Capabilities: principles and object capabilities

aslan@cs.au.dk

acknowledgements: Lau Skorstengaard

Administrativia

- We split capabilities over two classes
  - Today: capability principles and object capabilities
  - Next week: guest lecture by Aina on capability machines

Secure design principles [Saltzer & Schroeder' 1975]

Economy of mechanism	simple design makes it feasible to evaluate all possible paths
Fail-safe defaults	conservative defaults make security errors observable
Complete mediation	check every access to every object; skeptically examine performance gains that sacrifice securit
Open design	decoup … crecy for the design
Separation of privilege	if feasible, avoid single point of com romise
Least privilege	limit the damage from error (cf. "need-to-know")
Least common mechanism	every shared mechanism presents a new path to leak information
Psychological acceptability	design for human use

Every program and every user of the system should operate using the least set of privileges necessary to complete the job

Access control

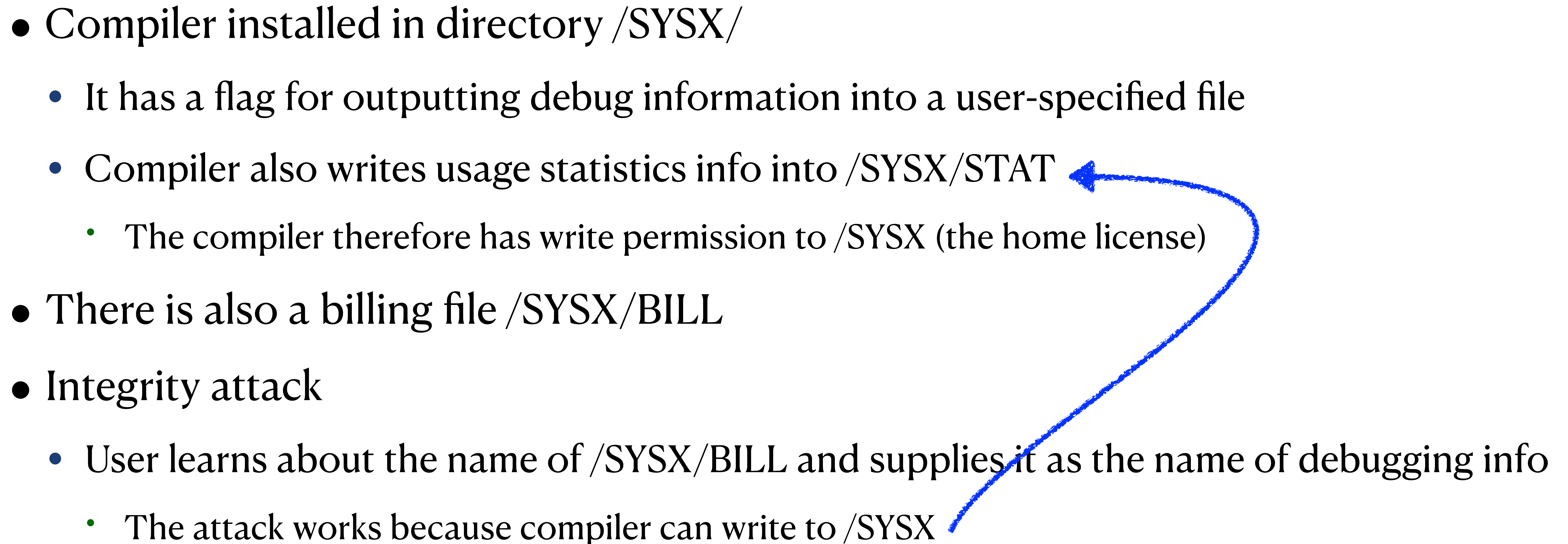
a form of authorization

- A mechanism for enforcing confidentiality and integrity
- Predefined security-sensitive *operations*
  - e.g., Read/Write/Execute
- Reference monitor (e.g., OS) consulted whenever predefined operations are invoked
- *Access control policy* specifies which operations a principal (subject) can do on an object

cf. F.B.Schneider's draft textbook on cybersecurity

Confused deputy

[Hardy, 1988]



Who is to blame in this example?

- The compiler should have scanned the specified output destination to make sure that the debug information was not written to a different directory?
    - No, it is useful to allow the user to write the debug info in a different directory.
- The compiler should have checked the the directory was not SYSX.
    - Again no, the SYSX folder may not have been invented when the compiler was written, and a user may have a legitimate reason to use SYSX.
- The compiler should have checked that the specified file for output was not SYSX/BILL!
    - No, that may not be the only sensitive file in SYSX, and it is not the compiler's job to enforce access control.

“The fundamental problem is that the compiler runs with authority stemming from two sources (that’s why the compiler is a confused deputy)”

the home license

Hardy, 1988

Permission and Authority

[Miller, 2006]

- *Permissions* specify direct access rights
- *Authority* is the ability to cause effects
    - Directly by permission, or
    - Indirectly by permitted interactions with other programs
- In confused deputy, the user has authority to write to SYSX/BILL despite having no permission
- Gap between permissions and authority $\Longrightarrow$ leakage of access rights
    - Can we make it so that “reasoning about permissions” ~ “reasoning about authority”?

