Defense and Attack Techniques against File-based TOCTOU Vulnerabilities: a Systematic Review

Razvan Raducu ORCID
Dept. of Computer Science and Systems Engineering, University of Zaragoza, Spain

Ricardo J. Rodríguez[1] ORCID
Dept. of Computer Science and Systems Engineering, University of Zaragoza, Spain

Pedro Álvarez ORCID
Dept. of Computer Science and Systems Engineering, University of Zaragoza, Spain

Abstract

File-based *Time-of-Check to Time-of-Use* (TOCTOU) race conditions are a well-known type of security vulnerability. A wide variety of techniques have been proposed to detect, mitigate, avoid, and exploit these vulnerabilities over the past 35 years. However, despite these research efforts, TOCTOU vulnerabilities remain unsolved due to their non-deterministic nature and the particularities of the different filesystems involved in running vulnerable programs, especially in Unix-like operating system environments. In this paper, we present a systematic literature review on defense and attack techniques related to the file-based TOCTOU vulnerability. We apply a reproducible methodology to search, filter, and analyze the most relevant research proposals to define a global and understandable vision of existing solutions. The results of this analysis are finally used to discuss future research directions that can be explored to move towards a universal solution to this type of vulnerability.

2012 ACM Subject Classification Security and privacy → Software security engineering; Security and privacy → Systems security

Keywords and phrases file-based race condition, TOCTOU vulnerability, avoidance techniques

Funding This work has been supported in part by the PDC2021-121072-C22 project granted by the Spanish Ministry of Economy and Competitiveness and by the University, Industry and Innovation Department of the Aragonese Government under *Programa de Proyectos Estratégicos de Grupos de Investigación* (DisCo research group, ref. T21-20R).
Razvan Raducu: The research of R. Raducu was also supported by the Government of Aragon under a DGA predoctoral grant (period 2021-2025).

1 Extended Abstract

Today, many applications are deployed on large-scale distributed systems and multi-core processors, which perform multiple tasks concurrently while sharing common resources such as memory, disk, or network. The intrinsic characteristics of the simultaneous execution of programs make them very difficult to write, test, and debug [11, 10], which facilitates the existence of concurrency bugs.

Concurrency bugs are caused by accesses to a shared resource between threads and processes without proper synchronization. These bugs can lead to vulnerabilities that, when triggered by adversaries, can cause a much broader impact on security, such as bypassing security checks, breaking the integrity of databases [20], hijacking the vulnerable program control flow execution, or escalating privileges [23], among others.

A common attack especially related to concurrency bugs is the privilege escalation attack, in which a malicious user gains access to other user accounts on the target system. The number

[1] Corresponding author.

of vulnerabilities related to privilege escalation has been increasing in recent years. For instance, in 2020 this type of vulnerability comprised 44% of all Microsoft vulnerabilities [5]. There are two main types of privilege escalation: *horizontal privilege escalation* attacks, in which an attacker expands their privileges by taking over another (non-privileged) user account and abusing the legitimate privileges granted to the other user; and *vertical privilege escalation* attacks, which involve increasing privileges/privileged access beyond what a user (or an application or other asset) already has.

Vertical privilege escalation attacks are commonly caused by a particular type of concurrency bug, called *race condition bugs*. The root cause of these bugs is a TOCTOU (Time-of-Check to Time-Of-Use) bug, which occurs when a program checks a particular characteristic of an object (e.g., whether the file exists), and later takes some action that assumes the checked characteristic still holds [6]. The window of opportunity that the program leaves between the time of check and the time of use is then exploited by an adversary. The adversary can increase this window by various means, such as overloading the system or creating specific inputs for the vulnerable program. In addition, TOCTOU vulnerabilities are present in different scenarios. For example, memory accesses involving the kernel [18, 19] (also known as double-fetch bugs), Remote Attestation [3, 4], Trusted Computing [7, 8], or file-based TOCTOU [6, 22], among others.

In this paper, we focus on file-based TOCTOU since they are one of the oldest known security flaws, dating back to the mid-70s [12, 1]. These types of race conditions, particularly common on Unix-like systems, occur due to the mapping from a filename to a unique inode and a device number. Although the mapping of the inode and device number to a file descriptor is race-free, the mapping of the filename to the inode and the device number is volatile since filenames and the underlying inode and device number may change on each system call invocation.

A well-known example of this kind of problem is `sendmail` [22], which used to look for a specific attribute of a mailbox file before adding new messages to it. Unfortunately, the verification and append operations are not an atomic unit. Consequently, if an adversary (the mailbox owner) replaces their mailbox file with a symbolic link to sensitive files (such as `/etc/passwd`, which contains information about system user accounts) between the verification and append operations, then `sendmail` will add email contents to `/etc/passwd`. As a result, the adversary can craft an email message to add a new user account with superuser privileges in the system.

