Automatic Placement of Authorization Hooks in the Linux Security Modules Framework

Vinod Ganapathy

vg@cs.wisc.edu

University of Wisconsin, Madison

Joint work with

Trent Jaeger

tjaeger@cse.psu.edu

Pennsylvania State University

Somesh Jha

jha@cs.wisc.edu

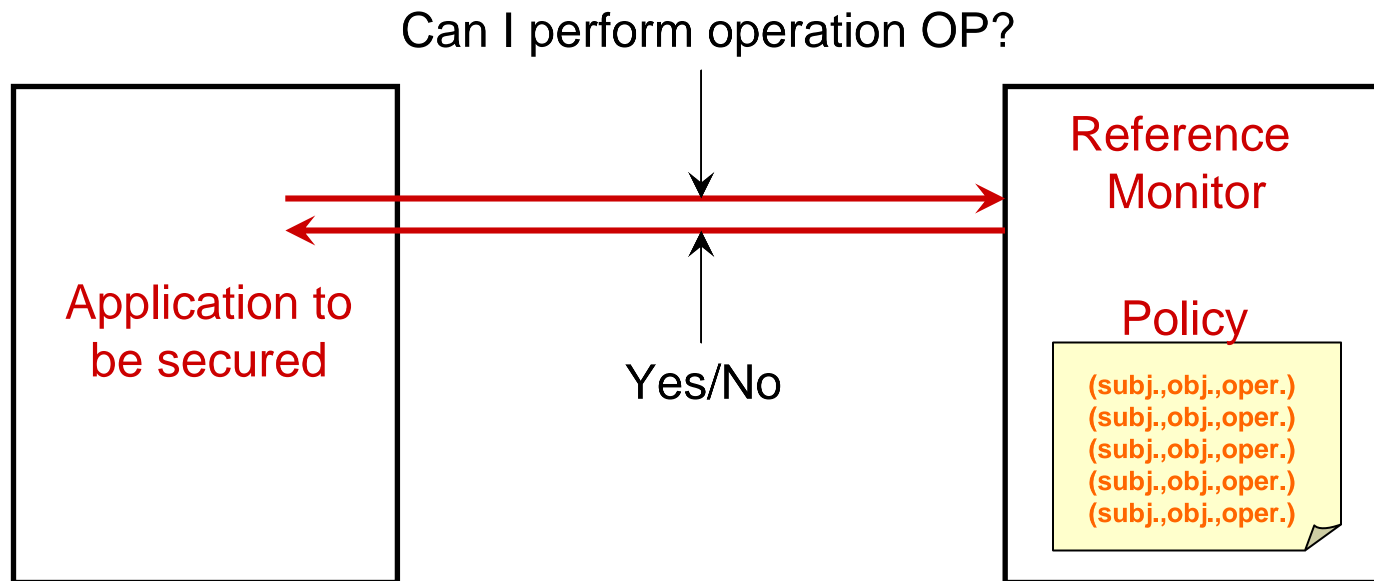
University of Wisconsin, Madison

Context of this talk

- Authorization policies and their enforcement
- Three concepts:
  - *Subjects* (e.g., users, processes)
  - *Objects* (e.g., system resources)
  - Security-sensitive *operations* on objects.
- Authorization policy:
  - A set of triples: (Subject, Object, Operation)
- Key question: How to ensure that the authorization policy is enforced?

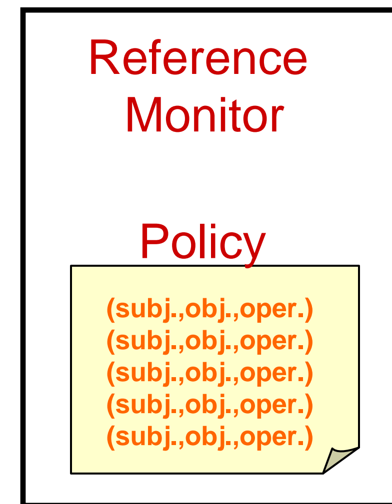
Enforcing authorization policies

- Reference monitor consults the policy.
- Application queries monitor at appropriate locations.



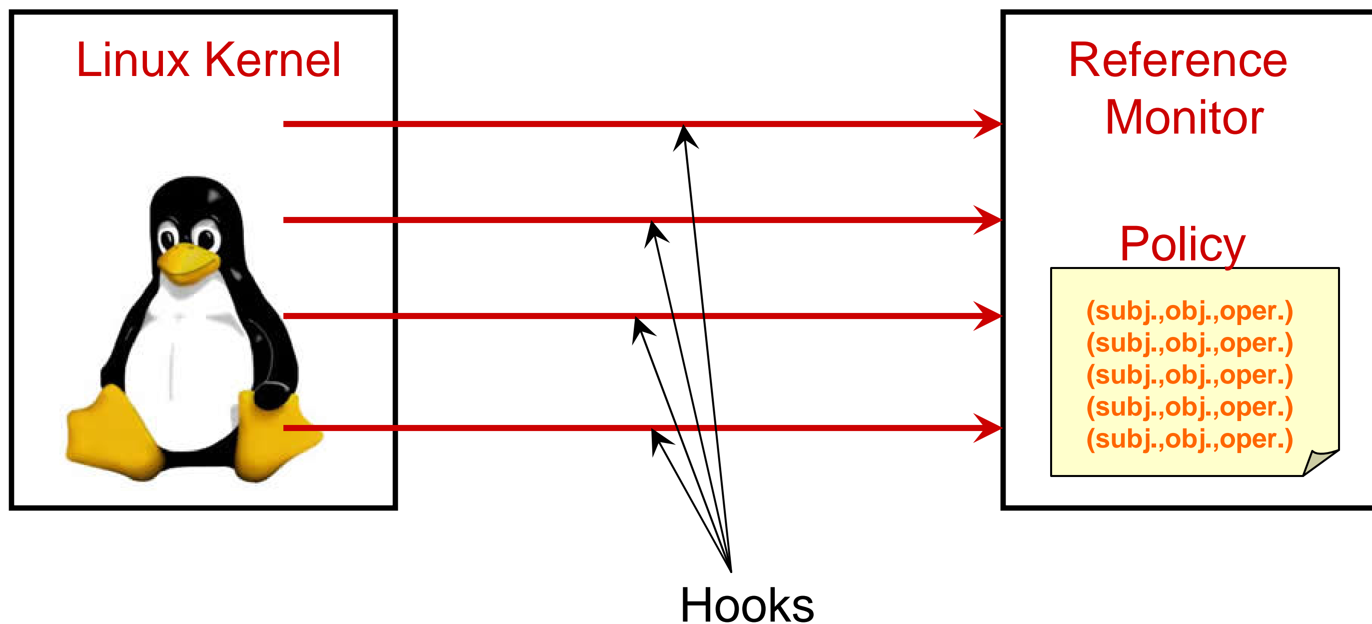
Linux security modules framework

- Framework for authorization policy enforcement.
- Uses a reference monitor-based architecture.
- Integrated into Linux-2.6



