ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

DynaRace

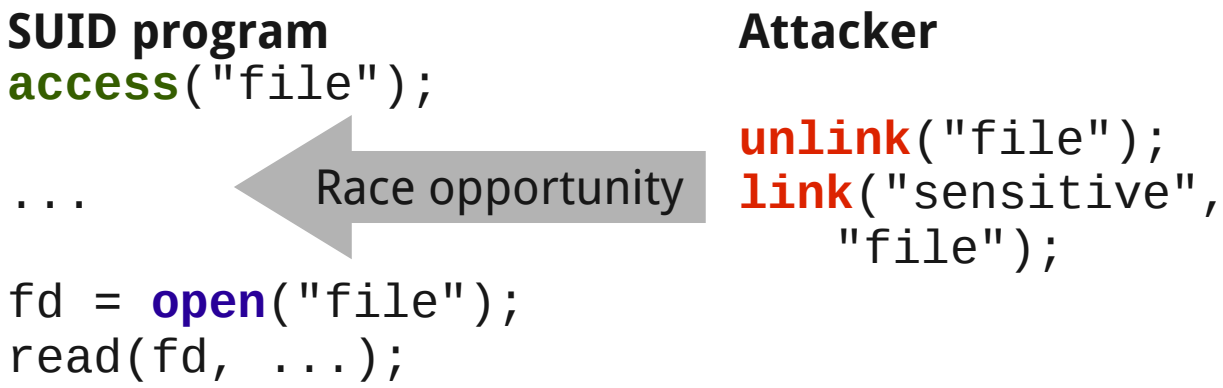
Protecting Applications Against TOCTTOU Races by User-Space Caching of File Metadata

Mathias Payer & Thomas R. Gross
Department of Computer Science
ETH Zürich, Switzerland

TOCTTOU races

Time Of Check To **Time of Use** (TOCTTOU) races for file accesses endanger integrity of applications

- The mapping between filename and inode is volatile
- Attacker uses delay between "**test**" and "**use**" system calls



