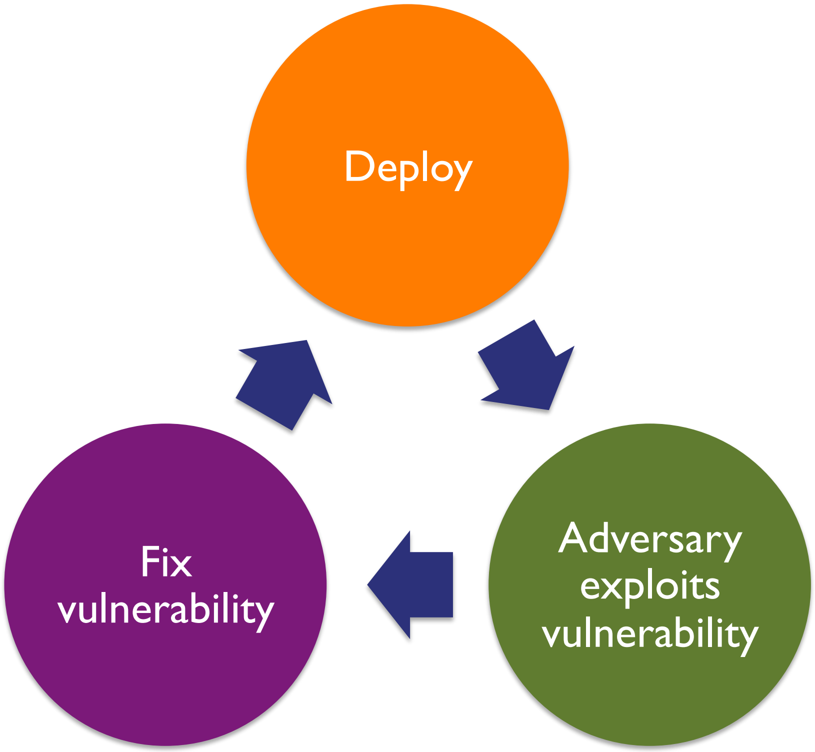
Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

Advanced Systems Security
Retrofitting for Security

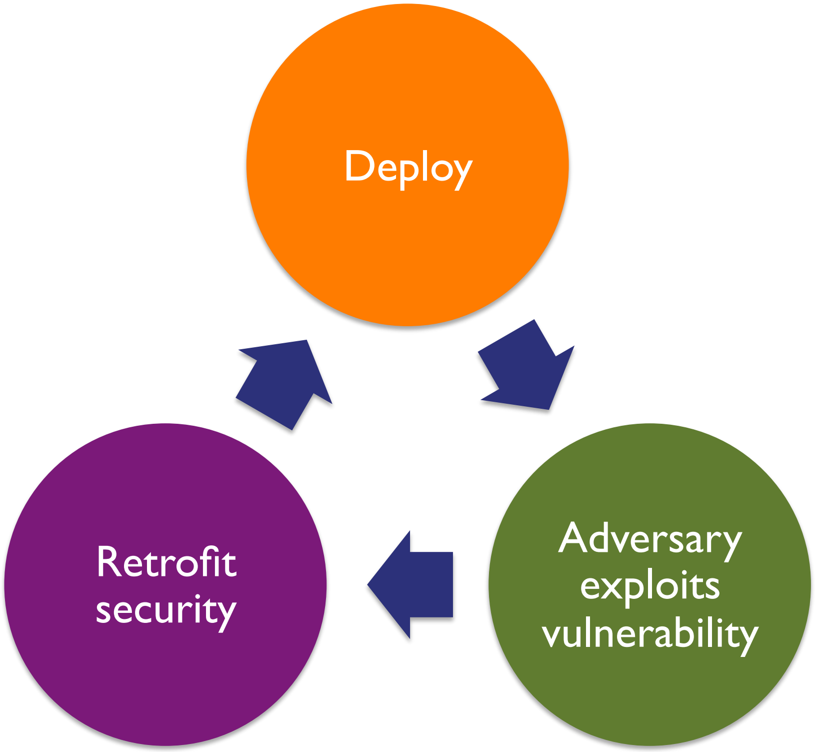
Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Retroactive Security



