

# A Formalisation of an Access Control Framework



joint work with Chunhan Wu and Xingyuan Zhang from the  
PLA University of Science and Technology in Nanjing

Christian Urban  
King's College London

# Access Control

- perhaps most known are Unix-style access control systems (root super-user, setuid mechanism)

```
> ls -ld . * */*
drwxr-xr-x 1 alice staff    32768  Apr  2 2010 .
-rw----r-- 1 alice students 31359  Jul 24 2011 manual.txt
-rwsr--r-x 1 bob   students 141359 Jun  1 2013 microedit
dr--r-xr-x 1 bob   staff    32768  Jul 23 2011 src
-rw-r--r-- 1 bob   staff    81359  Feb 28 2012 src/code.c
```


# Access Control

more fine-grained access control is provided by

- SELinux  
(security enhanced Linux developed by the NSA;  
mandatory access control system)
- Role-Compatibility Model  
(developed by Amon Ott;  
main application in the Apache server)

# Access Control

more fine-grained access control is provided by

- SELinux  
(security enhanced Linux developed by the NSA;  
mandatory access control system)
- Role-Compatibility Model   
(developed by Amon Ott;  
main application in the Apache server)

# Operations in the OS

using Paulson's inductive method a **state of the system** is a **trace**, a list of events (system calls):

$$[e_1, \dots, e_2]$$

$$e ::= \begin{array}{l} \textit{CreateFile } p f \quad | \quad \textit{ReadFile } p f \quad | \quad \textit{Send } p i \\ | \quad \textit{WriteFile } p f \quad | \quad \textit{Execute } p f \quad | \quad \textit{Recv } p i \\ | \quad \textit{DeleteFile } p f \quad | \quad \textit{Clone } p p' \quad | \quad \textit{CreateIPC } p i \\ | \quad \textit{ChangeOwner } p u \quad | \quad \textit{ChangeRole } p r \quad | \quad \textit{DeleteIPC } p i \\ | \quad \textit{Kill } p p' \end{array}$$

# Valid Traces

we need to restrict the traces to **valid traces**:

$$\frac{}{\text{valid } []} \quad \frac{\text{valid } s \quad \text{admissible } s e \quad \text{granted } s e}{\text{valid } (e::s)}$$

# Valid Traces

we need to restrict the traces to **valid traces**:

$$\frac{}{\text{valid } []} \quad \frac{\text{valid } s \quad \text{admissible } s e \quad \text{granted } s e}{\text{valid } (e::s)}$$


# Valid Traces

we need to restrict the traces to **valid traces**:




$$\frac{}{\text{valid } []} \quad \frac{\text{valid } s \quad \text{admissible } s \ e \quad \text{granted } s \ e}{\text{valid } (e::s)}$$

$$\frac{p \in \text{current\_procs } s \quad p' \notin \text{current\_procs } s}{\text{admissible } s \ (\text{Clone } p \ p')}$$



# Valid Traces

we need to restrict the traces to **valid traces**:

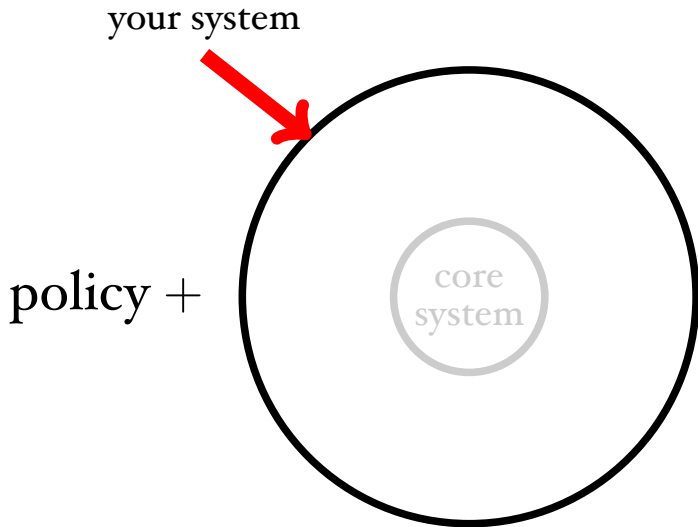

$$\frac{}{\text{valid } []} \quad \frac{\text{valid } s \quad \text{admissible } s e \quad \text{granted } s e}{\text{valid } (e::s)}$$

$$\frac{\text{is\_current\_role } s p r \quad \text{is\_file\_type } s f t \quad (r, t, \text{Execute}) \in \text{permissions}}{\text{granted } s (\text{Execute } p f)}$$