Motivation: Protect applications

Protect unmodified applications from TOCTTOU races

Cache metadata for accessed files

- Check and verify metadata on all file accesses
- User-space implementation

Metadata cache links filenames and inodes

- Stop potential file-based race attacks

Close the door to one popular attack vector

Outline

Motivation

DynaRace key idea

- File states capture permissions
- File resolution ensures safety

Implementation

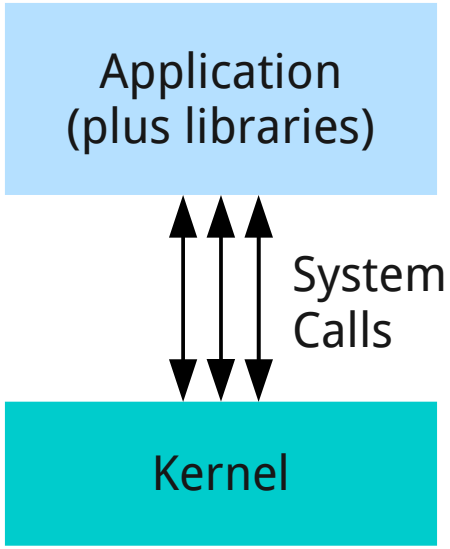
Evaluation

Related work

Conclusion

DynaRace key idea

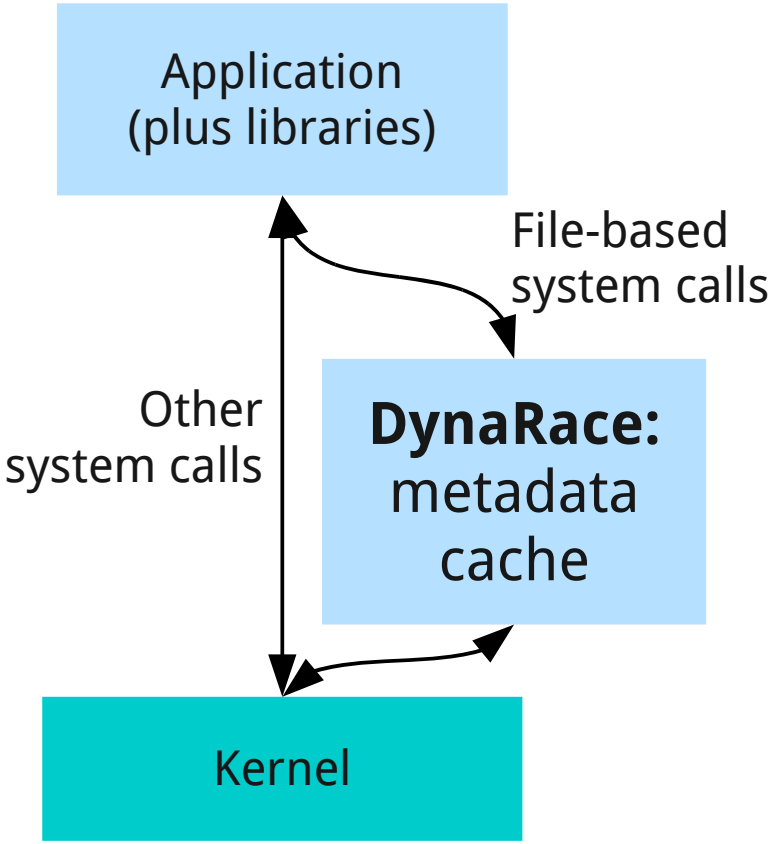
Keep state and metadata for all files



DynaRace key idea

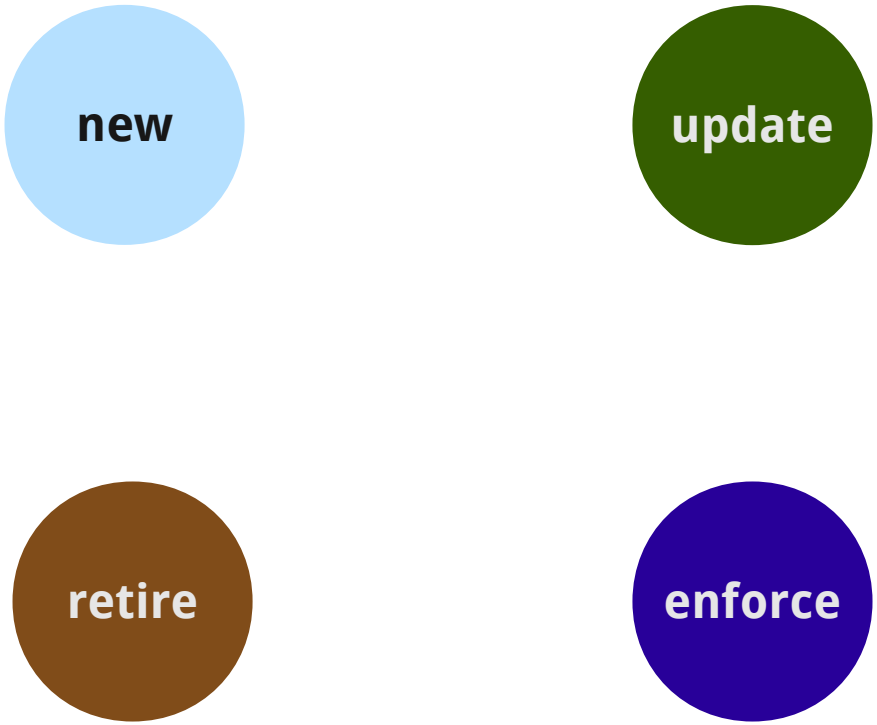
Keep state and metadata for all files

- Update metadata for new files
- Enforce metadata equality for known files



DynaRace file states

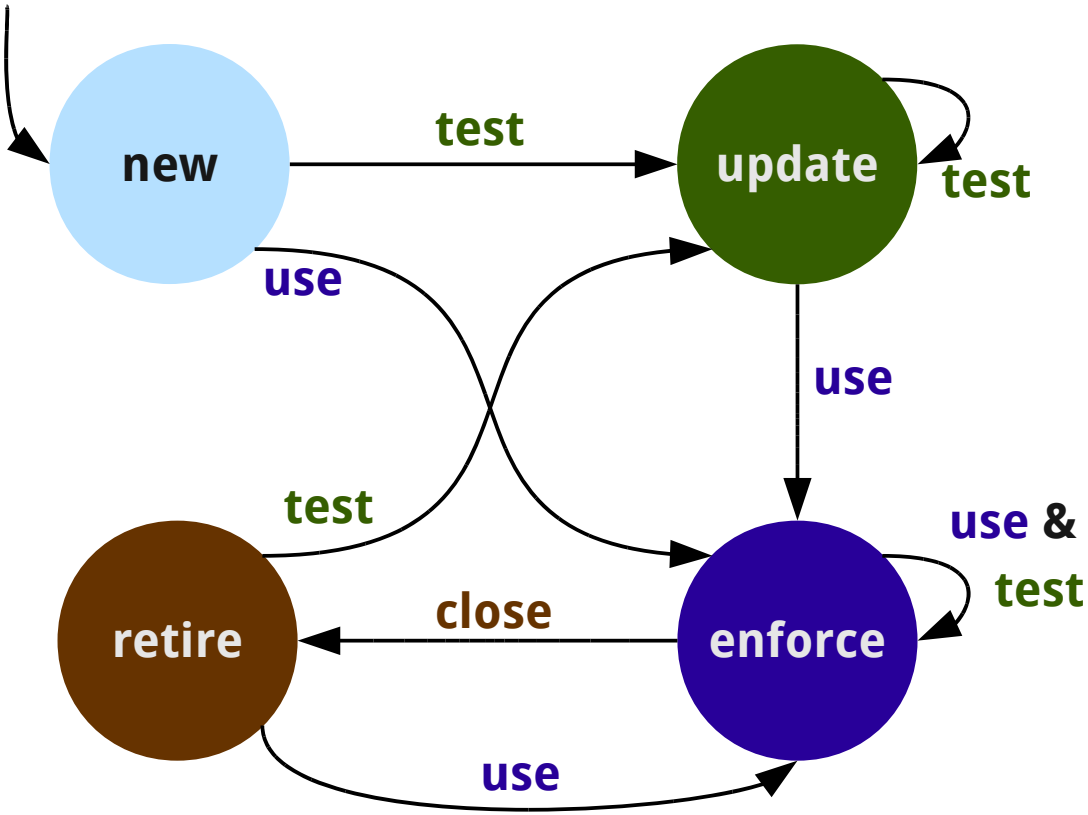
DynaRace keeps state for each accessed file



