Software Foundations of Security and Privacy (15-316, spring 2017)

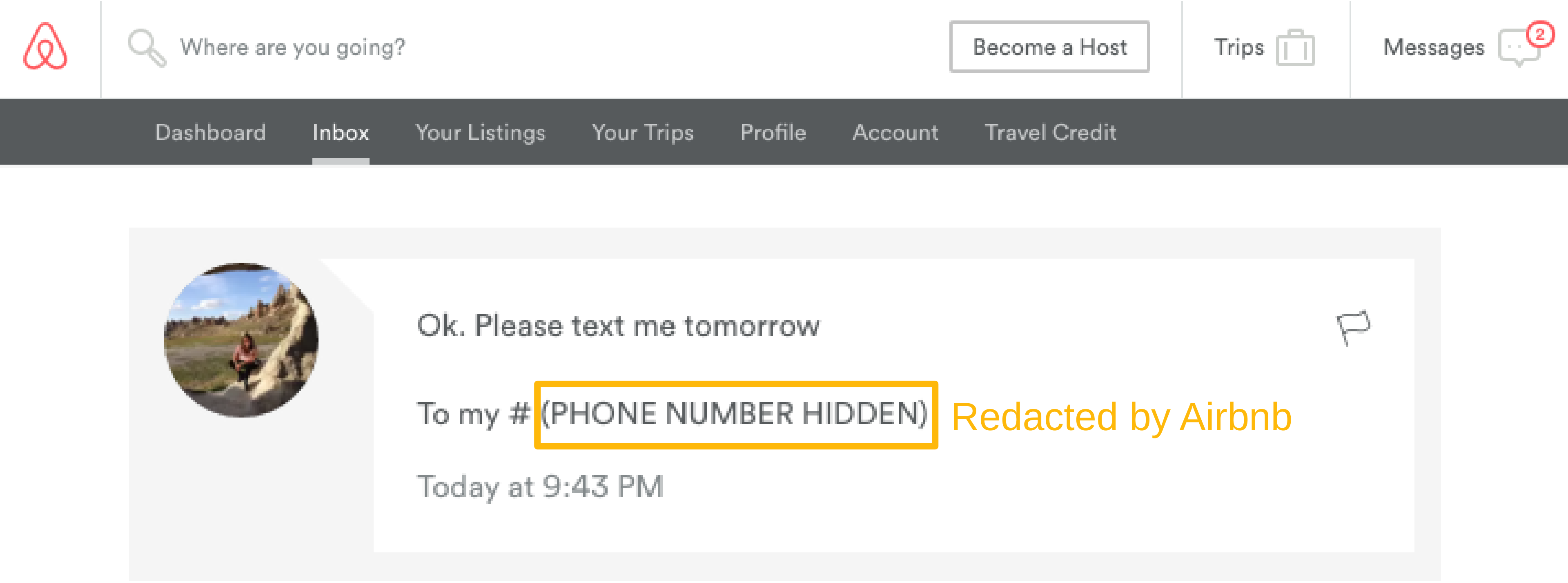
Lecture 11: Information Flow (1)

Jean Yang

jyang2@andrew.cmu.edu

Goal: Keep Secrets Secret

Example courtesy of Chelsea Voss



Ok. Please text me tomorrow

To my # (PHONE NUMBER HIDDEN)

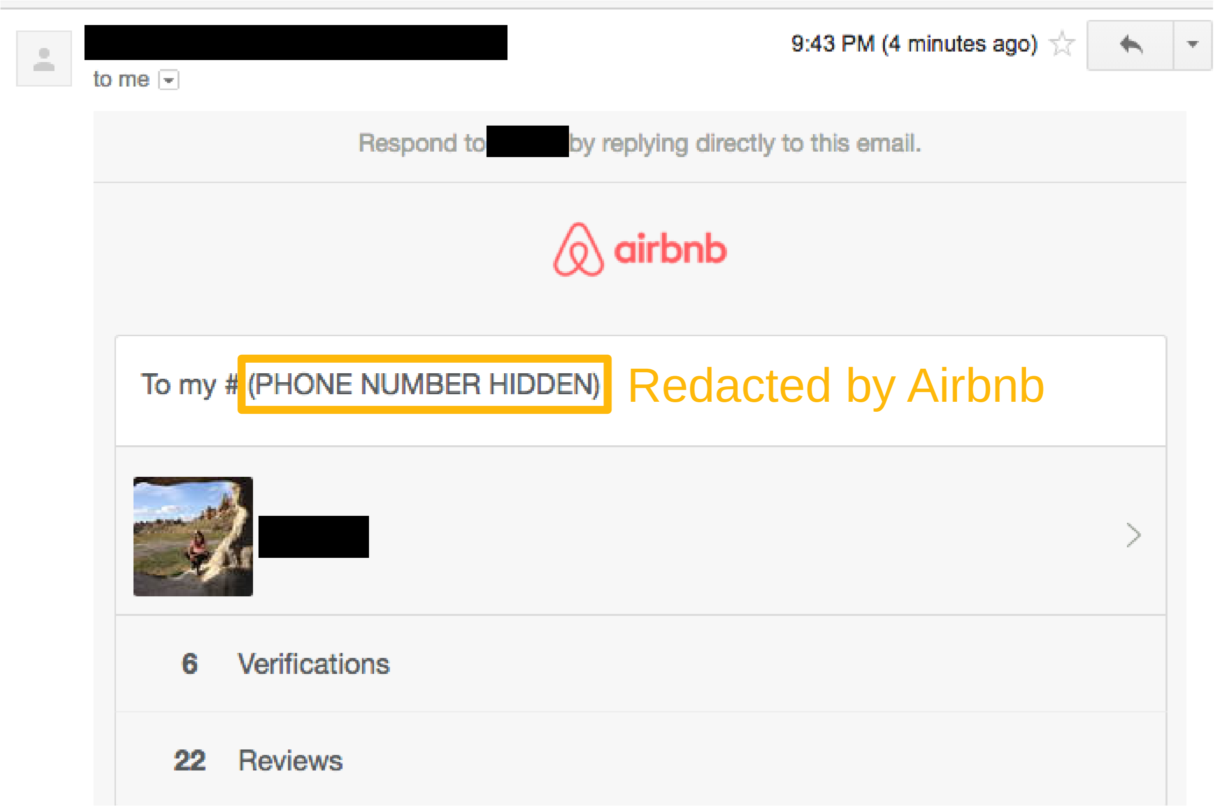
Today at 9:43 PM

Redacted by Airbnb

Airbnb has a policy of blocking phone numbers so communications happen through their application.

Redacting Phone Numbers…

Example courtesy of Chelsea Voss



[redacted]

to me

9:43 PM (4 minutes ago)

Respond to [redacted] by replying directly to this email.

To my # (PHONE NUMBER HIDDEN) Redacted by Airbnb

[redacted]

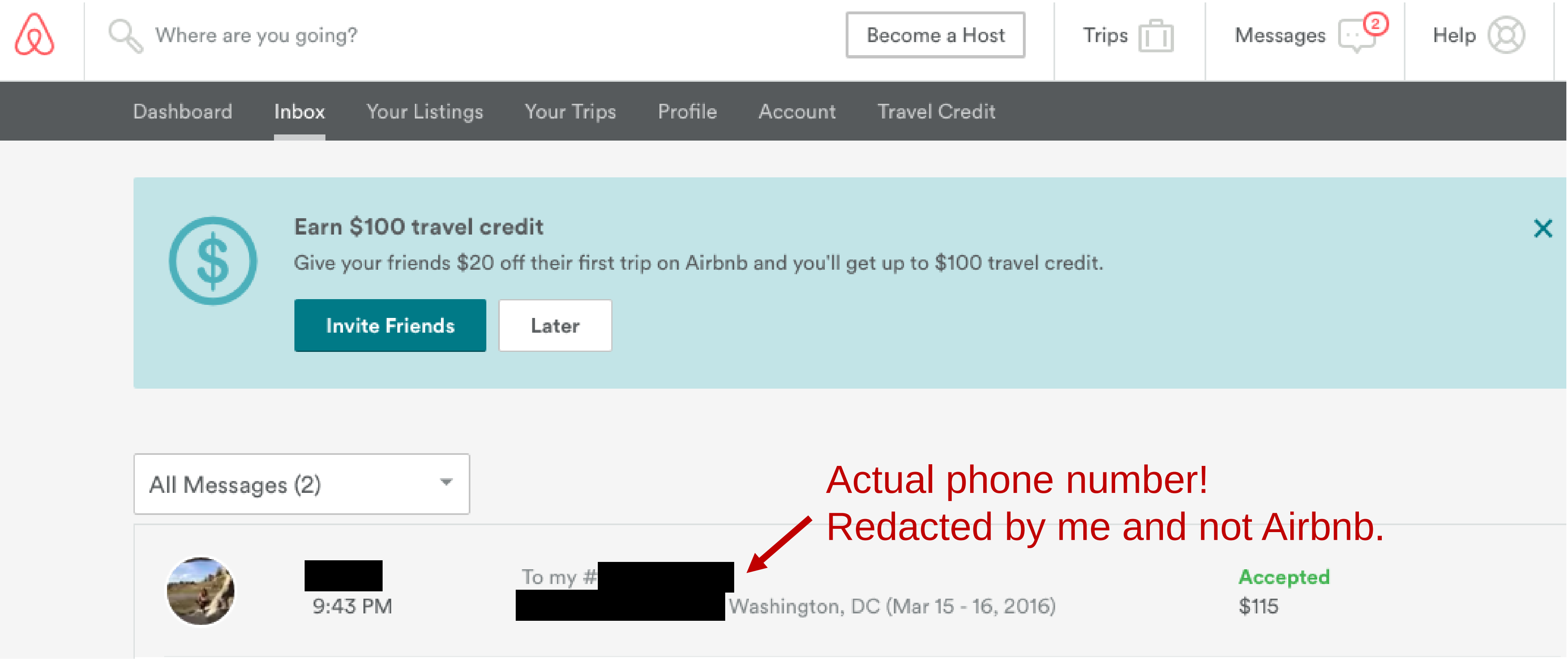
6 Verifications

22 Reviews

Phone number remains redacted in email view.

Missed a spot!

Example courtesy of Chelsea Voss



Earn $100 travel credit

Give your friends $20 off their first trip on Airbnb and you'll get up to $100 travel credit.