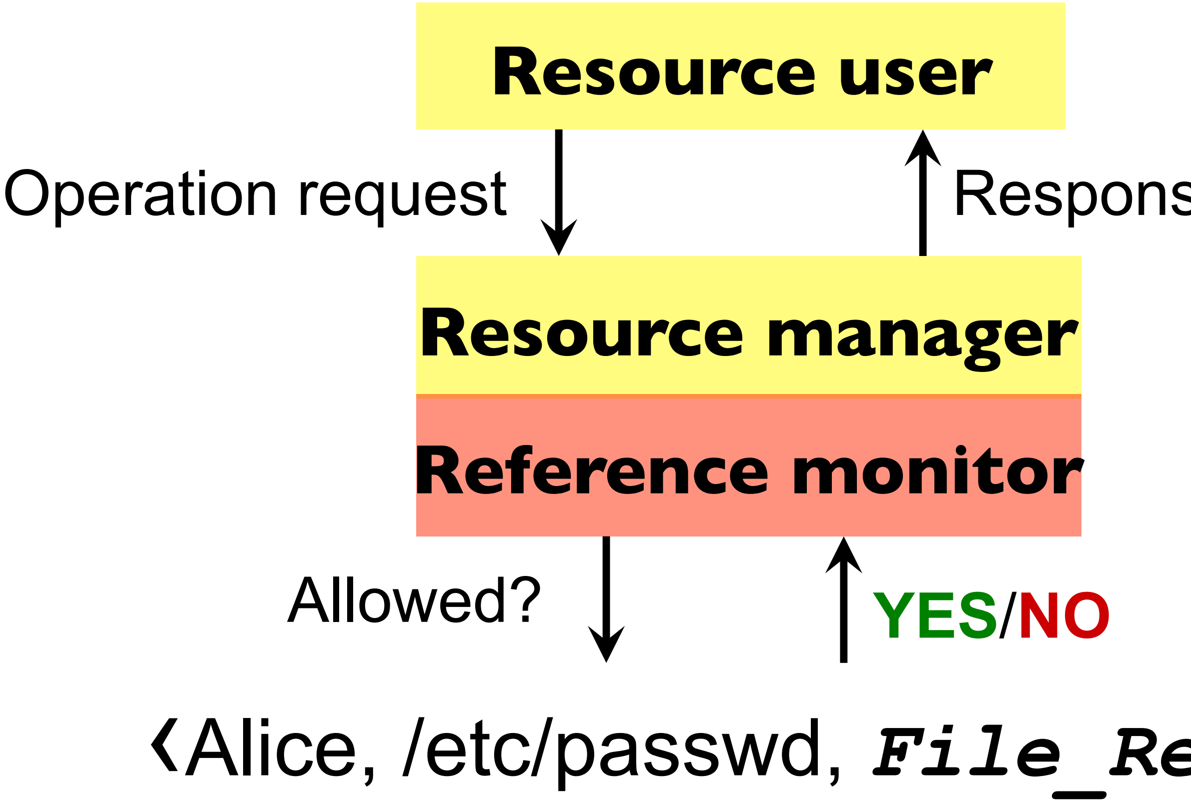
- “Penetrate and patch” as flaws are exposed as vulnerabilities

Retroactive Security



- Several codebases have been extended with security features retroactively
  - X Server, postgres, Apache, OpenSSH, Linux Kernel, browsers, etc.
- With a variety of security controls:
  - Privilege separation, Authentication, Auditing, Authorization, etc.

Authorizing Access



Resource user

Operation request

Response

Resource manager

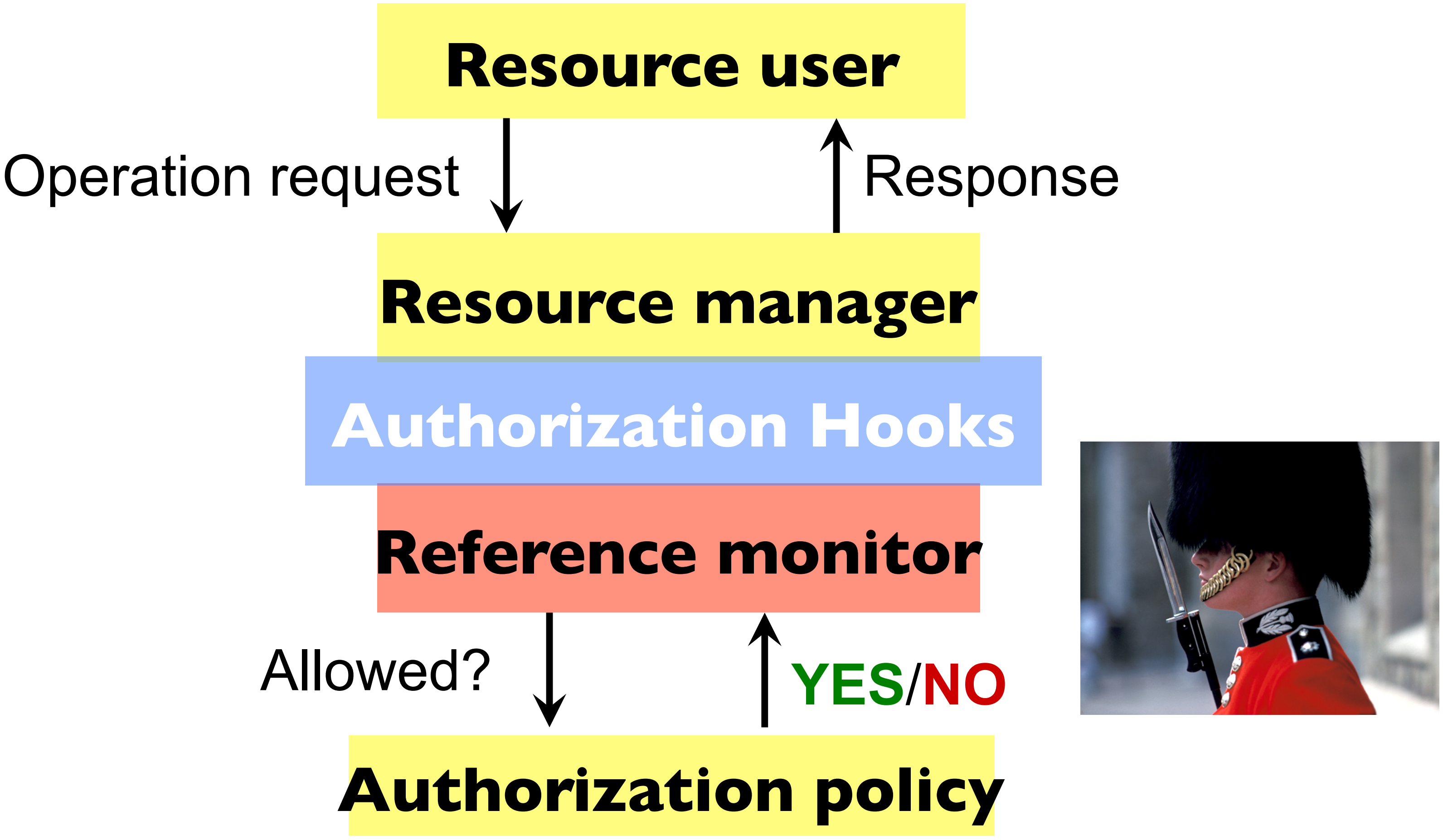
Reference monitor

Allowed?

