GameSS

EDUCATIONAL MATERIAL IN CYBER SECURITY

WHO IS THE MATERIAL FOR?

This material gives you a deep understanding of how information flows through software. This insight helps you avoid introducing information leaks into software and helps you reason about software security in general. It is suitable for professionals such as

- software developers,
- system administrators,
- operations engineers,
- security professionals.

WHO MADE THIS MATERIAL?

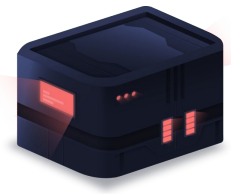
Willard Rafnsson

Associate Professor at the Department of Computer Science at the IT University of Denmark (ITU)

wilr@itu.dk

What are the main takeaways for this content?

- We have an obligation to control the flow of information through software, to protect customer- and company-assets.
- Information flows through software in a myriad of ways.
- A principled understanding of information flow helps us write more secure software.



let us begin… with a motivating example.

Your Spirit Animal

Step 1

Please pick your spirit animal to begin.

- ○ 🐱 Cat
- ○ 🐶 Dog

Next

Your Spirit Animal

Step 1

Please pick your spirit animal to begin.

- (•) 🐱 Cat
- ( ) 🐶 Dog

Next

Your Spirit Animal

Step 1

Please pick your spirit animal to begin.

- ○ 🐱 Cat
- ◉ 🐶 Dog

Next

Your Spirit Animal

Step

Please pick your spirit a

- 🐱 Cat
- 🐶 Dog

Next

who has learned **what** about you?

(you haven't clicked **next** yet)

(consider **remote origins** only; disregard on-device insecurities, and network-level observers)

Your Spirit Animal

Step

Please pick your spirit a

- ○ 🐱 Cat
- ◉ 🐶 Dog

Next

who has learned **what** about you?

(you haven't clicked **next** yet)

(consider **remote origins** only; disregard
on-device insecurities,
and network-level observers)

site's origin knows
date & time of page request,
IP addr, browser fingerprint, …

Your Spirit Animal

Step

Please pick your spirit a

- Cat
- Dog

Next

who has learned **what** about you?

(you haven't clicked **next** yet)

(consider **remote origins** only; disregard on-device insecurities, and network-level observers)

site's origin knows date & time of page request, IP addr, browser fingerprint, …

?⇒ where you are from (possibly who you are)

Your Spirit Animal

Step

Please pick your spirit a

- Cat
- Dog

Next

who has learned **what** about you?

(you haven't clicked **next** yet)

(consider **remote origins** only; disregard on-device insecurities, and network-level observers)

site's origin knows date & time of page request, IP addr, browser fingerprint, …

?⇒ where you are from (possibly who you are)

?⇒ you know Alice (who shared this URL on FB)

Your Spirit Animal

Step

Please pick your spirit a

- ○ 🐱 Cat
- ◉ 🐶 Dog

Next

who has learned **what** about you?

(you haven't clicked **next** yet)

(consider **remote origins** only; disregard on-device insecurities, and network-level observers)

site's origin knows date & time of page request, IP addr, browser fingerprint, …

?⇒ where you are from (possibly who you are)

?⇒ you know Alice (who shared this URL on FB)

?⇒ you are interested in animals (or surveys)

Terminal | Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>

    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
    <link rel="stylesheet" href="style.css">
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Terminal | Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
    <link rel="stylesheet" href="style.css">
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

date & time of page request,
IP addr, browser fingerprint, …

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Terminal Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>

    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
    <link rel="stylesheet" href="style.c
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

date & time of page request,
IP addr, browser fingerprint, …

let's look at this

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Terminal

Terminal

```
$(document).ready(function() {
    console.log("Setting cat behavior.");
    $("#cat").click( function(){
        $("#wrap").css('background-image', "url('https://i.kym-cdn.com/photos/images/original/002/375/173/84c.jpg')");
    });
    console.log("Setting dog behavior.");
    $("#dog").click( function(){
        $("#wrap").css('background-image', "url('https://i.kym-cdn.com/photos/images/original/000/581/296/c09.jpg')");
    });
});
```