[redacted] 9:43 PM

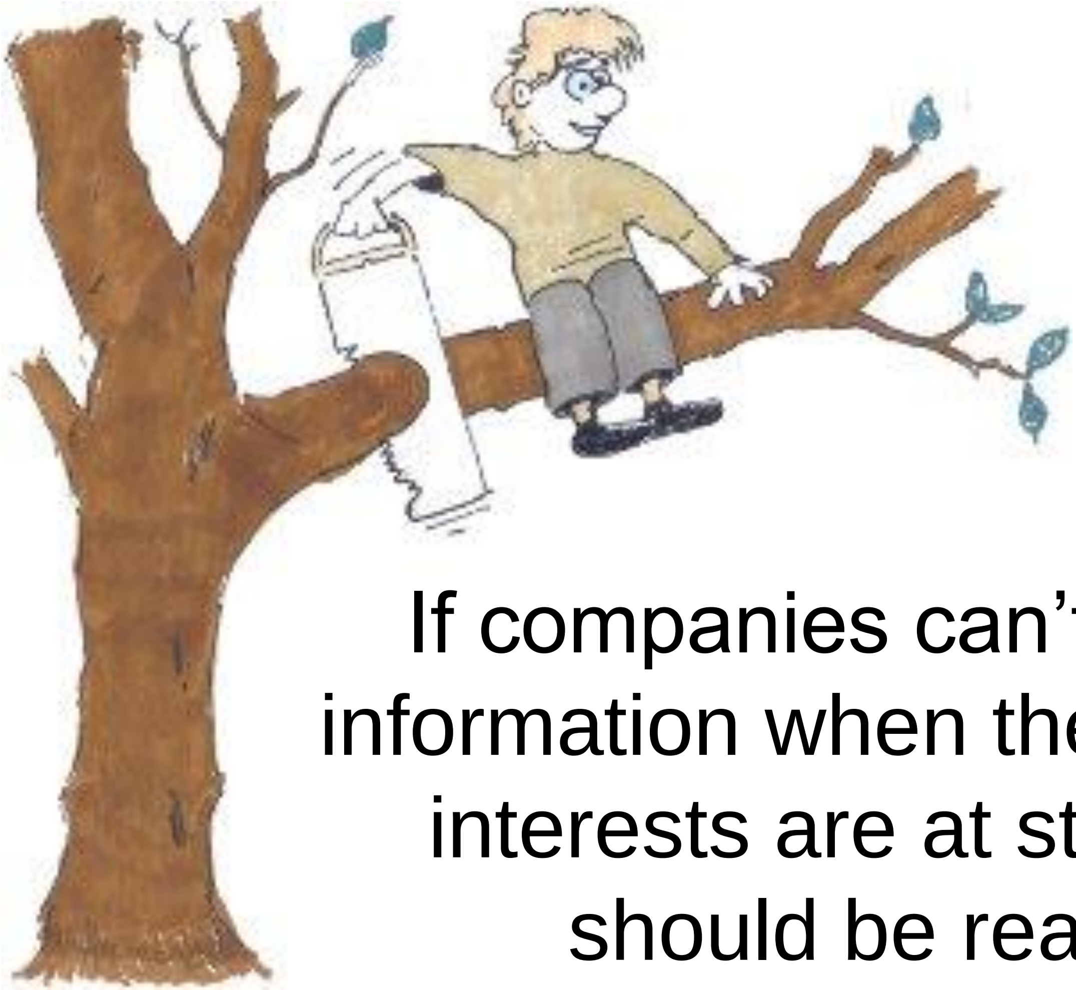
To my # [redacted] Washington, DC (Mar 15 - 16, 2016)

Accepted
$115

Actual phone number!
Redacted by me and not Airbnb.

Phone number is visible in message preview.

Main Takeaway

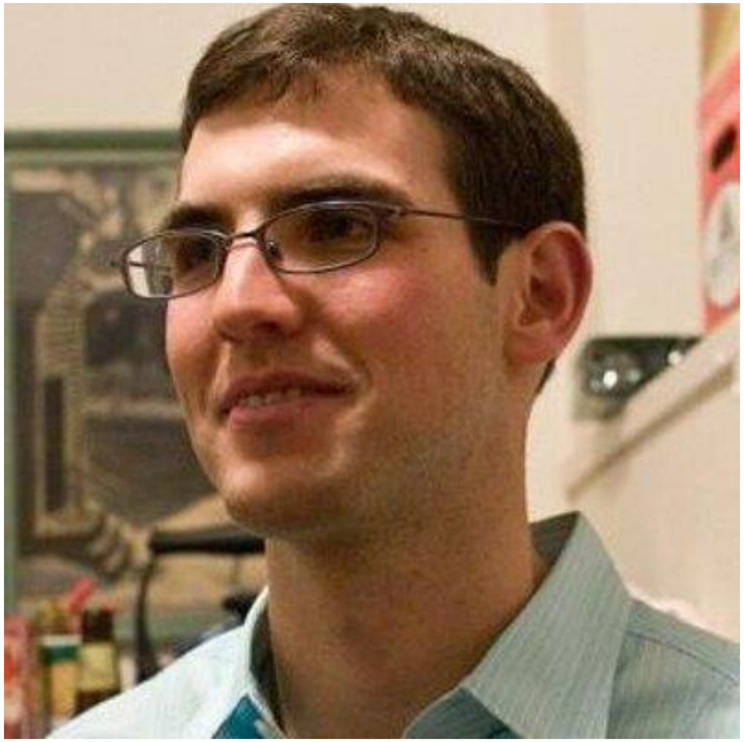


If companies can’t even protect information when their own financial interests are at stake, then we should be really afraid.

Alternative Takeaway

Companies don’t have the tools to prevent unauthorized information flows even when they are motivated to do so!

This Lecture: A Tribute to Max Krohn



@maxtaco on Twitter.

- Founded Thespark.com, OKCupid, and Keybase.
- Built OKWS for OKCupid as a PhD student at MIT.
- Continued using his research to make OKCupid’s backend better throughout his PhD.

Part One: What’s Wrong with Access Control?

Problems with Access Control

- Need to ensure the policy we are enforcing is the correct policy.
- Need to ensure we are enforcing policy according to appropriate principles under appropriate conditions.
  - Who are we showing the sensitive information to?
  - What computations have we done with the sensitive data before showing it?

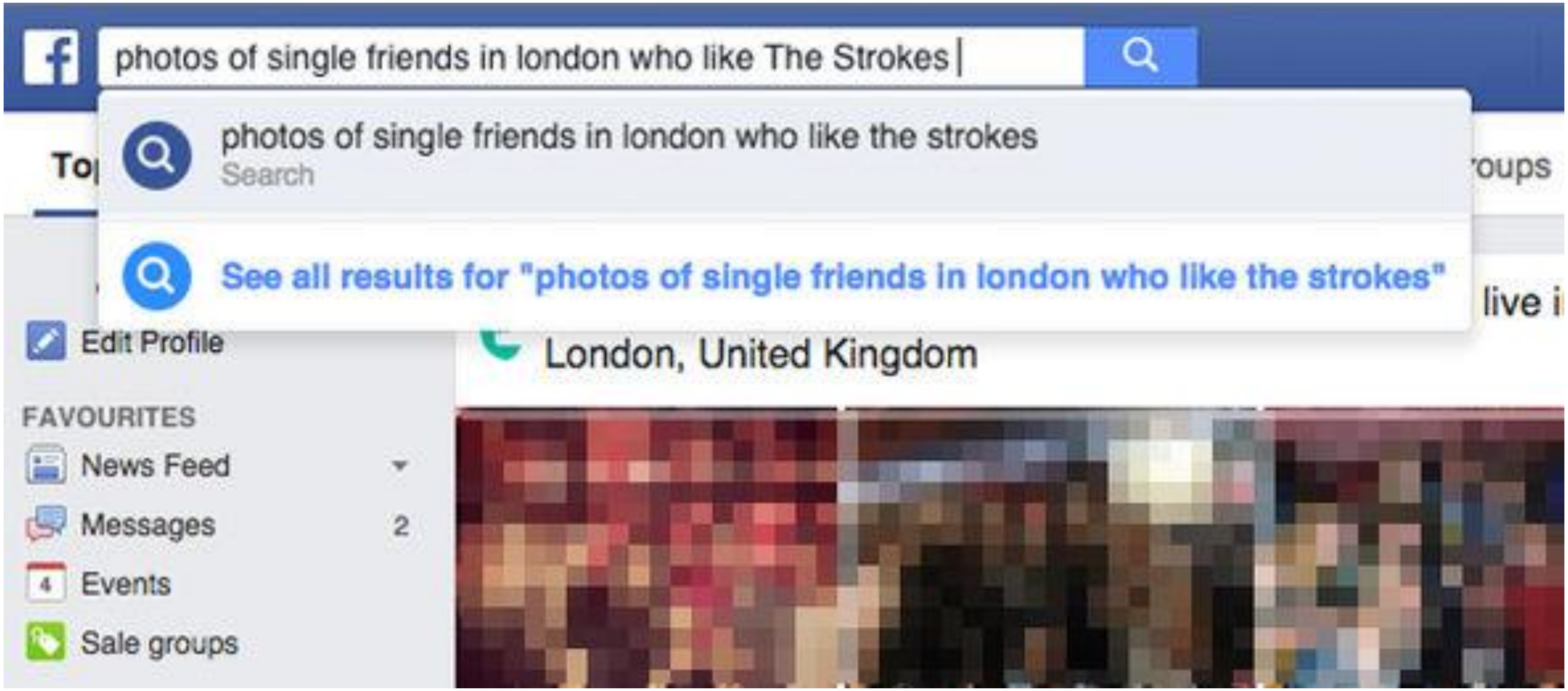
Limitations: Password Example

```
from: HotCRP
to: Eve
subject: Password Reminder

Dear Eve,
   Alice’s password is [redacted].

<3, HotCRP
```

Limitations: Search Interface



photos of single friends in london who like The Strokes

photos of single friends in london who like the strokes
Search

See all results for "photos of single friends in london who like the strokes"

London, United Kingdom

Problem: Access Control Does Not Help Track Eventual *Viewer*

Ways viewer may be unpredictable:

- Viewer determined by user input.
  - “Send mail to…”
- Viewer computed from code.
  - Send to all users in a group.

Problem: Access Control Does Not Address *Implicit Flows*

```
int x := <secret>
if (x > 0) {
    y := y+1;
}
```

Information flow from `x` to `y`!

Ways Implicit Flows May Arise

What are some examples where we could capture information flows indirectly?