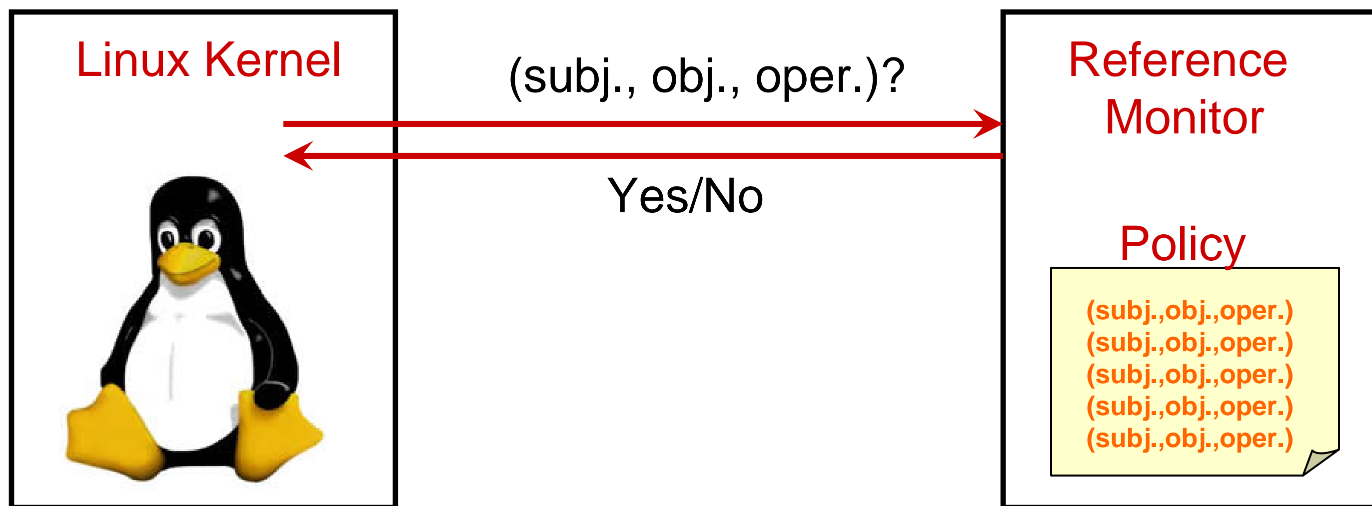
Linux security modules framework

- Reference monitor calls (*hooks*) placed appropriately in the Linux kernel.
- Each hook is an authorization query.



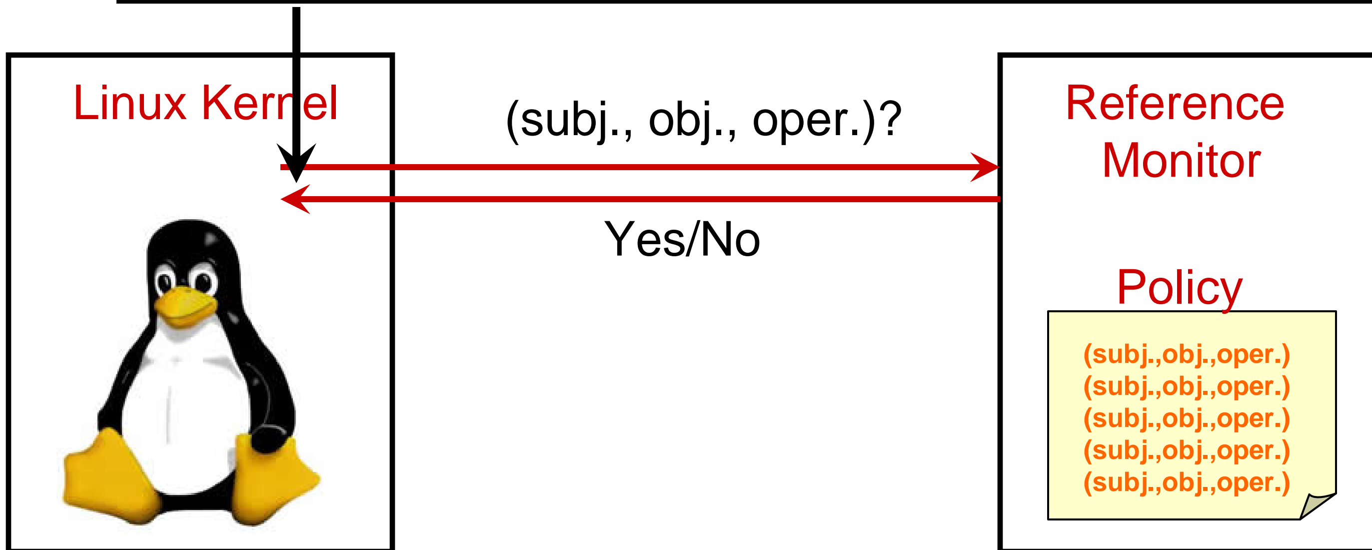
Linux security modules framework

- Authorization query of the form: (subj., obj., oper.)?
- Kernel performs operation only if query succeeds.



