Connecting the Dot Dots

Model Checking Concurrency in Capsicum

ASA-4
21 July 2010

Robert N. M. Watson
Jonathan Anderson

UNIVERSITY OF CAMBRIDGE

Introduction

- UNIX File System (UFS)
- Capsicum: practical capabilities for UNIX
- Whoops, a concurrency vulnerability
- Model checking file system containment
- Conclusions

The UNIX File System (UFS)

The UNIX file system

- Persistent object storage for UNIX
- Hierarchical, user-specified name space
- Also used for IPC
- DAC + MAC

➡ A security API

Typical APIs

- Open a file for I/O

```
int open(char *path, int flags, ...);
```

- Change directory

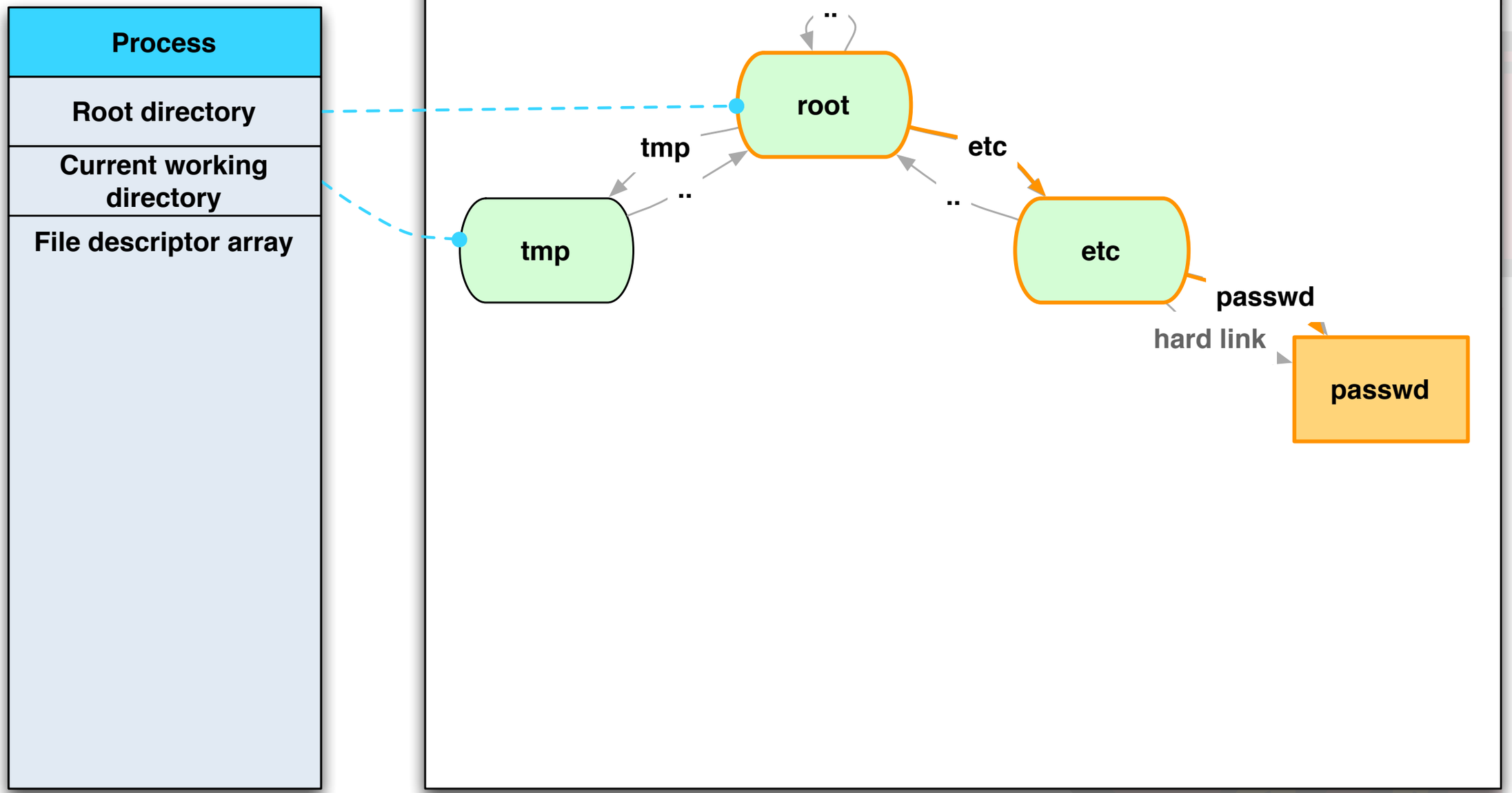
```
int chdir(char *path);
```

- Rename a file or directory

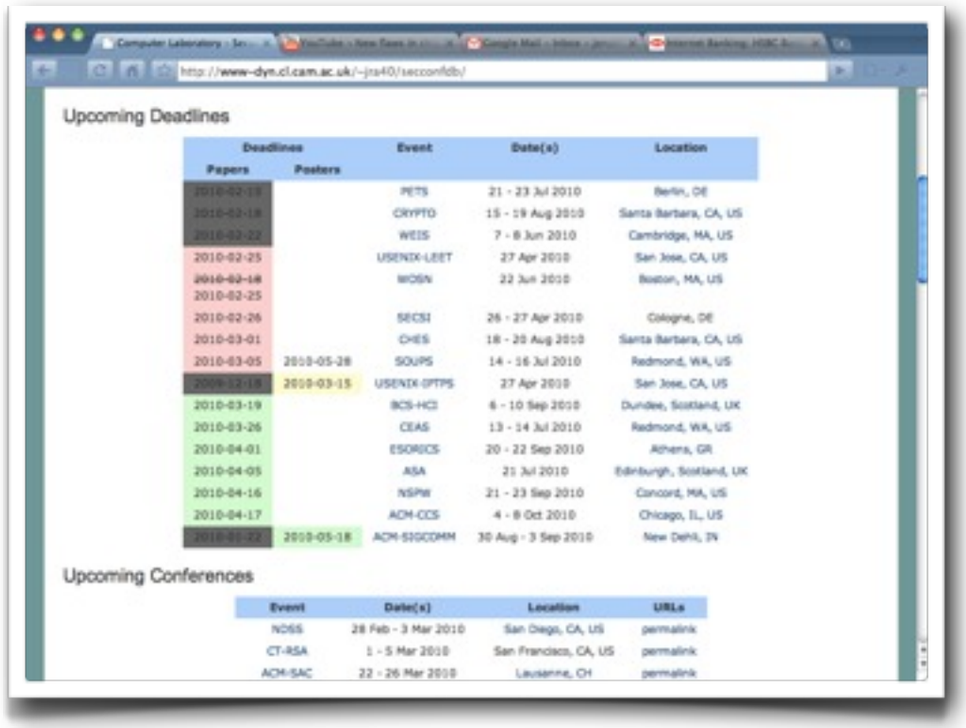
```
int rename(char *from, char *to);
```

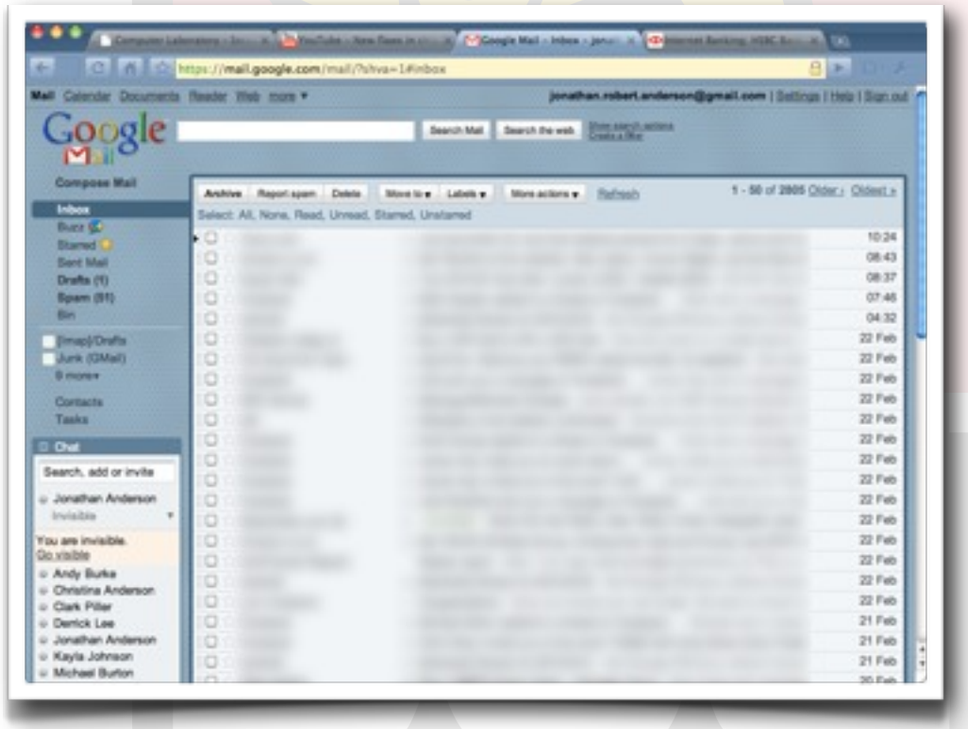
Looking up a path

open("/etc/passwd", O_RDONLY)