DynaRace file states

State transitions according to system calls groups

- **Test**: check a property, e.g., `access`, or `stat`
- **Use**: work with files, e.g., `open`, or `chmod`
- **Close**: retire files, e.g., `close`, or `unlink`

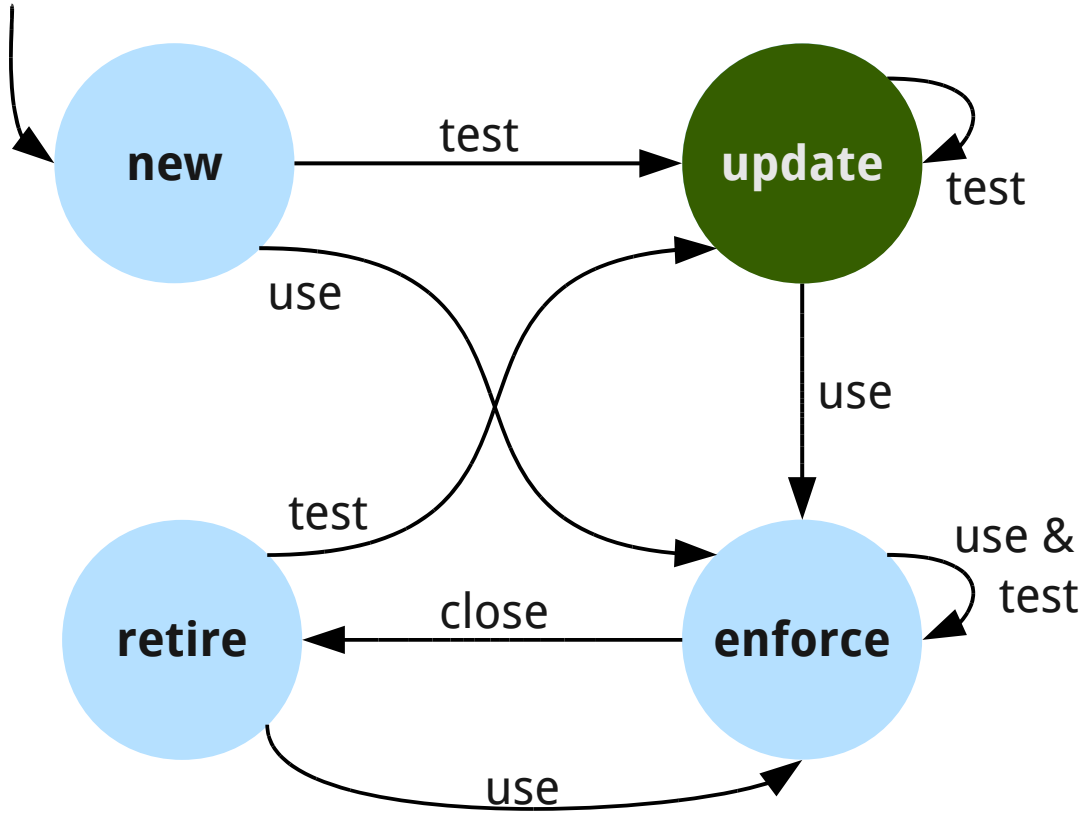


DynaRace file states: Example

SUID program

```
access("file");
...
fd = open("file");
read(fd, ...);
close(fd);
```

Metadata file cache:
`file` in `/tmp` [**update**]