Capability model

[Dennis and van Horn, 1965]

- Traditional *access control policies* separate designation (knowing a name) from permission
  - (a user may have no permission to SYSX/BILL but they know the file name)
- Main idea in capability model:
  - Combine designation and access right into *a capability*
  - Possession of a capability grants the access
    - It is therefore crucial that capabilities are *unforgeable*
      - Prevents creating authority out of thin air
    - Capabilities may be shared between components
      - They can also be *attenuated* to comply with the principle of least privilege
- Simplifies reasoning about authority
  - No capabilities means no access

Capability is an unforgeable token of authority

The confused deputy example and capabilities

- In a capability system
  - Compiler would have capability for the file containing statistics
  - User supplies capability instead of location
- Confused despite problem is no problem with capabilities :-)

Capabilities in operating systems

- KeyKOS [1992], EROS [1999], Coyotos [2004], PSOS[2003]
  - Pure-capability systems that use both OS capability and hardware capabilities
- seL4[2009]
  - Based on L4 microkernel
  - Formally verified in Isabelle
  - Uses capabilities as access control mechanisms
  - Does not rely on special hardware
- CheriBSD [2014]
  - Adoption of FreeBSD to run on CHERI architecture
  - Uses CHERI's capability-based memory protection

Object Capabilities

Object-capability model

based on [Miller, 2006]

- In object-oriented model, there is no need to describe access control as a separate concern
  - All we need is abstraction and modularity (that are highly emphasized already)
- Compare:
  - Principle of information hiding:
    - abstractions should hand out information only on a need to know basis
  - Principle of least privilege
    - authority should be handed out only on a need to know basis
- Access abstractions through encapsulation

Object-capability model

- Object model (recall from prior courses)
  - No distinction between subjects and objects – “everything is an object”
  - Objects consist of code and state
    - *State* is a mutable collection of *references* objects
  - Objects behave according to their code
  - Interaction between objects by sending messages over references
- Object-capability model
  - No distinction between subjects and objects – “everything is an object”
  - Objects consist of code and state
    - *State* is a mutable collection of *references* objects
  - Objects behave according to their code
  - Interaction between objects **only** by sending messages over references
    - Reference graph == access graph

Terminology

Object	Instance or data
Instance	Combination of data and code - Behavior of instance described by code
Reference	Access to an object indivisible combining - designation of object - access permission - means of access
Capability	Reference to non-data

Object-capability model: graph dynamics

References can only be obtained according to the following rules

Connectivity by

- Initial conditions
  - An initial access graph dictates the initial references in the system.
- Parenthood
  - At the moment of object creation, the creator holds the sole reference for the newly created object.
- Endowment
  - At object creation, the creator may make some of their references available
- Introduction
  - References can be sent in messages.
    - This is the only way to obtain existing references

Object orientation and object capabilities

- Not every language with objects follows the object-capability model
- Languages that do not follow object capability model
  - C++
    - possible to forge references by casting integers to pointers
  - C# and Java
    - public interface is circumvented when instance variables are assigned indirectly
- What does it take to make Java satisfy object-capability model?

Differences between Joe-E and Java

Joe-E programs are Java programs with added restrictions

- Part of reflection API disabled
  - Reflection API can circumvent private access modifiers
- Prohibit overwriting finalize ()
- Limit global scope
  - Java programs have *ambient authority* over outside world
  - In Joe-E
    - No ambient authority
    - Dependency injection required
- Tamed Java library
  - Library contains constructors, methods, and fields that grants arbitrary authority.
  - Remove these from library or make safe versions when possible
    - getParentFile()
    - File (File dir, String child) - File (null, path)

Differences between Joe-E and Java

Further restrictions

- Only allow static final Immutable field
  - Immutable
    - Joe-E tagging interface
    - Deeply immutable
  - Static fields not associated with object – ambiently available
- All exceptions must be Immutable
  - Means for transferring control and references unexpectedly
  - Immutable exceptions cannot transfer capabilities
- Prevent catching Errors
  - Prevent finally keyword – can be used to catch Errors
  - finally for Exceptions can be encoded with try and catch

Revoking and limiting authority

- In Joe-E and other object capability languages:
  - Capabilities cannot be restricted
    - Solved by facet patterns
    - Defines additional interface for manipulating state
  - Capabilities are irrevocable
    - Solved by: Revocable capability patterns
    - e.g., Java decorator pattern