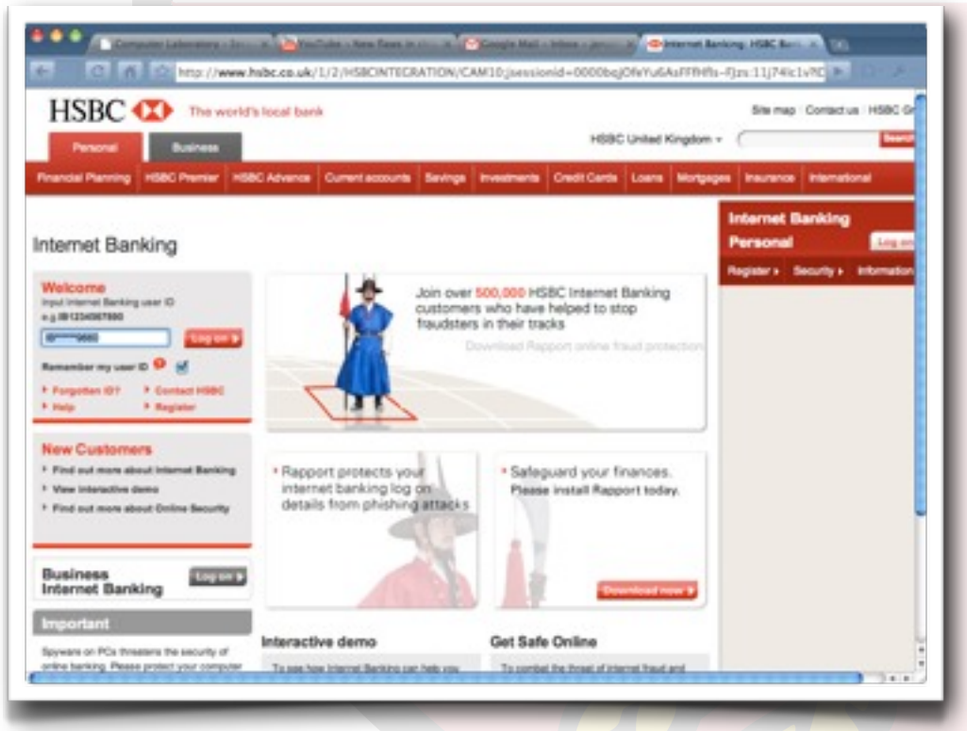


Capsicum: practical capabilities for UNIX



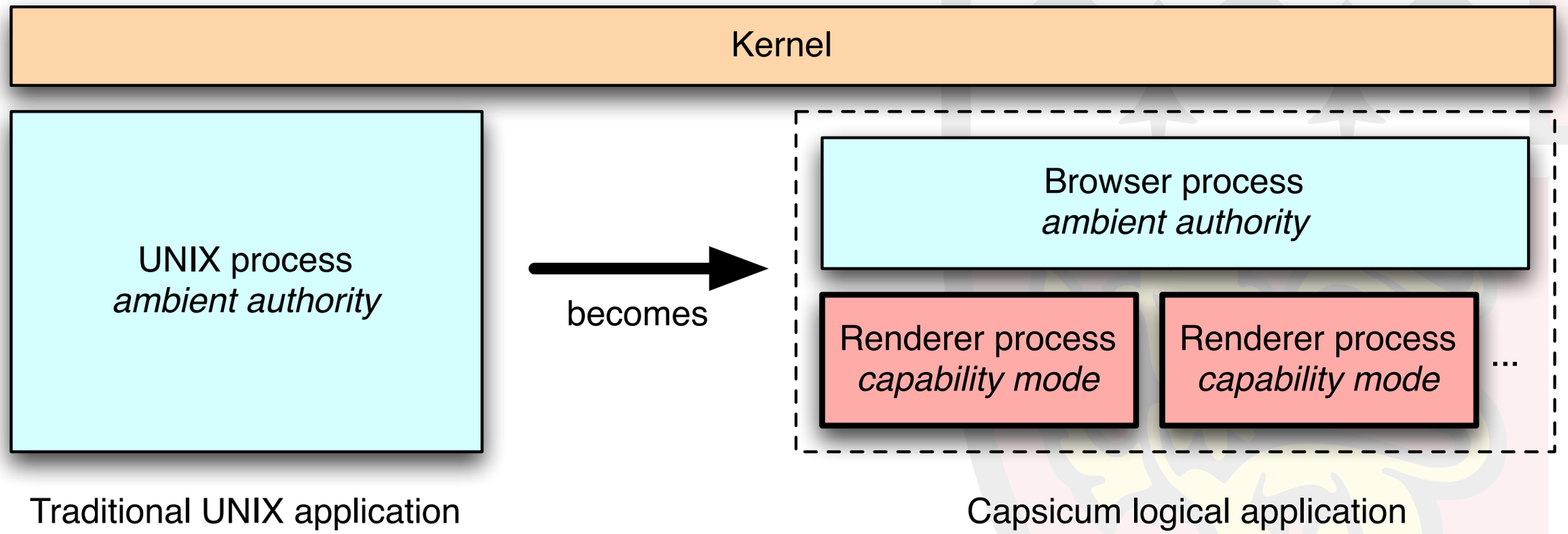


CVEs in Jan-Aug 2009	
Firefox	85
Safari	59
IE	48
Chrome	39
Flash	35
source; Justin Foster, OWASP	