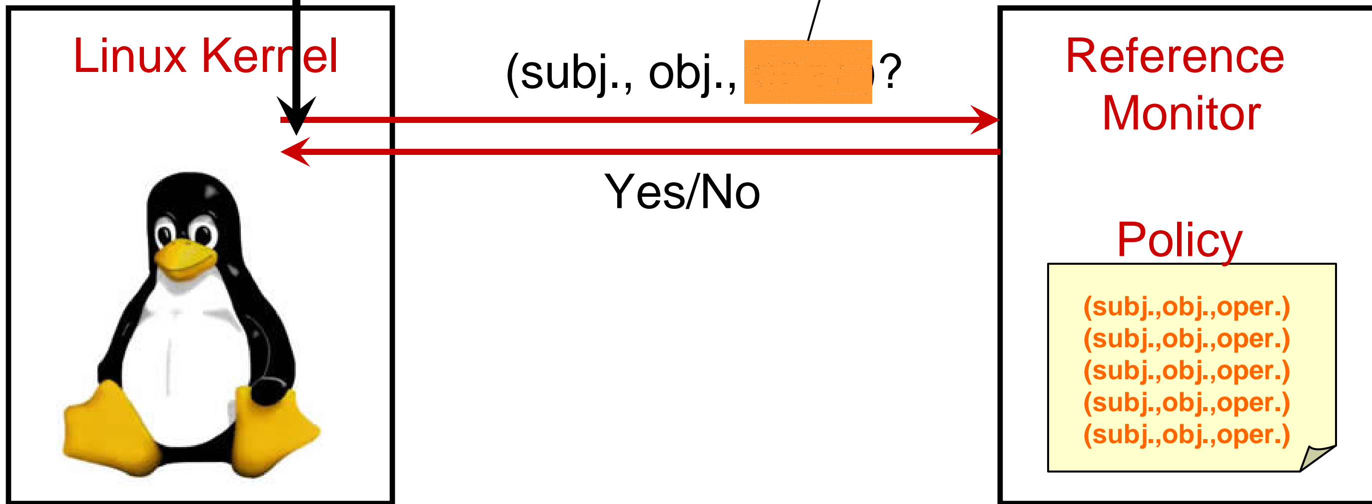
Virtual File System Code for Directory Removal

```
int vfs_rmdir(inode *dir, dentry *dentry) {
    …
    err = security_inode_rmdir(dir,dentry);
    if (!err) {
        dir->i_op->rmdir(dir,dentry);
    }
}
```



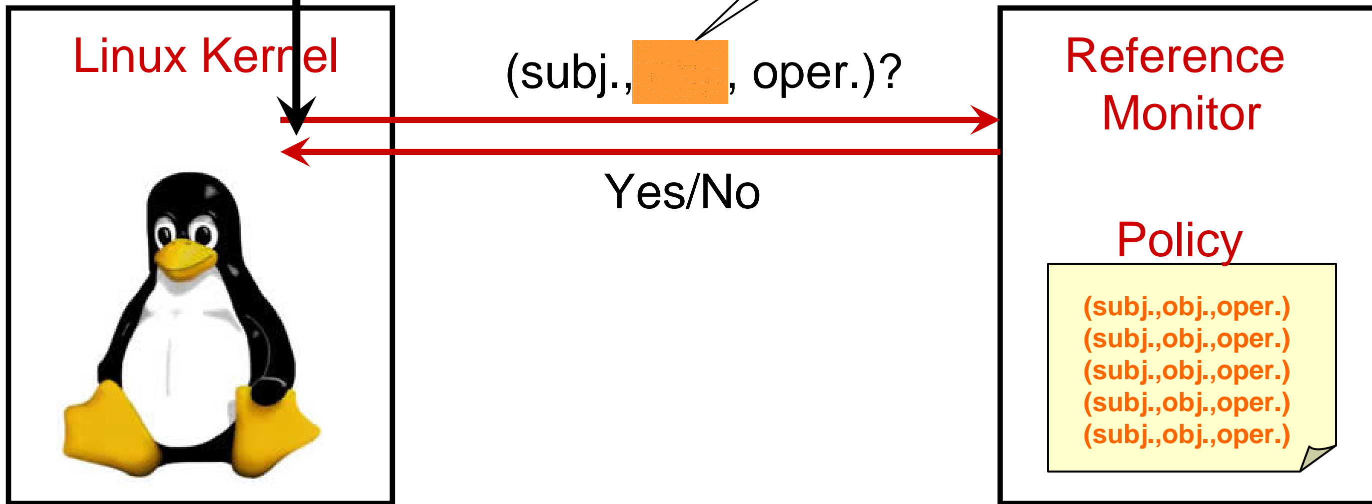
Virtual File System Code for Directory Removal

```
int vfs_rmdir(inode *dir, dentry *dentry) {
    …
    err = security_inode_rmdir(dir,dentry);
    if (!err) {
        dir->i_op->rmdir(dir,dentry);
    }
}
```



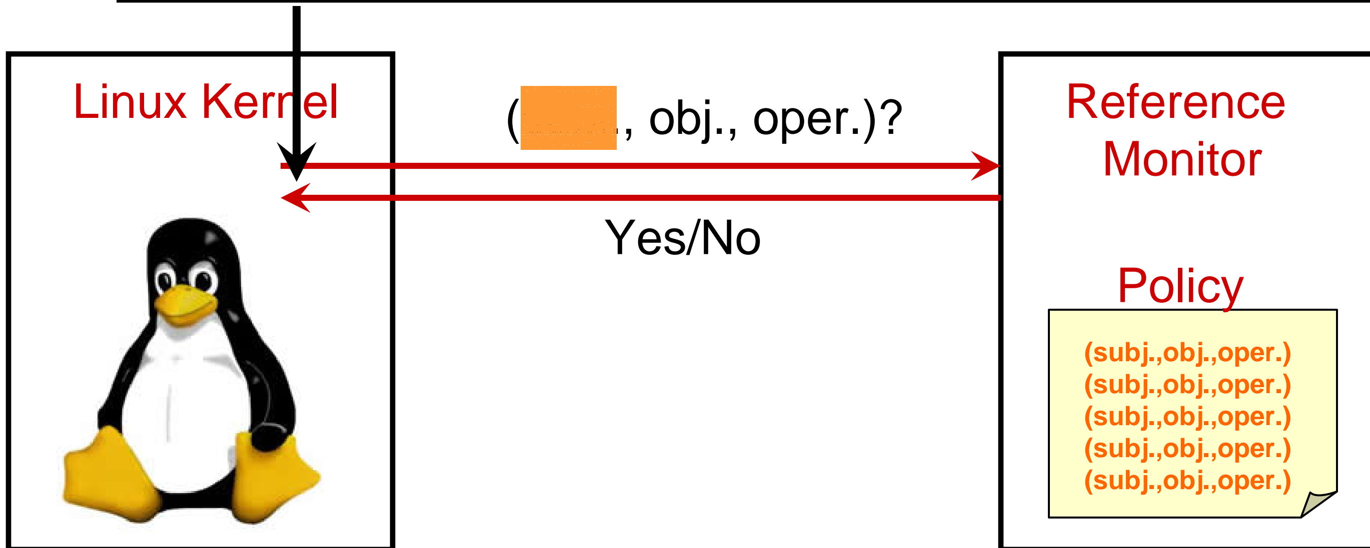
Virtual File System Code for Directory Removal

```
int vfs_rmdir(inode *dir, dentry *dentry) {
    …
    err = security_inode_rmdir(dir,dentry);
    if (!err) {
        dir->i_op->rmdir(dir,dentry);
    }
}
```



