Connecting the Dot-Dots: Model Checking Concurrency in a Security API

Robert N. M. Watson*
University of Cambridge
Cambridge, UK
robert.watson@cl.cam.ac.uk

Jonathan Anderson
University of Cambridge
Cambridge, UK
jonathan.anderson@cl.cam.ac.uk

Abstract

Capsicum is a lightweight operating system capability sandbox framework planned for inclusion in FreeBSD 9. While developing the system, we discovered a concurrency vulnerability in which colluding sandboxed applications could manipulate the file system lookup API to escape from sandboxing. To explore the problem further, we employed model checking to determine the minimum requirements for the semantics of file system access via UNIX APIs, and various strategies for limiting API semantics in order to close the vulnerability. We also discovered that near-identical vulnerabilities are present in at least one other piece of existing sandboxing technology.

1 Capsicum

Capsicum is a lightweight operating system capability framework that allows UNIX processes to abandon access to global namespaces, operating in a *capability mode*. For example, an Apache worker process entering *capability mode* is denied use of the `rename()` system call, but may continue to perform I/O on file descriptors held before entering the sandbox, and may even have new capabilities assigned to it via message passing. We allow directory capabilities to be passed to sandboxes, as shown in Figure 1, granting access for specified operations to the directory and its children objects. In effect, this allows delegations of the form "The sandbox may open for read any object under `/var/www`" or "The sandbox may read or write all files and directories under `/var/www/site1`.

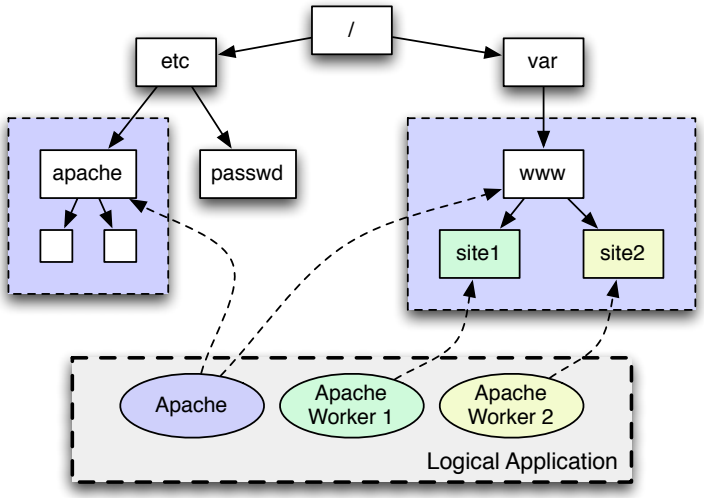

Figure 1: Partial delegation of a global filesystem namespace.

*Generously supported by a grant from Google, Inc.

To achieve this delegation, Capsicum only allows objects "below" the director capability can be accessed. This requires the UNIX path resolution routine, `namei()`, to implement an invariant: the parent object for a directory capability can never be named. In our initial implementation, we modified `namei()` to introduce a new constraint: any attempt to look up `..` relative to the starting directory capability would cause access control failure. This approach has a low implementation cost, and otherwise allows full file system semantics in the subtree, such as creating, renaming, removing, and opening files and directories.

2 The Vulnerability

After implementing Capsicum, we encountered a concurrency vulnerability exploiting non-atomicity in `namei()`: two threads can collude in manipulating the file system to escape the sandbox. Figure 2 illustrates how this can occur using two writable directory capabilities, one a subset of the other. When the threads simultaneously issue `openat()` and `renameat()` system calls, the invariant "the parent of a directory capability is unreachable" is violated without breaking the "can't lookup .. from the directory capability" constraint. In our example, `/var` is reachable despite neither capability granting access to it.

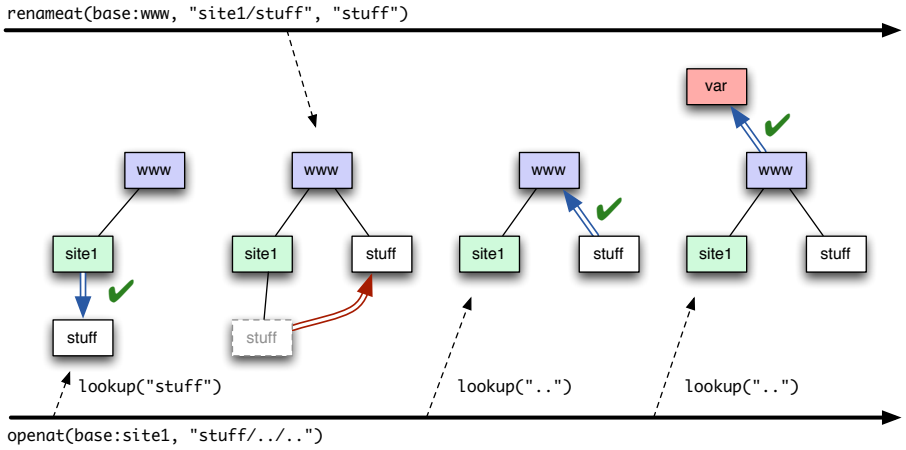

Figure 2: A malicious lookup which evades the constraint to break the invariant.

We have determined that this vulnerability affects other software providing partial namespace delegation. We employing a responsible disclosure process, but will provide complete details at the workshop.

3 Mitigation

Fundamentally, partial file system delegation with UNIX semantics is extremely tricky: paths are ephemeral traversal instructions, rather than first class objects. Therefore we had to consider fixes that limited UNIX semantics: our first pass disallows `..` in paths via directory capabilities, preventing cycles in modifications and traversals. However, it also breaks compatibility with existing applications. For instance, the Apache web server's configuration directory contains symbolic links from the `mods-enabled` directory to `mods-available`, e.g. `../mods-available/auth_basic.load`. Other semantic weakening is also sufficient, such as eliminating `renameat()`, and preventing concurrent namespace operations.

To better explore the problem, we employed the SPIN model checker to exhaustively test a model of the problem. We found that our broad-stroke solution was effective, but we also found other, more nuanced solutions to the security vulnerability.