UNIVERSITY OF CAMBRIDGE

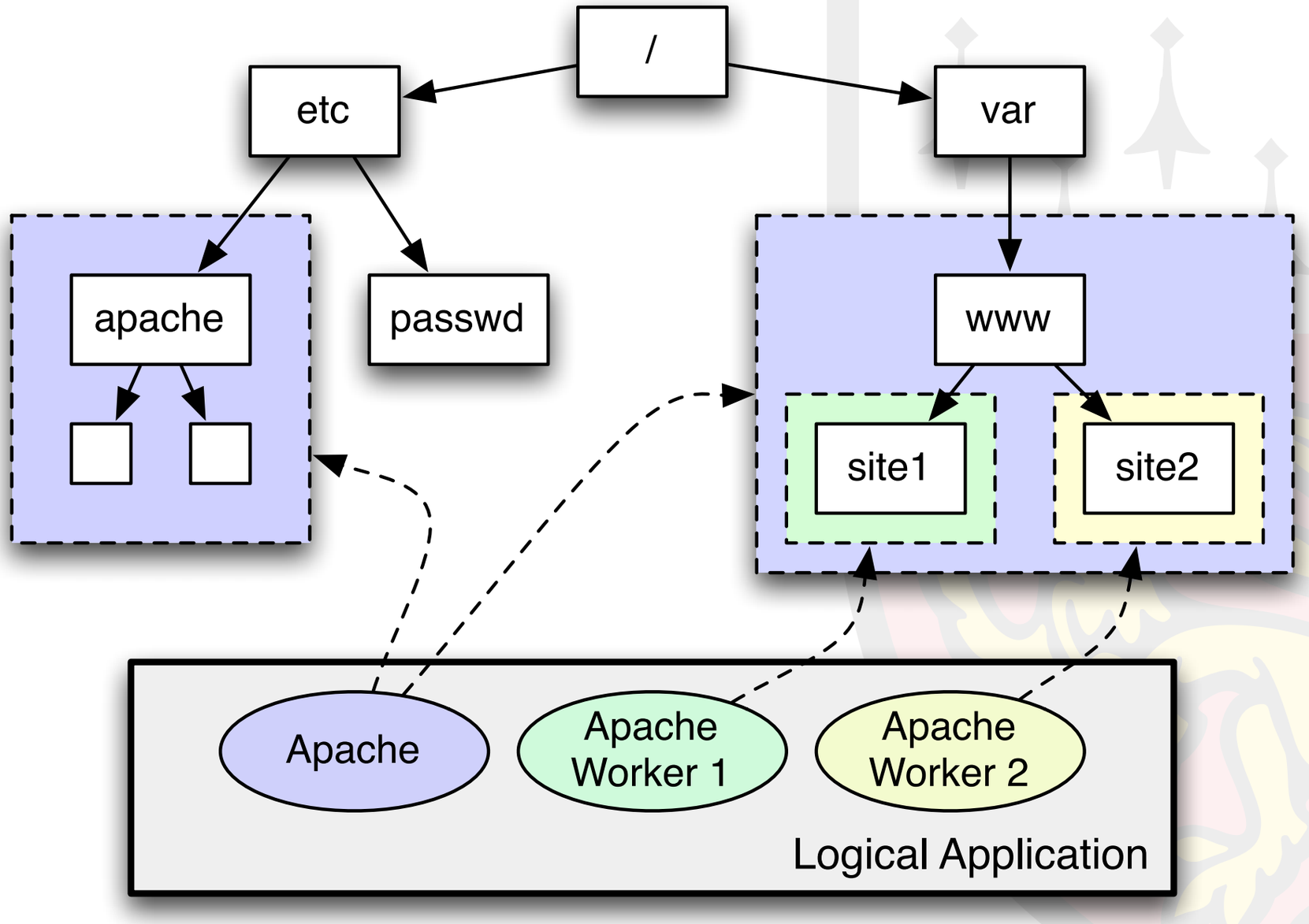
Logical applications



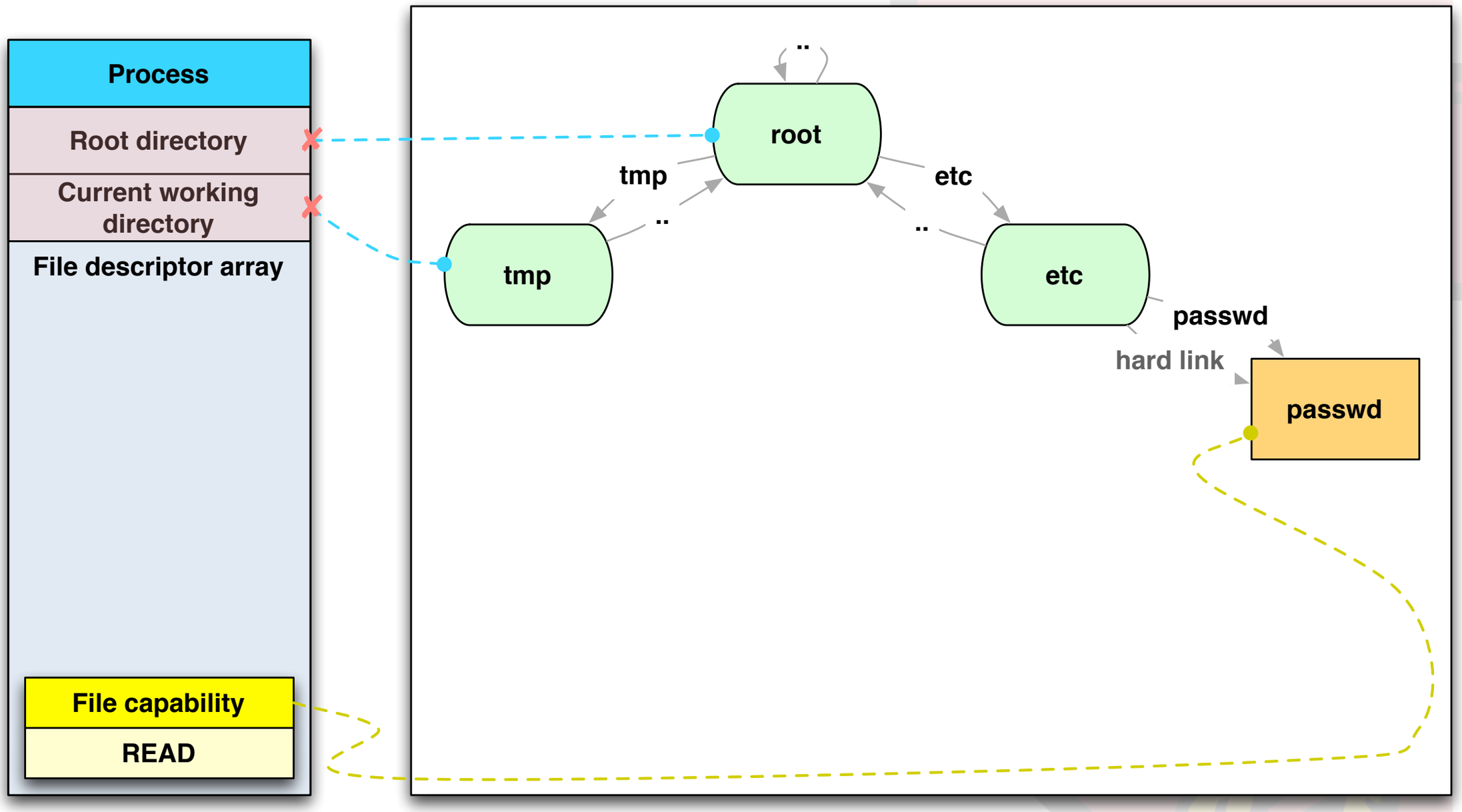