- Counting all users in a given location.
- Showing someone’s photo in health record search results if some disease diagnosis is positive.

Goal: Track Sensitive Values

Lindsay Lohan with parole ankle bracelet.

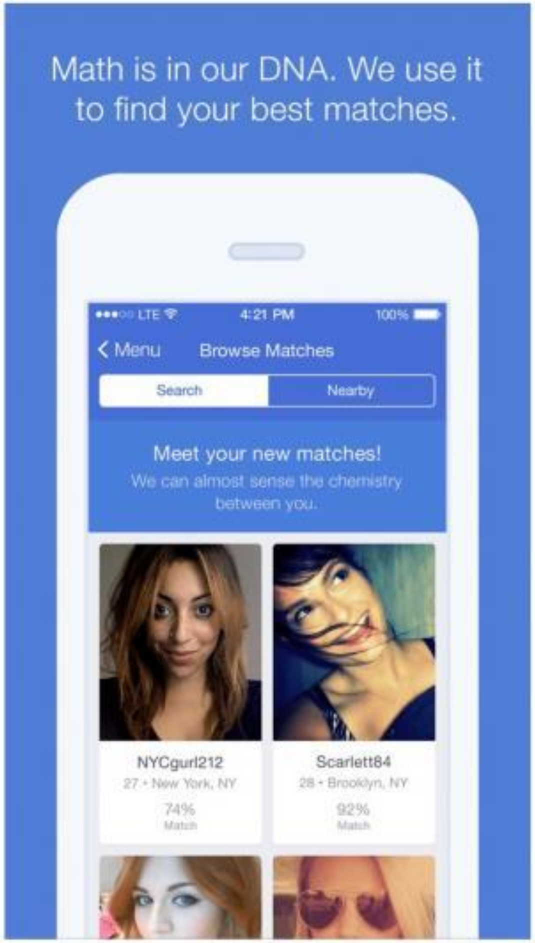
- Want to allow program to compute over sensitive values more or less freely.
- Want to prevent information from being released when there are unauthorized flows.

Some Questions

- What does access control give us?
- With access control, what is trusted?
- When isn’t access control enough?
- What do we need to address the viewer problem and the problem of implicit flows?

Part Two: Process Isolation with OKWS (Krohn 2004)

Real-World Motivation: Online Dating Involves Secrets



Math is in our DNA. We use it to find your best matches.

Meet your new matches!

We can almost sense the chemistry between you.

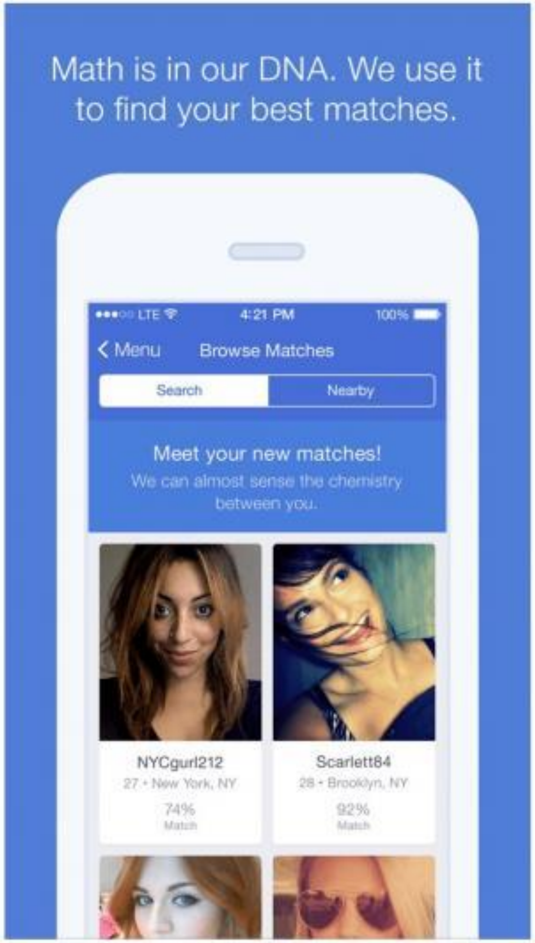
NYCgurl212
27 • New York, NY
74%
Match

Scarlett84
28 • Brooklyn, NY
92%
Match

Solution Requirements

- Needs to be able to run on Unix-based development server
- Needs to support all of the desired features of a production web server
- Needs to be fast enough to run OKCupid.com

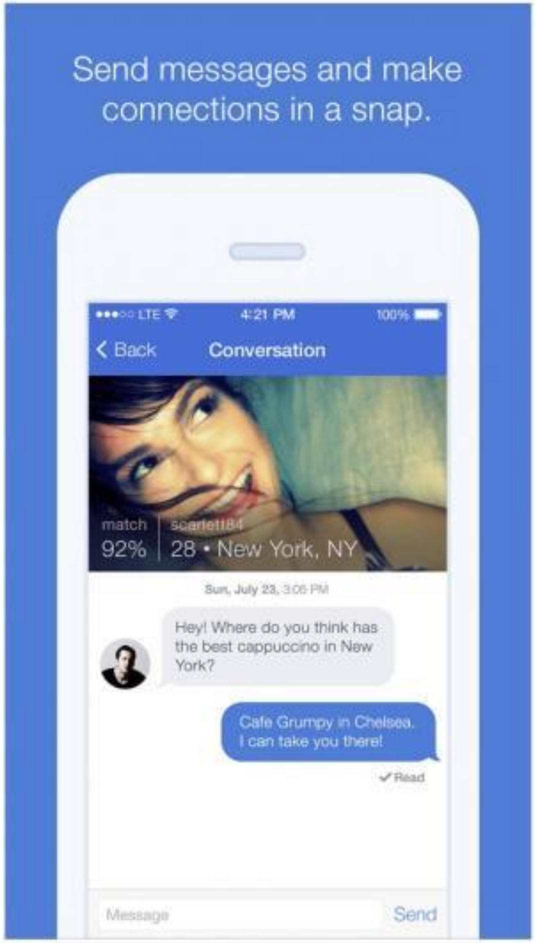
Solution: Process Isolation


Math is in our DNA. We use it to find your best matches.

Meet your new matches!
We can almost sense the chemistry between you.

Run orthogonal services (for instance “search” and “inbox” in different processes.

This way, buffer overflow in “inbox” won’t affect “profile” or “search!”


Send messages and make connections in a snap.

Hey! Where do you think has the best cappuccino in New York?

Cafe Grumpy in Chelsea. I can take you there!

Limitations

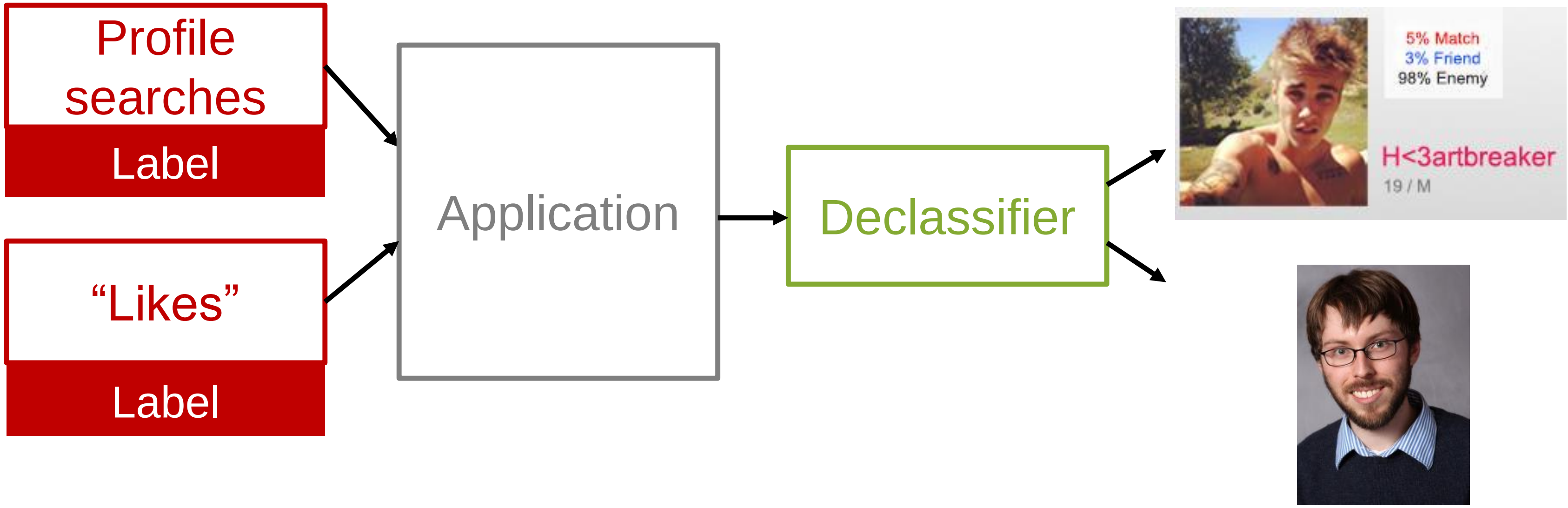
- Building secure systems with Unix is challenging because tools such as **`setuid`** and **`chroot`** conflict with common web server features such as embedded Python/Perl interpreters
- OKWS's security promises remain weak: if Bob comprises "inbox," can still read mail

Part Three: Decentralized Information Flow Control with Flume (Krohn *et al* 2007)

What If We Could Run On a Customized OS?

Decentralized Information Flow

Model for controlling information flow in systems with *mutual distrust* and *decentralized authority*. Sensitive data is *labelled* and can be *declassified* in a decentralized way.



