Time-of-check to time-of-use

In software development, **time-of-check to time-of-use** (**TOCTOU**, **TOCTTOU** or **TOC/TOU**) is a class of software bugs caused by a race condition involving the *checking* of the state of a part of a system (such as a security credential) and the *use* of the results of that check.

TOCTOU race conditions are common in Unix between operations on the file system,[1] but can occur in other contexts, including local sockets and improper use of database transactions. In the early 1990s, the mail utility of BSD 4.3 UNIX had an exploitable race condition for temporary files because it used the `mktemp()`[2] function.[3] Early versions of OpenSSH had an exploitable race condition for Unix domain sockets.[4] They remain a problem in modern systems; as of 2019, a TOCTOU race condition in Docker allows root access to the filesystem of the host platform.[5] In the 2023 Pwn2Own competition in Vancouver, a team of hackers were able to compromise the gateway in an updated Tesla Model 3 using this bug.[6]

Examples

In Unix, the following C code, when used in a `setuid` program, has a TOCTOU bug:

```
if (access("file", W_OK) != 0) {
    exit(1);
}

fd = open("file", O_WRONLY);
write(fd, buffer, sizeof(buffer));
```

Here, *access* is intended to check whether the real user who executed the `setuid` program would normally be allowed to write the file (i.e., *`access`* checks the real userid rather than effective userid).

This race condition is vulnerable to an attack:

Victim	Attacker

`1 if (access("file", W_OK) != 0) {`
`2 exit(1);`
`3 }`	

	After the access check, before the open, the attacker replaces `file` with a symlink to the Unix password file `/etc/passwd`: `symlink("/etc/passwd", "file");`

`5 fd = open("file", O_WRONLY);`
`6 write(fd, buffer, sizeof(buffer));`
Actually writing over `/etc/passwd`	

In this example, an attacker can exploit the race condition between the `access` and `open` to trick the `setuid` victim into overwriting an entry in the system password database. TOCTOU races can be used for privilege escalation to get administrative access to a machine.

Although this sequence of events requires precise timing, it is possible for an attacker to arrange such conditions without too much difficulty.

The implication is that applications cannot assume the state managed by the operating system (in this case the file system namespace) will not change between system calls.

Reliably timing TOCTOU

Exploiting a TOCTOU race condition requires precise timing to ensure that the attacker's operations interleave properly with the victim's. In the example above, the attacker must execute the `symlink` system call precisely between the `access` and `open`. For the most general attack, the attacker must be scheduled for execution after each operation by the victim, also known as "single-stepping" the victim.

In the case of BSD 4.3 mail utility and `mktemp()`,[2] the attacker can simply keep launching mail utility in one process, and keep guessing the temporary file names and keep making symlinks in another process. The attack can usually succeed in less than one minute.

Techniques for single-stepping a victim program include file system mazes[7] and algorithmic complexity attacks.[8] In both cases, the attacker manipulates the OS state to control scheduling of the victim.

File system mazes force the victim to read a directory entry that is not in the OS cache, and the OS puts the victim to sleep while it is reading the directory from disk. Algorithmic complexity attacks force the victim to spend its entire scheduling quantum inside a single system call traversing the kernel's hash table of cached file names. The attacker creates a very large number of files with names that hash to the same value as the file the victim will look up.

Preventing TOCTOU

Despite conceptual simplicity, TOCTOU race conditions are difficult to avoid and eliminate. One general technique is to use error handling instead of pre-checking, under the philosophy of EAFP – "It is easier to ask for forgiveness than permission" – rather than LBYL – "look before you leap". In this case there is no check, and failure of assumptions to hold are signaled by an error being returned.[9]

In the context of file system TOCTOU race conditions, the fundamental challenge is ensuring that the file system cannot be changed between two system calls. In 2004, an impossibility result was published, showing that there was no portable, deterministic technique for avoiding TOCTOU race conditions when using the Unix `access` and `open` filesystem calls.[10]

Since this impossibility result, libraries for tracking file descriptors and ensuring correctness have been proposed by researchers.[11]

An alternative solution proposed in the research community is for Unix systems to adopt transactions in the file system or the OS kernel. Transactions provide a concurrency control abstraction for the OS, and can be used to prevent TOCTOU races. While no production Unix kernel has yet adopted transactions, proof-of-concept research prototypes have been developed for Linux, including the Valor file system[12] and the TxOS kernel.[13] Microsoft Windows has added transactions to its NTFS file system,[14] but Microsoft discourages their use, and has indicated that they may be removed in a future version of Windows.[15]

File locking is a common technique for preventing race conditions for a single file, but it does not extend to the file system namespace and other metadata, nor does locking work well with networked filesystems, and cannot prevent TOCTOU race conditions.