YES/NO

〈Alice, /etc/passwd, ***File_Read***〉

Authorizing Access



Retrofitting is Hard

- For authorization
  - X11 ~ proposed 2003, upstreamed 2007, changing to date. [Kilpatrick *et al.*, ‘03]
  - Linux Security Modules ~ 2 years [Wright *et al.*, ’02]

Painstaking, manual procedure

> *At this point, SE-PostgreSQL has taken up a *__lot__* __of community resources__, not to mention an __enormous and doubtless frustrating amount of *the lead developer's* time and effort__, thus far __without a single committed patch, or even a consensus as to what it should (or could) do.__ Rather than continuing to blunder into the future, I think we need to do a reality check*
>
> \- http://archives.postgresql.org/message-id/20090718160600.GE5172@fetter.org

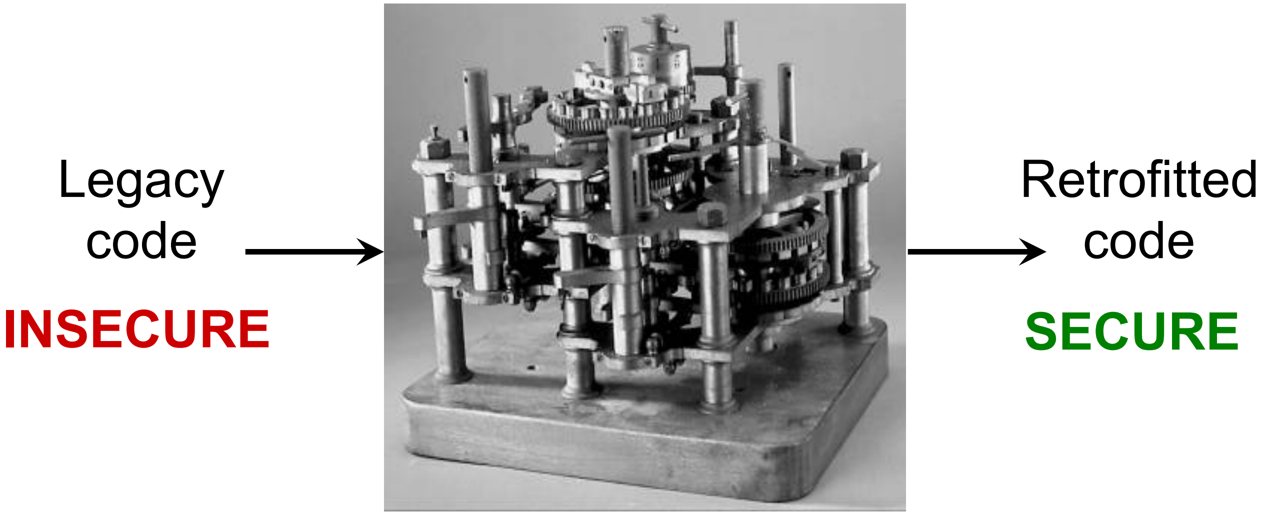
Retrofitting is Common

- Mandatory access control for Linux
  - Linux Security Modules [Wright *et al.*,'02]
- TrustedBSD, SEDarwin, sHype, XSM, …
- Secure windowing systems
  - Trusted X, Compartmented-mode workstation, X11/SELinux [Epstein *et al.*,'90][Berger *et al.*,'90][Kilpatrick *et al.*,'03]
- Java Virtual Machine/SELinux [Fletcher,'06]
- IBM Websphere/SELinux [Hocking *et al.*,'06]
- And more: Apache, PostgreSQL, dbus, gconf, …

Retrofitting Legacy Code

- What if you had to add security controls for a legacy program?