-UUU:@----F13 **code.js** All (17,0) (JavaScript) ----------

End of buffer

Terminal | Terminal

```
$(document).ready(function() {
    console.log("Setting cat behavior.");
    $("#cat").click( function(){
        $("#wrap").css('background-image', "url('https://i.kym-cdn.com/photos/images/original/002/375/173/84c.jpg')");
    });
    console.log("Setting dog behavior.");
    $("#dog").click( function(){
        $("#wrap").css('background-image', "url('https://i.kym-cdn.com/photos/images/original/000/581/296/c09.jpg')");
    });
});
```

knowyourmeme.com 's content delivery network

date & time of page request, IP addr, browser fingerprint, …

-UUU:@----F13 **code.js** All (17,0) (JavaScript) ----------

End of buffer

Terminal | Terminal

```
$(document).ready(function() {
    console.log("Setting cat behavior.");
    $("#cat").click( function(){
        $("#wrap").css('background-image', "url('https://i.kym-cdn.com/photos/images/original/002/375/173/84c.jpg')");
    });
    console.log("Setting dog behavior.");
    $("#dog").click( function(){
        $("#wrap").css('background-image', "url('https://i.kym-cdn.com/photos/images/original/000/581/296/c09.jpg')");
    });
});
```

knowyourmeme.com 's content delivery network

date & time of page request, IP addr, browser fingerprint, …

which animal we are interested in

(downloads one image or other, depending on choice)

```
-UUU:@----F13  code.js        All (17,0)     (JavaScript) ----------
End of buffer
```

Terminal | Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>

    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
    <link rel="stylesheet" href="style.css">
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Terminal | Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="https://www.googletagmanager.com/gtag/js?id=G-DNJN1PF3CS"></script>
    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Raleway">
    <link rel="stylesheet" href="style.css">
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

date & time of page request,
IP addr, browser fingerprint, …

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Terminal | Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="https://www.googletagmanager.com/gtag/js?id=G-DNJN1PF3CS"></script>
    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Rale
    <link rel="stylesheet" href="style.css">
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

date & time of page request,
IP addr, browser fingerprint, …

… and all our UI behavior
(scrolling, highlighting, clicking, time spent, …)

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Terminal | Terminal

```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Spirit Animal</title>
    <meta charset="utf-8">
    <script src="https://code.jquery.com/jquery-3.7.1.min.js"></script>
    <script src="https://www.googletagmanager.com/gtag/js?id=G-DNJN1PF3CS"></script>
    <script src="code.js"></script>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Rale
    <link rel="stylesheet" href="style.css">
  </head>
  <body><div id="wrap"><div id="content">
    <h1>Your Spirit Animal</h1>
    <h2>Step 1</h2>
    <p>Please pick your spirit animal to begin.</p>
    <form>
      <input type="radio" id="cat" name="animal" value="Cat"><label for="cat">🐱 Cat</label><br>
      <input type="radio" id="dog" name="animal" value="Dog"><label for="dog">🐶 Dog</label><br><br>
      <input type="button" value="Next">
    </form>
  </div></div></body>
</html>
```

date & time of page request,
IP addr, browser fingerprint, …

… and all our UI behavior
(scrolling, highlighting, clicking,
time spent, …)

it's just animal preference;
who cares?

-UUU:@----F13 **index.html** All (1,0) (HTML+) ----------

Mark set

Your Fitness Plan

Step 1

Please indicate your current fitness to begin.

- ○ 💪 I am in good health
- ○ ❤️ I have a health condition

Next

Your Fitness Plan

Step 1

Please indicate your current fitness to begin.