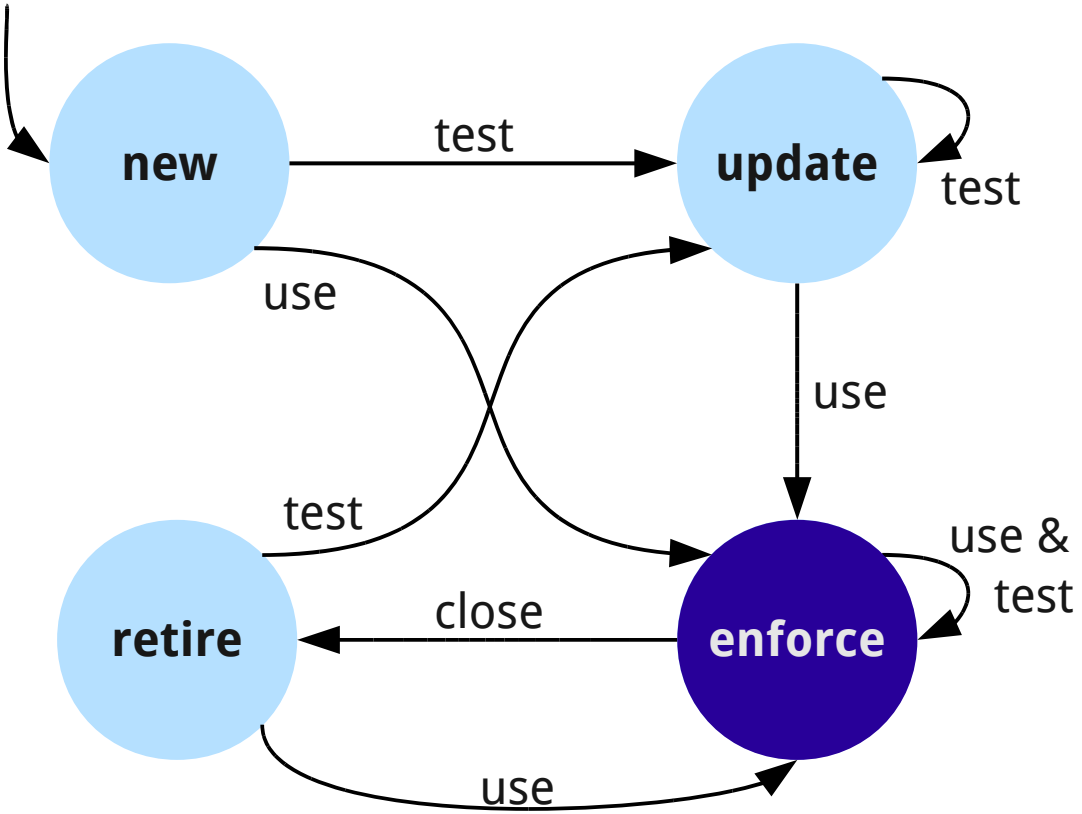


DynaRace file states: Example

SUID program

```
access("file");
...
fd = open("file");
read(fd, ...);
close(fd);
```

Metadata file cache:
`file` in `/tmp` [**enforce**]

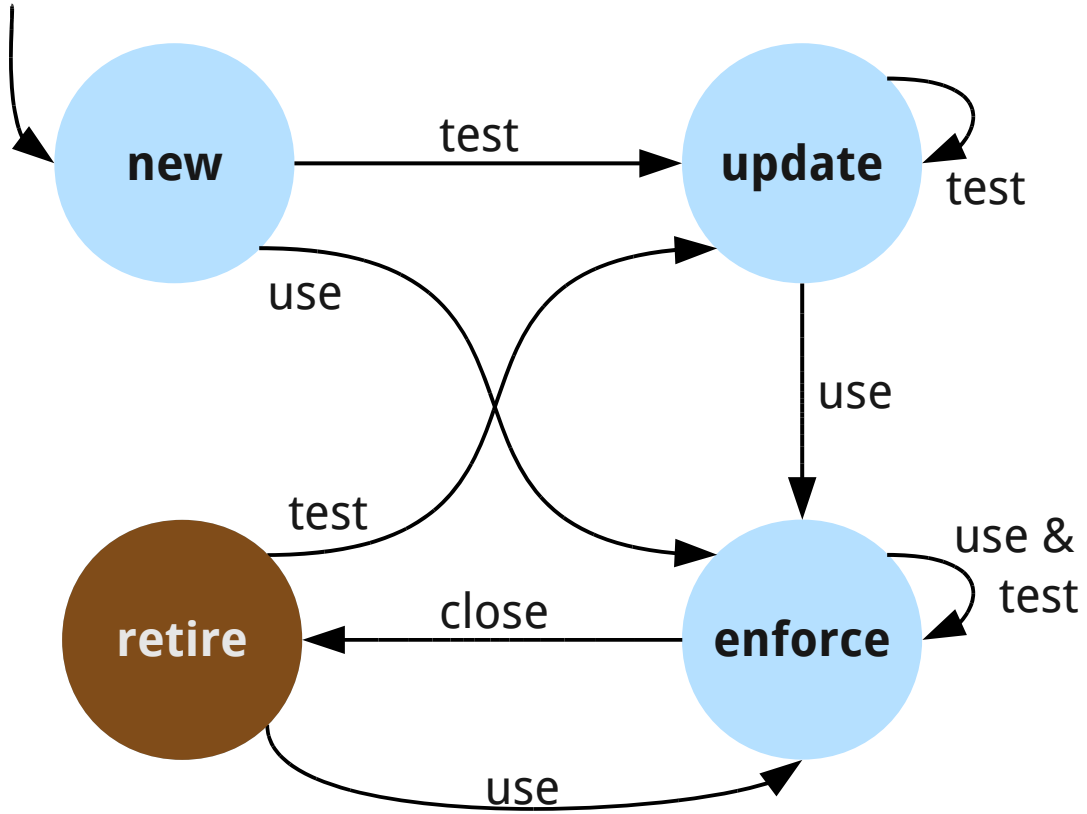


DynaRace file states: Example

SUID program

```
access("file");
...
fd = open("file");
read(fd, ...);
close(fd);
```