Need systematic techniques to retrofit legacy code for security



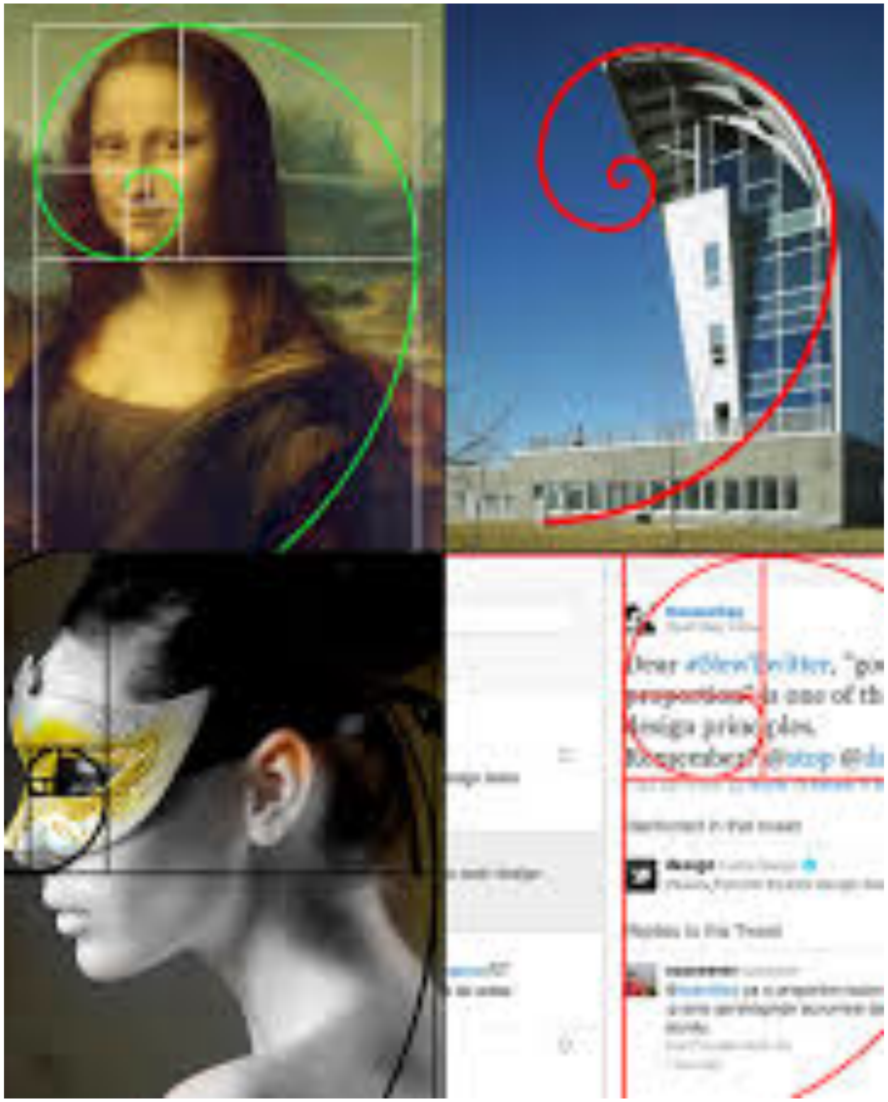
Legacy code

INSECURE

Retrofitted code

SECURE

Design for Security



- Perhaps retroactive security is the wrong approach
  - Too late to get right
- "Design for security" from the outset is the goal
  - But, how do we teach programmers to do that?
  - In a practical and time-effective manner
- Design methodologies may vary widely

What is Needed?

- Programs need multiple security controls

```
request_loop (client_data, private_data) {
  read(client_passwd, client_req );
  if (necessary ||
      compare_client(client_passwd,
                     private_data))
    access_object(client_req, client_data);
}
```

- Program reads `client_passwd` and `client_req`
- Don’t leak `private_data` used to check passwords
- Control client request’s access to `client_data`

What is Needed?

- Programs need multiple security controls

```
request_loop (client_data, private_data) {
  read(client_passwd, client_req );
  if (necessary ||
      compare_client(client_passwd,
                     private_data))
    access_object(client_req, client_data);
}
```

- Privilege separation between `compare_client` and `access_object`
- Authorization of `access_object`
- Auditing of execution of unsafe `client_req`

Past Efforts

- Automated Hook Placement:
  - Assumptions: **Training wheels**
    - (sensitive data types, hook code)

  [Ganapathy et al., 2005, 2006, 2007]

  [Sun et al., 2011, RoleCast 2011]

- Automated Hook Placement 2:
  - Assumptions: **Training wheels**
    - (constraint models of function and security)

  [Harris et al., 2010, 2013]

Security Goals

- Retrofit security controls automatically
  - From "security programs"
- Assist programmers in producing such security programs
  - From code analyses
- Compile such security programs into minimal cost code for enforcing the expected security goals correctly
  - Across security controls

Outline

- Let’s examine the problem of retrofitting for security
  - For authorization
- Then explore other security controls
  - For privilege separation and auditing
- Then, discuss how to retrofit across security controls
  - Step two

Retrofit for Authorization

We want to generate complete and minimal authorization hook placements mostly-automatically for legacy code

[CCS 2012] Divya Muthukumaran, Trent Jaeger, Vinod Ganapathy. Leveraging "choice" to automate authorization hook placement. In *Proceedings of the 19th ACM Conference on Computer and Communications Security (ACM CCS)*, October 2012.

