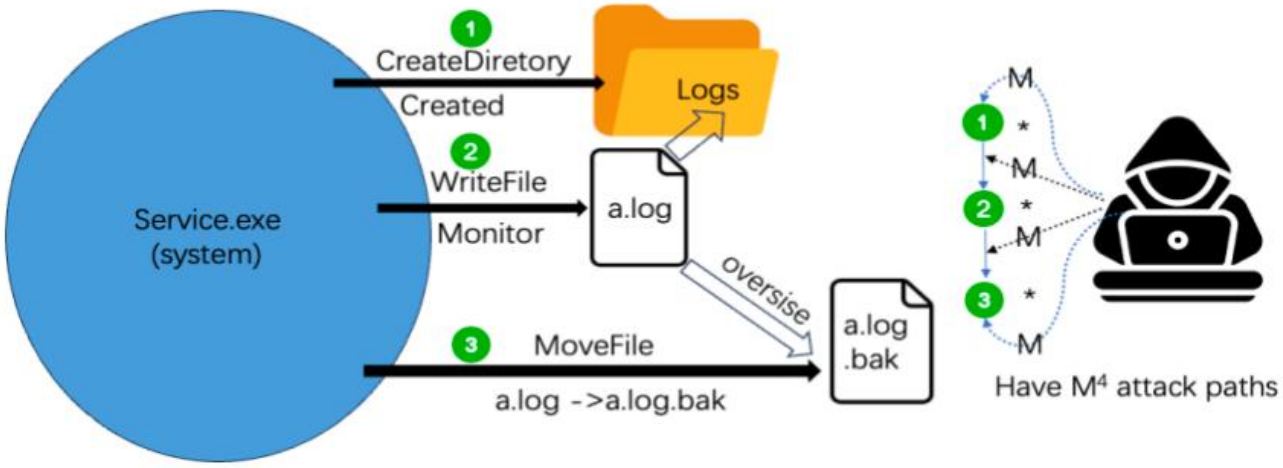
[1] https://www.zerodayinitiative.com/blog/2022/3/16/abusing-arbitrary-file-deletes-to-escalate-privilege-and-other-great-tricks

Empirical Study

- We conduct a comprehensive analysis of **145** LFVulns from the past five years
- **Q1: What is the root cause of LFVulns**
  - LFVulns stem from **improper symbolic link validation** in file operations.
- **Q2: What are the types of dangerous file operations that lead to LFVulns**
  - 1) Unsafe Creation, Write, and Overwriting (49, **33.8%**)
  - 2) Unsafe Copying and Moving (14, **9.7%**)
  - 3) Unsafe Access Control Configuration (14, **9.7%**)
  - 4) Unsafe Deletion (68, **46.8%**)
- **Q3: What factors hinder the detection and exploitation of LFVulns**
  - **File state constraints** are the primary obstacle to automating LFVulns detection and exploitation, affecting ***46% (66/145)*** of analyzed cases.

Automatically Detection and Exploitation Challenges

- **Challenge I:** Effectively solve file state constraints
  - Existing tools (e.g., PeachFuzz, StaticFAC) either suffer from static limitations (e.g., path explosion) or lack dynamic feedback on file states.
  - **No automated solution** effectively addresses this major barrier in LF vulnerability detection.
- **Challenge II:** Automate the exploitation of LFVulns
  -) **Precisely locating the point** in the file operation sequence for inserting attacker actions is non-trivial and incurs exponential complexity if exhaustively explored

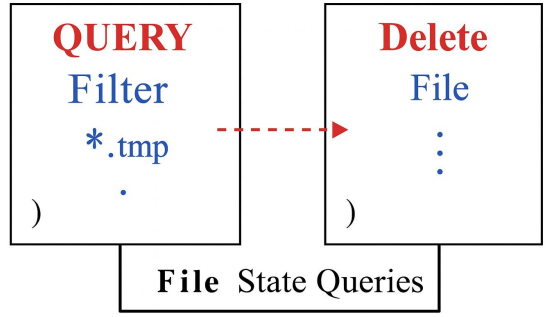


With M available file operations, the attacker has up to M^4 exploitation paths!