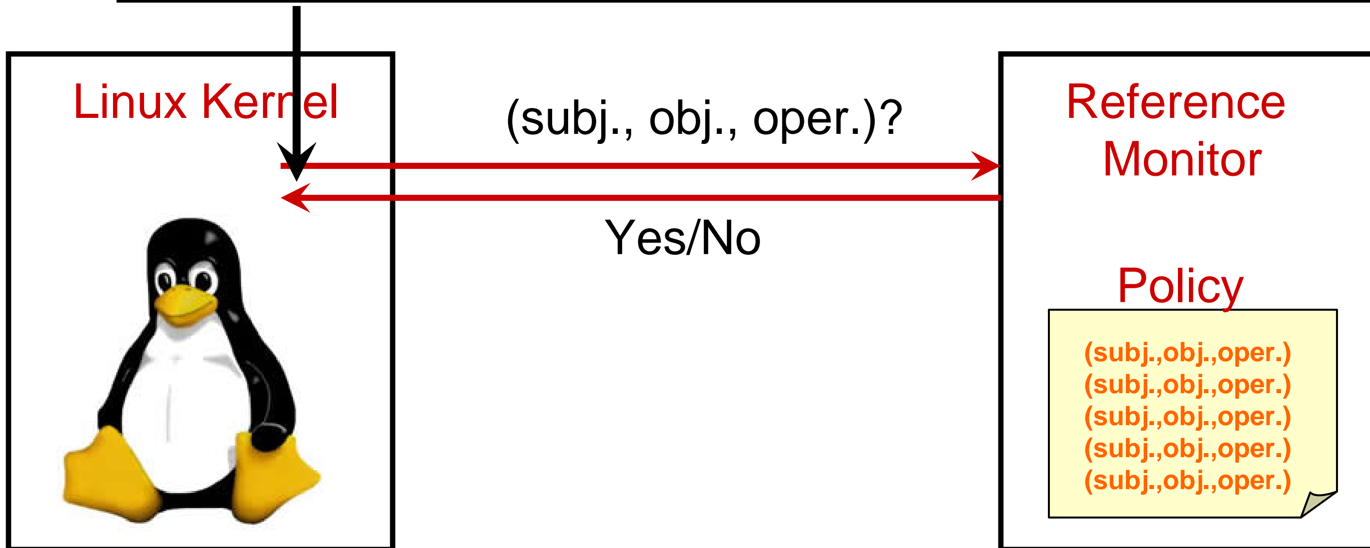
Virtual File System Code for Directory Removal

```
int vfs_rmdir(inode *dir, dentry *dentry) {
    …
    err = security_inode_rmdir(dir,dentry);
    if (!err) {
        dir->i_op->rmdir(dir,dentry);
    }
}
```



Virtual File System Code for Directory Removal

```
int vfs_rmdir(inode *dir, dentry *dentry) {
    …
    err = security_inode_rmdir(dir,dentry);
    if (!err) {
        dir->i_op->rmdir(dir,dentry);
    }
}
```



Key: Hooks must achieve *complete mediation.*

Hook placement is crucial

- Must achieve complete mediation.
  - Security-sensitive operations must be mediated by a hook that authorizes the operation.
- Current practice:
  - Hooks placed manually in the kernel.
  - Takes a long time: approx. 2 years for Linux security modules framework.
- Can this achieve complete mediation?
  - Prior work has found bugs in hook placement.
    [Zhang *et al.,* USENIX Security 2002, Jaeger *et al.,* ACM CCS 2002]

Main message of this talk

Static analysis can largely automate authorization hook placement and achieve complete mediation

Main message of this talk

Static analysis can largely automate authorization hook placement and achieve complete mediation

Reduces turnaround time of Linux Security Modules-like projects

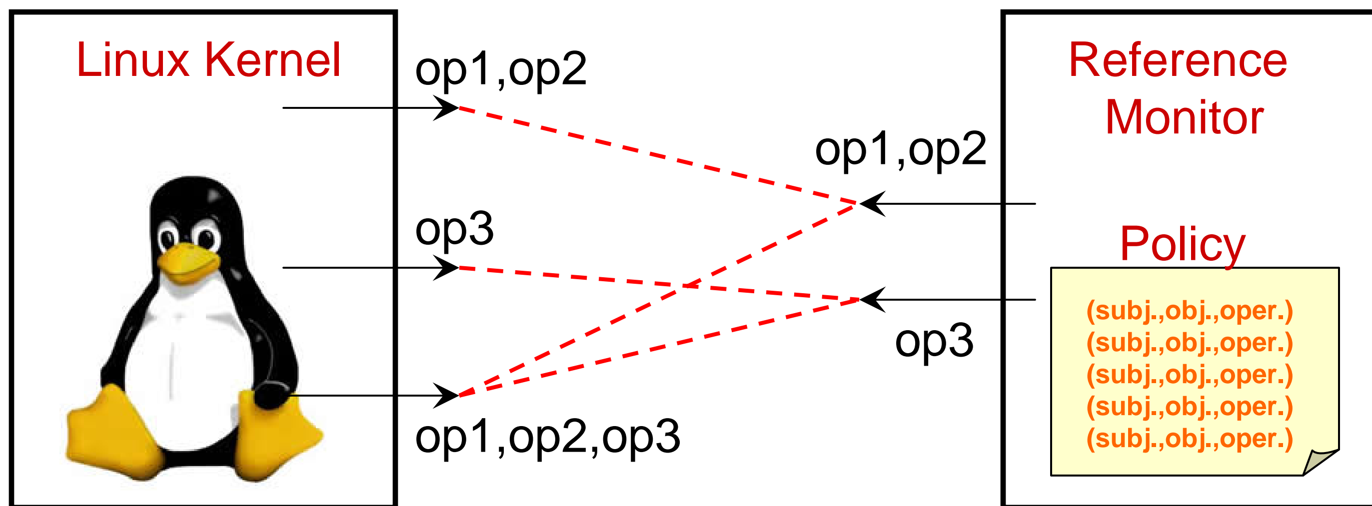
Main message of this talk

Static analysis can largely automate authorization hook placement and achieve <u>complete mediation</u>

Towards correctness by construction

Key intuition: Matchmaking

- Each kernel function performs an operation.
- Each hook authorizes an operation.
- Match kernel functions with appropriate hooks.

