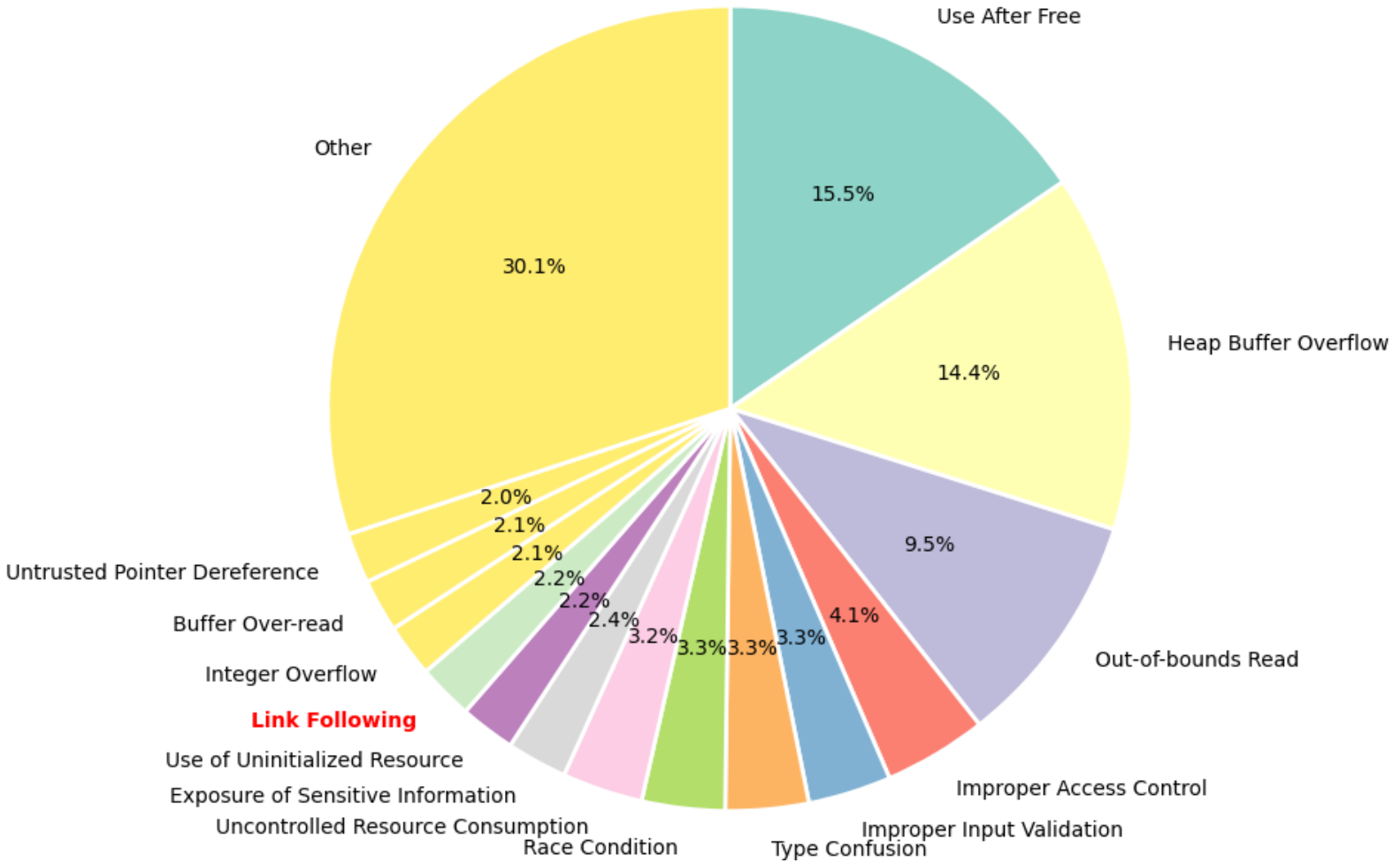
Windows plays Jenga: Uncovering Design Weaknesses in Windows File System Security

Dong-uk Kim*, Junyoung Park*, Sanghak Oh
Hyoungshick Kim, Insu Yun

ACM CCS
October 17, 2025

Vulnerability Statistics of Microsoft's Products (2025)



Since 2025, link following vulnerabilities have been reporting more frequently than **integer overflow**.

In-the-wild Exploitation

Windows Storage Elevation of Privilege Vulnerability

CVE-2025-21391
Security Vulnerability

Released: Feb 11, 2025

Assigning CNA: Microsoft

CVE.org link: CVE-2025-21391

Impact: Elevation of Privilege **Max Severity:** Important

Weakness: CWE-59: Improper Link Resolution Before File Access ('Link Following')

CVSS Source: Microsoft

Vector String: `CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:H/E:F/RL:O/RC:C`

Metrics: CVSS:3.1 7.1 / 6.6

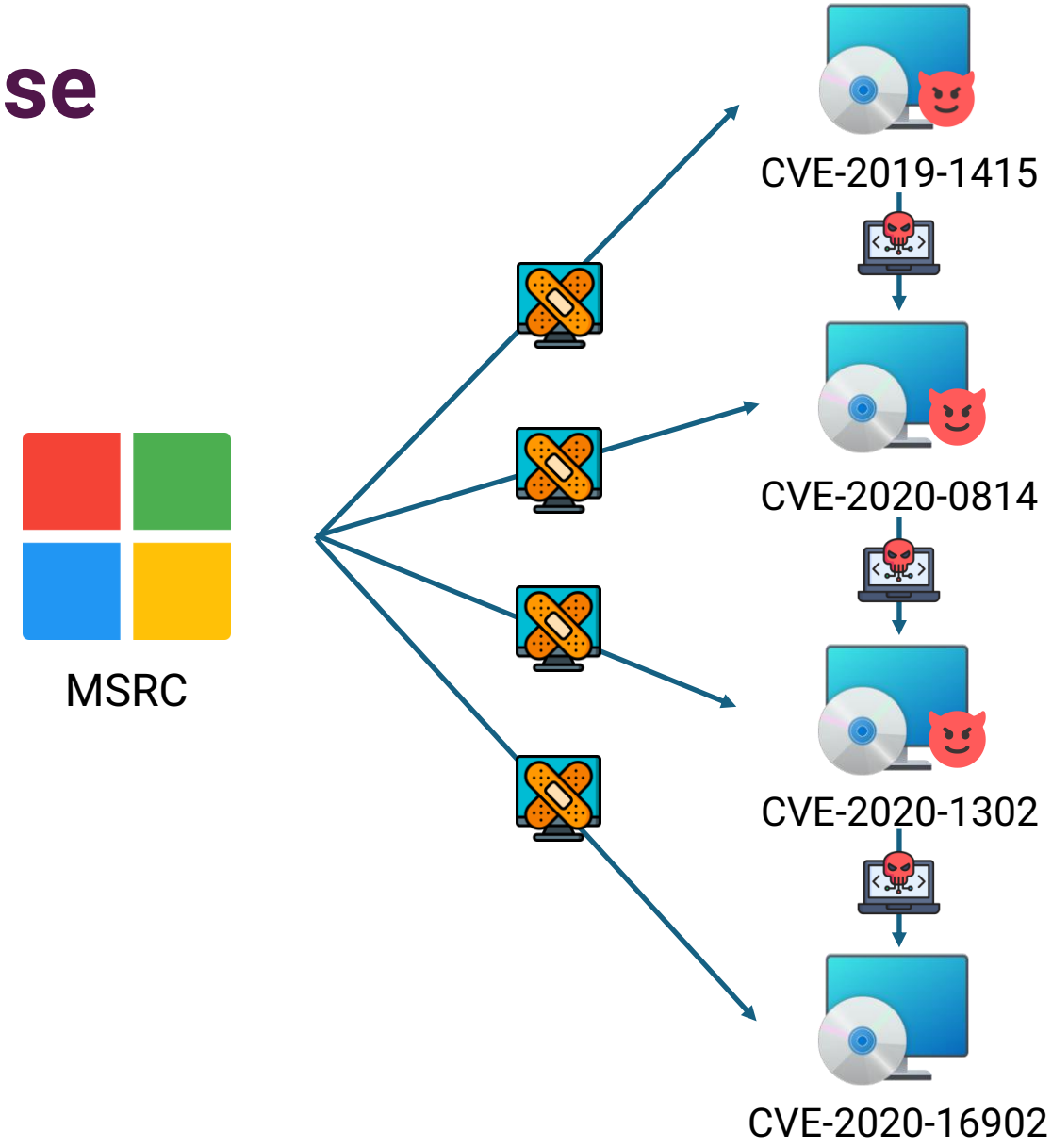
Exploitability

The following table provides an exploitability assessment for this vulnerability at the time of original publication.