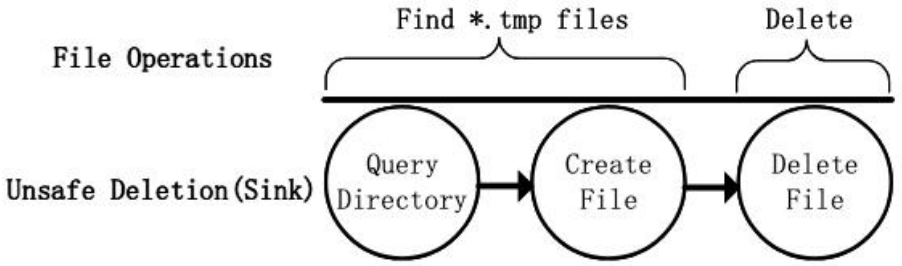
Our Idea: Automate detection and exploitation by separating the exploration and exploitation phases

- ***Key Observations***

1. File operations are often preceded by ***file state queries***, revealing implicit file state constraints.
2. Sinks are structurally *smaller and finite*, making them identifiable through subgraph matching within the larger file operation graph.



File State Queries before File Operations

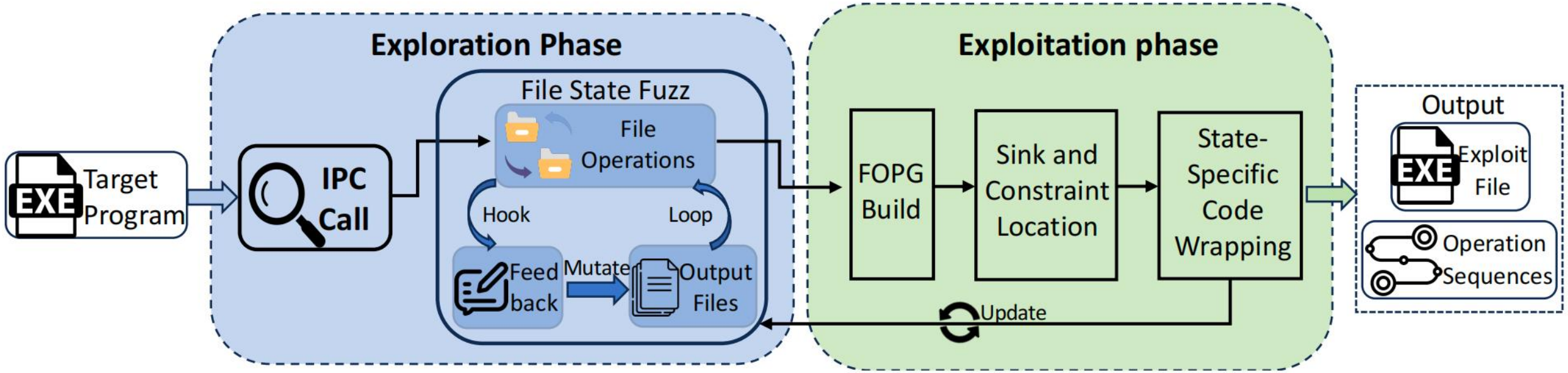


One Sink of Unsafe Deletion

LinkZard Overall Architecture

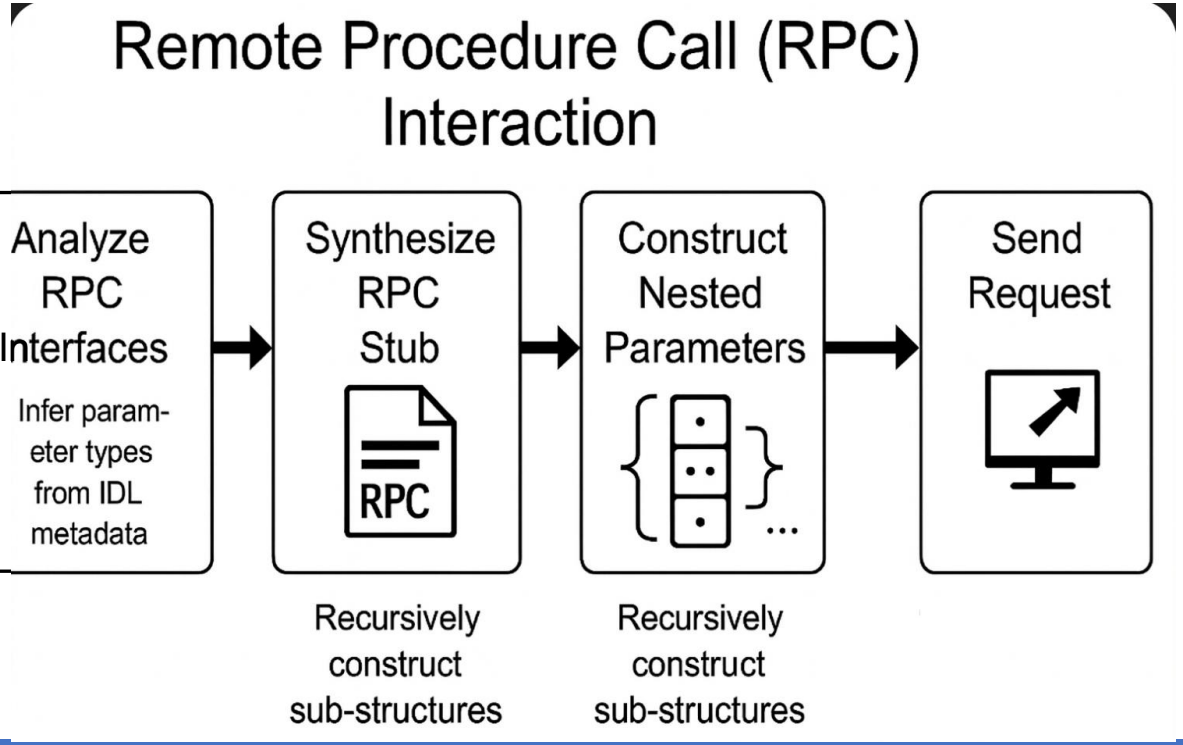
- **Phase I: Exploration**
  1. IPC Call
  2. Feedback-driven File State Fuzz

- **Phase II: Exploitation**
  1. *File Operation Primitive Graph* (FOPG) build
  2. Sink and Constraint Location
  3. State-Specific Code Wrapping



Stage I: Exploration

1. Exploration of LFVuln
   - Step 1: IPC Call
     - **Systematically mutates input parameters of privileged interfaces to explore diverse file operation paths**



- Extract RPC interfaces from *IDL* metadata
- Generate parameter values
- Recursively construct nested structures
- Invoke privileged interfaces to trigger file operations

Stage I: Exploration

2. File State Fuzz

- ***Goal:*** **Infer and solve file state constraints** to trigger LFVulns during privileged file operations
- Step 1: Instrument File State Queries
  - Instrument state query APIs to capture constraints such as *name patterns* or *file size*.
- Step 2: Fuzz & Mutate File States
  - Apply diverse mutation strategies to generate varied *query-relevant file states* and record the file operations
- Step 3: Validate Constraint Solving
  - Determine whether the inferred file state constraints are successfully solved by checking if *both the types and number of file operations increase* after file state mutations.

Stage II: Exploitation

1. *File Operation Primitive Graph* (FOPG) build
   - Construct the graph from ***exploration*** phase that models the sequence and dependency of file operation primitives to capture potential LFVulns.
2. Sink and Constraint Location
   - **Sink Location:** Identify vulnerable sink graph via subgraph matching on the FOPG
   - **File State Constraint Location:** determine whether associated file state constraints are **pre-sink** (required before the sink) or **on-sink** (required at the sink point for exploitation)
3. State-Specific Code Wrapping
   - **Based on the constraint type** (pre-sink or on-sink), LinkZard assemble tailored exploitation code.