Metadata file cache:
`file` in `/tmp` [**retire**]

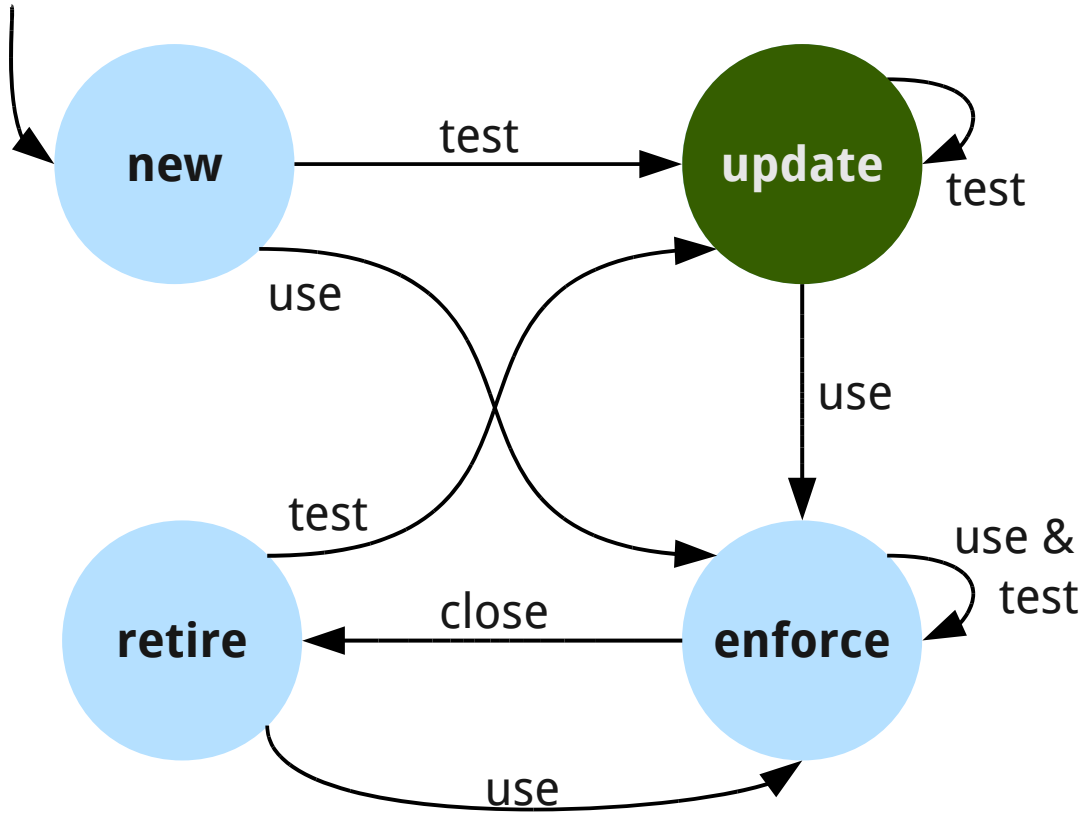


DynaRace file states: Example 2

SUID program

```
access("file");
...
fd = open("file");
read(fd, ...);
close(fd);
```

Metadata file cache:
`file` in `/tmp` [**update**]