Profile searches

Label

"Likes"

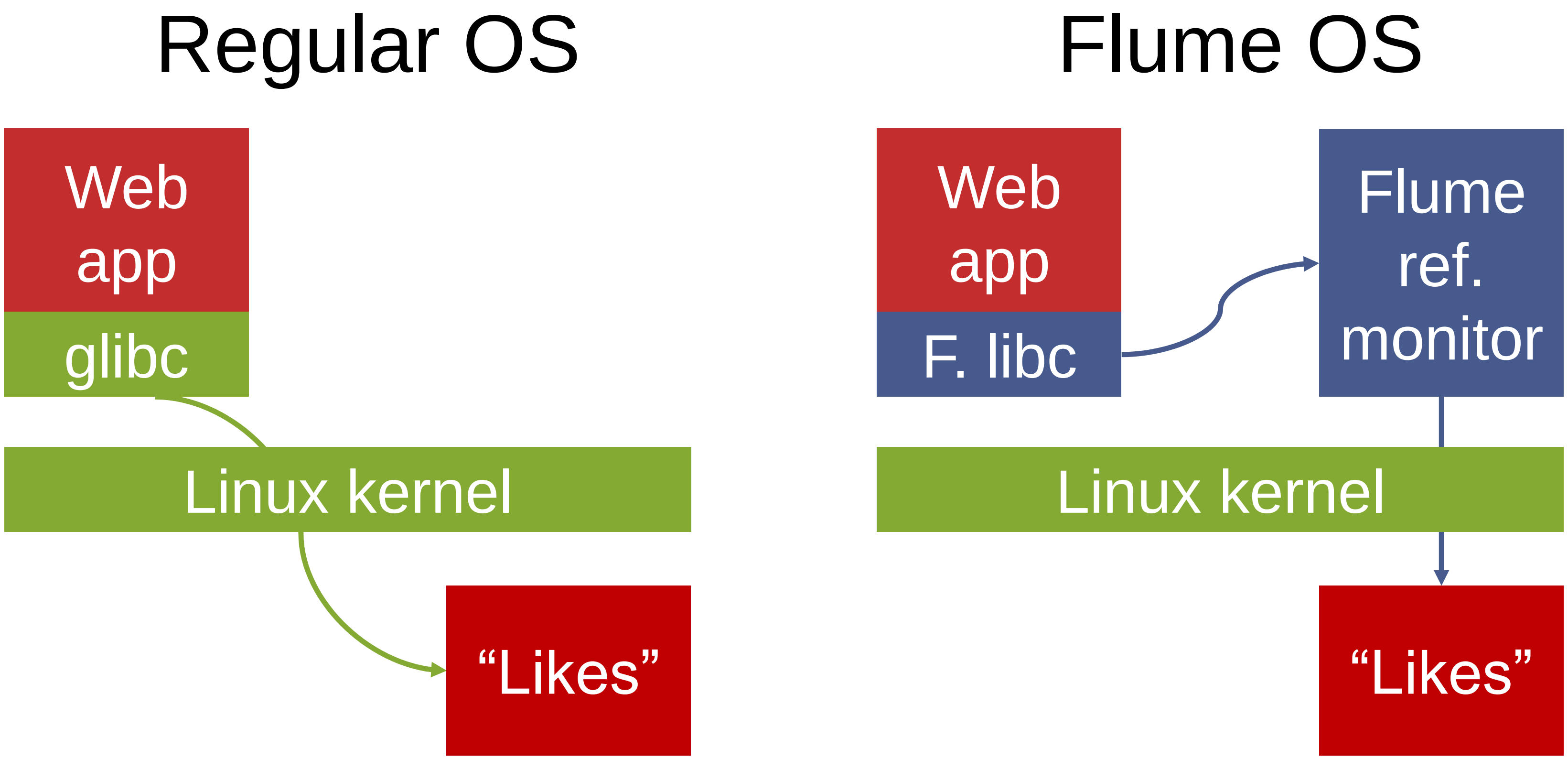
Label

Application

Declassifier

The Flume Operating System

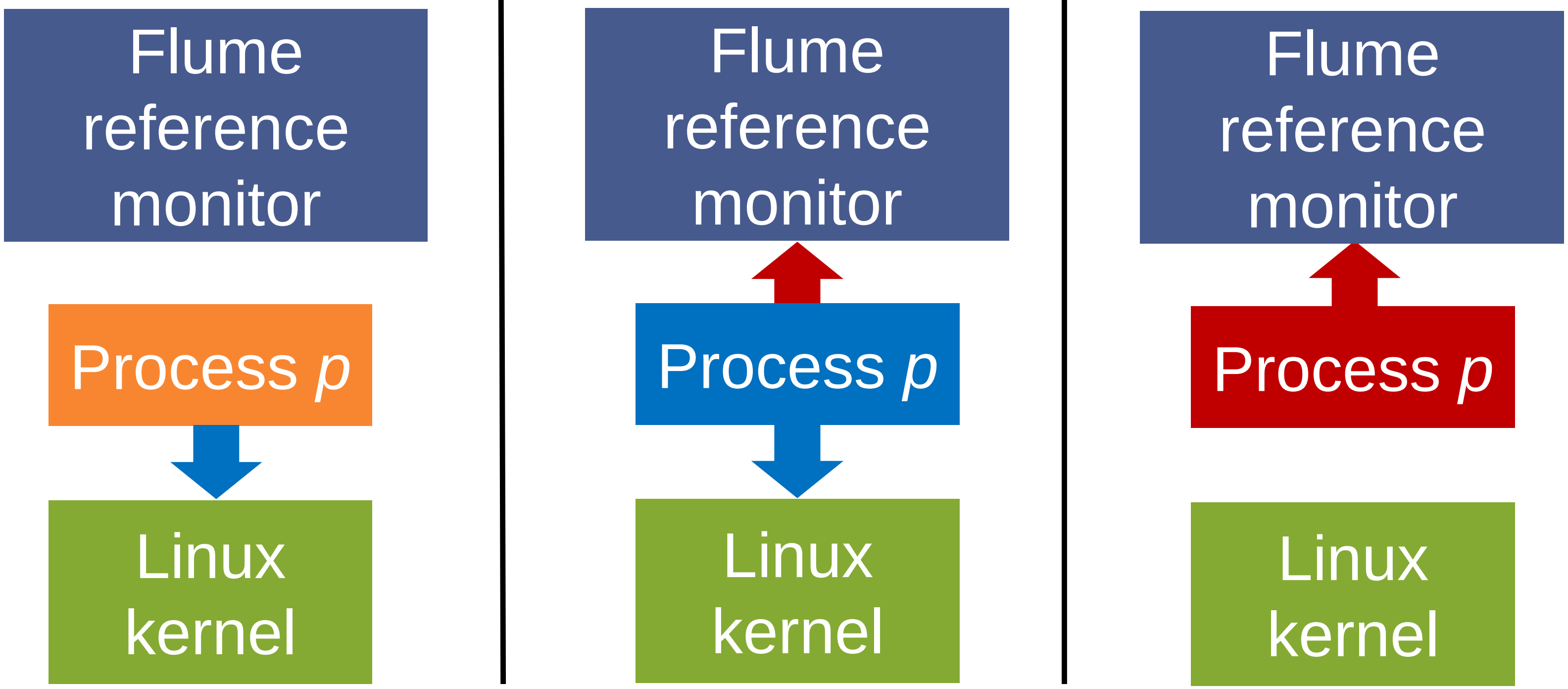
Based on system call delegation. Built in user-space with a few small kernel patches on top of Linux and OpenBSD.



Three Classes of Processes

Based on slide by Max Krohn

Flume-oblivious | Unconfined/mediators | Confined



Flume reference monitor

Process *p*

Linux kernel

Flume reference monitor

Process *p*

Linux kernel

Flume reference monitor

Process *p*

Linux kernel

Central Challenge

Accommodate process that use **existing communication interfaces** (for instance, socket and pipes) while specifying how and when they use their privileges.

- Awkward to modify each call to `read` or `write`.
- Conventional process interface full of channels that “leak” information, such as network sockets.

Solution: Labels + Endpoints

- *Label* processes with what they are allowed to read and write.
- Define how labels can be rewritten for *declassification* and *endorsement*.
- Represent each communication resource (for instance sockets and files) as an *endpoint* that specifies what subset of its privileges should be used when communicating.

Two Types of Processes

Untrusted

- Do most of the computation.
- Are constrained by, but possibly unaware of, DIFC controls.

Trusted

- Are aware of DIFC.
- Set up privacy and integrity controls that constrain untrusted processes.
- Have *privilege* to selectively violate classical information flow through *declassification* and *endorsement*.

Simple Label System

- **Goal:** track which secrets a process has accessed.
- **Mechanism:** each process gets a *secrecy label* summarizing the categories of data a process is assumed to have accessed.
  - { “Likes” } Tag
  - { “Financial reports” }
  - { “Likes” and “15-316 grades” } Label

Some Nomenclature

- **Confidentiality:** protecting sensitive reads.
  - When should a process be "authorized?"
  - Encryption provides end-to-end confidentiality, but it's difficult to compute on encrypted data
- **Integrity:** protecting sensitive writes.
  - Only authorized processes can write a file
  - Digital signatures provide end-to-end integrity, but cannot change signed data

Confidentiality and Integrity

- Secrecy label (S_p)
  - Specifies what data process *p* has read
  - “/usr/bin/login may read the password file”
- Integrity label (I_p)
  - Used to **endorse** the trustworthiness of *p*
  - “/usr/bin/login can only be updated by root”
  - “/usr/bin/login can only read user libs and config files endorsed by root”

Privilege

Ownership (O_p) regulates how *p* can update S_p and I_p.

- **Endorsement:** tags *p* can add to its labels (e.g. t^+)
- **Declassification:** tags *p* can remove from its labels (e.g. t^-)
- D_p is the set of tags that *p* can both add and remove

Secrecy and Integrity, More Formally

- **Secrecy:** "At some point process *p* added data with tag *s* to its address space."
  - $s \in S_p \Rightarrow \exists(data): p\ read\ data\ with\ tag\ s$
- **Integrity:** "All inputs to process *p* had tag *i*."
  - $i \in I_p \Rightarrow \forall(data): p\ read\ data\ with\ tag\ i$

Privilege, More Formally

“p can remove tag s from S_p and add tag *i* to I_p”

- $s \in t^- \Rightarrow p\ is\ trusted\ to\ declassify\ s$
- $i \in t^+ \Rightarrow p\ is\ trusted\ to\ endorse\ i$
- $t \in D_p \Rightarrow t \in t^- and\ t \in t^+$

Tags + Secrecy Labels

Based on slide by Max Krohn

P

Secrets P has viewed

$S_p = \{\}$ $S_p = \{ Finance \}$ $S_p = \{ Finance, HR \}$

$D_p = \{\}$ $D_p = \{ HR \}$

Tags P can add and remove from its label

```
change_label({Finance});
tag_t HR = create_tag();
change_label({});
change_l                  R});
change_l
```

Any process can

DIFC Rule: A process can create a new tag; gets ability to declassify it.

Declassification in action.