For `setuid` binaries, a possible solution is to use the `seteuid()` system call to change the effective user and then perform the `open()` call. Differences in `setuid()` between operating systems can be problematic.[16]

See also

- Linearizability

References

1. Wei, Jinpeng; Pu, Calton (December 2005). "TOCTTOU Vulnerabilities in UNIX-Style File Systems: An Anatomical Study" (https://www.usenix.org/conference/fast-05/tocttou-vulnerabilities-unix-style-file-systems-anatomical-study). USENIX. Retrieved 2019-01-14.
2. "mktemp(3)" (https://man7.org/linux/man-pages/man3/mktemp.3.html). *Linux manual page*. 2017-09-15.
3. Shangde Zhou(周尚德) (1991-10-01). "A Security Loophole in Unix" (https://archive.today/20130116041403/http://cdblp.cn/paper/UNIX%E7%9A%84%E4%B8%80%E4%B8%AA%E6%BC%8F%E6%B4%9E/94334.html). Archived from the original (http://cdblp.cn/paper/UNIX%E7%9A%84%E4%B8%80%E4%B8%AA%E6%BC%8F%E6%B4%9E/94334.html) on 2013-01-16.
4. Acheson, Steve (1999-11-04). "The Secure Shell (SSH) Frequently Asked Questions" (https://web.archive.org/web/20170213004928/http://www.employees.org/~satch/ssh/faq/TheWholeSSHFAQ.html). Archived from the original (http://www.employees.org/~satch/ssh/faq/TheWholeSSHFAQ.html) on 2017-02-13.
5. "Docker Bug Allows Root Access to Host File System" (https://duo.com/decipher/docker-bug-allows-root-access-to-host-file-system). *Decipher*. Duo Security. 28 May 2019. Retrieved 2019-05-29.
6. "Windows 11, Tesla, Ubuntu, and macOS hacked at Pwn2Own 2023" (https://www.bleepingcomputer.com/news/security/windows-11-tesla-ubuntu-and-macos-hacked-at-pwn2own-2023/). *BleepingComputer*. Retrieved 2023-03-24.
7. Borisov, Nikita; Johnson, Rob; Sastry, Naveen; Wagner, David (August 2005). "Fixing races for fun and profit: how to abuse atime". *Proceedings of the 14th Conference on USENIX Security Symposium*. **14**. Baltimore, MD: 303–314. CiteSeerX 10.1.1.117.7757 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.7757).
8. Xiang Cai; Yuwei Gui; Johnson, Rob (May 2009). "Exploiting Unix File-System Races via Algorithmic Complexity Attacks" (https://web.archive.org/web/20210518212029/https://www3.cs.stonybrook.edu/~rob/papers/races2.pdf) (PDF). *2009 30th IEEE Symposium on Security and Privacy*. Berkeley, CA. pp. 27–41. doi:10.1109/SP.2009.10 (https://doi.org/10.1109%2FSP.2009.10). ISBN 978-0-7695-3633-0. S2CID 6393789 (https://api.semanticscholar.org/CorpusID:6393789). Archived from the original (https://www3.cs.stonybrook.edu/~rob/papers/races2.pdf) (PDF) on 2021-05-18.
9. Martelli, Alex (2006). "Chapter 6: Exceptions". *Python in a Nutshell* (2 ed.). O'Reilly Media. p. 134. ISBN 978-0-596-10046-9.
10. Dean, Drew; Hu, Alan J. (August 2004). "Fixing Races for Fun and Profit: How to use access(2)". *Proceedings of the 13th USENIX Security Symposium*. San Diego, CA): 195–206. CiteSeerX 10.1.1.83.8647 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.8647).
11. Tsafrir, Dan; Hertz, Tomer; Wagner, David; Da Silva, Dilma (June 2008). "Portably Preventing File Race Attacks with User-Mode Path Resolution" (https://dominoweb.draco.res.ibm.com/c4028924309762d18525746e004a4feb.html). *Technical Report RC24572, IBM T. J. Watson Research Center*. Yorktown Heights, NY.
12. Spillane, Richard P.; Gaikwad, Sachin; Chinni, Manjunath; Zadok, Erez (February 24–27, 2009). "Enabling Transactional File Access via Lightweight Kernel Extensions" (https://www.fsl.cs.sunysb.edu/docs/valor/valor_fast2009.pdf) (PDF). *Seventh USENIX Conference on File and Storage Technologies (FAST 2009)*. San Francisco, CA.
13. Porter, Donald E.; Hofmann, Owen S.; Rossbach, Christopher J.; Benn, Alexander; Witchel, Emmett (October 11–14, 2009). "Operating System Transactions" (https://www.sigops.org/s/conferences/sosp/2009/papers/porter-sosp09.pdf) (PDF). *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP '09)*. Big Sky, MT.
14. Russinovich, Mark; Solomon, David A. (2009). *Windows Internals*. Microsoft Press. ISBN 978-0735648739.

15. "Alternatives to using Transactional NTFS" (https://web.archive.org/web/20220929200925/https://learn.microsoft.com/en-us/windows/win32/fileio/deprecation-of-txf). *Microsoft Developer Network*. Archived from the original (https://docs.microsoft.com/en-us/windows/win32/fileio/deprecation-of-txf) on 29 September 2022. Retrieved 10 December 2015.
16. Hao Chen; Wagner, David; Dean, Drew (2002-05-12). "Setuid Demystified" (https://people.eecs.berkeley.edu/~daw/papers/setuid-usenix02.pdf) (PDF).

Further reading

- Bishop, Matt; Dilger, Michael (1996). "Checking for Race Conditions in File Accesses" (http://nob.cs.ucdavis.edu/bishop/papers/1996-compsys/racecond.pdf) (PDF). *Computing Systems*. pp. 131–152.
- Tsafrir, Dan; Hertz, Tomer; Wagner, David; Da Silva, Dilma (2008). "Portably Solving File TOCTTOU Races with Hardness Amplification" (https://people.eecs.berkeley.edu/~daw/papers/tocttou-fast08.pdf) (PDF). *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST '08), San Jose (CA), February 26–29, 2008*. pp. 189–206.