Figure 1 illustrates this typical security flaw. The vulnerable code appears on the left side of the figure. On line 6 there is a check of the write permission on a file (identified by a string) with the `access` system call. Once the verification is successful, the file is opened (line 8) and certain data is appended to the file. If this program is run with *setuid* permission (i.e., users can run it with elevated privileges temporarily to perform a specific task), the adversary can take advantage of the race window between the operations on lines 6 and 8. An example of the exploit used by an adversary is shown on the right side of the figure. Suppose the exploit is run to write to the `/etc/passwd` file, which is a protected file in UNIX-based systems. If an adversary iteratively creates a symbolic link to `/etc/passwd` (line 14, right side) at the same time as the execution of the vulnerable program (line 12, right side), the race condition will eventually occur and the attack will succeed, appending new content to the protected file.

Specifically, file-based TOCTOU vulnerabilities[2] are file-based race conditions that occur

[2] In the rest of this paper, we refer to file-based TOCTOU vulnerabilities simply as TOCTOU vulnerabil-

```
// toctou.c
char *filename = argv[1];
// ...

// Check permissions
if(!access(filename, W_OK)){
        // Open the file
        file = fopen(filename, "a+");

        // Write to file the user input
        fwrite(buffer, sizeof(char), strlen(buffer), file);
        fwrite("\n", sizeof(char), 2, file);
        fclose(file);
}else
        printf("No permission, exiting!\n");
```

```
#!/bin/bash
# exploit.sh
# (execute it as: ./exploit.sh /etc/passwd)
TEMPFILE="temp.file"
OLD_LS=`ls -l $1`
NEW_LS=`ls -l $1`

while [ "$OLD_LS" == "$NEW_LS" ]
do
    rm -f $TEMPFILE
    echo "From␣user" > $TEMPFILE
    echo "TOCTOU␣success" | ./toctou $TEMPFILE > /dev/null &
    unlink $TEMPFILE
    ln -s $1 $TEMPFILE &
    NEW_LS=`ls -l $1`
done
```

Figure 1 Example of a file-based TOCTOU vulnerability (top) and exploit (bottom).

Table 1 Common Weakness Enumerations related to TOCTOU.

CWE ID	Vulnerability
CWE-59	*Improper Link Resolution Before File Access ('Link Following')*
CWE-61	*UNIX Symbolic Link (Symlink) Following.*
CWE-62	*UNIX Hard Link*
CWE-362	*Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')*
CWE-363	*Race Condition Enabling Link Following*
CWE-367	*Time-of-check Time-of-use (TOCTOU) Race Condition* (not only file-based TOCTOU)
CWE-386	*Symbolic Name not Mapping to Correct Object*
CWE-706	*Use of Incorrectly-Resolved Name or Reference*

on filesystems with *weak* synchronization mechanisms (that is, they do not provide methods to ensure that filesystem objects remain unchanged between consecutive interactions with them). Given the non-deterministic nature of race conditions, the success of an attack is highly dependent on the precise and timely actions of the attacker at any given time during the execution of the vulnerable program. Furthermore, the occurrence of this type of vulnerability also depends on certain system calls being executed in a specific order, as well as environmental conditions [21, 22]. Therefore, the reproducibility of these vulnerabilities is typically very difficult.

Despite the age of this security flaw, numerous vulnerabilities are still reported each year related to TOCTOU vulnerabilities. For example, at the time of writing, a query to find TOCTOU-related vulnerabilities returns 786 results in the National Vulnerability Database [14] and 120 results in the MITRE CVE search engine [13], with the newest being only a few days old in both cases. This clearly shows that it is still a significant security problem and that the CVE release for TOCTOU vulnerabilities is common in the software industry. Furthermore, this vulnerability affects projects of any size, such as open-source projects [9], and major software vendors [17, 16, 2]. *The proof of the pudding is in the eating*: as shown in Table 1, there are several Common Weakness Enumeration (CWE) entries related to TOCTOU. CWEs represent a common language for discussing, finding, and addressing the causes of software security vulnerabilities, currently maintained by the MITRE Corporation. Each individual CWE represents only one type of vulnerability. This paper aims to systematically review the scientific literature in order to find techniques to mitigate TOCTOU vulnerabilities, as well as techniques to exploit these vulnerabilities. Specifically, we review the literature to find out what techniques have been proposed, how they are implemented, how they detect TOCTOU vulnerabilities, which operating system they target, and whether any source code or software tool is available to reproduce the experimental results.

In summary, our contributions are the following:

- We conduct a comprehensive review of the literature on defense and attack solutions against TOCTOU vulnerabilities. In particular, we found 37 articles proposing some kind of defense solution and only 4 articles proposing attacks against TOCTOU.

ities.

- We propose a taxonomy for TOCTOU defenses and attacks, according to when they perform the vulnerability detection/exploitation and at what level they operate. Furthermore, we classify TOCTOU attacks based on the attack vector they exploit.
- We highlight future research trends and directions regarding defense solutions for TOCTOU vulnerabilities. Our proposals cover modifying current operating system calls to make them race-free and security focused, modifying the kernel to avoid the use of filenames, and the use of transactional filesystems.

The full version of this paper (with a full discussion of the systematic review) was published in [15].