Universe of Tags:

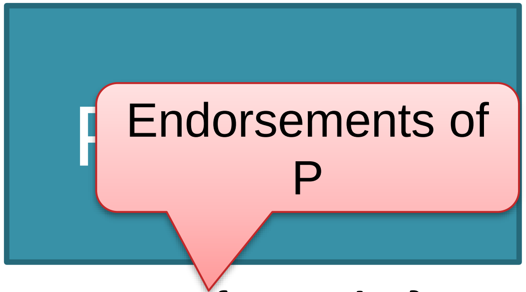
HR

Finance

SecretProjects

Tags + Integrity Labels

Based on slide by Max Krohn



Endorsements of P

I_p = {Apple}

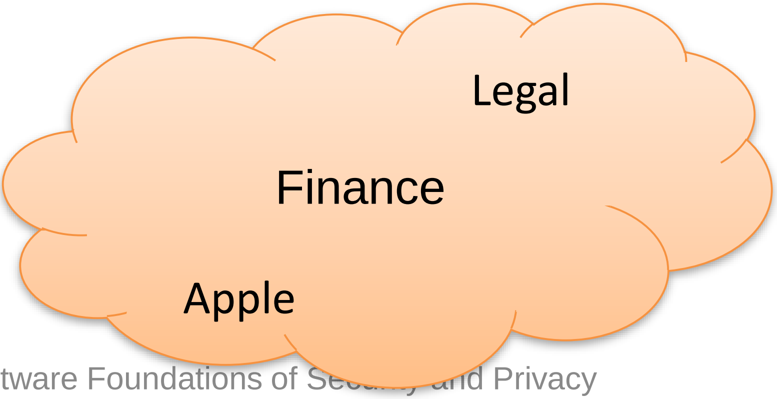
D_p = {}

Tags P can add and remove from its label

```
change_label({});
```

Any process can remove any tag from its label.

Universe of Tags:



Tags + Integrity Labels

Based on slide by Max Krohn

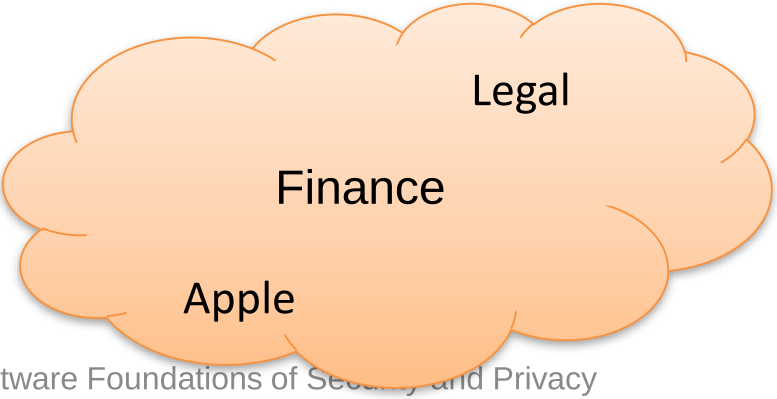
Process *p*

$I_p = \{\}$

$D_p = \{\}$

```
change_label({});
```

Universe of Tags:



Legal

Finance

Apple

Tags + Integrity Labels

Based on slide by Max Krohn

Process *p*

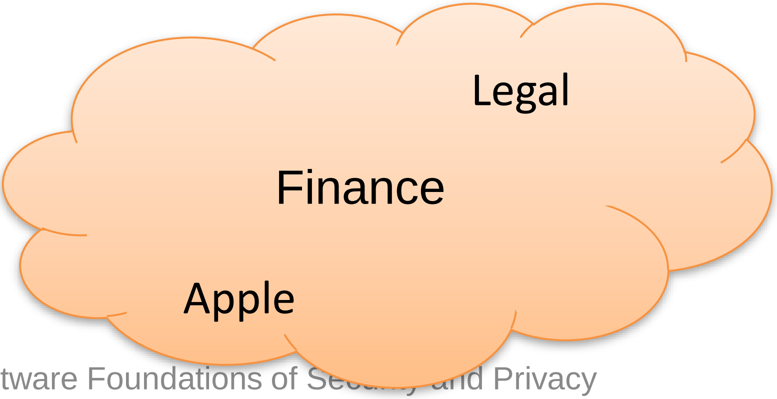
$I_p = \{\}$

$D_p = \{\}$

```
change_label({});

tag_t HR = create_tag();
```

Universe of Tags:



Tags + Integrity Labels

Based on slide by Max Krohn

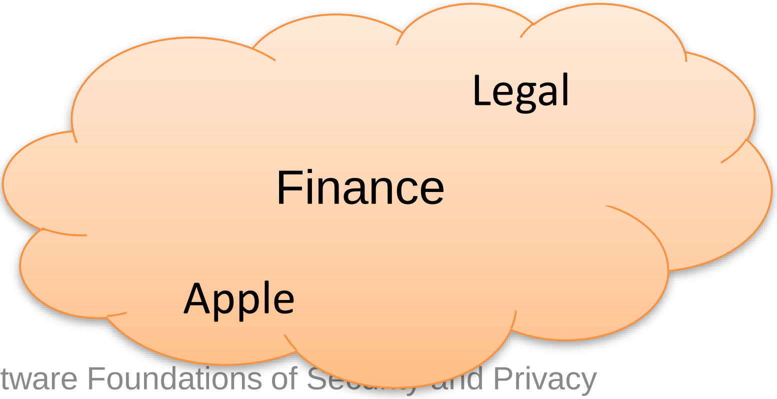
Process *p*

$I_p = \{\}$

$D_p = \{\}$

```
change_label({});
tag_t HR = create_tag();
```

Universe of Tags:



Legal

Finance

Apple

Tags + Integrity Labels

Based on slide by Max Krohn

Process *p*

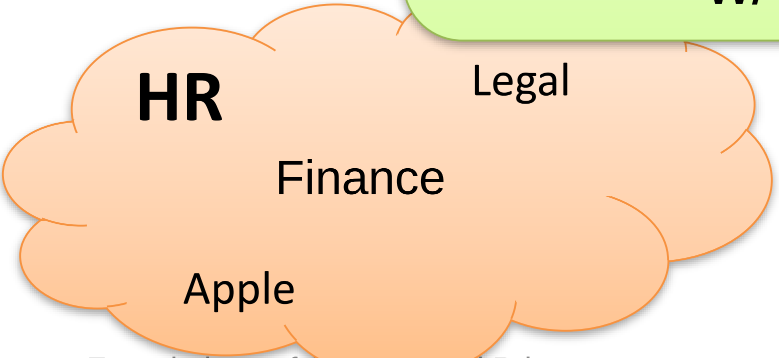
I_p = {}

D_p = {HR}

```
change_label({});
tag_t HR = create_tag();
```

DIFC Rule: A process can create a new tag; gets ability to endorse w/ it.

Universe of Tags:



Tags + Integrity Labels

Based on slide by Max Krohn

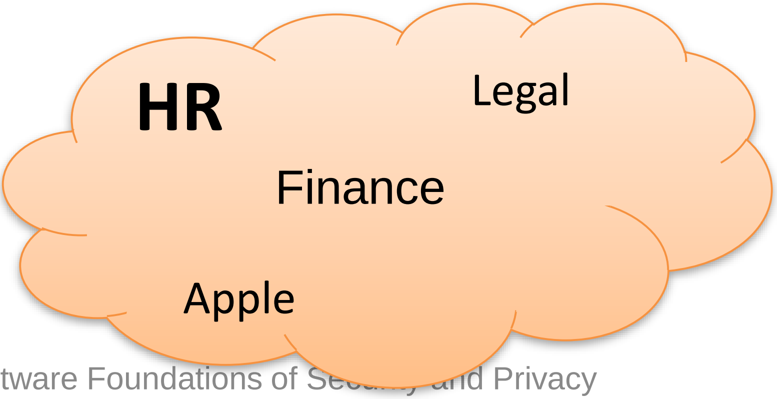
Process *p*

$I_p = \{\}$

$D_p = \{HR\}$

```
change_label({});
tag_t HR = create_tag();
change_label({HR});
```

Universe of Tags:



Tags + Integrity Labels

Based on slide by Max Krohn

Process *p*

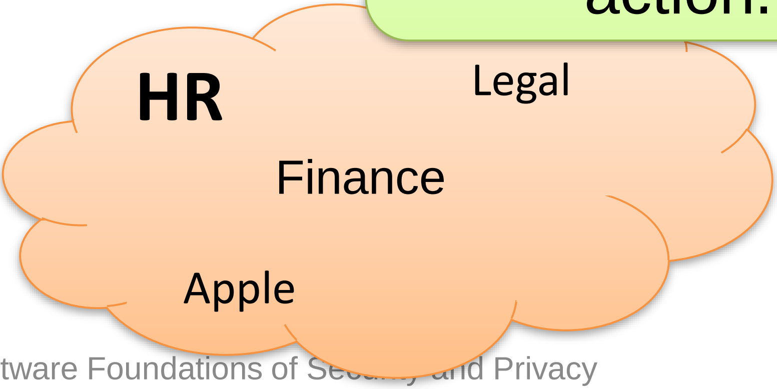
I_p = {HR}

D_p = {HR}

```
change_label({});
tag_t HR = create_tag();
change_label({HR});
```

DIFC: Endorsement in action.