Publicly disclosed
No
Exploited
Yes
Exploitability assessment
Exploitation Detected

https://msrc.microsoft.com/update-guide/vulnerability/CVE-2025-21391

Patch Case



: Windows Installer Service

: Vulnerability Patch

: Patch Bypass

https://ssd-disclosure.com/ssd-advisory-windows-installer-elevation-of-privileges-vulnerability/

Motivation

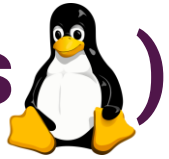
Why do Windows file system vulnerabilities continue to occur again and again?

Research Questions

- **RQ1**: How are Windows and other OS(Linux) file system designed differently?
  - **A1:** Windows and Linux have **four key discrepancies** in file systems.
    - 1) Link traversal with JDSymlink
    - 2) Advisory and mandatory locks
    - 3) Permission inheritance
    - 4) Permission inversion

Research Questions

- **RQ1**: How are Windows and other OS(Linux) file system designed differently?
  - **A1:** Windows and Linux have **four key discrepancies** in file systems.
    - 1) Link traversal with JDSymlink
    - 2) Advisory and mandatory locks
    - 3) Permission inheritance
    - 4) Permission inversion

Discrepancies of File System ( vs )

- File Link Traversal
- **File Locking Mechanism***
- File Permissions
  - Permission Inheritance
  - **Permission Inversion***

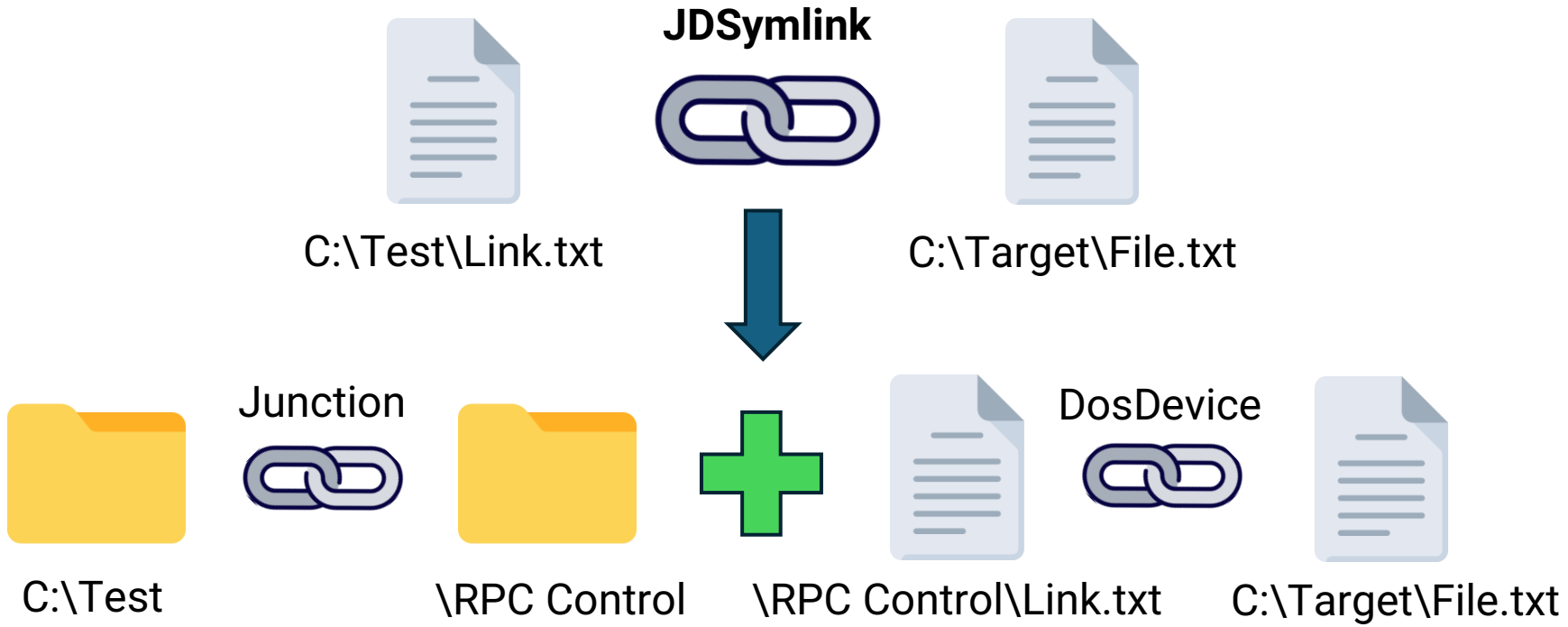
***: Newly identified in our work**

Links in Windows

- **NTFS Symbolic Link**
  - Similar to **Linux symbolic links** in functionality.
  - Windows requires **Administrator privileges** to create symbolic links.
- **NTFS Junction**
  - Work similarly **hard links**.
  - Only reference **directories**, **not files**.
- **DosDevice**
  - A special type of symbolic link that maps a simpler name to a device or file path.
  - For example, the name **C:** is assigned to a device **\Device\HarddiskVolumeX**.

Links in Windows

- **JDSymlink**
  - Combination with a **NTFS Junction** and **DosDevice.**

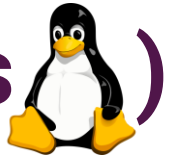


Link Traversal Behaviors (vs)

Operation	Linux		Windows		JDSymlink
	System calls/flags	Linux symbolic link	Windows APIs/flags	NTFS symbolic link	
open	`open` [39]	✓	`CreateFile` [55]	✓	✓
open (w/ a flag)	`O_NOFOLLOW` [39]	✗	`FILE_FLAG_OPEN_REPARSE_POINT` [55]	✗	✓
remove	`remove` [40]	✗	`DeleteFile` [58]	✗	✓
rename	`rename` [41]	✗	`MoveFile` [56]	✗	✓

✓/✗: follows/does not follow links.

Most file-related Windows APIs follows **JDSymlink**.

Discrepancies of File System ( vs )

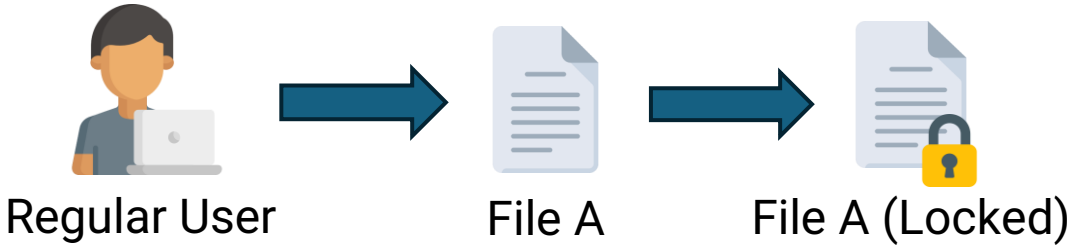
- File Link Traversal
- **File Locking Mechanism***
- File Permissions
  - Permission Inheritance
  - **Permission Inversion***