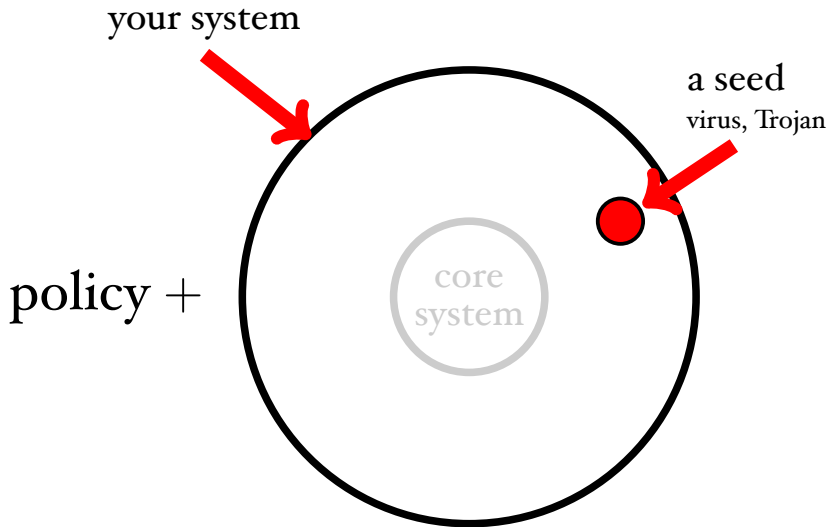
# Design of AC-Policies

*"what you specify is what you get but not necessarily what you want..."*

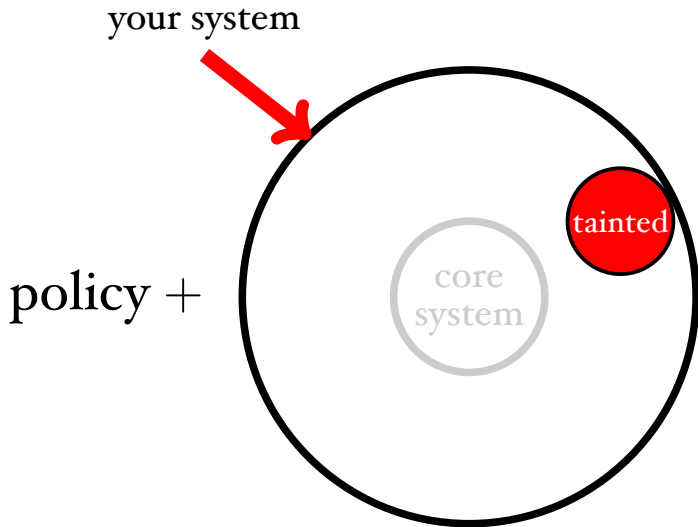
# Testing Policies



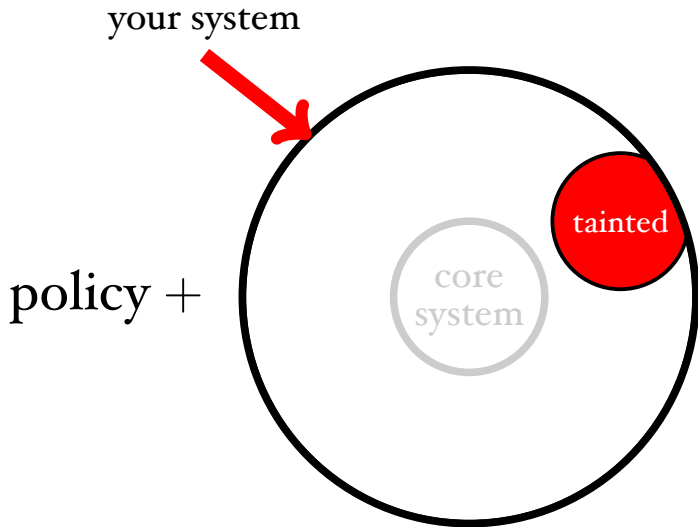
# Testing Policies



# Testing Policies



# Testing Policies

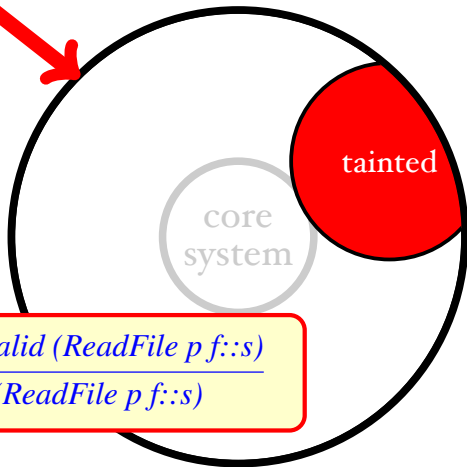


# Testing Policies

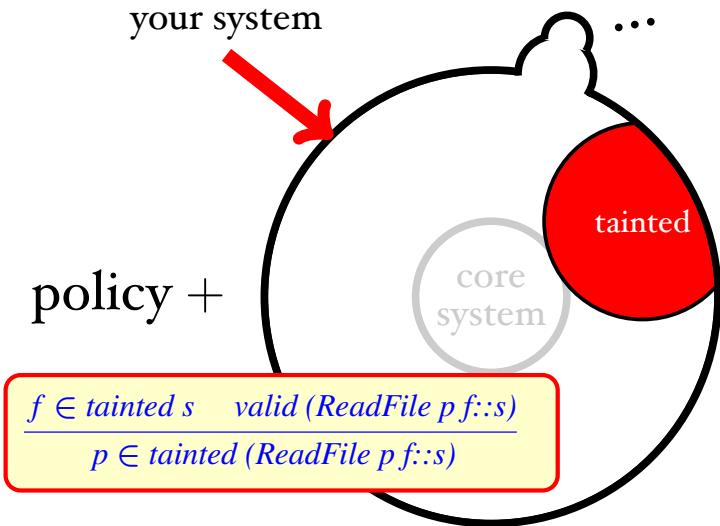
your system



policy +


$$\frac{f \in \text{tainted } s \quad \text{valid} (\text{ReadFile } p f :: s)}{p \in \text{tainted} (\text{ReadFile } p f :: s)}$$

# Testing Policies





# A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - suppose a tainted file has type *bin* and
  - there is a role *r* which can both read and write *bin*-files

# A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - suppose a tainted file has type *bin* and
  - there is a role *r* which can both read and write *bin*-files
  - then we would conclude that this tainted file can spread

# A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - suppose a tainted file has type *bin* and
  - there is a role *r* which can both read and write *bin*-files
  - then we would conclude that this tainted file can spread
  - but if there is no process with role *r* and it will never be created, then the file actually does not spread

# A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - suppose a tainted file has type *bin* and
  - there is a role *r* which can both read and write *bin*-files
  - then we would conclude that this tainted file can spread
  - but if there is no process with role *r* and it will never been created, then the file actually does not spread
- **our solution:** take a middle ground and record precisely the information of the initial state, but be less precise about every newly created object.

# Results about our Test

- we can show that the objects (files, processes, ...) we need to consider are only finite (at some point it does not matter if we create another *bin*-file)

## **Thm (Soundness)**

If our test says an object is taintable, then it is taintable in the OS, and we produce a sequence of events that show how it can be tainted.

# Results about our Test

- we can show that the objects (files, processes, ...) we need to consider are only finite (at some point it does not matter if we create another *bin*-file)