- (•) 💪 I am in good health
- ( ) ❤️ I have a health condition

Next

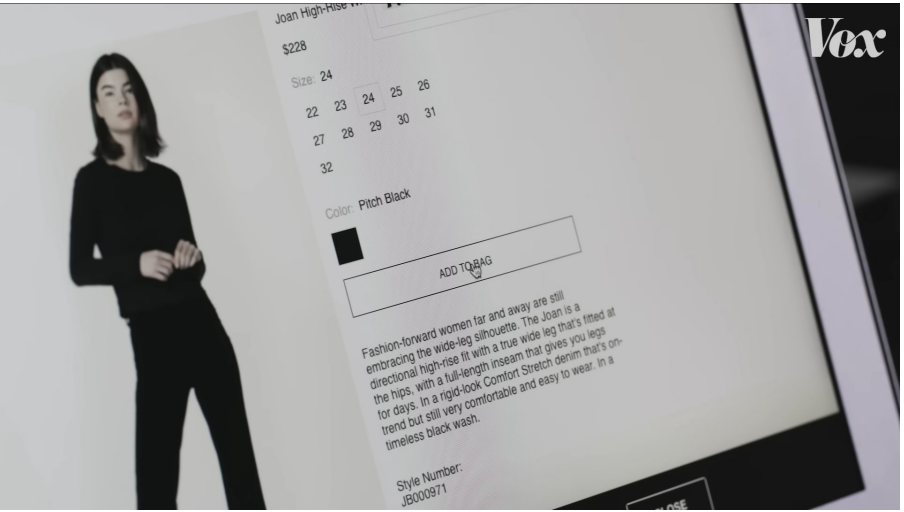
Your Fitness Plan

Step 1

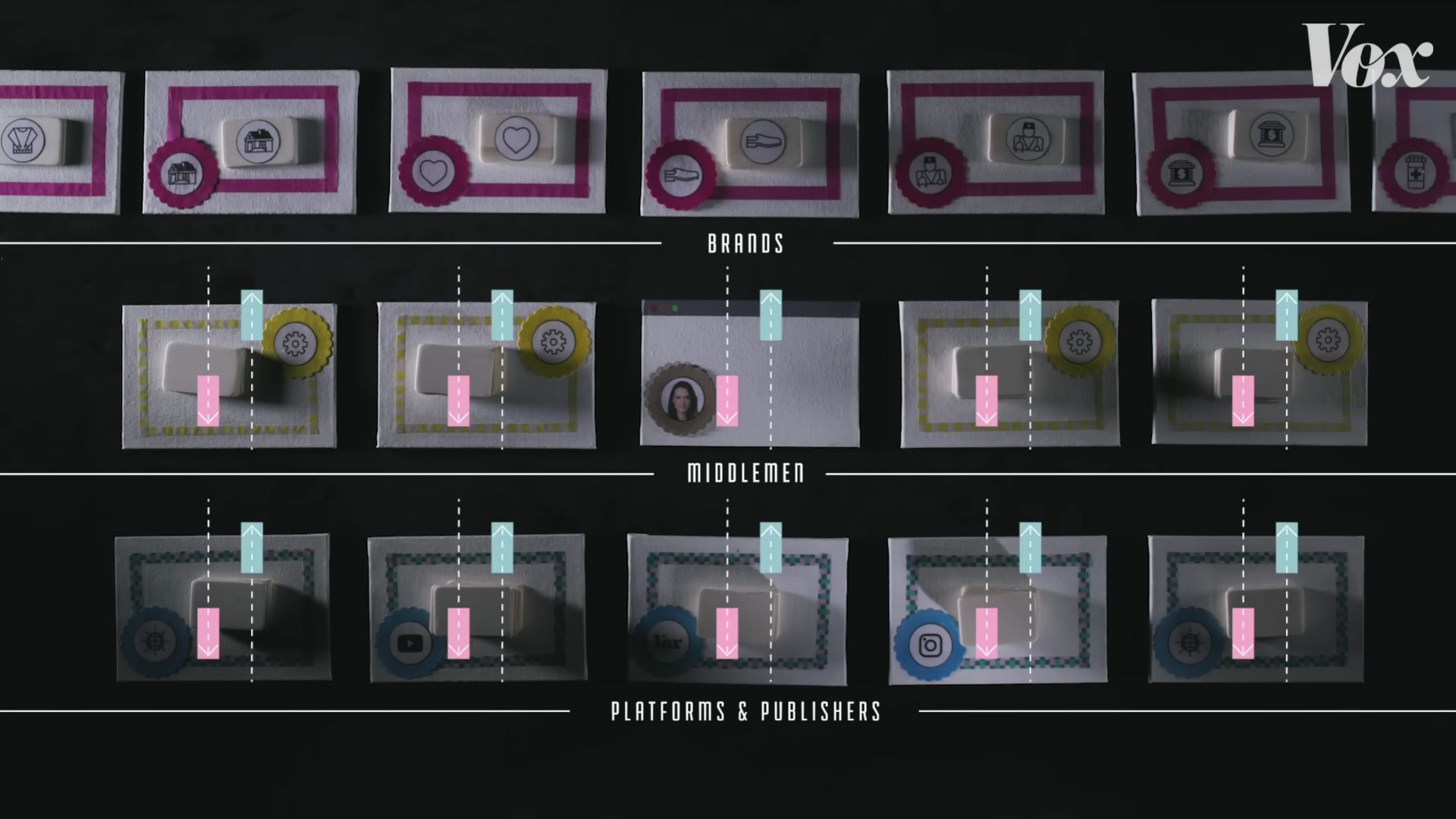
Please indicate your current fitness to begin.