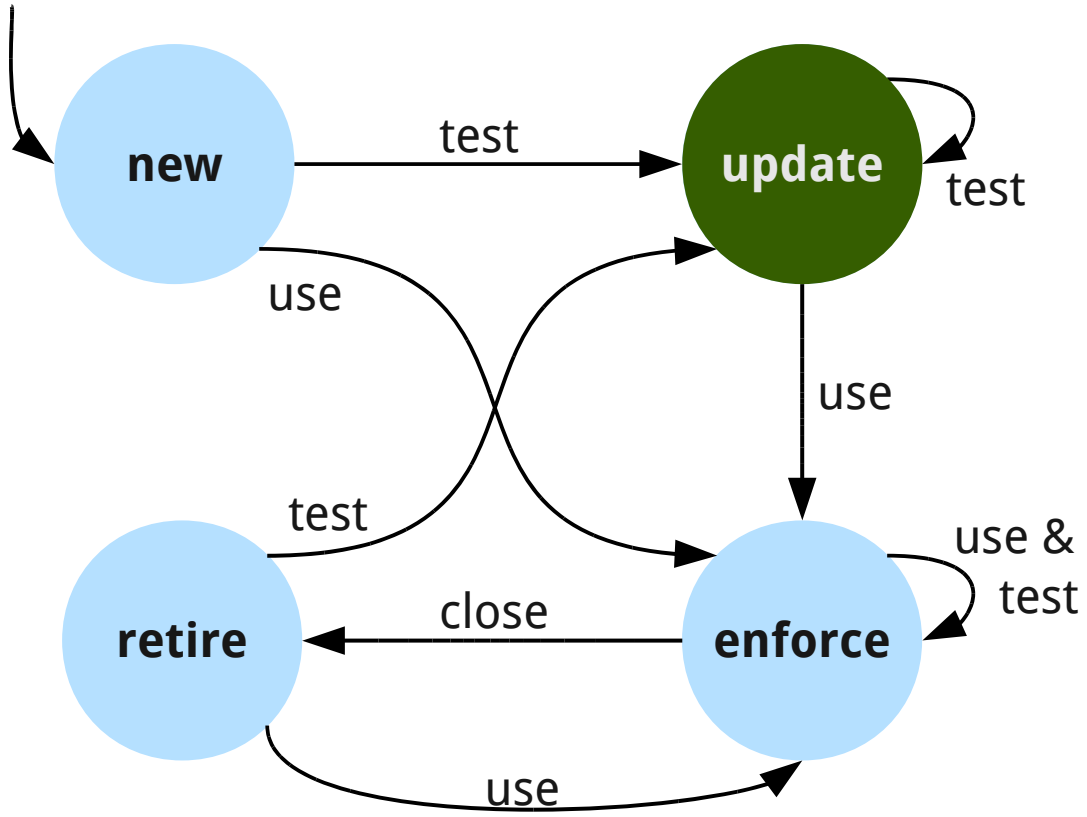
DynaRace file states: Example 2

Race

SUID program

```
access("file");
...
fd = open("file");
read(fd, ...);
close(fd);
```

Metadata file cache:
`file` in `/tmp` [**update**]



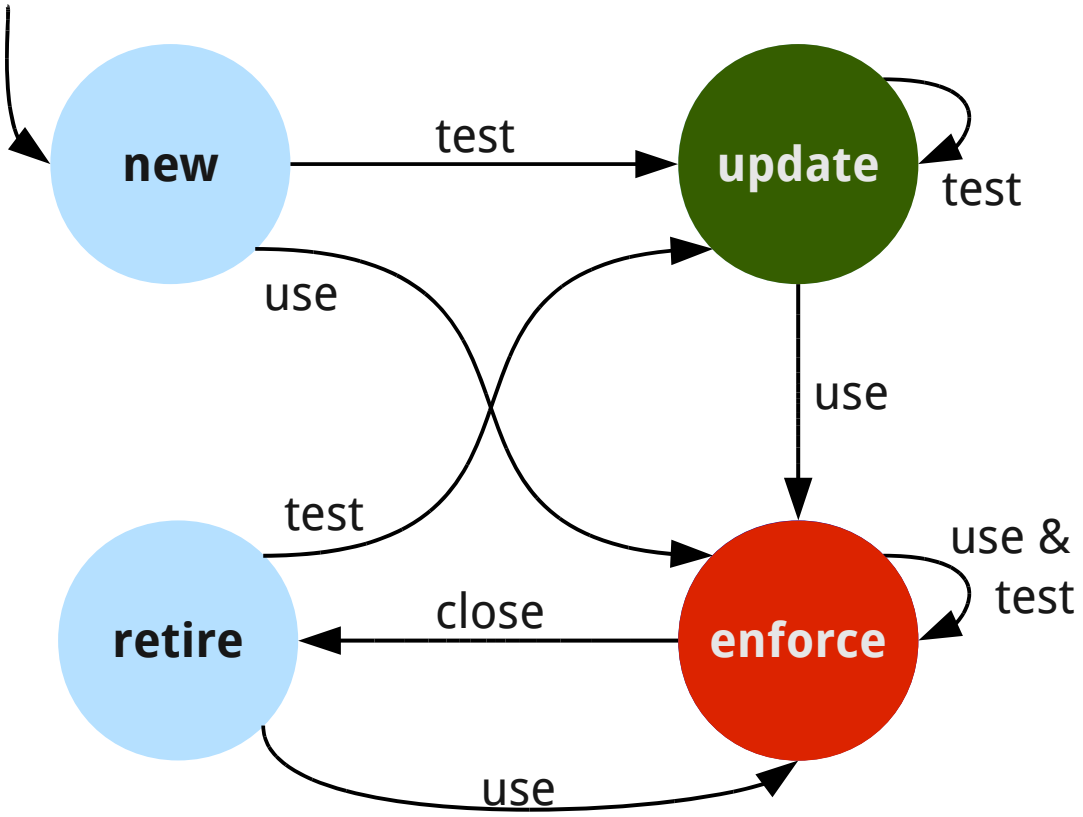
DynaRace file states: Example 2

Race

SUID program

```
access("file");
...
fd = open("file");
read(fd, ...);
close(fd);
```

Metadata file cache:
`file` in `/tmp` [**enforce**]