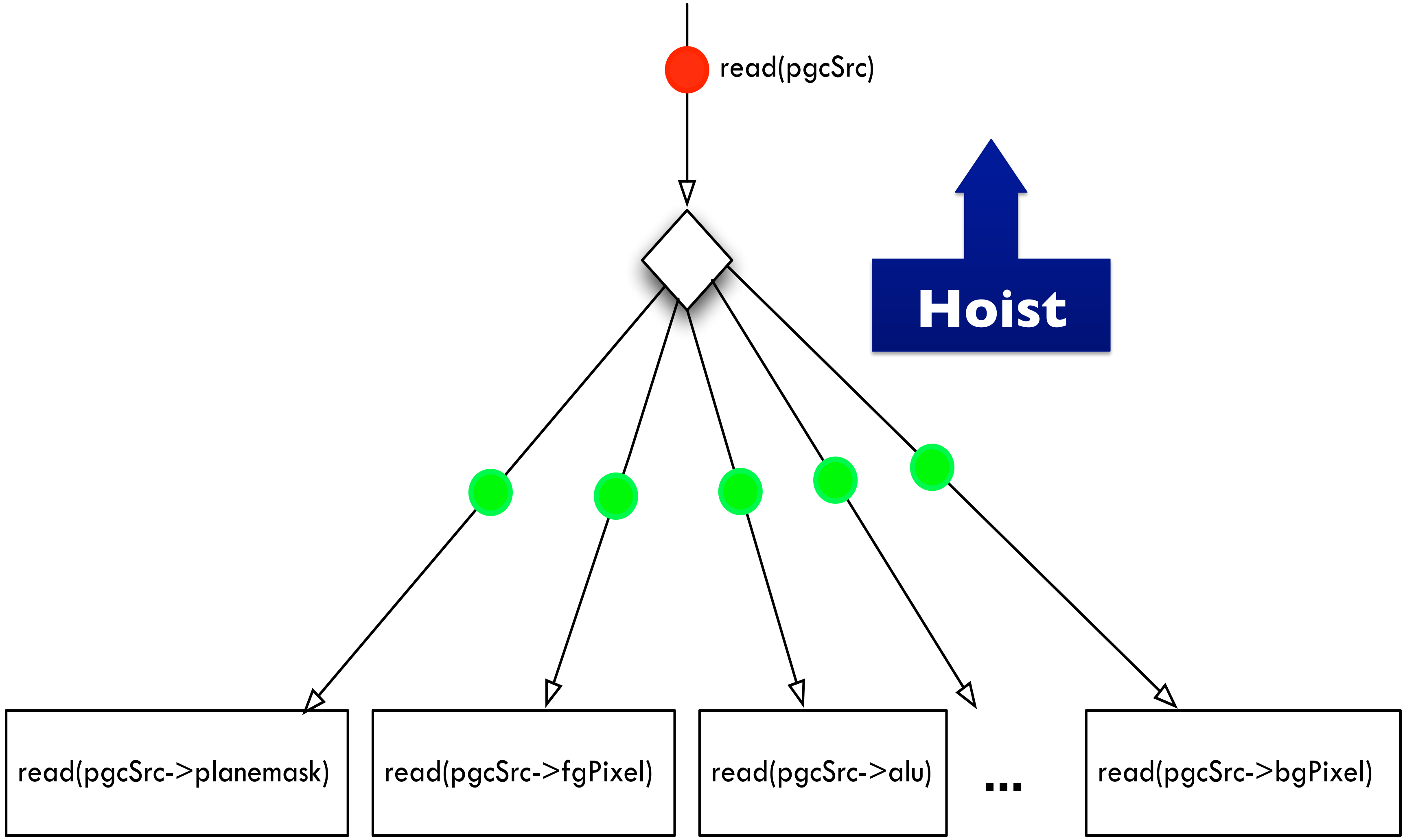
[ESSoS 2015] Divya Muthukumaran, Nirupama Talele, Trent Jaeger, Gang Tan. Producing hook placements to enforce expected access control policies. In *Proceedings of the 2015 International Symposium on Engineering Secure Software and Systems (ESSoS)*, March 2015.

Placement Comparison

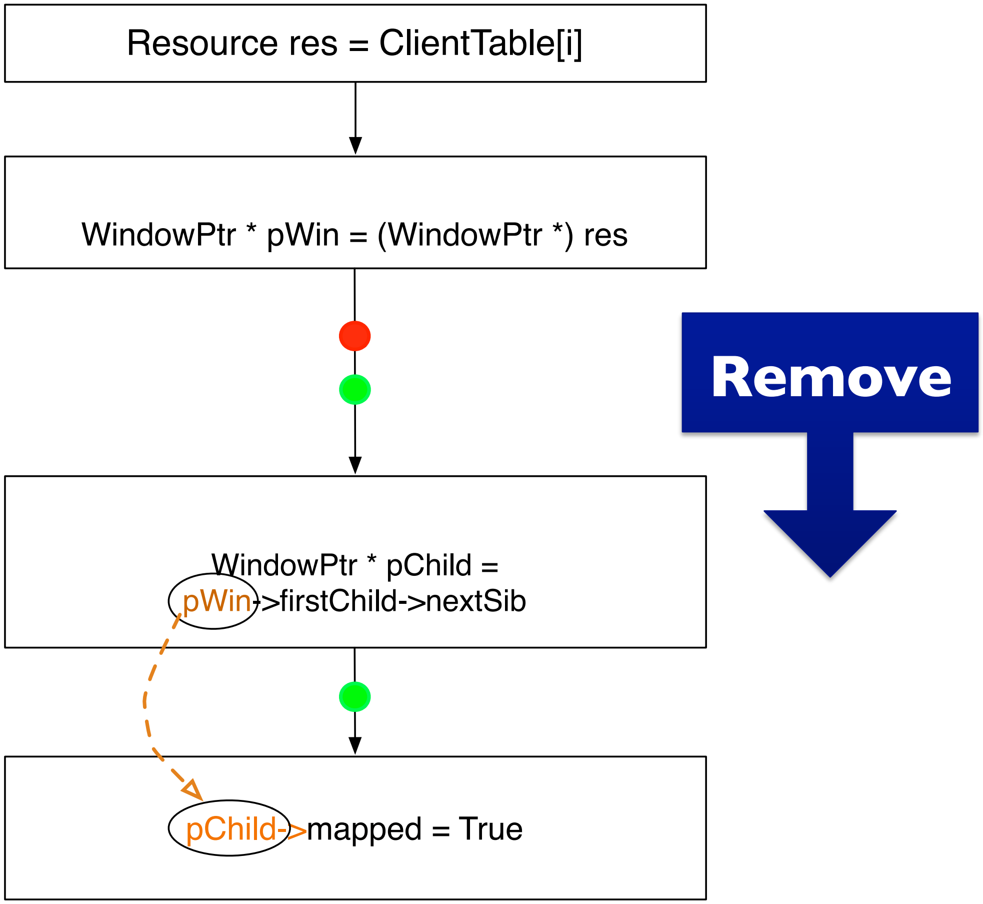
- Based on CCS 2012 Method
- X Server:
  - Manual: 201 hooks
  - Automated: 532 hooks
- Postgres:
  - Manual: ~370
  - Automated: 579

What does this mean?