Facet pattern in Joe-E

Restricting authority

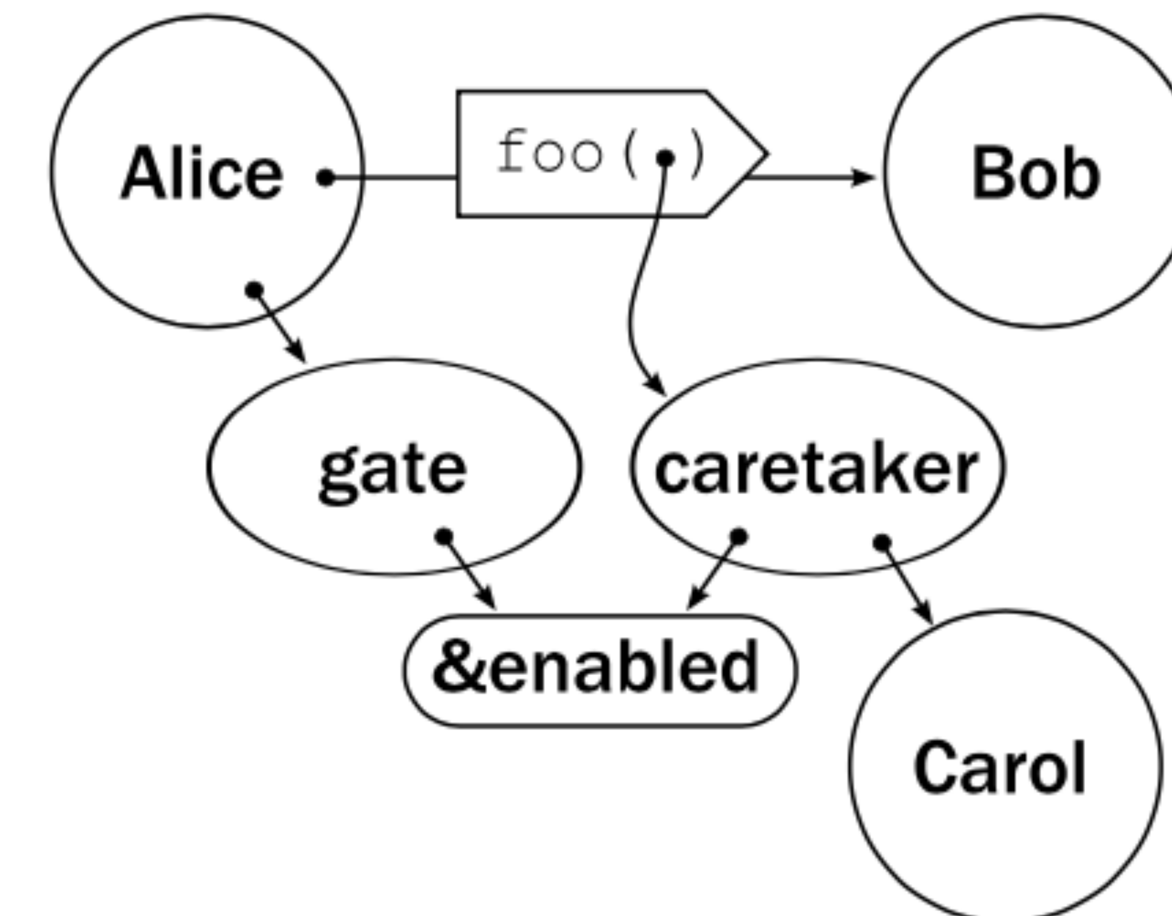
```
class Queue {
  public Object dequeue() {
      ...
  }
  public void enqueue(Object o) {
      ...
  }
}

class Queue {
  public Object dequeue() {
      ...
  }
  public void enqueue(Object o) {
      ...
  }
  public Receiver enqueuer() {
      return new Receiver() {
          public void receive(Object x) {
              enqueue(x);
          }
      };
  }
}
```

Caretaker pattern in E

revocation with capabilities

```
def makeCaretaker(target) {
    var enabled := true
    def caretaker {
        match [verb, args] {
            if (enabled) {
                E.call(target, verb, args)
            } else {
                throw("disabled")
            }
        }
    }
    def gate {
        to enable()  { enabled := true  }
        to disable() { enabled := false }
    }
    return [caretaker, gate]
}
```



Membrane pattern in E

Revocation with capabilities

```
def makeMembrane(target) {
    var enabled := true
    def wrap(wrapped) {
        if (Ref.isData(wrapped)) {
            # Data provides only irrevocable knowledge, so don't
            # bother wrapping it.
            return wrapped
        }
        def caretaker {
            match [verb, args] {
                if (enabled) {
                    def wrappedArgs := map(wrap, args)
                    wrap(E.call(wrapped, verb, wrappedArgs))
                } else {
                    throw("disabled")
                }
            }
        }
        return caretaker
    }
    def gate {
        to enable()  { enabled := true }
        to disable() { enabled := false }
    }
    return [wrap(target), gate]
}
```

Object-capability languages

- E, Joe-E
- Google Caja
  - Capability safe subset of JavaScript
  - SafeEcmaScript
    - https://github.com/tc39/proposal-ses
- Shill
  - Capability-based shell scripting language
- Midori

Summary

- Capability concept
  - Motivation: example of confused deputy
  - Capabilities
    - unforgeable token of authority
    - combine designation and access
- Object Capability Model
  - Principle of encapsulation coincide with least privilege
  - Capability safety requires restrictions on many language features
- Patterns for programming with capabilities