DynaRace file resolution

Resolve files in race-free manner*

- Resolve the path **atom** by **atom**
- Check if the atom is in the cache
  - Enforce metadata according to state
- Update atom's metadata
- Use recursion to follow links

```
Resolving
/tmp/.X0-lock

/
tmp/ in /
.X0-lock in /tmp/
```

* Files are resolved similar to the `check_use` mechanism by Tsafrir et al. [FAST'08, IBM TR RC24572]

Outline

Motivation

The DynaRace approach

Implementation

Evaluation

Related work

Conclusion

DynaRace prototype implementation

Prototype implementation uses user-space virtualization

- Additional virtualization layer between application and OS

Libdetox* rewrites executed application code

- File-based system calls replaced with DynaRace functions
- Metadata and state cache in VM layer
- Linux x86 implementation

* Libdetox implements software-based fault isolation using dynamic BT by Payer et al. [VEE'11]

DynaRace prototype implementation

Libdetox

- Total loc: 15'130
  - Translation tables loc: 4'907
- Comments: 5'015

DynaRace (for subset of system calls)

- Total loc: 441
- Comments: 372
- Changes to libdetox per redirected system call: 2 loc

Outline

Motivation

The DynaRace approach

Implementation

Evaluation

- Apache performance
- X.org bug study

Related work

Conclusion

Apache performance

Apache 2.2 on Ubuntu 10.04 LTS using `ab` benchmark

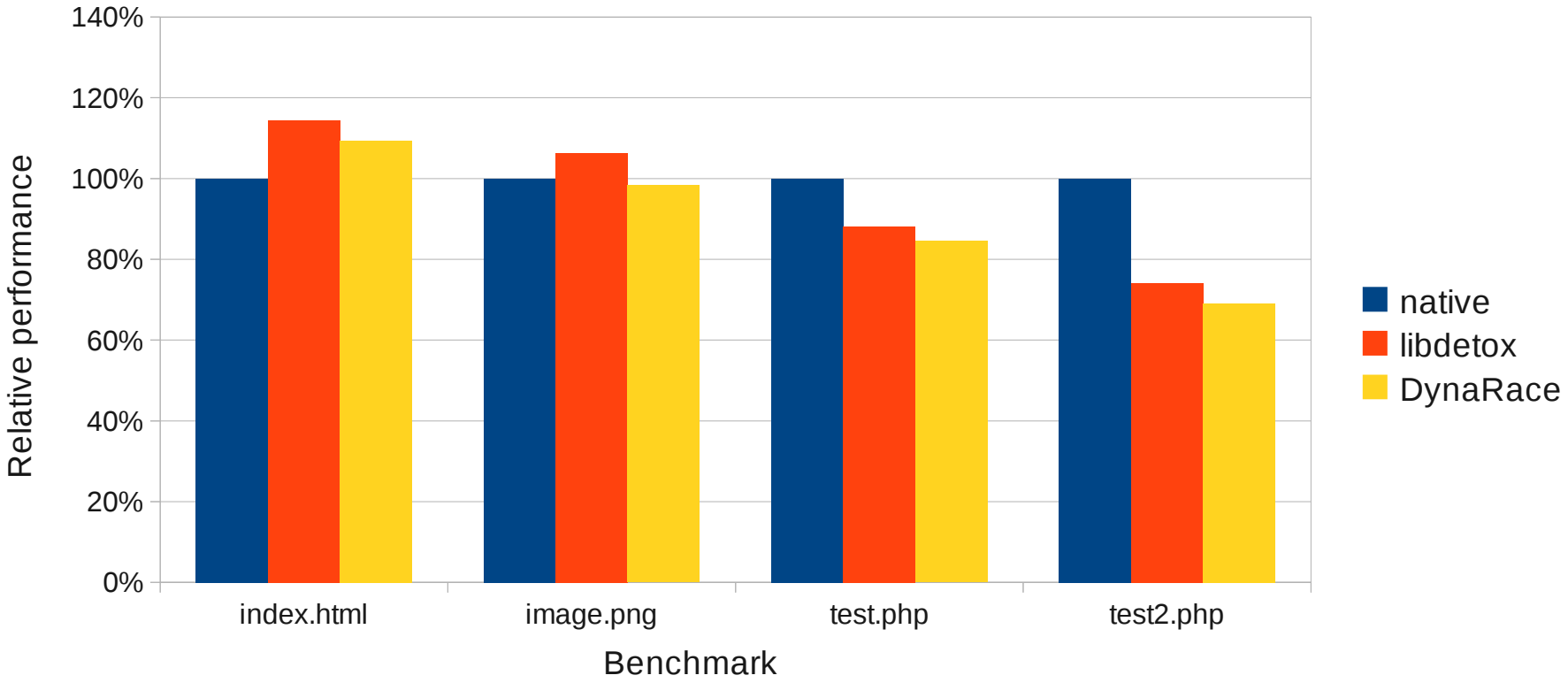
- Core i7 950 CPU @ 3.07GHz, in 32bit x86 mode
- `ab` executes with two concurrent connections
- Each file is downloaded 100,000 times
  - `index.html` 5kB HTML
  - `image.png` 1MB raw data
  - `test.php` short PHP script (90B output)
  - `test2.php` long PHP script (49kB output)