***: Newly identified in our work**

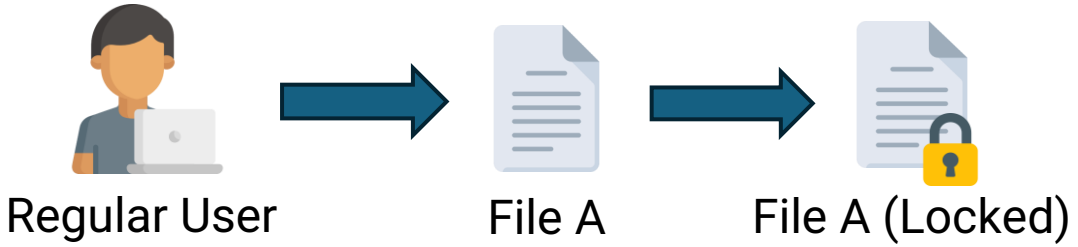
Advisory vs Mandatory Locks

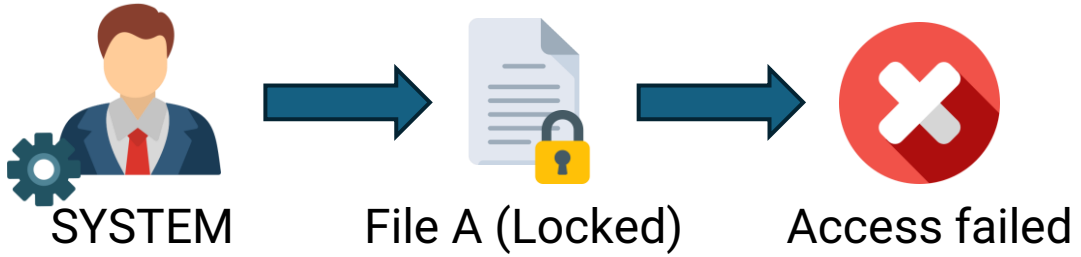
- **Advisory Locks**
  - Require processes to **voluntarily use** the locking mechanism.
  - Must explicitly call functions to acquire and release the locks.
- **Mandatory Locks**
  - **Enforce** the locking mechanism at the kernel level.
  - Denying access to locked files even for processes that do not explicitly use the locking mechanism.

Example of Lock Behaviors (vs)

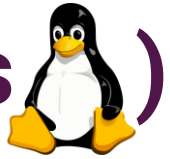


Advisory Locks ( )





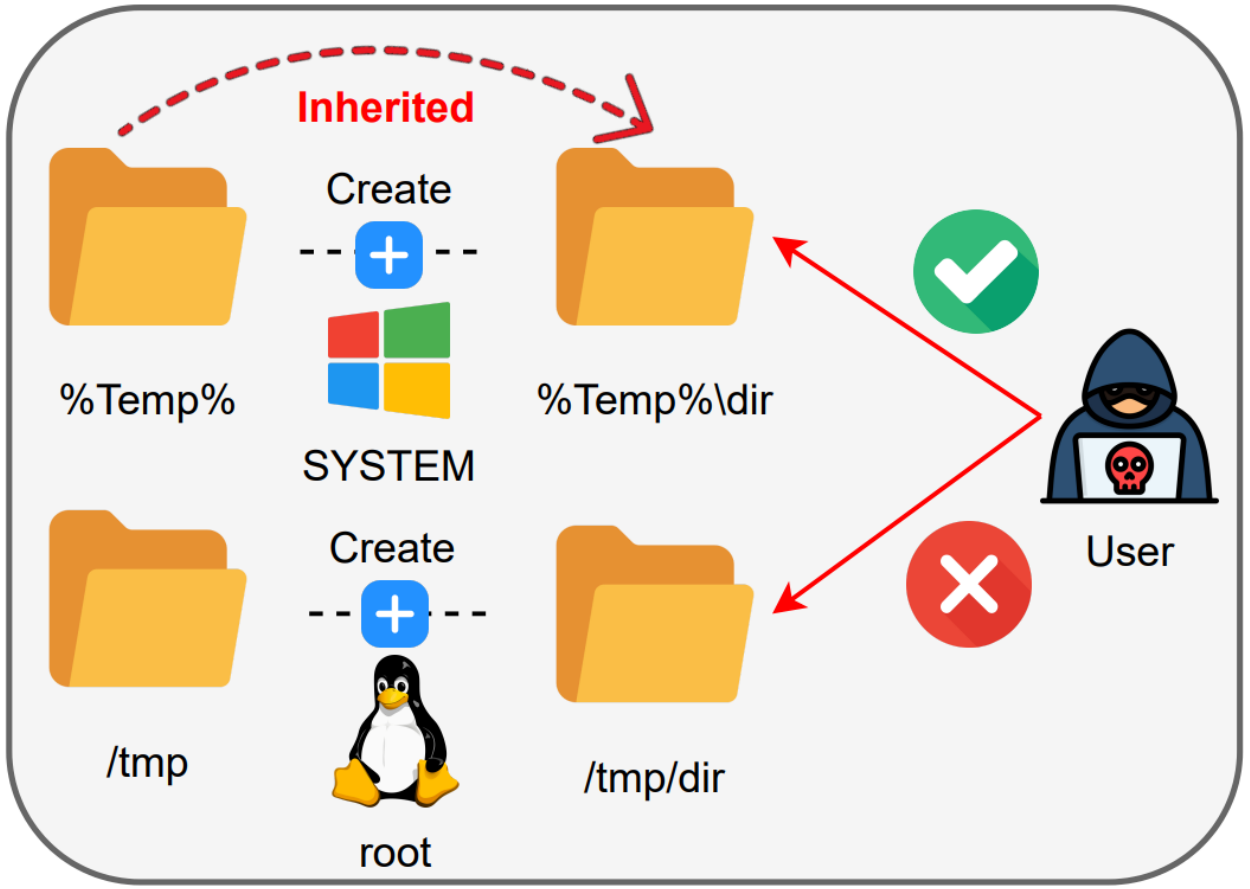
Mandatory Locks ( )

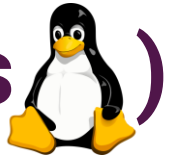
Discrepancies of File System ( vs )

- File Link Traversal
- **File Locking Mechanism***
- File Permissions
  - Permission Inheritance
  - **Permission Inversion***

***: Newly identified in our work**

File Permission Inheritance (vs)