Universe of Tags:



HR

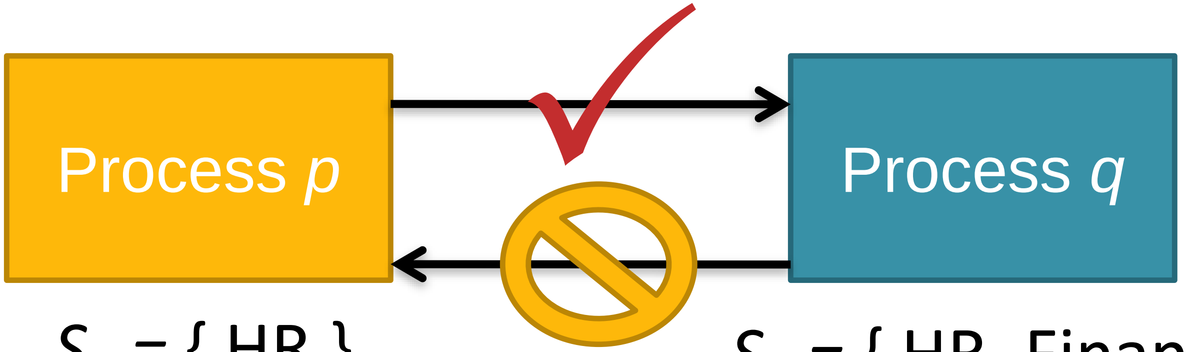
Legal

Finance

Apple

Communication Rule

Based on slide by Max Krohn



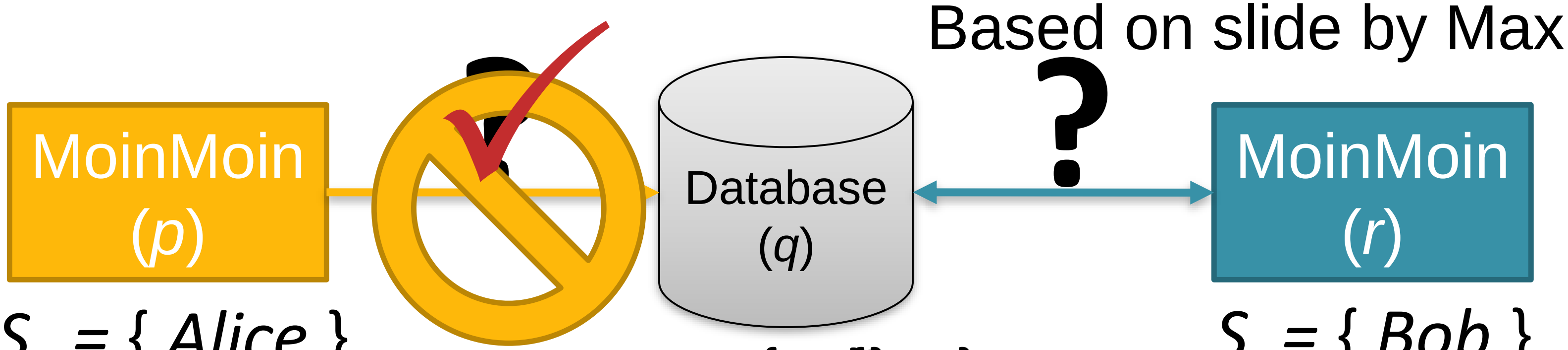
$S_p = \{ \text{HR} \}$

$S_q = \{ \text{HR, Finance} \}$

p can send to *q* iff $S_p \subseteq S_q$

Flume Communication Rule

Based on slide by Max Krohn



$S_p = \{ \, Alice \, \}$

$S_q = \{ \, Alice \, \}$

$S_q = \{\}$

$D_q = \{ \, Alice, Bob \, \}$

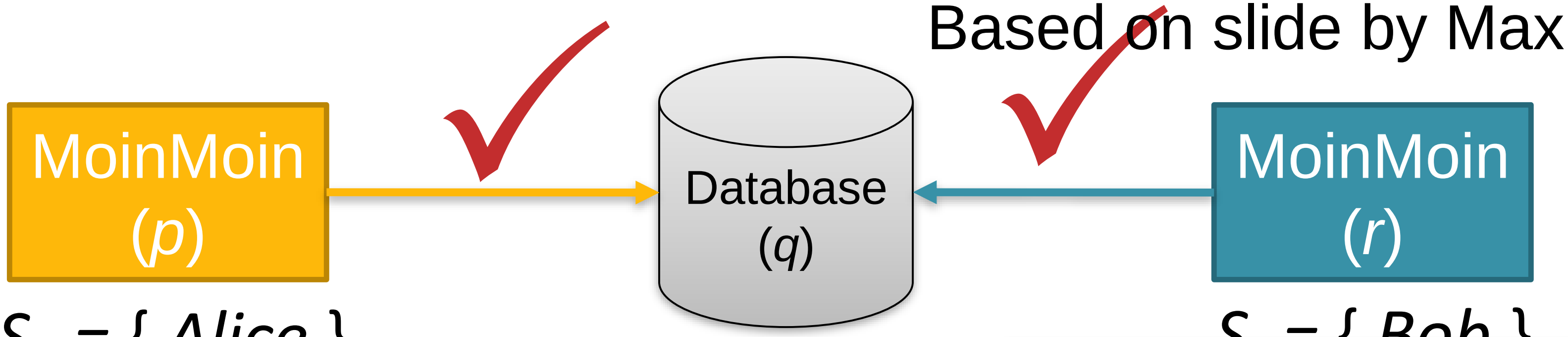
$S_r = \{ \, Bob \, \}$

$$S_p \not\subseteq S_q$$

1. *q* changes to $S_q = \{ \, Alice \, \}$
2. *p* sends to *q*
3. *q* changes back to $S_q = \{\}$

Flume Communication Rule

Based on slide by Max Krohn



$S_p = \{ Alice \}$

$S_r = \{ Bob \}$

$S_q = \{\}$

$D_q = \{ Alice, Bob$

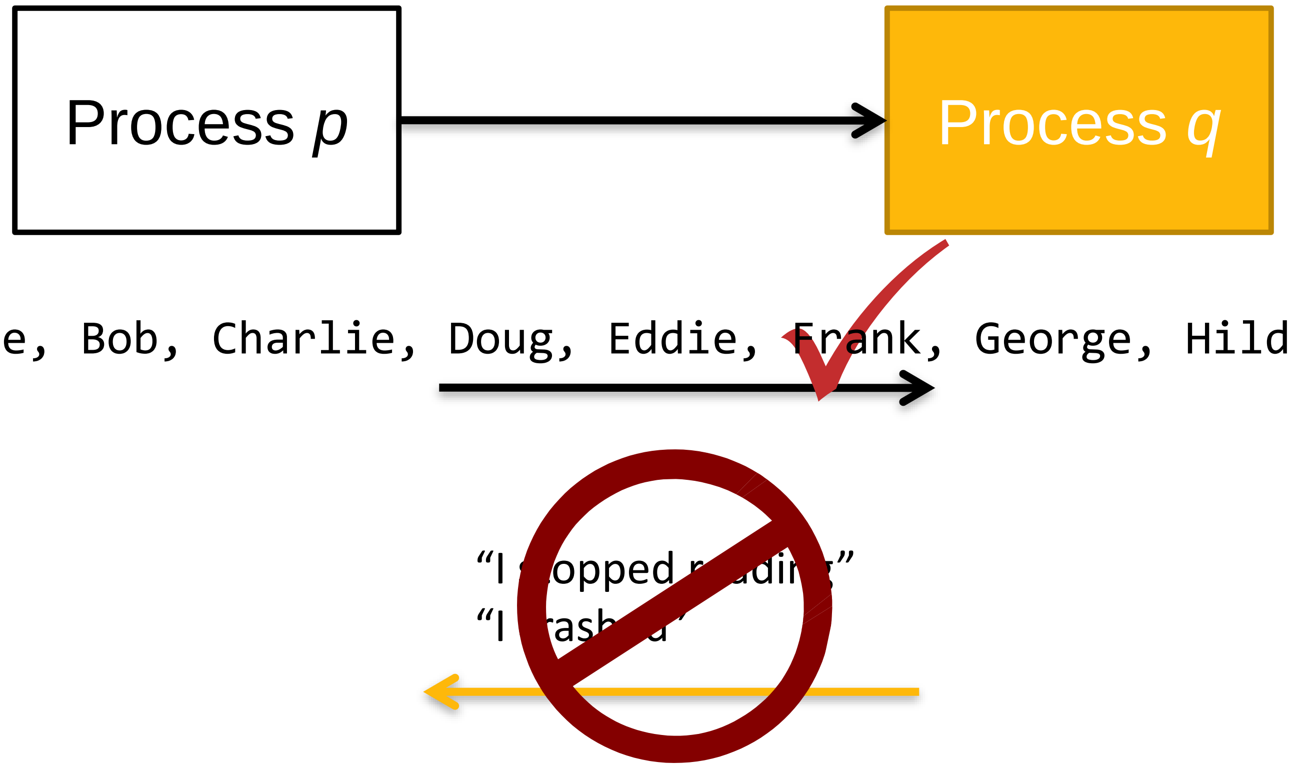
Senders get extra latitude

Receivers get extra latitude

- *p* can send to *q* iff:
  - In IFC: $S_p \subseteq S_q$
  - In Flume: $S_p - D_p \subseteq S_q \cup D_q$

The Unexpected Behavior Problem

Based on slide by Max Krohn



Process *p*

Process *q*

“Fire Alice, Bob, Charlie, Doug, Eddie, Frank, George, Hilda, Ilya…”

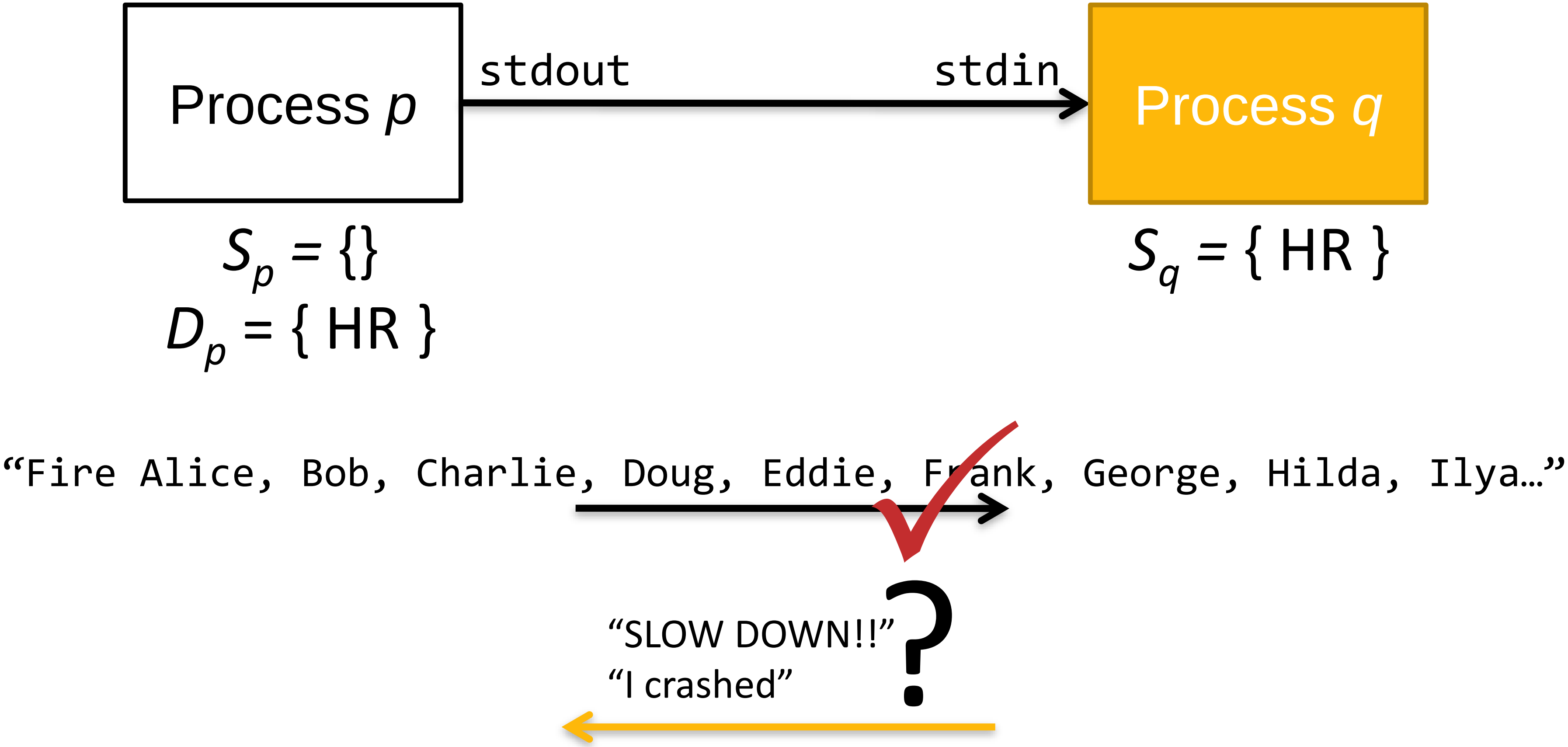
“I stopped reading”