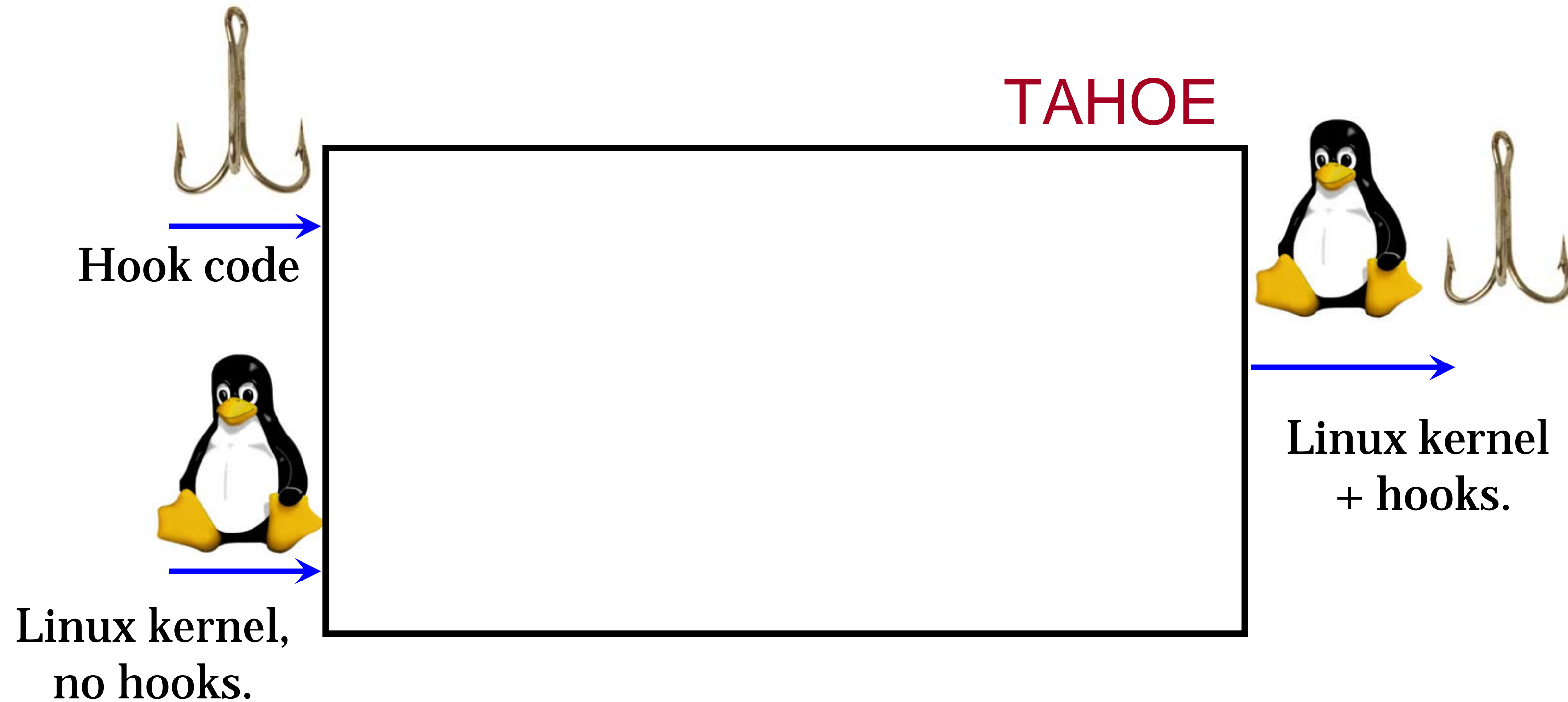


Tool for Authorization Hook Placement

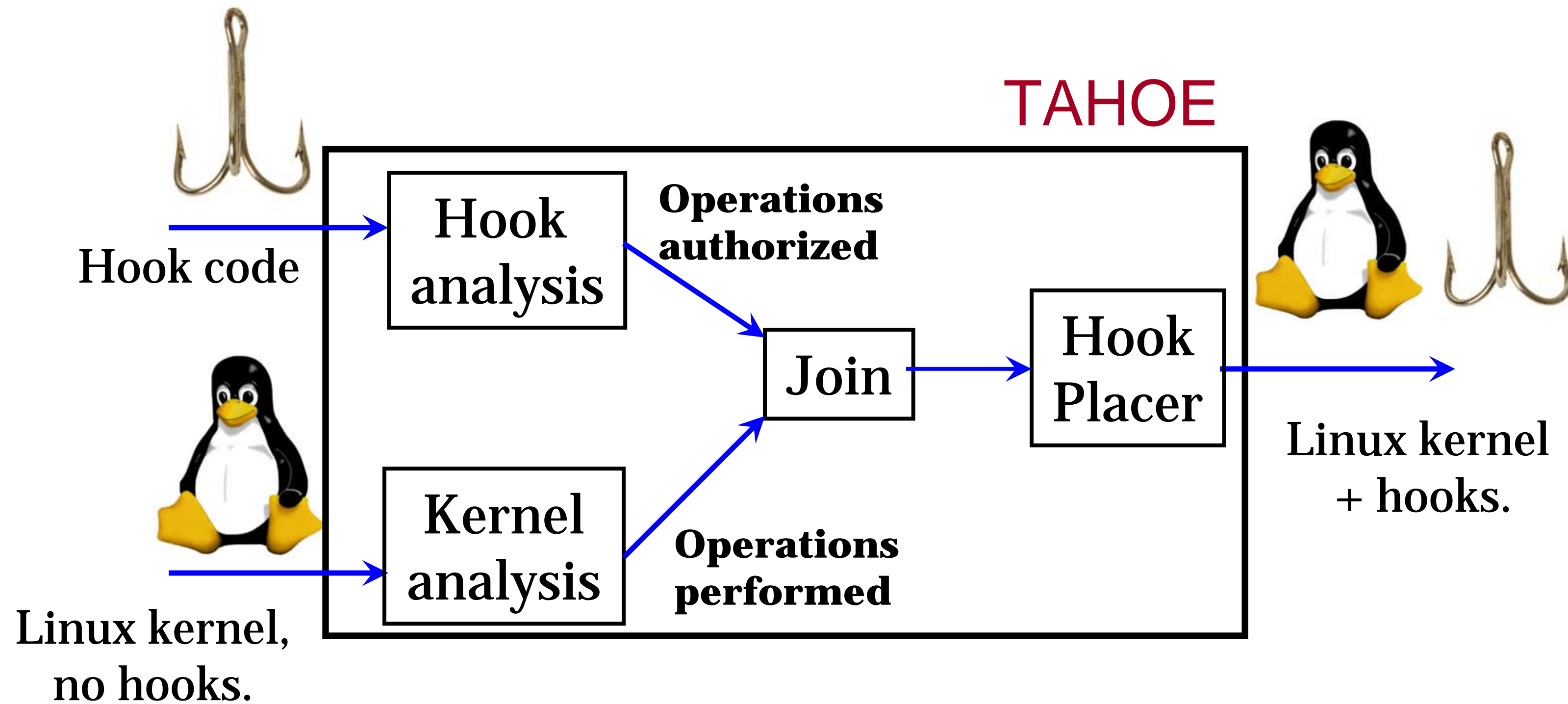
- Input:
  - A set of security-sensitive operations.
  - Source code of reference monitor hooks.
  - Source code of the Linux kernel, without hooks placed.