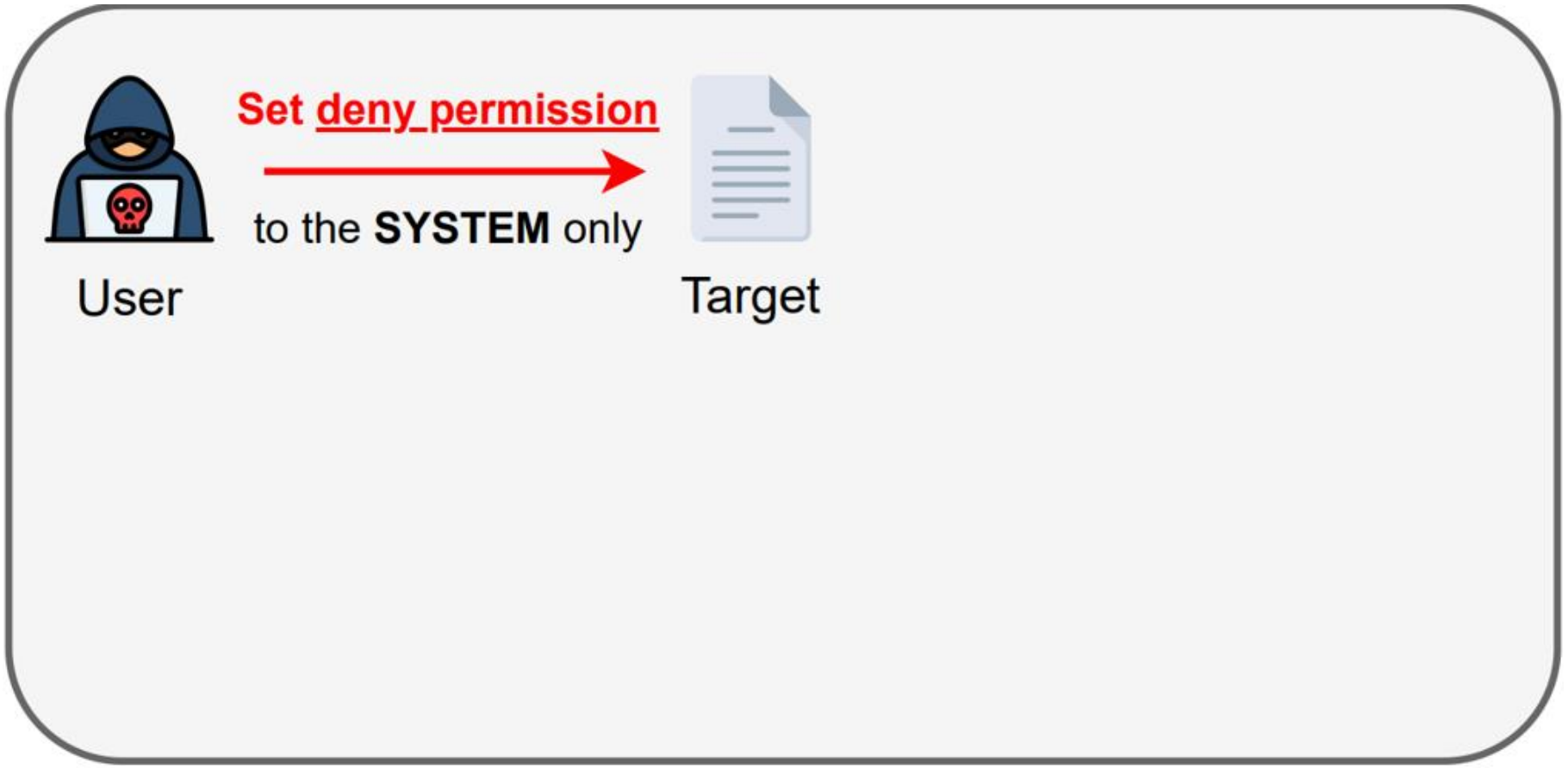


Discrepancies of File System ( vs )

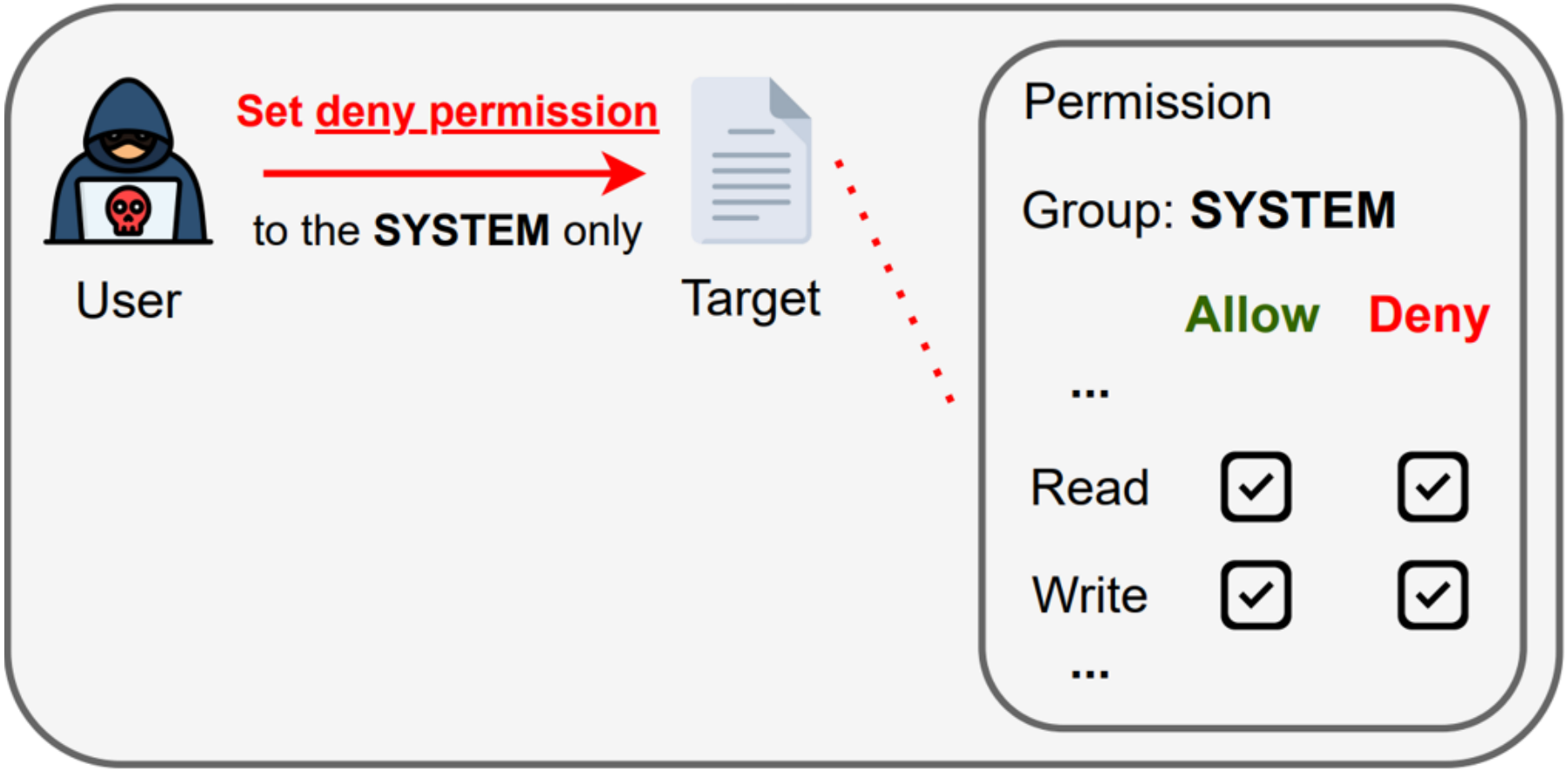
- File Link Traversal
- **File Locking Mechanism***
- File Permissions
  - Permission Inheritance
  - **Permission Inversion***