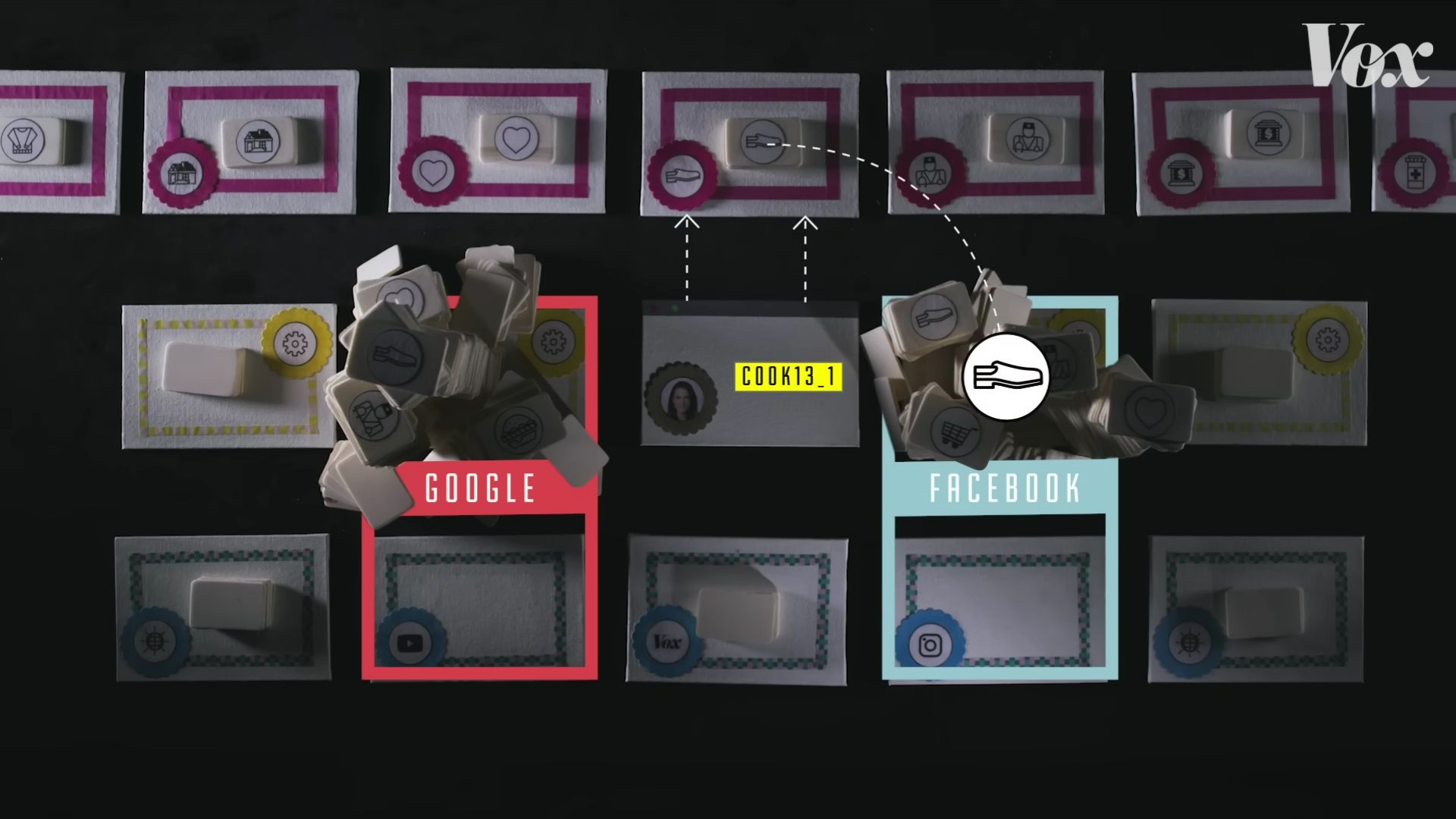
- ○ 💪 I am in good health
- ◉ ❤️ I have a health condition