- Output:
  - Linux kernel with hooks placed.

Tool for Authorization Hook Placement



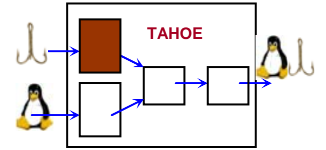
Tool for Authorization Hook Placement



Security-sensitive operations

- We use the set of operations from the LSM implementation of SELinux.
- Comprehensive set of operations on resources:
  - **`FILE_READ`**
  - **`DIR_READ`**
  - **`FILE_WRITE`**
  - **`DIR_WRITE`**
  - **`SOCKET_RECV_MESG`**
  - **`SOCKET_LISTEN`**
  - … (504 such operations)

Authorization hook analysis



- Analyze source code of hooks and:
  - Recover the operations authorized.
  - Conditions under which they are authorized.

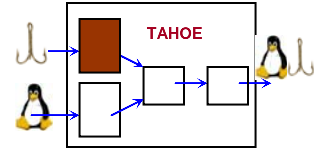
- Example:

```
int selinux_inode_permission(struct *inode, int mask) {
    op = 0;
    // s = info about process requesting operation

    if (mask & MAY_EXEC) op |= DIR_SEARCH;
    if (mask & MAY_WRITE) op |= DIR_WRITE;
    if (mask & MAY_READ) op |= DIR_READ;

    Query_Policy(s, inode, op);
}
```

Authorization hook analysis



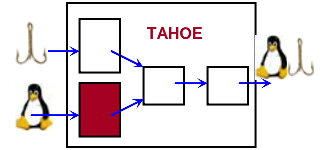

```
int selinux_inode_permission(struct *inode, int mask) {
    op = 0;
    // s = info about process requesting operation

    if (mask & MAY_EXEC) op |= DIR_SEARCH;
    if (mask & MAY_WRITE) op |= DIR_WRITE;
    if (mask & MAY_READ) op |= DIR_READ;

    Query_Policy(s, inode, op);
}
```

- Flow-and-context-sensitive static analysis:
  - **DIR_READ** authorized if `**mask & MAY_READ**'
  - **DIR_WRITE** authorized if `**mask & MAY_WRITE**'
  - **DIR_SEARCH** authorized if `**mask & MAY_EXEC**'

Linux kernel analysis



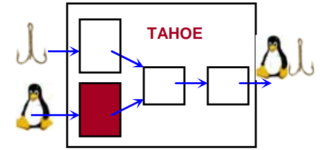
- Analyze Linux kernel to determine the security-sensitive operations performed by each function.
- More challenging than hook analysis.
- Example:

Virtual File System Code for Directory Removal

```
int vfs_rmdir (struct inode *dir, struct dentry *dentry) {
        ...
        dir->i_op->rmdir(dir, dentry);
        ...
}
```

Points to physical file system code

Example

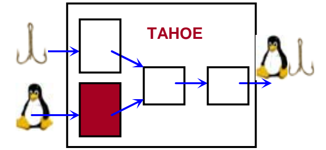


- How to infer the security-sensitive operations performed by `dir->i_op->rmdir(dir,dentry)`?

```
int vfs_rmdir (struct inode *dir, struct dentry *dentry) {
        ...
        dir->i_op->rmdir(dir, dentry);
        ...
}
```

```
$ ls foo/
bar/
```

Example

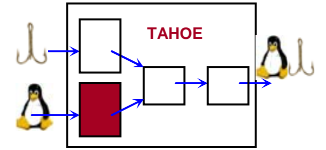


- How to infer the security-sensitive operations performed by `dir->i_op->rmdir(dir,dentry)`?

```
int vfs_rmdir (struct inode *dir, struct dentry *dentry) {
        ...
        dir->i_op->rmdir(dir, dentry);
        ...
}
```

```
$ cd foo/
$ rmdir bar/
```

Example



- How to infer the security-sensitive operations performed by `dir->i_op->rmdir(dir,dentry)`?

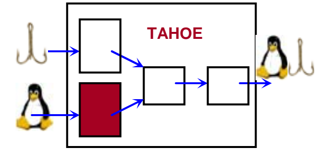
```
int vfs_rmdir (struct inode *dir, struct dentry *dentry) {
      ...
      dir->i_op->rmdir(dir, dentry);
      ...
}
```

- Removing ba
  - Lookup of entry for o.
  - Removing (and henc ing to) `foo`'s data structures.

How to extract this information?

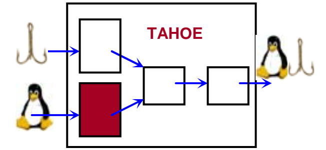
- `rmdir` involves `DIR_SEARCH`, `DIR_RMDIR` and `DIR_WRITE`.

Key observation



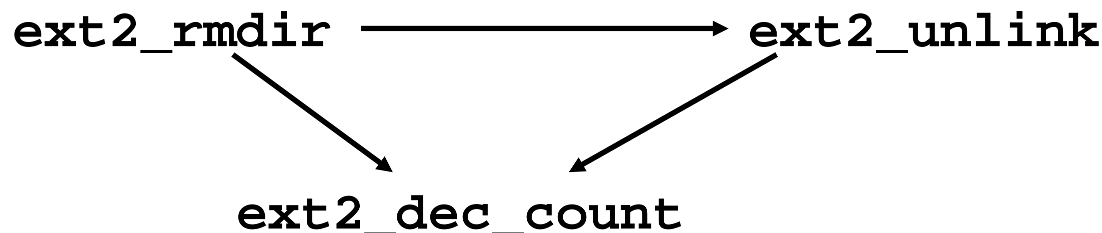
- Each security sensitive operation typically involves certain *idiomatic events*.
- Examples:
  - `DIR_WRITE :-` *Set* `inode->i_ctime &` *Call* `address_space_ops->prepare_write()`
  - `DIR_SEARCH :-` *Read* `inode->i_mapping`
  - `DIR_RMDIR :-` *Set* `inode->i_size TO 0 &` *Decrement* `inode->i_nlink`
- These rules are called *Idioms:*
  - Boolean combination of code-patterns.
  - Idiom language resembles Datalog.