***: Newly identified in our work**

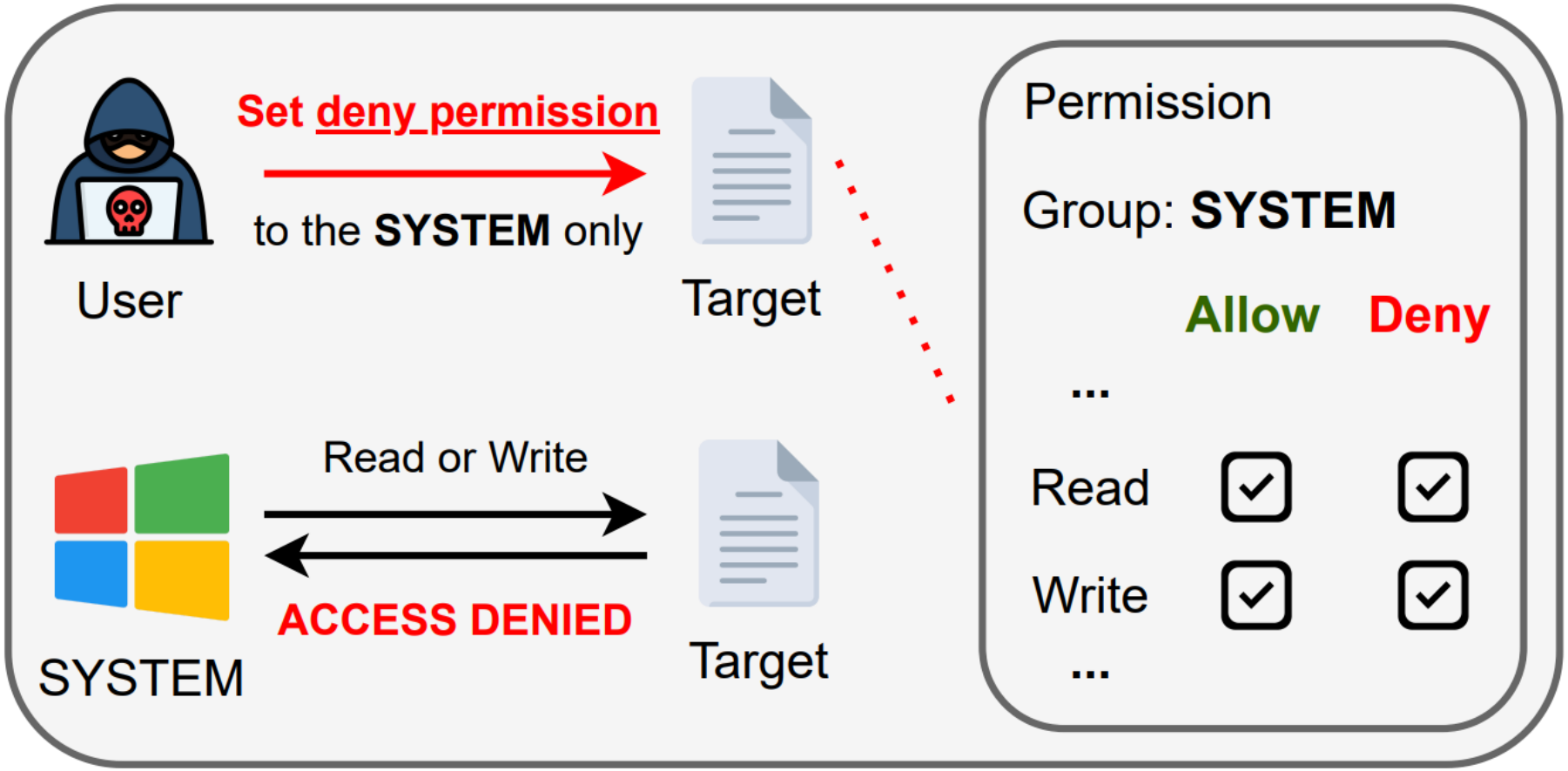
File Permission Inversion in 



File Permission Inversion in



File Permission Inversion in



Research Questions

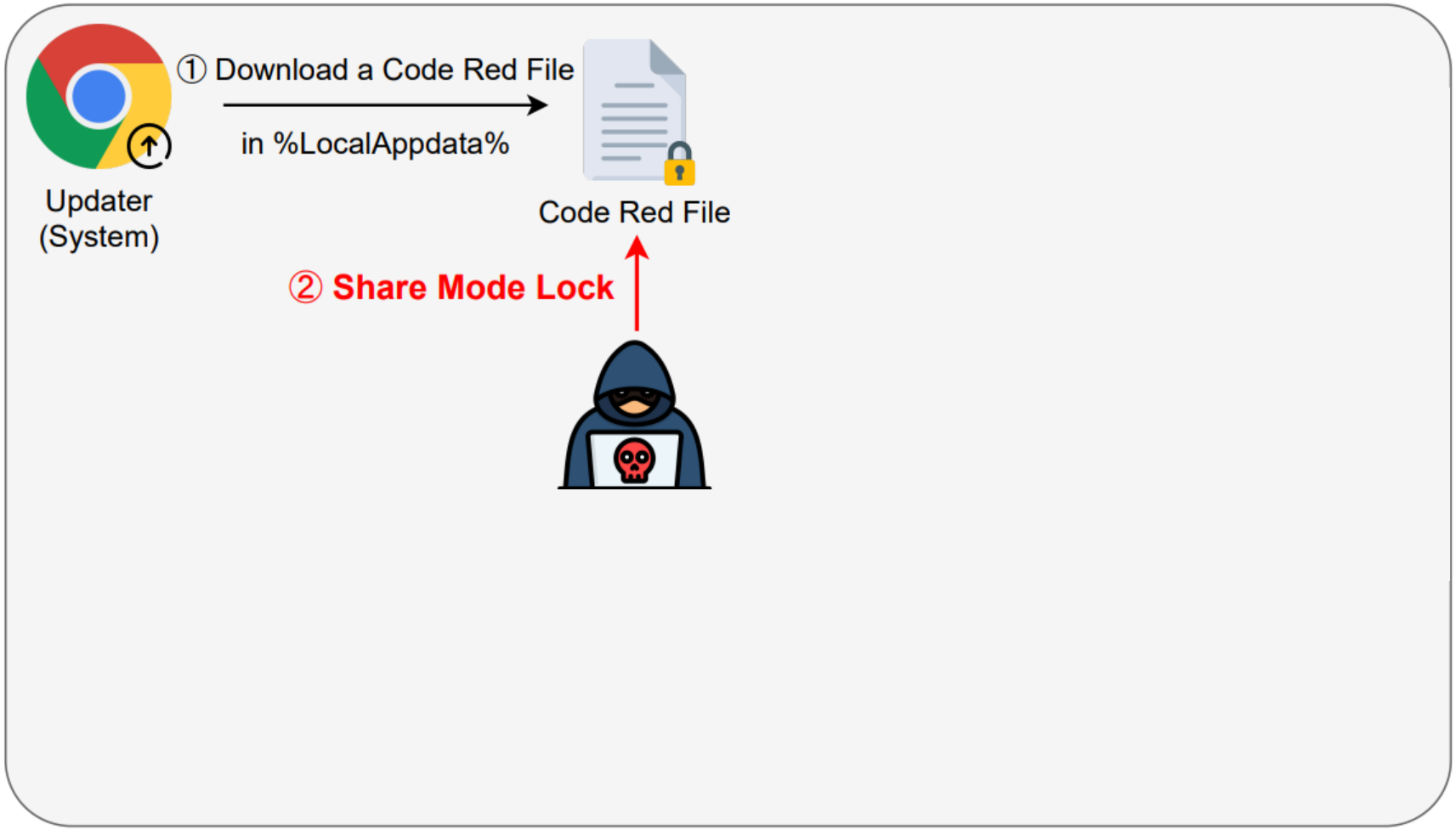
- **RQ2**: How serious are these discrepancies?
  - **A2**: We demonstrated that these discrepancies can lead to critical security vulnerabilities from **11 different vendors**.
  - Additionally, we showed that such vulnerabilities could arise in various component such as **browser sandboxes, antivirus, and software updater**.

Real-World Cases

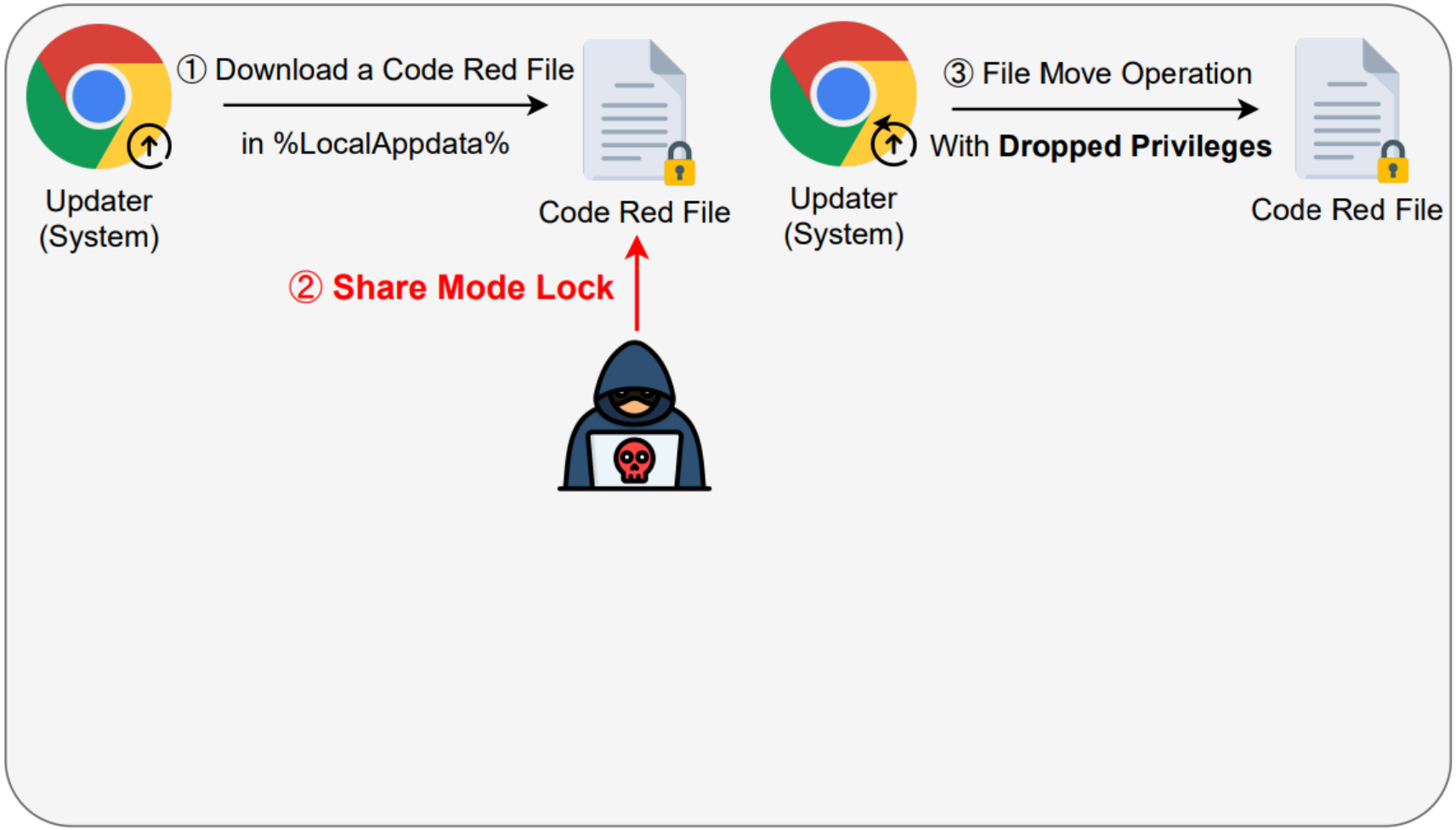
#	Software	Status	CVE ID	Impact
1	Google Chrome	Fixed	CVE-2024-1694	EoP
2	Google Chrome	Fixed	CVE-2024-7977	EoP
3	Mozilla Firefox	Fixed	CVE-2025-2817	EoP
4	Mozilla Firefox	Confirmed	-	Sandbox Escape
5	Oracle Java	Fixed	CVE-2025-50063	EoP
6	NVIDIA App	Confirmed	-	EoP
7	Samsung Magician	Fixed	CVE-2025-32098	EoP
8	Docker Desktop	Fixed	CVE-2025-3224	EoP
9	360 Total Security Antivirus	Fixed	-	DoS
10	Gen Digital CCleaner	Fixed	CVE-2025-3025	EoP
11	libcurl	Fixed	-	EoP
12	MalwareBytes Antivirus	Duplicate	CVE-2024-6260	EoP
13	Foxit Reader	Fixed	CVE-2024-38393	EoP