“I crashed”

Exposing failure messages can leak information!

The Unexpected Behavior Problem

Based on slide by Max Krohn



Process *p*

stdout

stdin

Process *q*

$S_p = \{\}$

$D_p = \{ HR \}$

$S_q = \{ HR \}$

“Fire Alice, Bob, Charlie, Doug, Eddie, Frank, George, Hilda, Ilya…”

“SLOW DOWN!!”

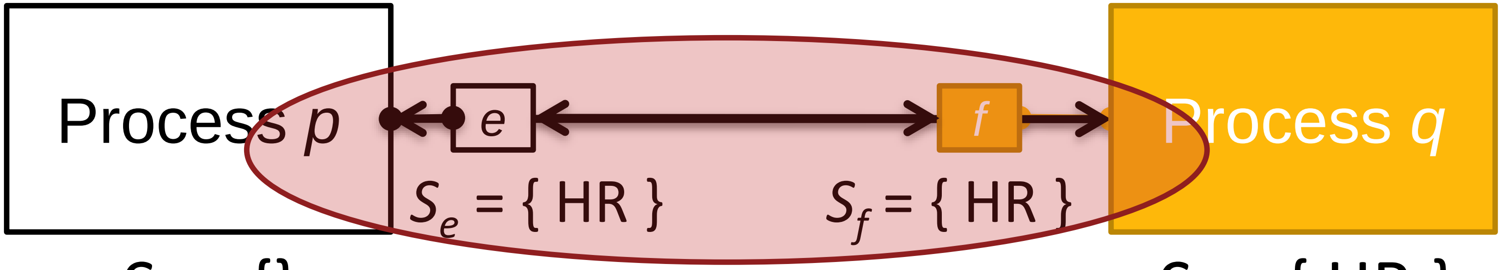
“I crashed”

?

Slide by Max Krohn

Solution: Endpoint Abstraction

Based on slide by Max Krohn



$S_e = \{ \text{HR} \}$ $S_f = \{ \text{HR} \}$

$S_p = \{\}$

$D_p = \{ \text{HR} \}$

$S_q = \{ \text{HR} \}$

- If $S_e \subseteq S_f$, then allow *e* to send to *f*
- If $S_f \subseteq S_e$, then allow *f* to send to *e*
- If $S_f = S_e$, then allow bidirectional flow

“Fire Alice, Bob, Charlie, Doug, Eddie, Frank, George, Hilda, Ilya…”

“SLOW DOWN!!”

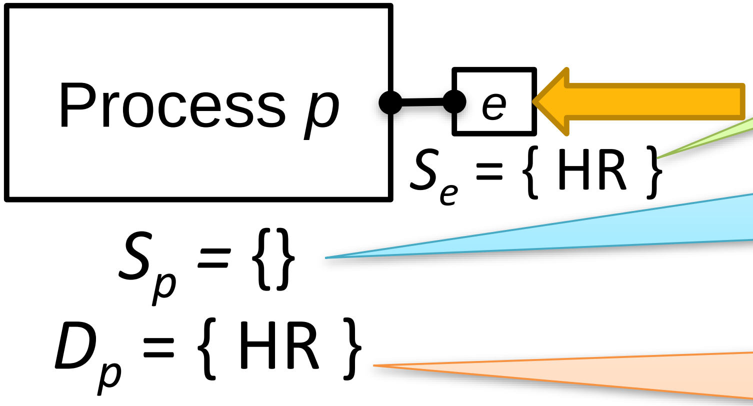
“I crashed”

Benefits of Endpoints

- Simplifies application programming. When a process attempts and fails to adjust labels on its endpoints, system can safely report errors.
- Make many declassification and endorsement decisions *explicit*. Flume processes must explicitly mark file descriptors that serve as avenues for declassification/endorsement.

Endpoints Declassify Data

Based on slide by Max Krohn



S_e = { HR }

S_p = {}

D_p = { HR }

Data enters process p with secrecy { HR }

But p keeps its label S_p = {}

Thus p needs HR $\in D_p$

Slide by Max Krohn

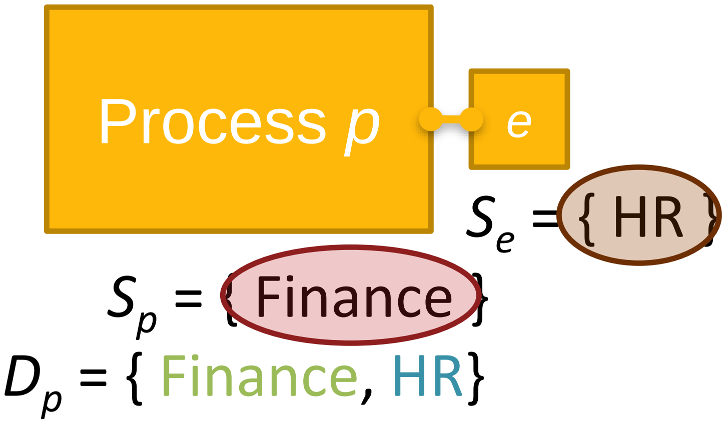
Endpoint Invariant

Based on sl Krohn

- For any tag $t \in S_p$ and $t \notin S_e$
- Or any tag $t \in S_e$ and $t \notin S_p$
- It must be that $t \in D_p$

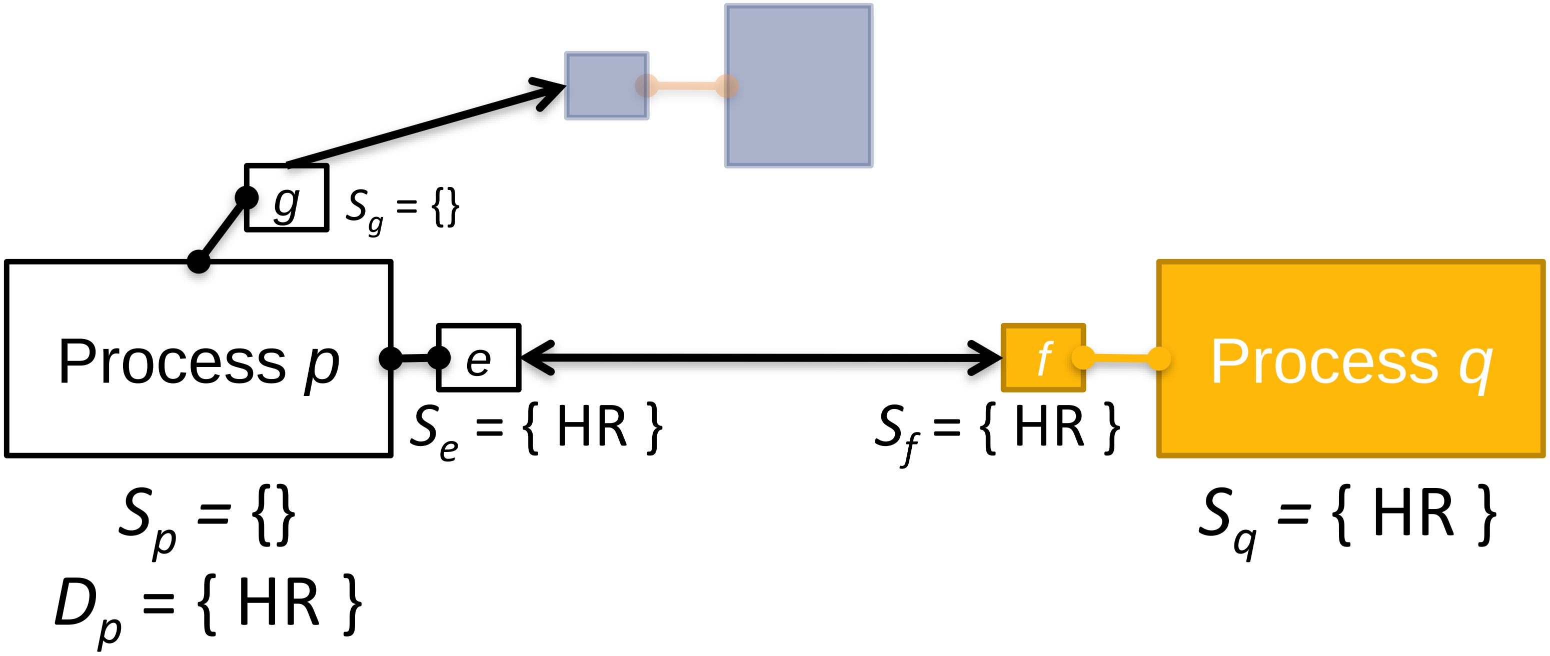
Export inf.

Import inf.