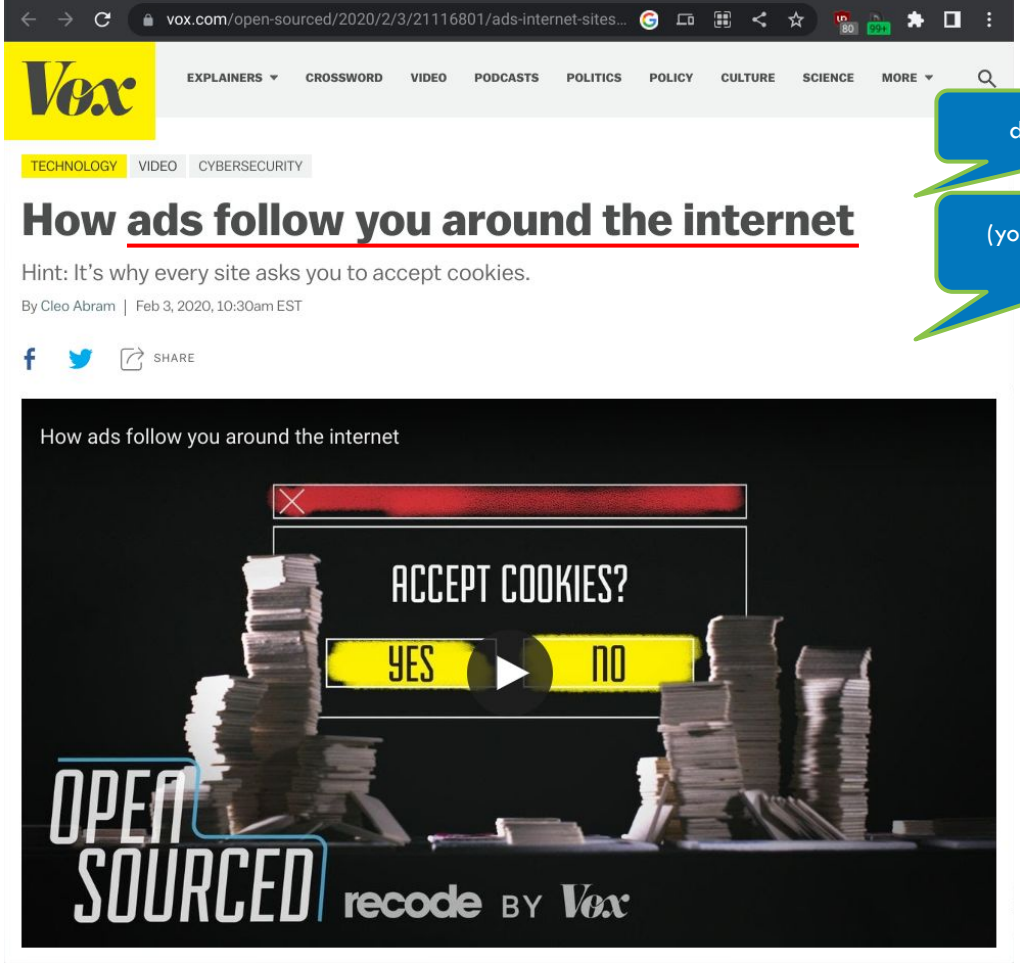
Next


Vex
$228
Size: 24
22 23 24 25 26
27 28 29 30 31
32
Color: Pitch Black
ADD TO BAG
Fashion-forward women far and away are still embracing the wide-leg silhouette. The Joan is a directional high-rise fit with a true wide leg that's fitted at the hips, with a full-length inseam that gives you legs for days. In a rigid-look Comfort Stretch denim that's on-trend but still very comfortable and easy to wear. In a timeless black wash.
Style Number:
JB000971


Vex
nsored
Hide ad
Why am I seeing this ad?


Vox
BRANDS
MIDDLEMEN
PLATFORMS & PUBLISHERS


Vox
COOK13_1
GOOGLE
FACEBOOK



data collection

(you are constantly tracked)

video (6 min) at:
https://www.vox.com/open-sourced/2020/2/3/21116801/ads-internet-sites-cookies

slate.com/technology/2023/04/data-broker-inference-privacy-le...

Incognito (2)

SLATE

News & Politics Culture Technology Business Human Interest Podcasts

ADDRESSING IT HEAD-ON

FOLLOW US

future tense

How Shady Companies Guess Your Religion, Sexual Orientation, and Mental Health

And sell that data to the highest bidder.

BY JUSTIN SHERMAN

APRIL 26, 2023 • 12:55 PM

data brokers

(you are constantly profiled)

propublica.org/article/health-insurers-are-vacuuming-up-detail...

Incognito

PROPUBLICA Graphics & Data Newsletters About

Get the Big Story Join

Racial Justice Health Care Politics Criminal Justice More... Series Video Impact Search

data buyers

(businesses use your data against you)

Justin Volz, special to ProPublica

HEALTH INSURANCE HUSTLE

Health Insurers Are Vacuuming Up Details About You — And It Could Raise Your Rates

as in **check, regulate**

you must *control the flow of* customer *information.*

What is the GDPR?

The General Data Protection Regulation is a European Union law that was implemented May 25, 2018, and requires organizations to safeguard personal data and uphold the privacy rights of anyone in EU territory. The regulation includes seven principles of data protection that must be implemented and eight privacy rights that must be facilitated. It also empowers member state-level data protection authorities to enforce the GDPR with sanctions and fines. The GDPR replaced the 1995 Data Protection Directive, which created a country-by-country patchwork of data protection laws. The GDPR, passed in European Parliament by overwhelming majority, unifies the EU under a single data protection regime.