Hook Hoisting



Hook Removal



Resource res = ClientTable[i]

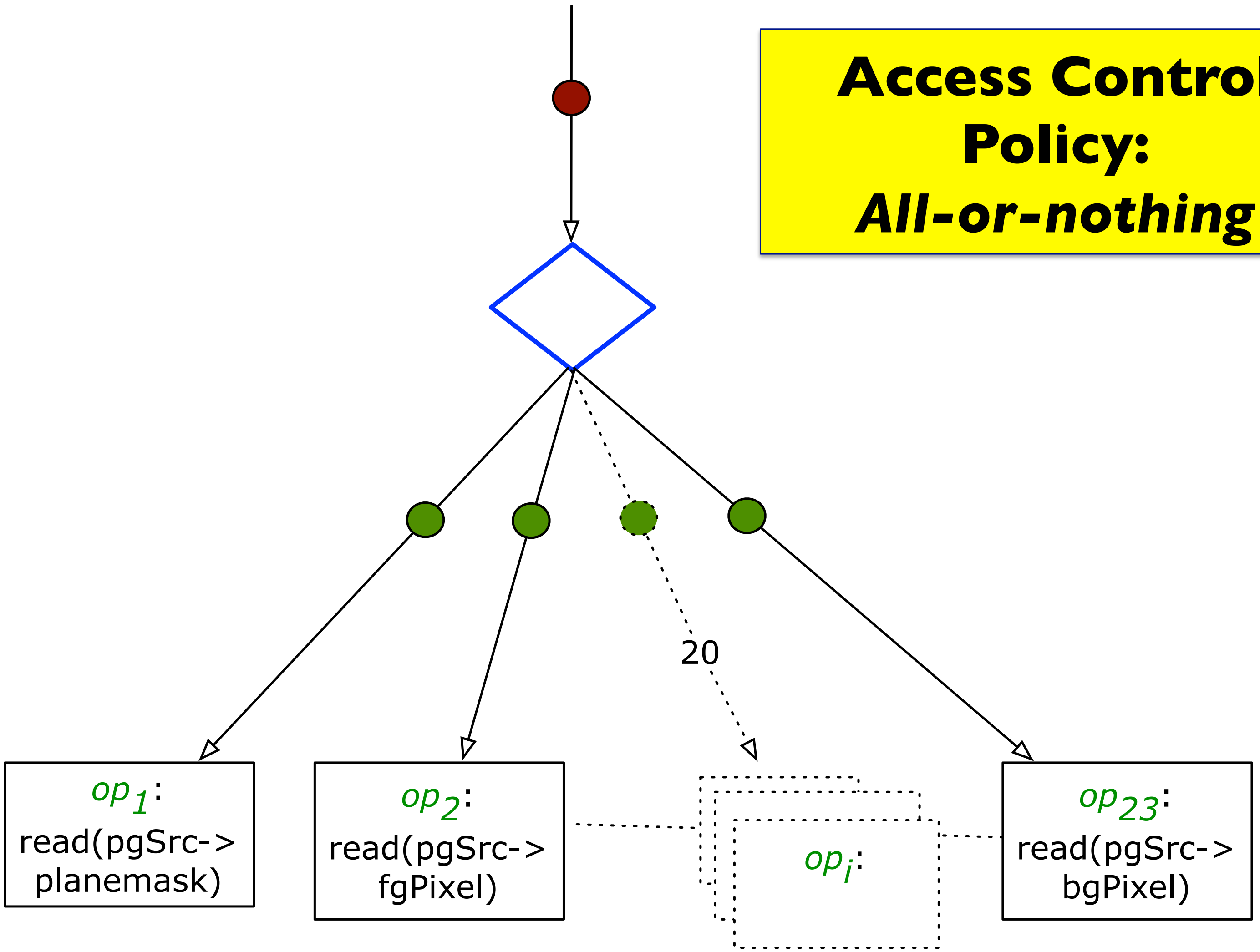
WindowPtr * pWin = (WindowPtr *) res

WindowPtr * pChild = pWin->firstChild->nextSib

pChild->mapped = True

Remove

Relate to Access Control


20
op_1: read(pgSrc-> planemask)
op_2: read(pgSrc-> fgPixel)
op_i:
op_{23}: read(pgSrc-> bgPixel)