Endpoints Labels Are Independent

Based on slide by Max Krohn



Evaluation

Based on slide by Max Krohn

- Does Flume allow adoption of Unix software?
  - 1,000 LOC launcher/declassifier
  - 1,000 out of 100,000 LOC in MoinMoin changed
  - Python interpreter, Apache, unchanged
- Does Flume solve security vulnerabilities?
  - Without our knowing, MoinMoin wiki case studies inherited two ACL bypass bugs from MoinMoin
  - Both are not exploitable in Flume’s MoinMoin
- Does Flume perform reasonably?
  - Performs within a factor of 2 of the original on read and write benchmarks

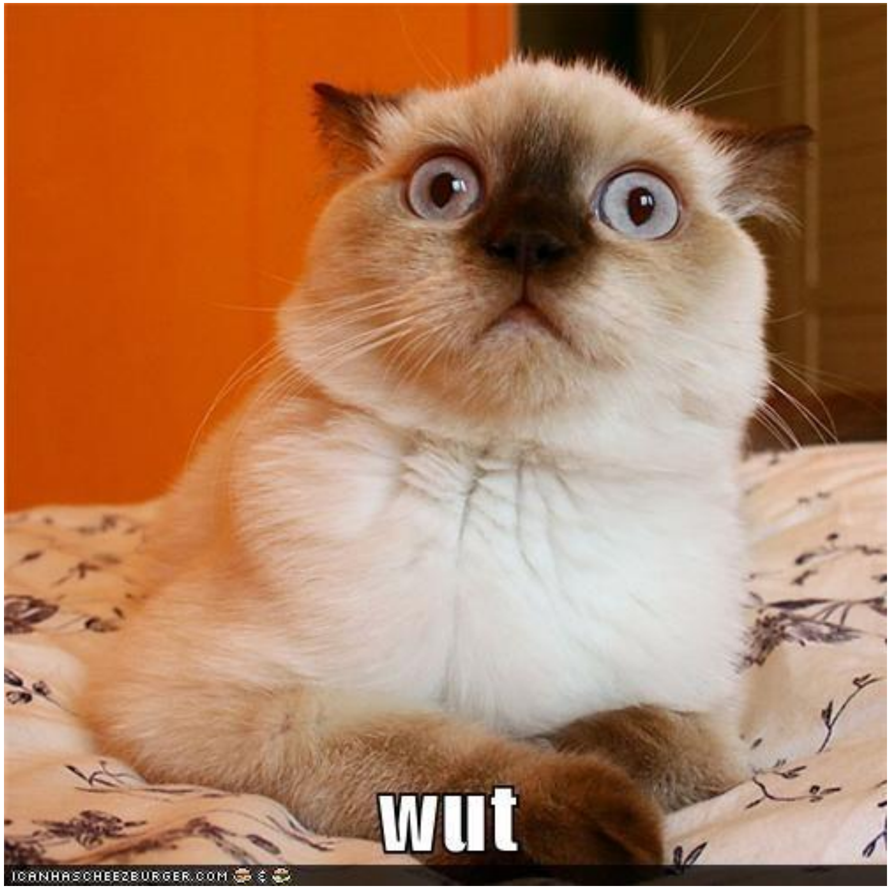
Slide by Max Krohn

Limitations

Based on slide by Max Krohn

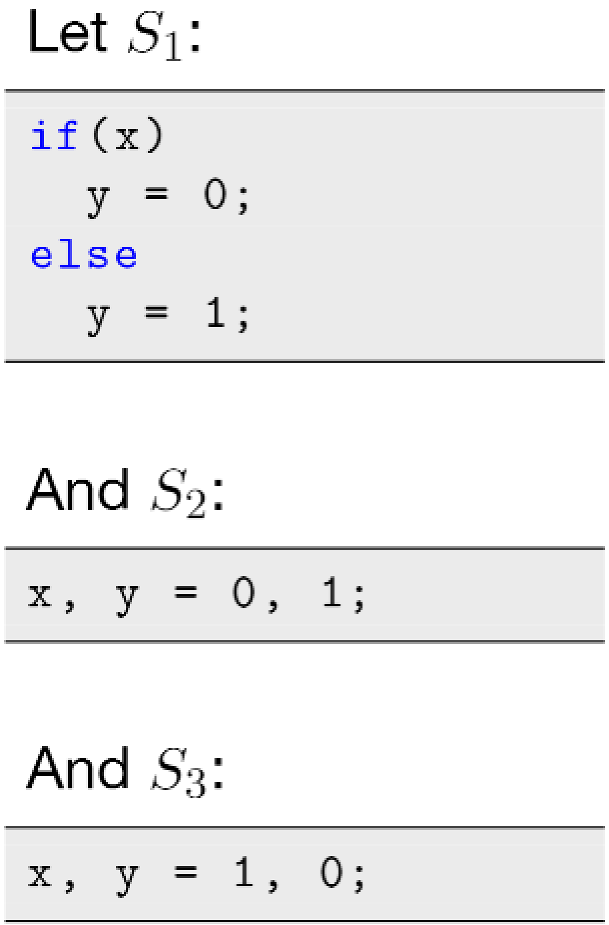
- Bigger TCB than HiStar / Asbestos
  - Linux stack (Kernel + glibc + linker)
  - Reference monitor (~22 kLOC)
- Covert channels via disk quotas
- Confined processes like MoinMoin don't get full POSIX API.
  - `spawn()` instead of `fork()` & `exec()`
  - `flume_pipe()` instead of `pipe()`

Slide by Max Krohn



Part Four: How Do We Have a Reference Monitor for IFC?

Recall: Information Flow Isn’t EM-Enforceable



Let S_1:

```
if(x)
   y = 0;
else
   y = 1;
```

And S_2:

```
x, y = 0, 1;
```

And S_3:

```
x, y = 1, 0;
```

- Recall that we can only distinguish between these three programs if we can choose *two* values for *x*.
- Information flow is a *hyperproperty.*
- Information flow is a *2-safety* property that is finitely refutable over *pairs* of traces.

Desired Guarantee

Non-interference: observable program behavior should not depend on confidential data.

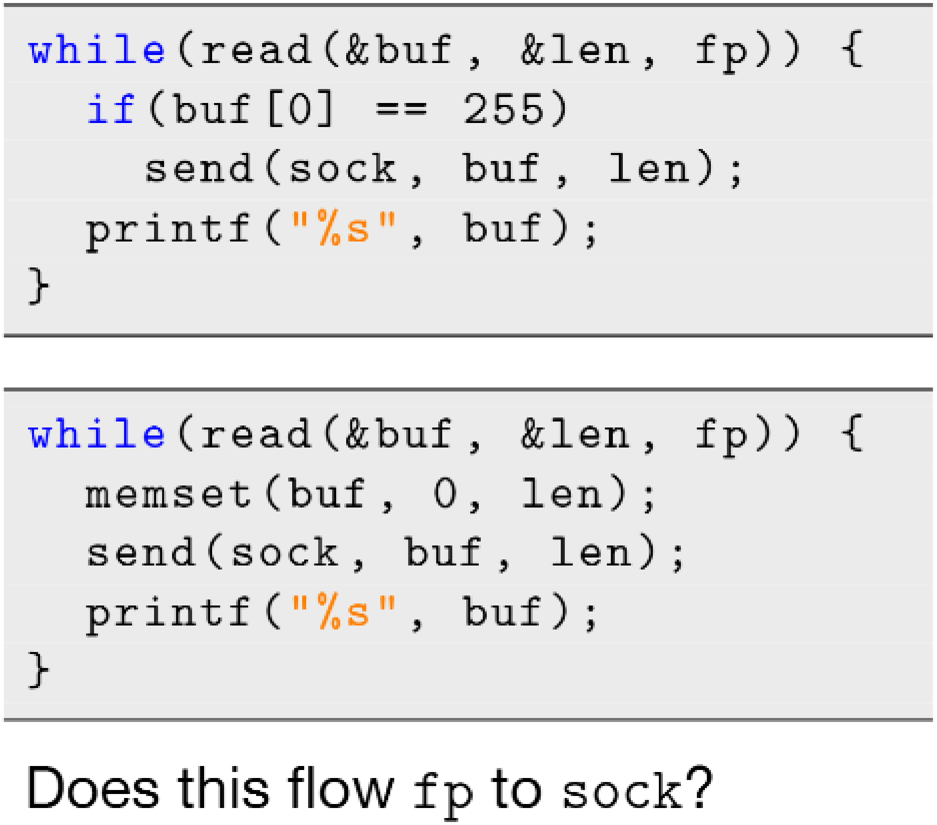
More formally, for a deterministic program P:

$$\forall M_1, M_2: \quad M_1 =_L M_2 \wedge (P, M_1) \rightarrow^* M_1' \wedge (P, M_2) \rightarrow^* M_2' \Rightarrow M_1' =_L M_2'$$

Low-confidentiality projections of initial memory are equivalent.

Low-confidentiality projections of result memory are equivalent.

But Also Recall!



```
while(read(&buf, &len, fp)) {
  if(buf[0] == 255)
    send(sock, buf, len);
  printf("%s", buf);
}
```

```
while(read(&buf, &len, fp)) {
  memset(buf, 0, len);
  send(sock, buf, len);
  printf("%s", buf);
}
```

Does this flow `fp` to `sock`?

- We discussed “no send after read” policy earlier.
- Ideally want to prevent fp from flowing into sock.
- But second program does not flow fp into sock! Check was conservative.

"Platonic" Information Flow is Fine-Grained!

Things that should be allowed under our definition of non-interference:

- Sensitive value gets sent to process, but it doesn't actually use it.
- Sensitive value gets sent to process and it uses it, but doesn't send it out across a specific endpoint.

So What Is Flume Doing?

- Flume **overapproximates** programs that can leak information.
- Flume tracks information flow at the **process level**, rather than at the granularity of individual reads/writes.

Question: What flows does Flume prevent, that may be otherwise allowed?

Part Five: Wrapping Up on Coarse-Grained IFC

Information Flow Involves Tracking Across the Program

Internet joke about how Lindsay Lohan tried to cover up her bracelet.

In a DIFC system, it is not only sensitive values that are tracked, but *any value* whose value depends on a sensitive value. Tracking becomes very fashionable!

Granularity Matters

- Precise information flow is not a safety property.
- We can, however, *overapproximate* information flow using reference monitors.
- There are tradeoffs between precision of tracking and programmer/runtime overhead!

Discussion Questions

- How does access control fall short? (Why do we need information flow?)
- What are the tradeoffs of the label abstraction? The endpoint abstraction?
- What are the tradeoffs of using DFIC systems in general?
- How do the techniques we learned about prevent the leaks we discussed?