Chrome Updater EoP (CVE-2024-1694)

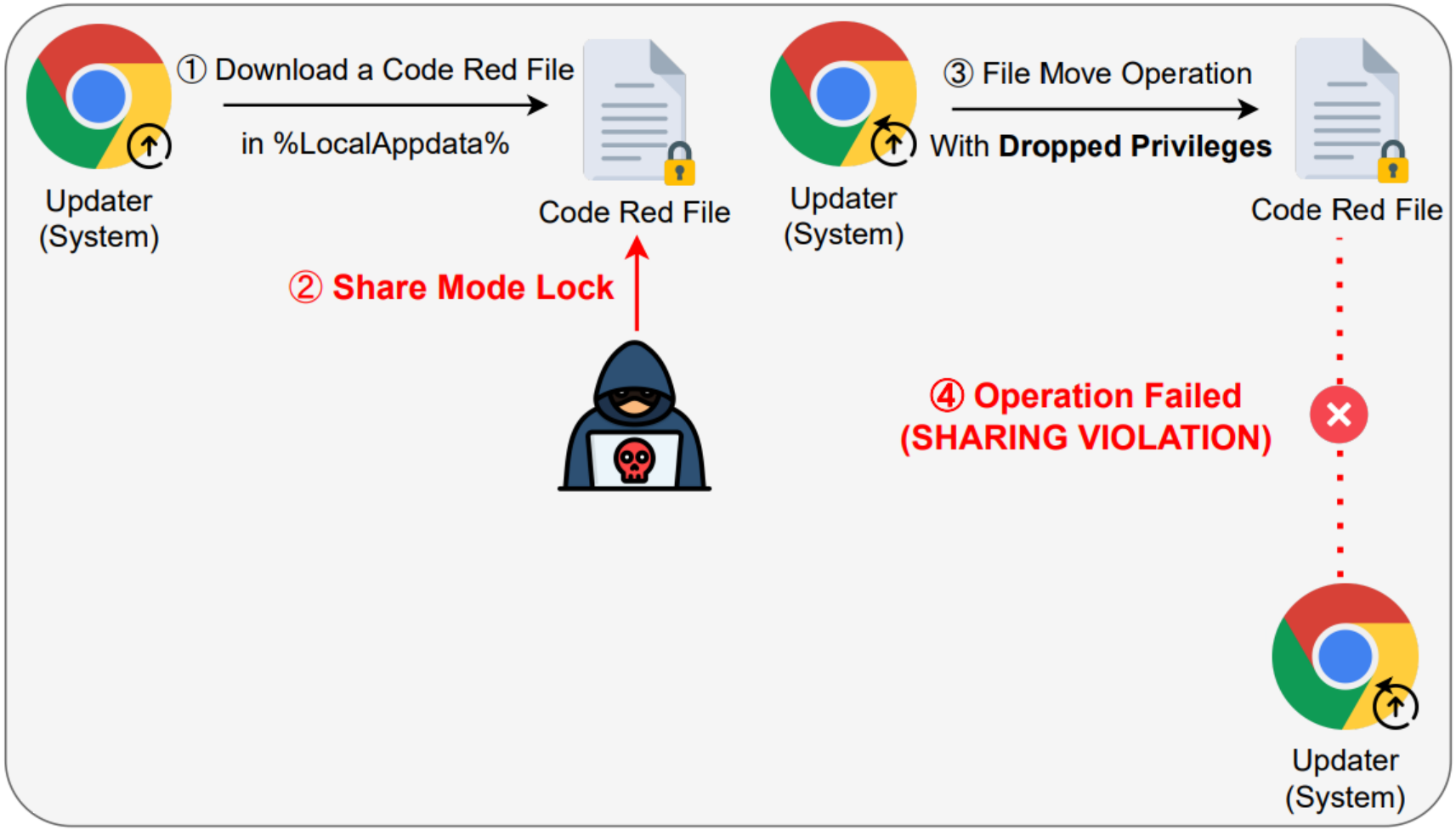
Chrome Updater EoP (CVE-2024-1694)



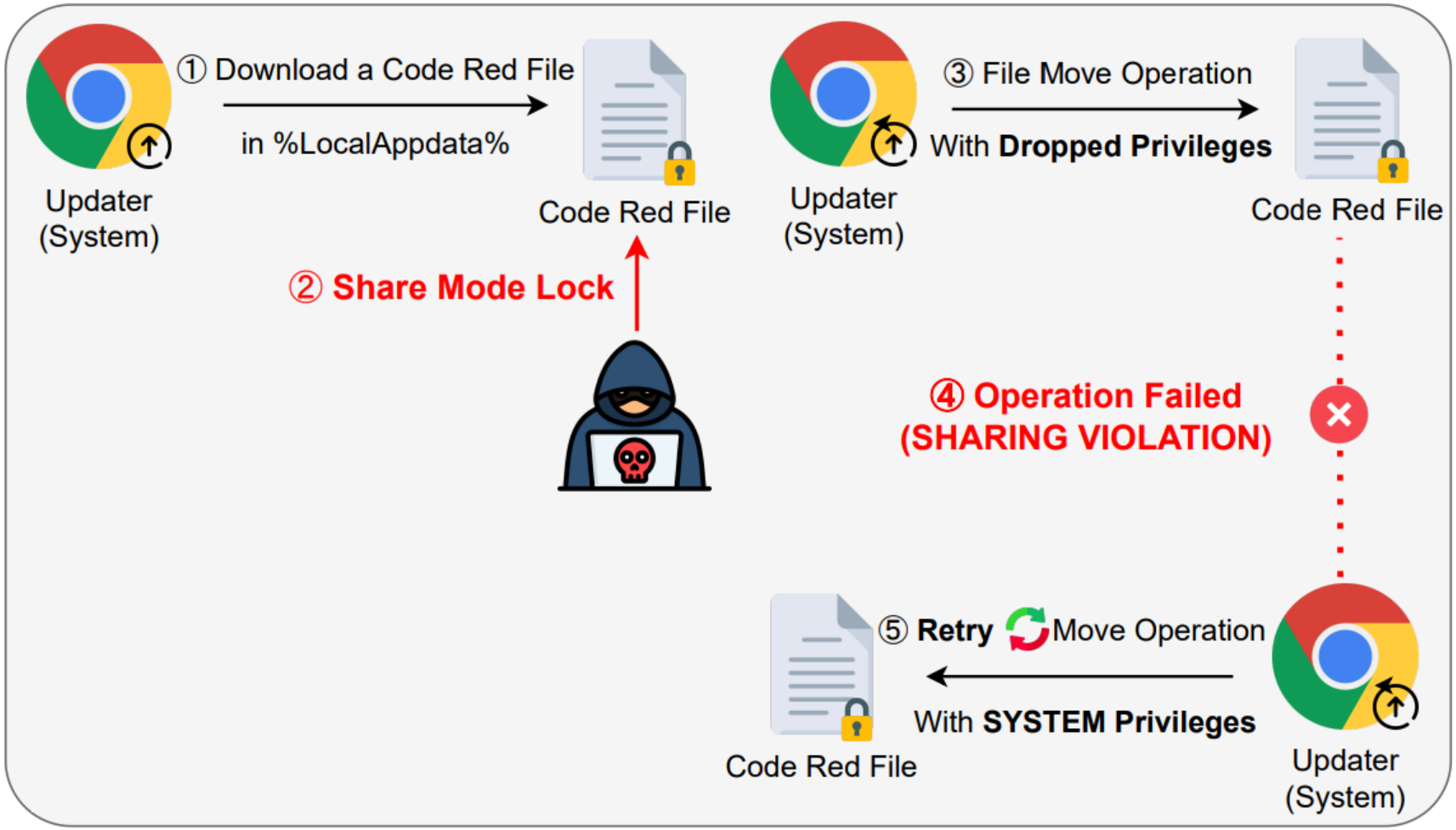
Chrome Updater EoP (CVE-2024-1694)