Access Control Policy: ***All-or-nothing***

Authorization Constraints

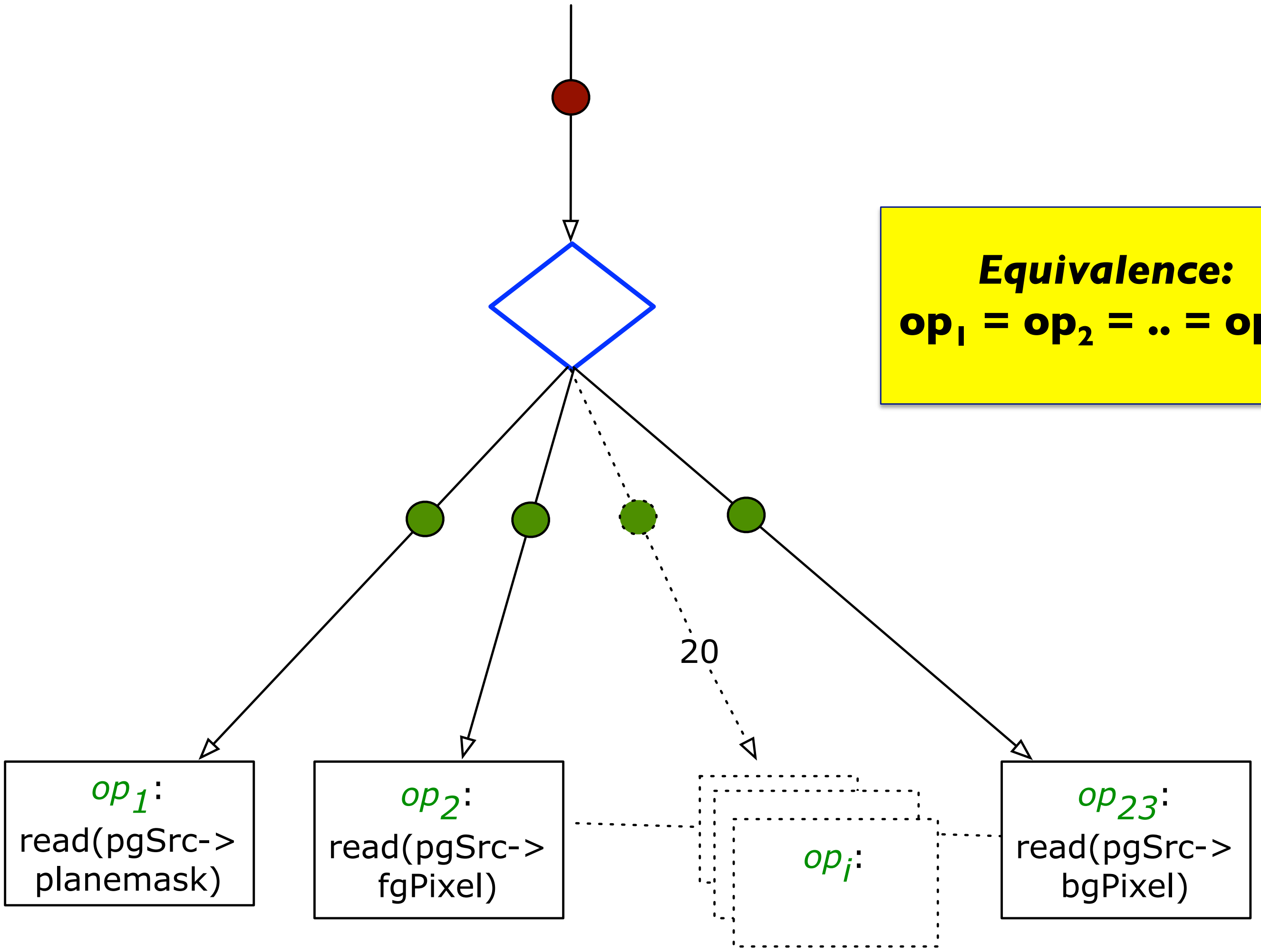
- *Allowed(o)*: Subset of subjects in *U* that are allowed to perform operation *o*.
- Constraint 1:
  - *Allowed(o1) = Allowed(o2)*, then ***o1 equals o2***
- Constraint 2:
  - *Allowed(o1)* ⊂ *Allowed(o2)*, then ***o1 subsumes o2***

Authorization Constraints

- *Allowed(o)*: Subset of subjects in *U* that are allowed to perform operation *o*.
- Constraint 1:
  - *Allowed(o1) = Allowed(o2)*, then ***o1 equals o2***
- Constraint 2:
  - *Allowed(o1)* $\subset$ *Allowed(o2)*, then ***o1 subsumes o2***

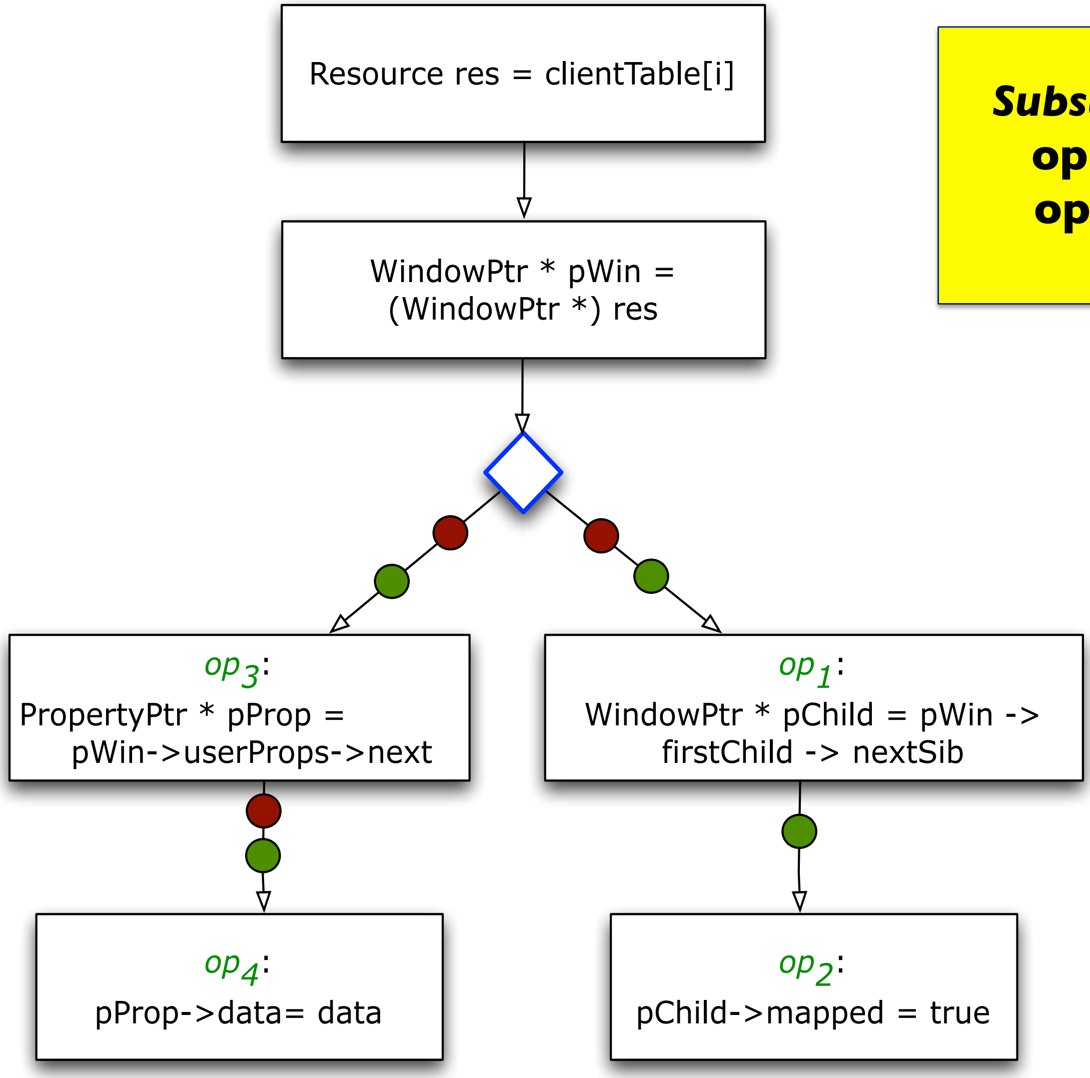
Set of Authorization Constraints limit the access control policies that can be enforced