Read our summary of the GDPR for an overview of the law.

GDPR's goals are to enhance individuals' control and rights over their personal information

you must *control the flow of* company *assets.*

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

this category is discouraged today for being too broad. but see children here:

Weakness ID: 200
Vulnerability Mapping: DISCOURAGED
Abstraction: Class

View customized information: Conceptual | Operational | Mapping Friendly | **Complete** | Custom

Description

Extended Description

Alternate Terms

Relationships

Relevant to the view "Research Concepts" (CWE-1000)

Nature	Type	ID	Name
ChildOf	C	668	Exposure of Resource to Wrong Sphere
ParentOf	B	201	Insertion of Sensitive Information Into Sent Data
ParentOf	B	203	Observable Discrepancy
ParentOf	B	209	Generation of Error Message Containing Sensitive Information
ParentOf	B	213	Exposure of Sensitive Information Due to Incompatible Policies
ParentOf	B	215	Insertion of Sensitive Information Into Debugging Code
ParentOf	B	359	Exposure of Private Personal Information to an Unauthorized Actor
ParentOf	B	497	Exposure of Sensitive System Information to an Unauthorized Control Sphere
ParentOf	B	538	Insertion of Sensitive Information into Externally-Accessible File or Directory
ParentOf	B	1258	Exposure of Sensitive System Information Due to Uncleared Debug Information
ParentOf	B	1273	Device Unlock Credential Sharing
ParentOf	B	1295	Debug Messages Revealing Unnecessary Information
CanFollow	V	498	Cloneable Class Containing Sensitive Information
CanFollow	V	499	Serializable Class Containing Sensitive Data
CanFollow	B	1272	Sensitive Information Uncleared Before Debug/Power State Transition

Identifying JavaScript Code and Gathering JavaScript Files

Programmers often hardcode sensitive information with JavaScript variables on the frontend. Testers should check HTML source code and look for JavaScript code between `<script>` and `</script>` tags. Testers should also identify external JavaScript files to review the code (JavaScript files have the file extension `.js` and name of the JavaScript file usually put in the `src` (source) attribute of a `<script>` tag).