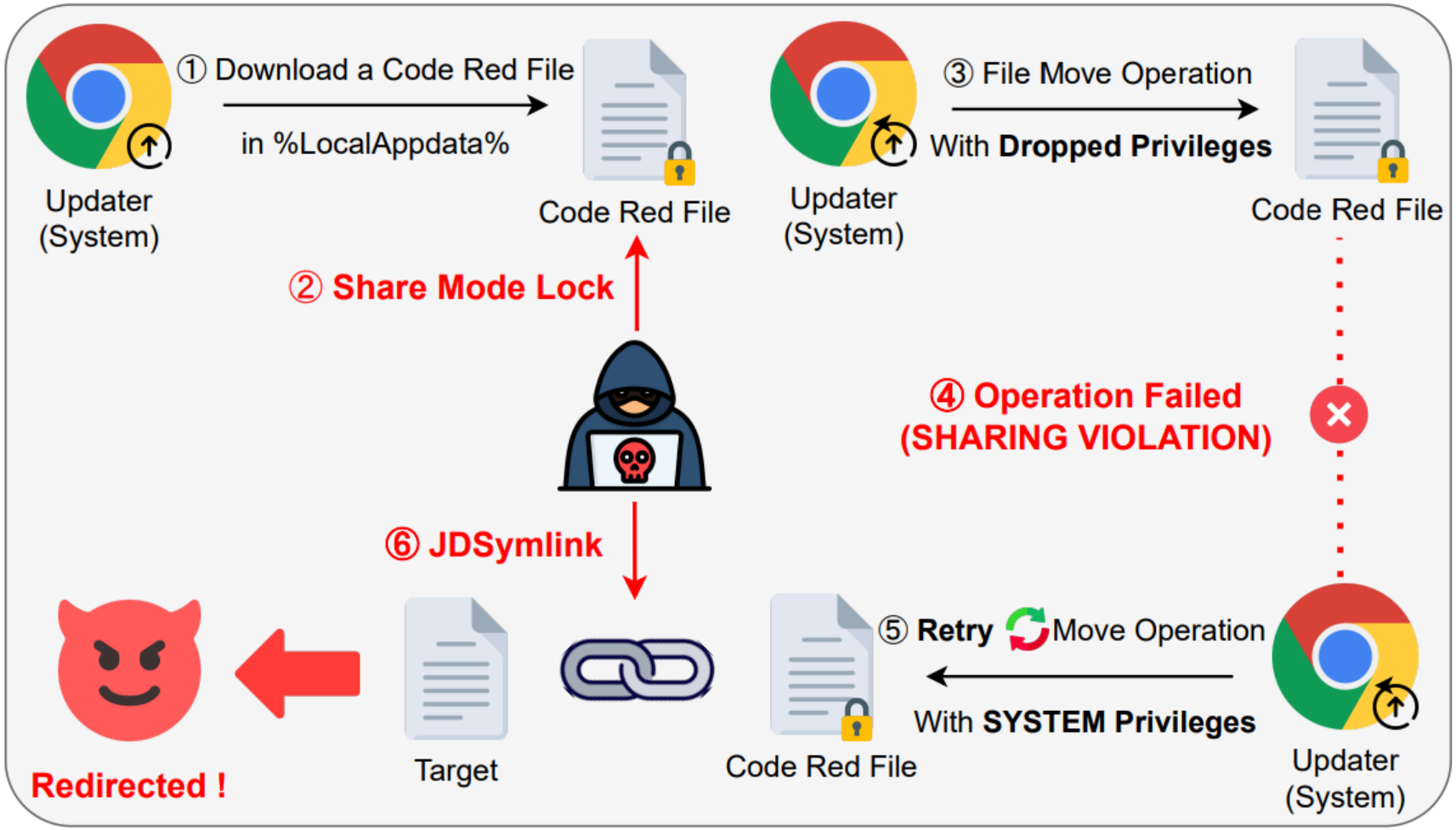
Chrome Updater EoP (CVE-2024-1694)



Chrome Updater EoP (CVE-2024-1694)



Chrome Updater EoP (CVE-2024-1694)



Research Questions

- **RQ3**: How do the existing compatibility layers handle these discrepancies?
  - **A3**: We found **27 non-compliant**[1] and **9 inconsistent**[2] behaviors in six languages.
    - All operations in Windows always follow **JDSymlinks**
  - Moreover, **no compatibility layers** are available for file locks and file permissions.

[1] **Non-compliant**: does not align with the specification
[2] **Inconsistent** : behaves differently on Windows and Linux and not clearly defined in specification

Compatibility Layers

Table 4: Behaviors of file-related functions on Windows and Linux.

Platforms (Link Type)	Languages	open			create		remove	copy						rename	
		N	P	XC	N	XC	N	N(S)	N(D)	P(S)	P(D)	I(S)	I(D)	N(S)	N(D)
Specification	C	✓	✗	✗	✓	-	✗	-	-	-	-	-	-	✗	✗
	C++	✓	-	-	-	-	✗	✓	✓	✗	✓	✗	✗	✗	✗
	Python	✓	✗	✗	-	-	?	✓	?	✗	?	-	-	✗	✗
	Java	?	-	-	?	-	?	✓	✗	✗	?	-	-	?	?
	Go	?	-	?	?	-	?	-	-	-	-	-	-	?	?
	Rust	?	-	✗	?	✗	✗	✓	✓	-	-	-	-	✗	✗
Linux	C	✓	✗	✗	✓	-	✗	-	-	-	-	-	-	✗	✗
	C++	✓	-	-	-	-	✗	✓	✗	✗	✗	✗	✗	✗	✗
	Python	✓	✗	✗	-	-	✗	✓	✓	✗	✓	-	-	✗	✗
	Java	✓	-	-	✓	-	✗	✓	✗	✗	✗	-	-	✗	✗
	Go	✓	-	✗	✓	-	✗	-	-	-	-	-	-	✗	✗
	Rust	✓	-	✗	✓	✗	✗	✓	✓	-	-	-	-	✗	✗
Windows (NTFS Symlink)	C	✓	-	✓	✓	-	✗	-	-	-	-	-	-	✗	✗
	C++	✓	-	-	-	-	✗	✓	✓	✗	✢	✗	★	✗	✗
	Python	✓	-	✓	-	-	✗	✓	✓	✗	✓	-	-	✗	✗
	Java	✓	-	-	✓	-	✗	✓	✗	✗	✗	-	-	✗	✗
	Go	✓	-	✓	✓	-	✗	-	-	-	-	-	-	✗	✗
	Rust	✓	-	✗	✓	✗	✗	✓	✓	-	-	-	-	✗	✗
Windows (JDSymlink)	C	✓	-	✓	✓	-	✓	-	-	-	-	-	-	✓	✻
	C++	✓	-	-	-	-	✓	✓	✓	✓	✓	✓	✓	✓	✻
	Python	✓	-	✓	-	-	✓	✓	✓	✓	✓	-	-	✓	✻
	Java	✓	-	-	✓	-	✓	✓	✓	✓	✓	-	-	✓	✻
	Go	✓	-	✓	✓	-	✓	-	-	-	-	-	-	✓	✻
	Rust	✓	-	✓	✓	✓	✓	✓	✓	-	-	-	-	✓	✻