Linux kernel analysis

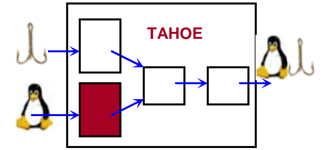


- *Flow-insensitive*, inter-procedural search for code patterns.
- Example: Call-graph of **`ext2`** file system

```
ext2_rmdir (struct inode *dir, struct dentry *dentry)
{
        ext2_unlink(…);
        …
        ext2_dec_count(…);
}
```

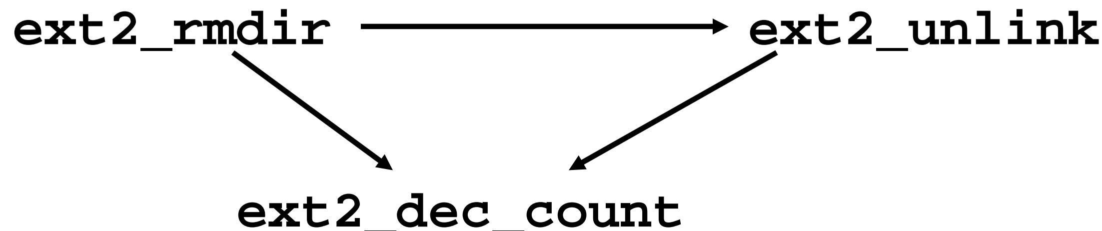


Linux kernel analysis

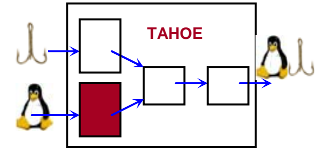


- *Flow-insensitive*, inter-procedural search for code patterns.
- Example: Call-graph of **`ext2`** file system

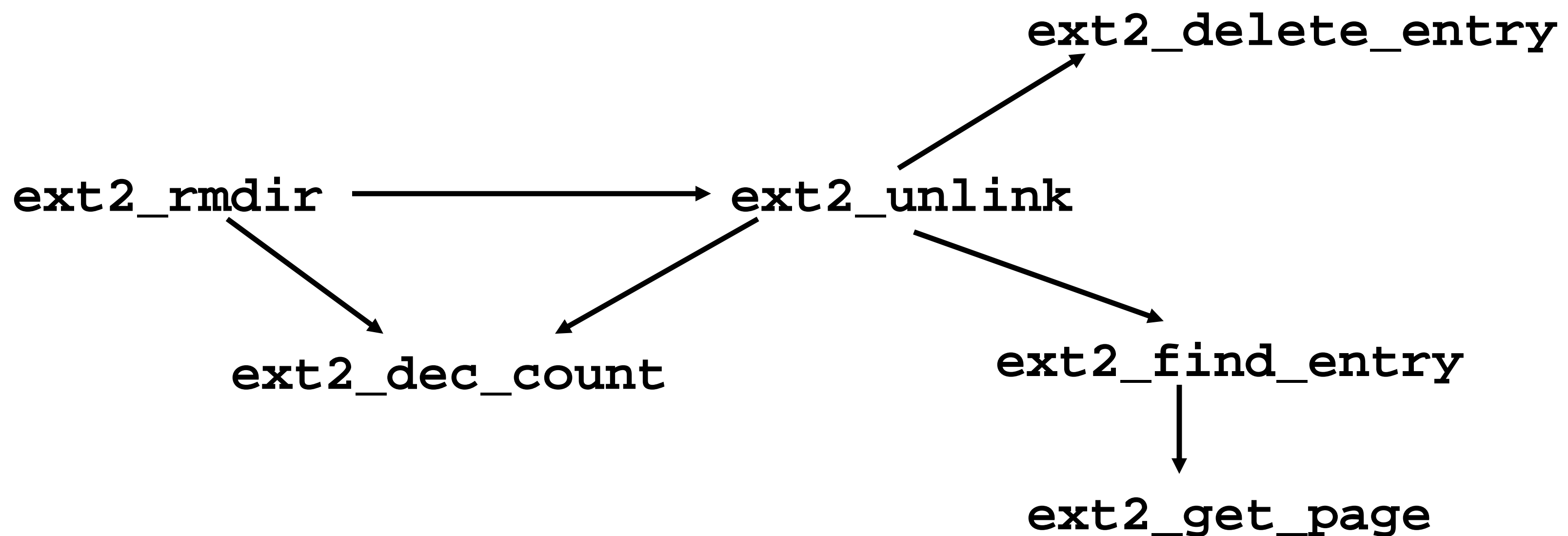
```
ext2_rmdir (struct inode *dir, struct dentry *dentry)
{
        ext2_unlink(…);
        …
        ext2_dec_count(…);
}
```



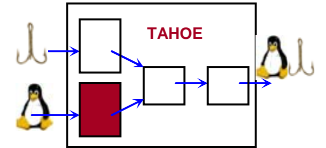
Linux kernel analysis



- *Flow-insensitive*, inter-procedural search for code patterns.
- Example: Call-graph of **`ext2`** file system



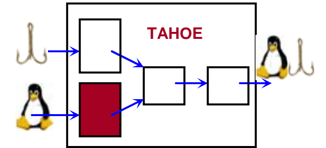
Linux kernel analysis



- *Flow-insensitive*, inter-procedural search for code patterns.
- Example: Call-graph of **`ext2`** file system

```
page->mapping->a_ops->prepare_write()
ext2_delete_entry
inode->i_size = 0
ext2_rmdir
ext2_unlink
ext2_dec_count
inode->i_nlink--
ext2_find_entry
ext2_get_page
mapping = inode->i_mapping
```

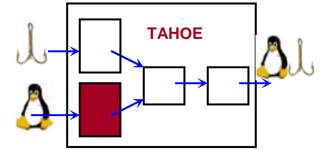
Linux kernel analysis



- `DIR_RMDIR :-` *Set* `inode->i_size` to 0 & *Decrement* `inode->i_nlink`
- Example: Call-graph of `ext2` file system

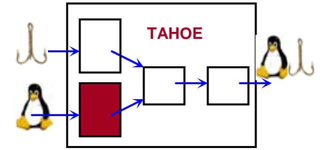
```
page->mapping->a_ops->prepare_write()
ext2_delete_entry
inode->i_size = 0
ext2_rmdir
ext2_unlink
ext2_dec_count
ext2_find_entry
inode->i_nlink--
ext2_get_page
mapping = inode->i_mapping
```

Result with ext2_rmdir



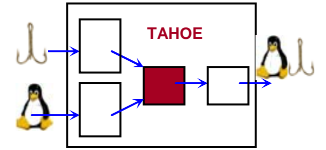
- *Flow-insensitive*, inter-procedural search for code patterns.
- Results:

Idioms

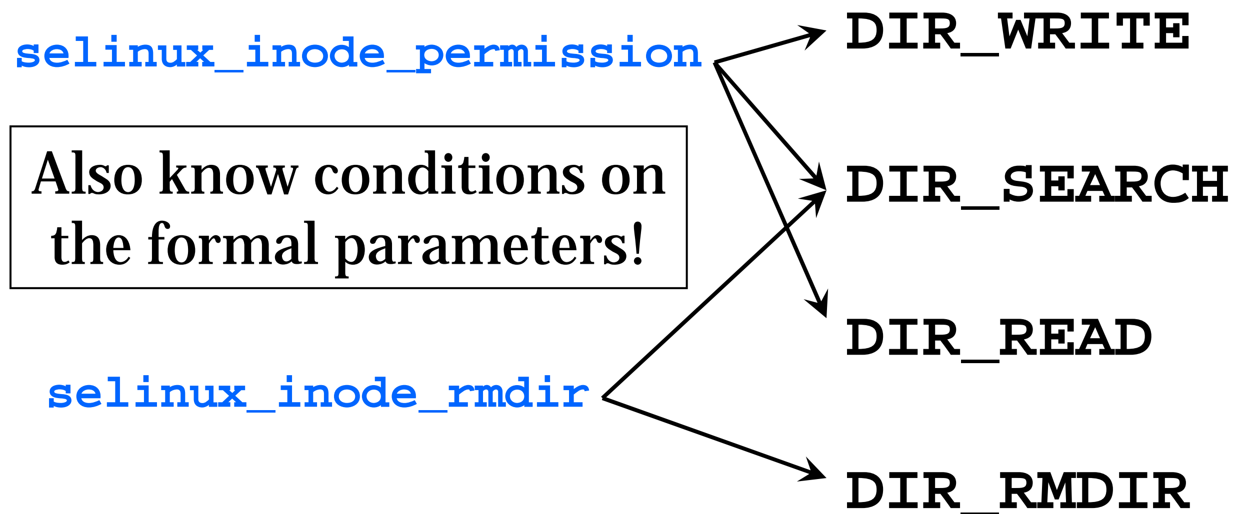


- Currently specified manually by us:
  - We wrote 150 idioms in a week.
  - We expect that a kernel developer can write these faster and more precisely.
- Difference from manual hook placement:
  - Only knowledge of kernel required.
  - One-time activity for the kernel: can reuse results for different reference monitors.
- Current work: Automating idiom writing.

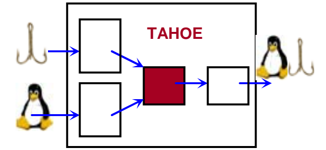
Combining results



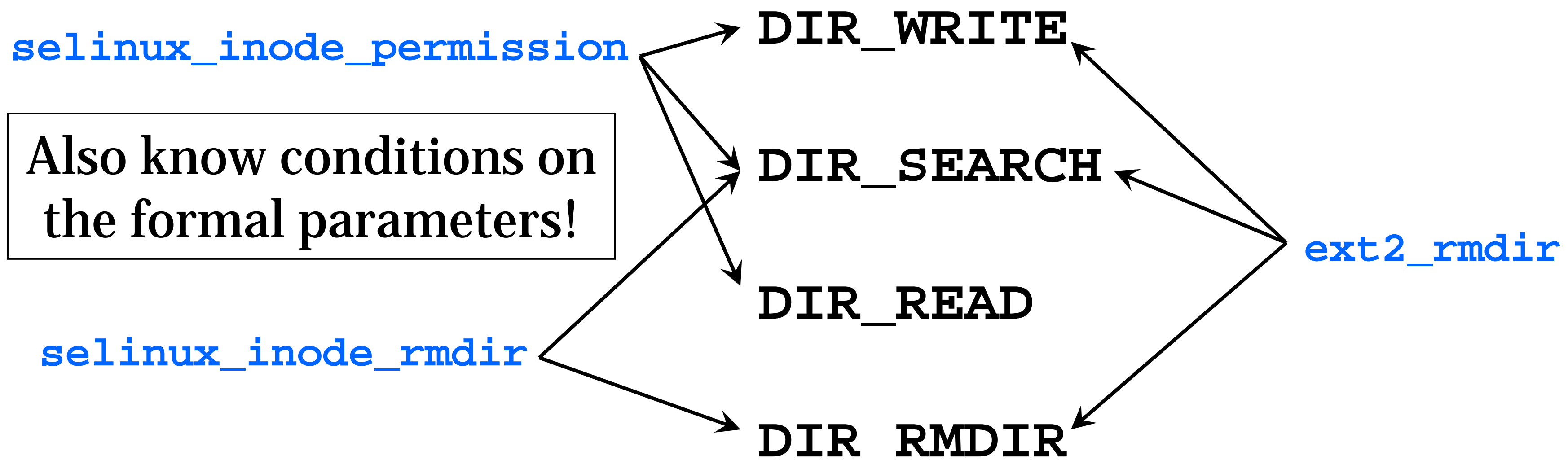
- From authorization hook analysis



Combining results

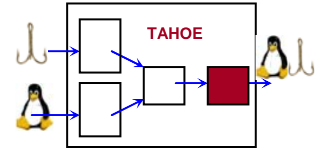


- From kernel analysis



Protect **ext2_rmdir** with
- **selinux_inode_rmdir**
- **selinux_inode_permission**(**MAY_WRITE**)

Placing hooks



- Naïve (but correct) approach:
  - Place hooks at each function call in the kernel using join analysis results.
  - May lead to redundant checks.

- TAHOE works differently:
  - Identifies a small set of *controlled functions*.
  - Suffices to place hooks to protect these.

- See paper for details.

Results

- Wrote idioms for `inode` and `socket` operations
- Tested with SELinux reference monitor and Linux kernel version 2.4.21

Hook type	Num.	Num. Locs	False pos.	False neg.
`inode`	26	40	13	4
`socket`	12	12	4	0

- False positives and negatives mainly because of imprecision in idioms.

Future work

- Hook placement for general-purpose servers
  - Example: X server.
  - Must enforce authorization policies on X clients.
  - Example: Prevent a "cut-and-paste" from a high-security `xterm` to a low-security `xterm`.
- Hundreds of such servers: database servers, web servers,…
- Manual hook placement?
  - *Simply infeasible*!

Summary of important ideas

- Can largely automate authorization hook placement using static analysis.
- Key idea: Matchmaking based on security-sensitive operations.
- TAHOE: A tool for LSM-hook placement.

Thank You

Contact Information

Vinod Ganapathy	vg@cs.wisc.edu
Trent Jaeger	tjaeger@cse.psu.edu
Somesh Jha	jha@cs.wisc.edu

Web-site

http://www.cs.wisc.edu/~vg/papers/ccs2005a/