Evaluation

- Dataset
  - **42** known LFVulns applications
    - Successfully reproduced described in our empirical study
  - **120** commercial closed-source programs
    - All programs having over **50**K downloads
- Result Overview
  - LinkZard achieves a **90.48**% detection recall and an **86.84**% exploitation success rate
  - **55** 0day vulnerabilities across **49** program
  - **15** assigned CVE identifiers

Baselines	Detection					Exploitation
	TP	FP	FN	Prec (%)	Recall (%)	Success Rate (%)
Jerry-Ext	24	10	18	70.59%	57.14%	/
LinkZard	38	0	4	100.00%	90.48%	86.84% (33/38)

RQ1: Effective

Baselines	TP	FP	FN	Prec (%)	Recall (%)
$\text{LinkZard}_{\text{NF}}$	19	0	23	100.00	45.24
$\text{LinkZard}_{\text{NS}}$	38	10	4	79.17	90.48
LinkZard	38	0	4	100.00	90.48

RQ3: Ablation Study

Evaluation

RQ2: 0days in commercial closed-source programs

#	Software Name	Vendor	Download	Sec. Risk	Status
1	WeCom	Tencent	250M	Dos	Fixed
2	Tencent Meeting	Tencent	200M	Dos	Fixed
3	WeChat Input	Tencent	20M	Dos	Fixed
4	Foxit PDF	Foxit Software	9.8M	LPE	CVE-2024-38***
5	Foxit PDF	Foxit Software	9.8M	Dos	Fixed
6	Adobe *** Software	Adobe	7.6M	Dos	Confirmed
7	W*** Software	Elastic	3.2M	LPE	Confirmed
8	Microsoft PC Manager	Microsoft	3.1M	LPE	CVE-2024-49***
9	Microsoft ***	Microsoft	3.1M	Dos	Confirmed
10	iTunes for Windows	Apple	2.7M	LPE	CVE-2024-44***
11	H*** Software	Elastic	2.5M	LPE	Confirmed
12	Microsoft OfficePlus	Microsoft	2.4M	LPE	CVE-2024-38***
13	H*** Software	Elastic	2.4M	LPE	Confirmed
14	M*** Software	Elastic	1.7M	LPE	Confirmed
15	Azure Software 1	Microsoft	1.7M	LPE	CVE-2024-38***
16	Azure Software 2	Microsoft	1.7M	LPE	Fixed
17	F*** Software	Elastic	1.5M	LPE	Confirmed
18	V*** Software	Veeam	1.3M	LPE	Confirmed
19	Sonos Controller S2	Sonos	1.3M	Dos	Confirmed
20	O*** Software	ManageEngine	1.0M	LPE	CVE-2024-98**
21	E*** Software 1	ManageEngine	1.0M	LPE	Fixed
22	E*** Software 2	ManageEngine	1.0M	LPE	Fixed
23	A*** Software	ManageEngine	1.0M	LPE	Fixed
24	P*** Software	Trend Micro	635K	LPE	Reported
25	T*** Software	Trend Micro	594K	Dos	Reported
26	MalwareBytes Adwcleaner	MalwareBytes	521K	LPE	Confirmed
27	TeamCity	JetBrains	483K	LPE	CVE-2024-43***

- **25** have been acknowledged with **$39,600** in bug bounty rewards
- The discovered vulnerabilities span a wide spectrum of high-impact software ecosystems, including:
  - Popular commercial applications with over 5 million downloads, such as Tencent Meeting, Enterprise WeChat, Foxit PDF
  - Critical system components, including default **Windows** services (e.g., *Windows Image Acquisition*) and **Intel** driver-level middleware
  - Well-known vendors, such as **Microsoft**, **Apple**, **Intel**, **JetBrains**, **VMware**, and **Tencent**, all exhibited multiple previously unknown LF vulnerabilities

Conclusion

- We present the first empirical study and threat model of LF vulnerabilities, revealing their **root causes** and characteristics.
- Guided by key insights, we design LinkZard, **an automated system that detects and exploits LFVulns** without manual effort.
- LinkZard uncovered **55** zero-day LF vulnerabilities in **120** real-world applications, with **15** CVEs assigned and **$39,600** in rewards received.

Thanks for listening to our presentation!

For more details, welcome to follow our paper.

Pig in a Poke: Automatically Detecting and Exploiting Link Following Vulnerabilities in Windows File Operations

Bocheng Xiang, Yuan Zhang, Fengyu Liu, Hao Huang, Zihan Lin, Min Yang
Fudan University, China

Any Question:

bcxiang23@m.fudan.edu.cn