Equivalence


20
op_1: read(pgSrc->planemask)
op_2: read(pgSrc->fgPixel)
op_i:
op_{23}: read(pgSrc->bgPixel)

Equivalence:
$op_1 = op_2 = .. = op_{23}$

Subsumption



Resource res = clientTable[i]

WindowPtr * pWin = (WindowPtr *) res

op$_3$: PropertyPtr * pProp = pWin->userProps->next

op$_1$: WindowPtr * pChild = pWin -> firstChild -> nextSib

op$_4$: pProp->data= data

op$_2$: pChild->mapped = true

Subsumption:

$op_1 > op_2$

$op_3 \not> op_4$

Build Retrofitting Policies

- How do programmers build retrofitting policies?
  - Hundreds of hooks could be removed
-

Build Retrofitting Policies

- However, there are common policy assumptions
  - E.g., object flows – if two operations produce the same data flow, such as from the object to the client (read), then they may be assumed to be equivalent
  - Under this constraint, we could still enforce MLS
- Apply "constraint selectors" to collect such authorization constraints from code
  - Removes up to 2/3 of the unnecessary hooks

Retrofitting for Authorization

- (1) Identify security-sensitive operations
  - Mostly-automated identification of operations [CCS 2012]
- (2) Produce retrofitting policy
  - Produce default authorization hook placement for SSOs
  - Apply constraint selectors for high-level policy constraints
  - Interactive selection of other authorization constraints
- (3) Generate minimal* authorization hook placement
  - Based on retrofitting policy (* modulo assumptions)
- (4) Validate reference monitor concept relative to retrofitting policies and correct transformation

Other Security Controls

- Retrofitting for Privilege Separation and Auditing

Shen Liu, Gang Tan, Trent Jaeger. PtrSplit: Supporting General Pointers in Automatic Program Partitioning. In Proceedings of the 24th ACM Conference on Computer and Communications Security (ACM CCS), October 2017.

Sepehr Amir-Mohammadian, Stephen Chong, Christian Skalka. Correct Audit Logging: Theory and Practice. In Proceedings of the 5th International Conference on Principles of Security and Trust, 2016.

Retrofitting for Auditing

Retrofitting for Auditing

- *Audit logs* are intended to provide information about programs to support:
  - Accountability and proof of authorization.
  - Surveillance and intrusion detection.
  - Dynamic analysis for performance/security evaluation.
- Current practice missing crucial foundational elements:
  - What is the *formal relation* between a program and its audit log?
  - What *policy* specifies audit log generation?

Retrofitting for Auditing

- We propose an information algebraic semantics of auditing that takes as input:
  - An arbitrary program p in a given language.
  - A *logging policy LP* that specifies conditions for logging particular events. (i.e., retrofitting policy)
- This semantics, written *genlog*(p, *LP*) denotes a set of *information*. An audit log L is *sound* (resp. *complete*) with respect to the policy iff:
  - $L \leq \mathit{genlog}(p, \mathit{LP})$ (resp. $\mathit{genlog}(p, \mathit{LP}) \leq L$) where $\leq$ is an *information containment relation.*

Putting it all together

- Retrofit for multiple security controls
  - Claim: reasoning about retrofitting policies across security controls enables benefits

Retrofitting for All

- Benefits of retrofitting policies
  - Separate security program from functional program
    - Prevent errors in integration of the two – even for updates
    - Make policy enforcement expectations explicit
  - Leverage the relationships between security controls
    - Remove redundant security controls
    - Use security controls to improve retrofitting policies
- Bottom line: there is no silver bullet - programmers will need to add such security controls

Summary

- Problem: Place Security Controls in Legacy Code
  - Hard to do manually
- Insights:
  - Program expectations of security controls into "retrofitting policies" or "security programs"
  - Retrofit programs automatically to minimize cost, validate correctness of security and function
  - Apply across a set of security controls for coherent "Defense in Depth"
- Targets: Authorization, Privilege Separation, and Auditing
- Future: How shall programmers "Design/program for security" ?