## **Thm (Soundness)**

If our test says an object is taintable, then it is taintable in the OS, and we produce a sequence of events that show how it can be tainted.

## **Thm (Completeness)**

If an object is taintable in the OS and *undeletable*<sup>\*</sup>, then our test will find out that it is taintable.

\* an object is *undeletable* if it exists in the initial state, but there exists no valid state in which it could have been deleted

# Why the Restriction?

- assume a process with *ID* is tainted but gets killed by another process
- after that a process with the same *ID* gets *re-created* by cloning an untainted process
- clearly the new process should be considered *untainted*

# Why the Restriction?

- assume a process with *ID* is tainted but gets killed by another process
- after that a process with the same *ID* gets *re-created* by cloning an untainted process
- clearly the new process should be considered *untainted*

unfortunately our test will not be able to detect the difference (we are less precise about newly created processes)



# Why the Restriction?

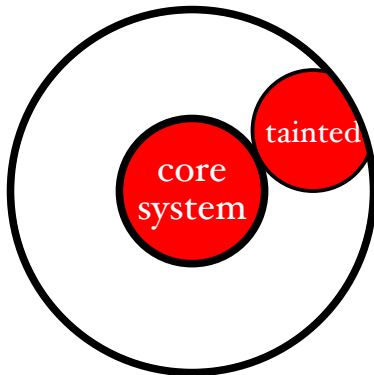
- assume a process with *ID* is tainted but gets killed by another process
- after that a process with the same *ID* gets *re-created* by cloning an untainted process
- clearly the new process should be considered *untainted*

unfortunately our test will not be able to detect the difference (we are less precise about newly created processes)

Is this a serious restriction? We think not ...

# Core System

Admins usually ask whether their policy is strong enough to protect their core system?



core system files are typically undeletable

# Conclusion

- we considered the Role-Compatibility Model used for securing the Apache Server  
13 events, 13 rules for OS admisibility, 14 rules for RC-granting, 10 rules for tainted
- we can scale this to SELinux  
more fine-grained OS events (inodes, hard-links, shared memory, ...)  
22 events, 22 admisibility, 22 granting, 15 taintable

# Conclusion

- we considered the Role-Compatibility Model used for securing the Apache Server
  - 13 events, 13 rules for OS admisibility, 14 rules for RC-granting, 10 rules for tainted
- we can scale this to SELinux
  - more fine-grained OS events (inodes, hard-links, shared memory, ...)
  - 22 events, 22 admisibility, 22 granting, 15 taintable
- hard sell to Ott (who designed the RC-model)
- hard sell to the community working on access control (beyond *good science*)

**Thanks!**  
**Questions?**