References

1 R. P. Abbott, J. S. Chin, J. E. Donnelley, W. L. Konigsford, S. Tokubo, and D. A. Webb. Security Analysis and Enhancements of Computer Operating Systems. Technical Report NBSIR 76-1041, Institute of Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, MD, April 1976.

2 Adobe. Adobe Security Bulletin. [Online; `https://helpx.adobe.com/security/products/creative-cloud/apsb20-11.html`], March 2020. Accessed on January 13, 2022.

3 Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. Collective Remote Attestation at the Internet of Things Scale: State-of-the-Art and Future Challenges. *IEEE Communications Surveys Tutorials*, 22(4):2447–2461, 2020.

4 Orlando Arias, Dean Sullivan, Haoqi Shan, and Yier Jin. LAHEL: Lightweight Attestation Hardening Embedded Devices using Macrocells. In *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 305–315, 2020.

5 BeyondTrust. Microsoft Vulnerabilities Report 2021. [Online; `https://www.beyondtrust.com/assets/documents/BeyondTrust-Microsoft-Vulnerabilities-Report-2021.pdf`], March 2021. Accessed on May 30, 2021.

6 Matt Bishop and Michael Dilger. Checking for Race Conditions in File Accesses. *Computing Systems*, 9(2):131–152, 1996.

7 Sergey Bratus, Nihal D'Cunha, Evan Sparks, and Sean W Smith. TOCTOU, Traps, and Trusted Computing. In *International Conference on Trusted Computing*, volume 4968, pages 14–32, Berlin, Heidelberg, March 2008. Springer-Verlag. `doi:10.1007/978-3-540-68979-9_2`.

8 Xiaolin Chang, Bin Xing, and Ying Qin. Formal Analysis of a Response Mechanism for TCG TOCTOU Attacks. In *2012 Fourth International Conference on Multimedia Information Networking and Security*, pages 19–22, 2012.

9 Flysystem. Time-of-check Time-of-use (TOCTOU) Race Condition in league/flysystem. [Online; `https://github.com/thephpleague/flysystem/security/advisories/GHSA-9f46-5r25-5wfm`], June 2021. Accessed on January 13, 2022.

10 E.A. Lee. The Problem with Threads. *Computer*, 39(5):33–42, 2006.

11 Charles E. McDowell and David P. Helmbold. Debugging Concurrent Programs. *ACM Comput. Surv.*, 21(4):593–622, dec 1989.

12 W. S. McPhee. Operating System Integrity in OS/VS2. *IBM Systems Journal*, 13(3):230–252, 1974.

13 MITRE. MITRE CVE - TOCTOU Search Results. [Online; `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=file-based+TOCTOU`], January 2022. Accessed on January 13, 2022.

14 National Vulnerability Database. NVD - TOCTOU Search Results. [Online; `https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cwe_id=CWE-59&isCpeNameSearch=false`], January 2022. Accessed on January 13, 2022.

15 Razvan Raducu, Ricardo J. Rodríguez, and Pedro Alvarez. Defense and Attack Techniques against File-based TOCTOU Vulnerabilities: a Systematic Review. *IEEE Access*, 10:21742–21758, 2022.

16 Red Hat. CVE-2021-30465- Red Hat Customer Portal. [Online; `https://access.redhat.com/security/cve/cve-2021-30465`], May 2021. Accessed on January 13, 2022.

17 VMWare. VMSA-2020-0023.3. [Online; `https://www.vmware.com/security/advisories/VMSA-2020-0023.html`], October 2020. Accessed on January 13, 2022.

18 Pengfei Wang, Jens Krinke, Kai Lu, Gen Li, and Steve Dodier-Lazaro. How Double-Fetch Situations Turn into Double-Fetch Vulnerabilities: A Study of Double Fetches in the Linux Kernel. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1–16, Vancouver, BC, aug 2017.

19 Pengfei Wang, Kai Lu, Gen Li, and Xu Zhou. DFTracker: Detecting Double-Fetch Bugs by Multi-Taint Parallel Tracking. *Front. Comput. Sci.*, 13(2):247–263, April 2019.

20 Todd Warszawski and Peter Bailis. ACIDRain: Concurrency-Related Attacks on Database-Backed Web Applications. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 5–20, New York, NY, USA, 2017. Association for Computing Machinery.

21 Jinpeng Wei and Calton Pu. TOCTTOU vulnerabilities in unix-style file systems: An anatomical study. In *4th USENIX Conference on File and Storage Technologies (FAST 05)*, page 12, San Francisco, CA, December 2005.

22 Jinpeng Wei and Calton Pu. Modeling and Preventing TOCTTOU Vulnerabilities in Unix-style File Systems. *Computers & Security*, 29(8):815–830, 2010. `doi:10.1016/j.cose.2010.09.004`.

23 Junfeng Yang, Ang Cui, Sal Stolfo, and Simha Sethumadhavan. Concurrency Attacks. In *Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism (HotPar'12)*, page 15, USA, 2012. USENIX Association.