✓/✗: follows/does not follow the link, ?: not specified, -: not available.
: conformant (i.e., compliant or consistent), : non-compliant, : inconsistent among implementations.
N: no flags, P: link traversal prevention, I: ignore link (skip), XC: exclusive creation (O_EXCL | O_CREAT). S/D: source/destination.
✢: follows link when file → link, but not when link → link.
★: does not follow link when file → link, but errors in link → link.
✻: partially follows JDSymlinks — follows junctions of JDSymlinks, but not DosDevices.

Research Questions

- **RQ4**: How aware are developers of these discrepancies, and how do they respond?
  - **A4**: Developers have limited awareness of OS-level file system discrepancies.
    - Rarely implement **effective mitigation** strategies.
    - None were aware of Windows-specific **file locking mechanisms**.

Research Questions

- **RQ4**: How aware are developers of these discrepancies, and how do they respond?
  - **A4**: Developers have limited awareness of OS-level file system discrepancies.
    - **Rarely implement** effective mitigation strategies.
    - None were aware of Windows-specific **file locking mechanisms**.

User Study Result

- **Task**: Simple logging system with high privileges
  - 19 out of 21 participants successfully implemented the requirement.
  - **Every** participant's code had security vulnerabilities, including:
    - Unchecked insecure links
    - Insecure use of temporary directories
    - Unsafe file system API usage

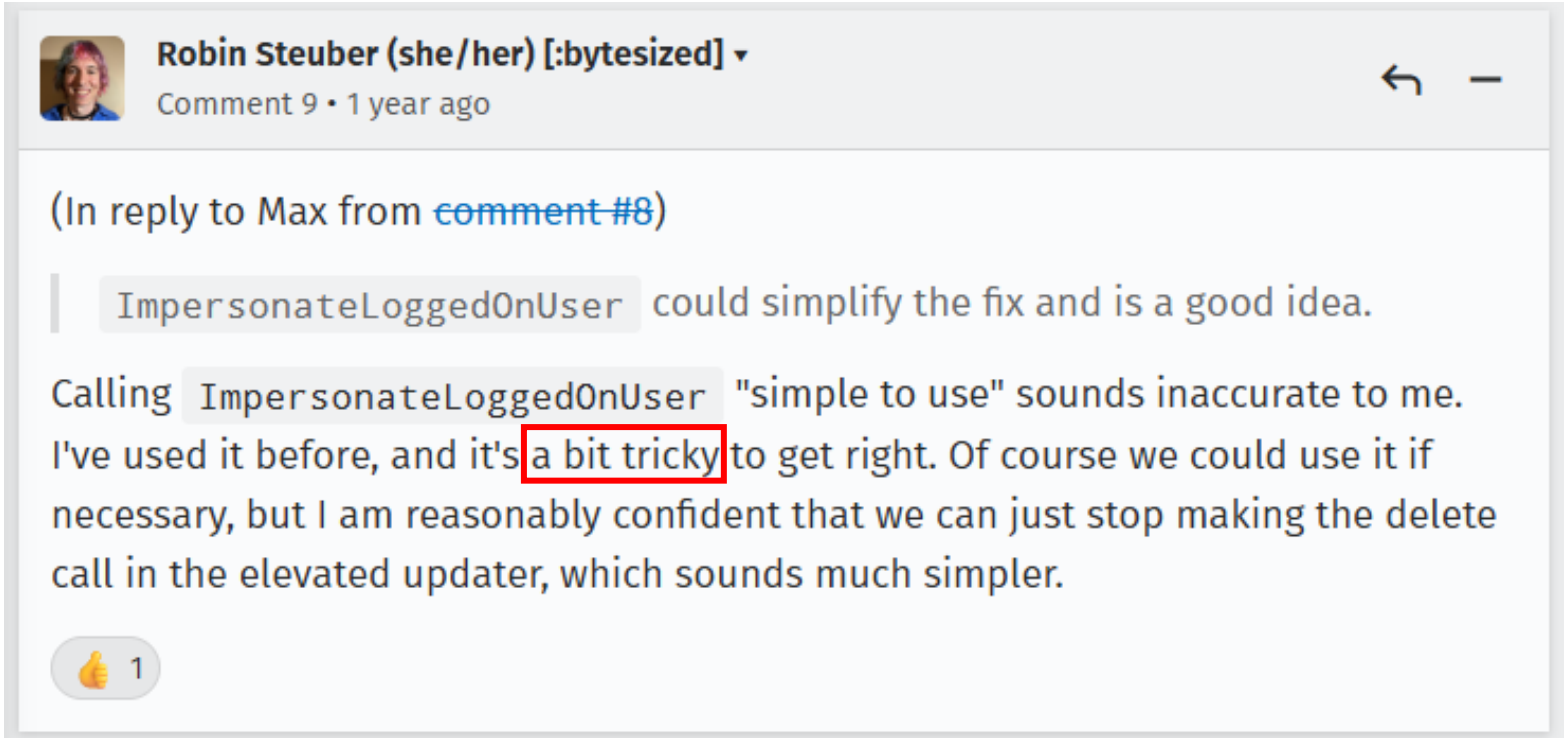
User Study Result

- **Interview**: We asked participants about their awareness of discrepancies.
  - Cross-OS knowledge: **None** of the 21 participants had knowledge of Windows vs Linux file system discrepancies. Only **two** mentioned superficial differences.
  - File locking: **No participants** demonstrated understanding risks.
  - Invalid access control: **9 participants** recognized the risks and suggested proper mitigation.

Research Questions

- **RQ5**: What would be the countermeasures to mitigate these vulnerabilities?
  - **A5**: Existing countermeasures have limitations.
    - Even experienced developers struggle to implement effective countermeasures.
    - We believe the root cause lies in fundamental Windows file system **design weaknesses**.
    - Therefore, **Microsoft** should address this directly rather than through targeted patches.

Challenges of Mitigating the Vulnerabilities



A comment from the Firefox developer about **Impersonate**

Challenges of Mitigating the Vulnerabilities

In terms of issuing a CVE, we are still considering. Tentatively, this could be scored as CVSS:4.0/AV:L/AC:H/AT:P/PR:L/UI:A/VC:H/VI:H/VA:H/SC:H/SI:H/SA:H (7.3 HIGH) but some more research is being done. We might consider this purely as security hardening since the underlying problem is a Windows bug.

We will follow up as soon as we have updates.

Security @ Docker

A response from the Docker developer about **CVE-2025-3224**

Responses form Microsoft

Microsoft Security Response Center <secure@microsoft.co... 1월 23일 (목) 오전 7:02

나에게

Hello,

Thank you again for submitting this issue to Microsoft. Currently, MSRC prioritizes vulnerabilities that are assessed as "Important" or "Critical" severities for immediate servicing. After careful investigation, this case has been assessed as **low** severity and does not meet MSRC's bar for immediate servicing. However, we have shared the report with the team responsible for maintaining the product or service. They will take appropriate action as needed to help keep customers protected.

Here is some information on Microsoft's security vulnerability servicing criteria that may help you in your future research:

Conclusion

- **Comprehensive analysis** of Windows file system design weaknesses.
- **Four critical discrepancies** → high-impact vulnerabilities in real-world software.
- **Compatibility layers** and **developers** are unable to properly handle these discrepancies,
- **Existing countermeasures** remain limited in their effectiveness.
- Microsoft should consider **redesigning** the file system to provide a secure foundation for the Windows ecosystem.