Traditional UNIX application

Capsicum logical application

Hierarchical delegation with capabilities



File capabilities



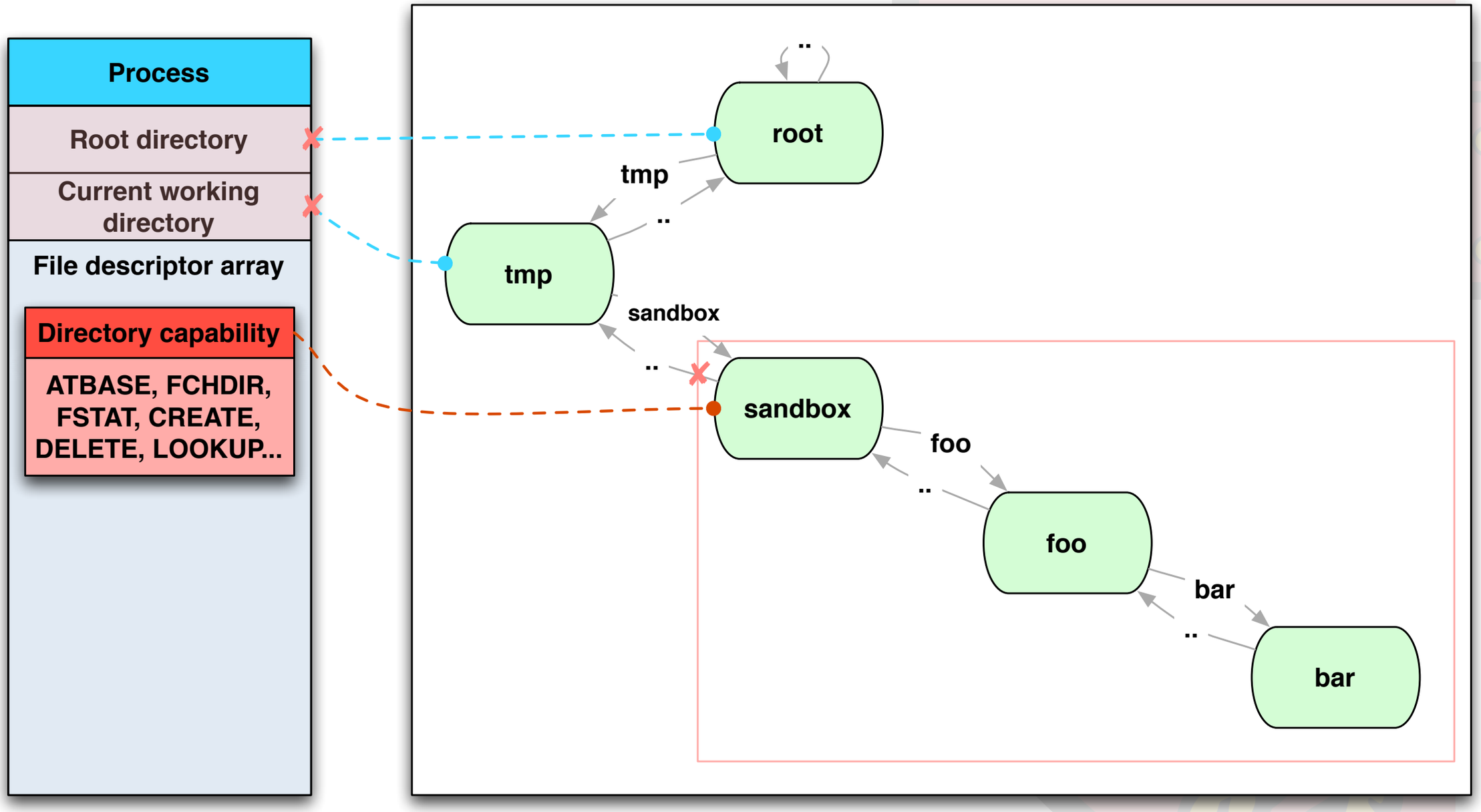
at(2) APIs

- Variations accepting directory descriptors:

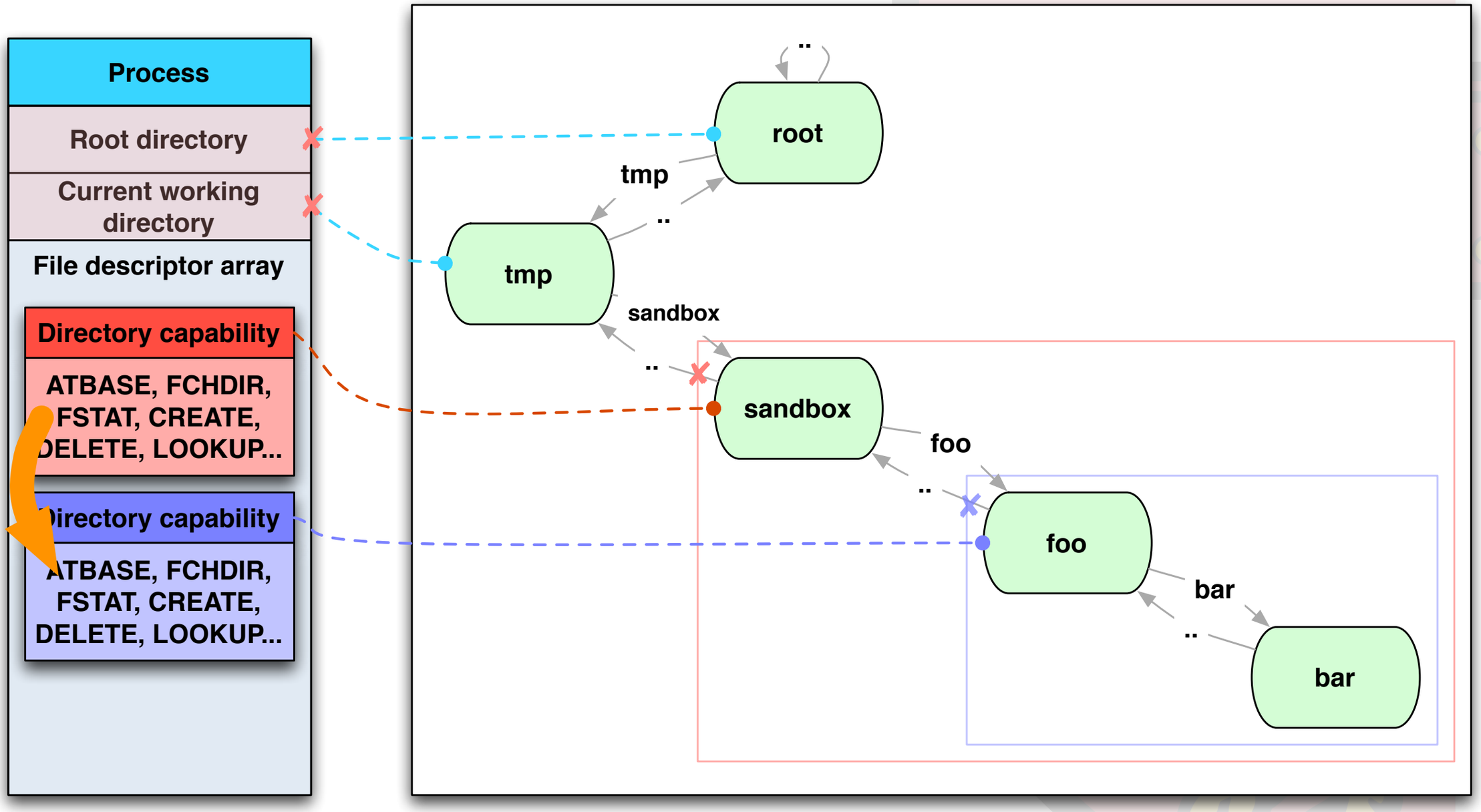
```
int renameat(int fromfd, char *from,
             int tofd, char *to);
```

- Avoid intermediate lookup state/costs
- Use at(2) calls to delegate directories
- Grant rights to objects **under** capability

Directory delegation



Derived capabilities



Implementing *under*

- Reject at(2) on absolute paths
- Reuse existing **`namei`** lookup code
- Require directory capability argument
- If ".." is looked up relative to starting directory, return **`ENOTCAPABLE`**

A concurrency vulnerability

Concurrency

- Multiple computational processes **execute at the same time** and may **interact with each other**
- Concurrency leads to the **appearance of non-determinism**

Concurrency vulnerabilities

- When **incorrect concurrency management** leads to **vulnerability**
  - Violation of **specifications**
  - Violation of **user expectations**
  - **Passive** - leak information or privilege
  - **Active** - allow adversary to extract information, gain privilege, deny service...