Apache performance

3 different configurations:

- Native: native, unmodified execution of Apache
- Libdetox: Apache running in Libdetox sandbox
- DynaRace: Libdetox + DynaRace protection

Apache performance



Overhead of DynaRace comparable to libdetox

Speedup due to better code layout

Overall performance penalty is tolerable

X.org security exploit

X.org protected with DynaRace

P1

```
...
lfd = open(tmp, O_CREAT|O_EXCL|O_WRONLY, 0644);
...
if(lfd < 0) {
  unlink(tmp);
}
...
write(lfd, pid_str, 11);
/* unchecked relaxation */
chmod(tmp, 0444);
...
```

tmp lock file: `/tmp/.X0-lock`

P1 metadata file cache:
`.X0-lock` in `/tmp` [**enforce**]

* in `os/utils.c` [CVE-2011-4029]

X.org security exploit

X.org protected with DynaRace

```
...
lfd = open(tmp, O_CREAT|O_EXCL|O_WRONLY, 0644);
...
if(lfd < 0) {
  unlink(tmp);
}
...
write(lfd, pid_str, 11);
/* unchecked relaxation */
chmod(tmp, 0444);
...
```

P2

P1 *zzz*

tmp lock file: `/tmp/.X0-lock`

P1 metadata file cache:
`.X0-lock` in `/tmp` [**enforce**]

P2 metadata file cache:
`.X0-lock` in `/tmp` [**enforce**]

* in `os/utils.c` [CVE-2011-4029]

X.org security exploit

X.org protected with DynaRace

```
...
lfd = open(tmp, O_CREAT|O_EXCL|O_WRONLY, 0644);
...
if(lfd < 0) {
  unlink(tmp);
}
...
write(lfd, pid_str, 11);
/* unchecked relaxation */
chmod(tmp, 0444);
...
```

P x

P1 *zzz*

tmp lock file: `/tmp/.X0-lock`
File removed by P2

P1 metadata file cache:
`.X0-lock` in `/tmp` [**enforce**]

P2 metadata file cache:
`.X0-lock` in `/tmp` [**retire**]

* in `os/utils.c` [CVE-2011-4029]

X.org security exploit

X.org protected with DynaRace

```
...
lfd = open(tmp, O_CREAT|O_EXCL|O_WRONLY, 0644);
...
if(lfd < 0) {
  unlink(tmp);
}
...
write(lfd, pid_str, 11);
/* unchecked relaxation */
chmod(tmp, 0444);
...
```

P1 *zzz*

Attacker links **`/tmp/.X0-lock`** to a sensitive file (e.g., **`/etc/shadow`**)

P1 metadata file cache:
`.X0-lock` in `/tmp` [**enforce**]

* in `os/utils.c` [CVE-2011-4029]

X.org security exploit

X.org protected with DynaRace

```
...
lfd = open(tmp, O_CREAT|O_EXCL|O_WRONLY, 0644);
...
if(lfd < 0) {
  unlink(tmp);
}
...
write(lfd, pid_str, 11);
/* unchecked relaxation */
chmod(tmp, 0444);
...
```

P1

tmp lock file: **`/tmp/.X0-lock`** links to **`/etc/shadow`**

P1 metadata file cache:
`.X0-lock` in `/tmp` [**enforce**]

Metadata mismatch for `.X0-lock`
P1 is terminated with race exception
Attacker is not successful

* in `os/utils.c` [CVE-2011-4029]

Outline

Motivation

The DynaRace approach

Implementation

Evaluation

Related work

Conclusion

Related work

Mazières and Kaashoek change OS to support inode-based file access [HotOS'97]

- Implemented as new system calls

Tsafrir et al. implement safe user-space path resolution [FAST'08, IBM TR RC24572]

- Safe path resolution needs program changes

Chari et al. ensure that given path elements are safe to open by the current user [NDSS'10]

- Introduces manipulators as new concept, needs program changes

More related work in the paper

Conclusion

DynaRace protects unmodified applications from file-based TOCTTOU races

- Files checked depending on state and metadata cache
- Enforces metadata equality for currently used files

Binary translator rewrites unsafe system calls

- User-space cache protects application

Removes the burden of race protection from the programmer

Thank you for your attention

?

Implementation alternatives

Kernel implementation

- No BT overhead
- Additional code & complexity in kernel

libc-based implementation

- No BT overhead
- Potential coverage problem

Ptrace-based implementation

- Easy interception of system calls
- Injecting code for DynaRace system call replacements is difficult

Apache performance

	native*	libdetox**	DynaRace**
index.html	1464	-14.5%	-9.4%
image.png	48	-6.3%	1.6%
test.php	1773	11.9%	15.5%
test2.php	463	25.9%	30.9%

* requests per second
** relative overhead/speedup compared to native