

RaceGuard: Kernel Protection From Temporary File Race Vulnerabilities

Crispin Cowan, Steve Beattie, Chris Wright, and Greg
Kroah-Hartman

In USENIX Security Symposium, Washington, DC, August
2001

WireX Communications, Inc. <http://wirex.com/>



Temporary File Race Vulnerabilities

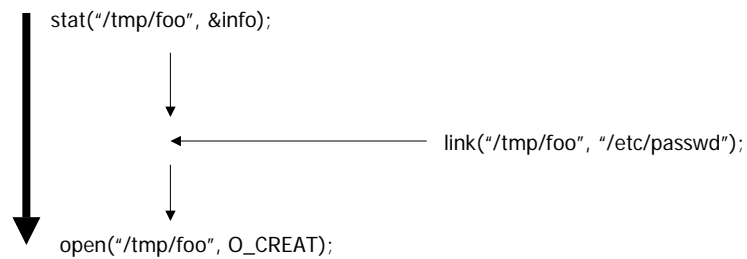
- Basic form:

A privileged program first probes the state of the file system, and then takes some action. The attacker races into the gap between the probe and the action, changes the state of the file system such that the victim program's action will have an unintended effect.

Temporary File Creation

Victim program

Attacker



Result: the file, /etc/passwd is overwritten.

A variation (dangling symlink) : link to non-existence file whose existence has security implication.
(e.g. /etc/nologin)

RaceGuard Design

- Each process keeps a cache of non-existent file names from the result of stat() calls.
- If the creat() call following the stat() call hits a file that already exists, and the name matches a name in the cache, then it is a race attack so abort the operation.
- If no conflict with the cache, remove the name from the cache to prevent the "false positive" events (in case the same name is used for creation multiple times)



RaceGuard Implementation

- Mediates three basic types of syscalls
 - 1) File probing system calls
e.g. stat(), lstat(), access(),...
 - 2) File creating system calls
e.g. open(), creat(), mkdir(), link(),...
 - 3) Process creation/removal
e.g. fork(), exit()
- Per-process & Inter-process RaceGuard cache management.



RaceGuard Implementation Cache Management Policy

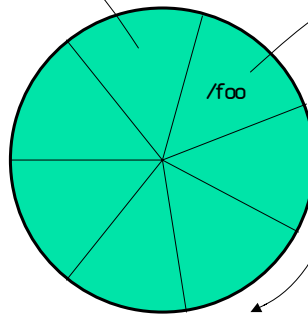
- 7 pre-allocated entries per process
 - small but sufficient in support of the cache management policy.
- Aggressive cache clearing policy
 - heavy use of cache invalidation upon successful creation of temporary files
- Conservative cache population policy
 - Interfering legitimate software by false positives is much more critical than failing to detect

Cache Management Policy

cache eviction policy

Cache is a circular buffer

If no empty slot found, eject this entry



Most recent entry
(start scan from this
entry)

Scan for first empty slot

RaceGuard Implementation

Cache Management Policy

- Races between processes
 - child processes inherit cache from parent process upon fork().
 - child processes clearing their cache entries notify their parent to also clear.
(aggressive cache clearing policy)
 - child processes do not populate their parents' cache as this could cause false positive. (conservative cache population policy)



RaceGuard Implementation

- Some file creating syscalls are not subject to race conditions. E.g. `mkdir()`, `link()`, etc.
 - they fail when the entry already exists. However, any successful file creation by those syscalls is checked against the cache and cleared if a match occurs.
(aggressive cache clearing policy)
- Some file probing syscalls are excluded.
E.g. it's possible to check for a file's existence with `chmod()` call, but it's an uncommon practice.
(conservative approach)



RaceGuard Implementation

A pathological case

- Cache flooding when executing external programs. E.g. `/bin/java`
If `$PATH` has many entries, many calls to `stat()` will occur, filling the cache with useless entries.



Security Testing

- To do deterministic testing, `mktemp()` library call has been “doctored”:

1) Pauses the caller for 30 seconds.

2) Produces file name that is easy to guess (even print it out)



Compatibility Testing

- **Goal:** Make sure RaceGuard does not interfere legitimate software that are not being actively subjected to actual race attacks.



Compatibility Testing

- Examples:

- 1) Mozilla:

- makes heavy use of temporary files for caching web content. Re-use of same names caused false positive. Fixed by cache clearing upon successful file creations.

- 2) A shell script Red Hat uses:

- parent process does the probe, and a child process creates the temporary files.

Fixed by making child processes clearing their cache entries also clear their parents' cache entries.



Performance Testing

- RaceGuard Microbenchmark Results

(overhead imposed on system calls)

System Call	Without RaceGuard	With RaceGuard	% Overhead
Stat non-existent file	4.3 microseconds	8.8 microseconds	104%
Open non-existent file	1.5 milliseconds	1.44 milliseconds	-4%
Fork	161 microseconds	183 microseconds	13%



Performance Testing

- Kernelstone Macrobenchmark, in Seconds
(imposed overhead on programs that make intensive use of many temporary files)

	Real Time	User Time	System Time
Without RaceGuard	10,700	8838	901
With RaceGuard	10,742	8858	904
%Overhead	0.4%	0.2%	0.3%

Time to build SRPM of the Linux kernel which incorporates thousand forks and temporary files.



Conclusions

- RaceGuard protects vulnerable programs against temporary file race attacks with minimal compatibility and performance overhead
- RaceGuard is available as a GPL'd patch to the Linux Kernel.