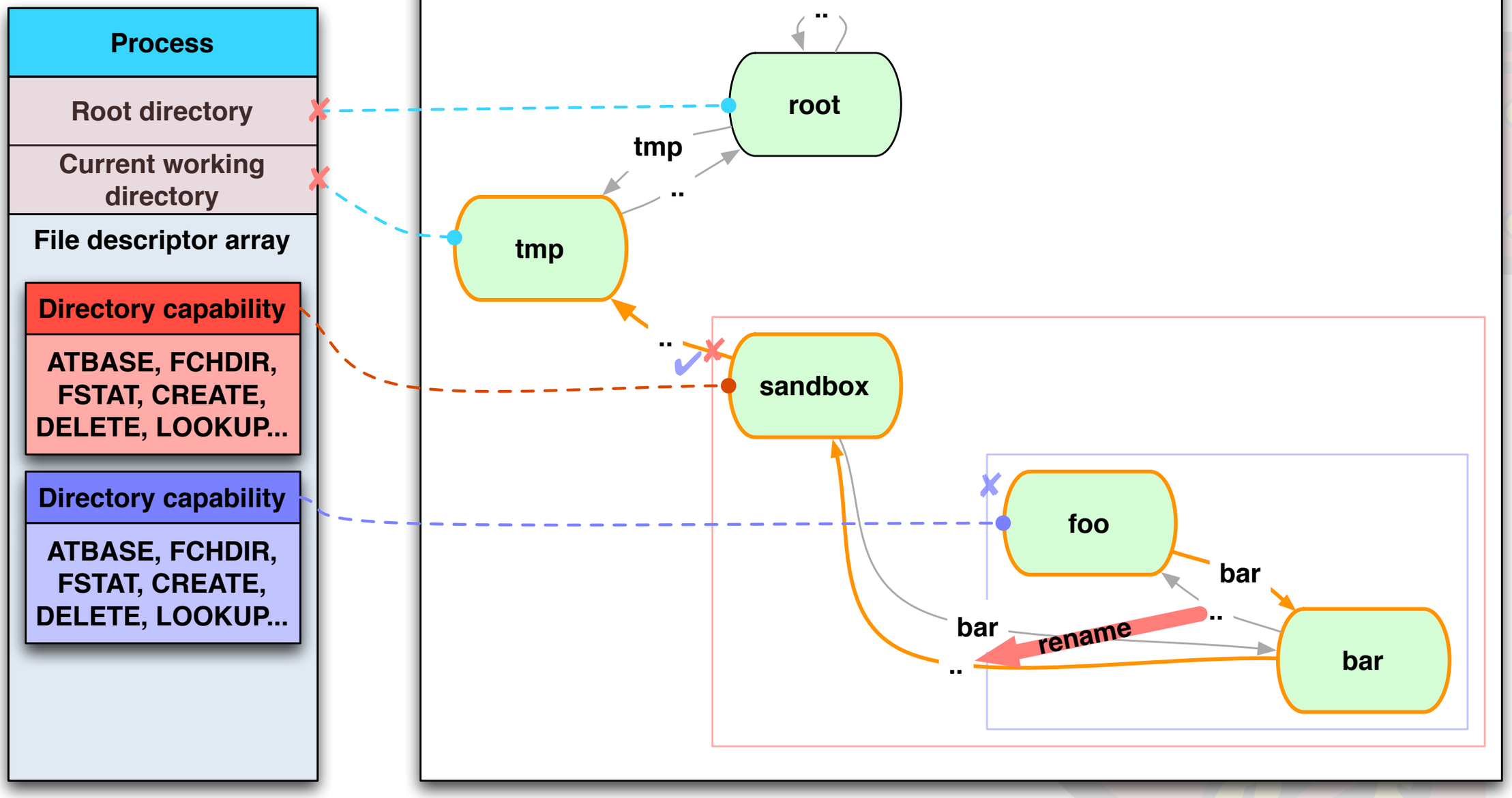
From concurrency bug to security bug

- Vulnerabilities in security-critical interfaces
  - Races on arguments and interpretation
    - Atomic “check” and “access” not possible
- Data consistency vulnerabilities
  - Stale or inconsistent security metadata
  - Security metadata and data inconsistent

Concurrency attacks on APIs

- System call API bridges "untrusted" userspace and "trusted" kernel
- Attacker's goal to manipulate APIs and trigger security incorrectness in "trusted" implementation
- In software, usually done using multiple client threads/processes and system calls, LPCs, or RPCs to a multithreaded server

openat(foofd, "bar/../..");
renameat(foofd, "bar", sandboxfd, "bar");



Concurrency vulnerabilities

- Most race conditions are time-of-check-to-time-of-use (TOCTTOU)
- This vulnerability is **not** TOCTTOU
- Bisbey 1978 "unexpected concurrency"
- Security failure due to programmer not understanding concurrency opportunity

The vulnerability

- Bypass containment if any writable directory capabilities passed to sandbox
- Dual-core notebook required ~100,000 loops to exploit
- Exploits non-atomic namespace lookup relative to other operations
  - A performance feature we can't remove!

Why formal methods?

- Serious but subtle concurrency vulnerability with unclear implications
- Namespace containment widely used in UNIX; `chroot` and beyond
- Want to show that other combinations of name space calls can't trigger similar vulnerabilities

Model checking

- Summary: Clarke, Emerson, Sifakis 2007 Turing Award lecture; Comm ACM 2009
- Finite state machine represents system under analysis
- Express safety properties in temporal logic
- Exhaustively check model conformance
- Common in protocol, hardware verification

The goal

- Model the relationship between the attacker and the file system implementation
- Want to explore all possible interleavings of events the attacker can trigger
- Validate critical assertions

The model

- Selected SPIN model checker
- 222-line Promela model of system/attacker
- Model a finite set of concurrent processes, each with a limited system call vocabulary
- Finite-sized file system (8 nodes);
  Initial path configuration similar to picture
- Assertion: multi-“..” lookups fail

Solution space

Approach	Performance	Functionality	Security
Remove subtree delegation	✔	✘	✔
Namespace walk	✘	✔	✘
Limit namespace concurrency	✘	✔	✔ (NFS: ✘)
Limit “..”	✔	✘	✔

Limitations

- Hand-crafted Promela mode - significantly different semantics and implementation from kernel code
- Finite process count
- Limited system call vocabulary
- Limited file system size
- Want stronger “can’t name root” assertion

Conclusion

- Capsicum: practical capabilities for UNIX
- Concurrency vulnerability with serious real-world implications for Capsicum
- Applied model checking
- Improved our confidence in security / performance / functionality trade-off

Q&A