Check JavaScript code for any sensitive information leaks which could be used by attackers to further abuse or manipulate the system. Look for values such as: API keys, internal IP addresses, sensitive routes, or credentials. For example:

```
const myS3Credentials = {
  accessKeyId: config('AWSS3AccessKeyID'),
  secretAccessKey: config('AWSS3SecretAccessKey'),
};
```

The tester may even find something like this:

```
var conString = "tcp://postgres:1234@localhost/postgres";
```

When an API Key is found, testers can check if the API Key restrictions are set per service or by IP, HTTP referrer, application, SDK, etc.

For example, if testers find a Google Map API Key, they can check if this API Key is restricted by IP or restricted only per the Google Map APIs. If the Google API Key is restricted only per the Google Map APIs, attackers can still use that API Key to query unrestricted Google Map APIs and the application owner must pay for that.

```
<script type="application/json">
...
{"GOOGLE_MAP_API_KEY":"AIzaSyDUEBnKgwiqMNpDplT6ozE4Z0XxuAbqDi4",
"RECAPTCHA_KEY":"6LcPscEUiAAAAHOwwM3fGvIx9rsPYUq62uRhGjJ0"}
...
</script>
```


OWASP testing guide has examples of information leaks

WHO IS SECURE INFORMATION FLOW FOR?

software developers - you already put thought into this when writing software.

a principled understanding of secure information-flow equips you to do information-flow control systematically, which helps you write more secure software.
this grows the kit with which you can solve programming challenges.

in this module:

- information flow,
- secure information flow,
- writing software with secure information flow.

conceptually
part 1

in practice
part 2

information flow security complements existing security mechanisms.

here's how.


firewall?
(& other isolation/hardening approaches)


firewall?
(& other isolation/hardening approaches)
reduces attack surface, but
does not secure running services
(listening on open ports).

antivirus?

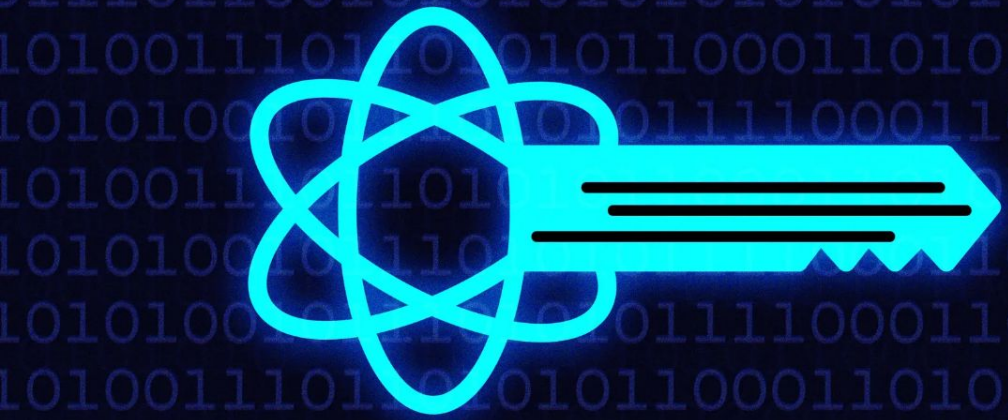
antivirus?

only known vulnerabilities.
(does not enforce
confidentiality or integrity)


network monitoring,
intrusion detection, ...
ALERT
Intrusion Detected
0 1 0 1 1 0 1 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1
1 0 1 0 0 1 0 1 0 1 1 1 1 0 0 1 0 1 0 1 1 1

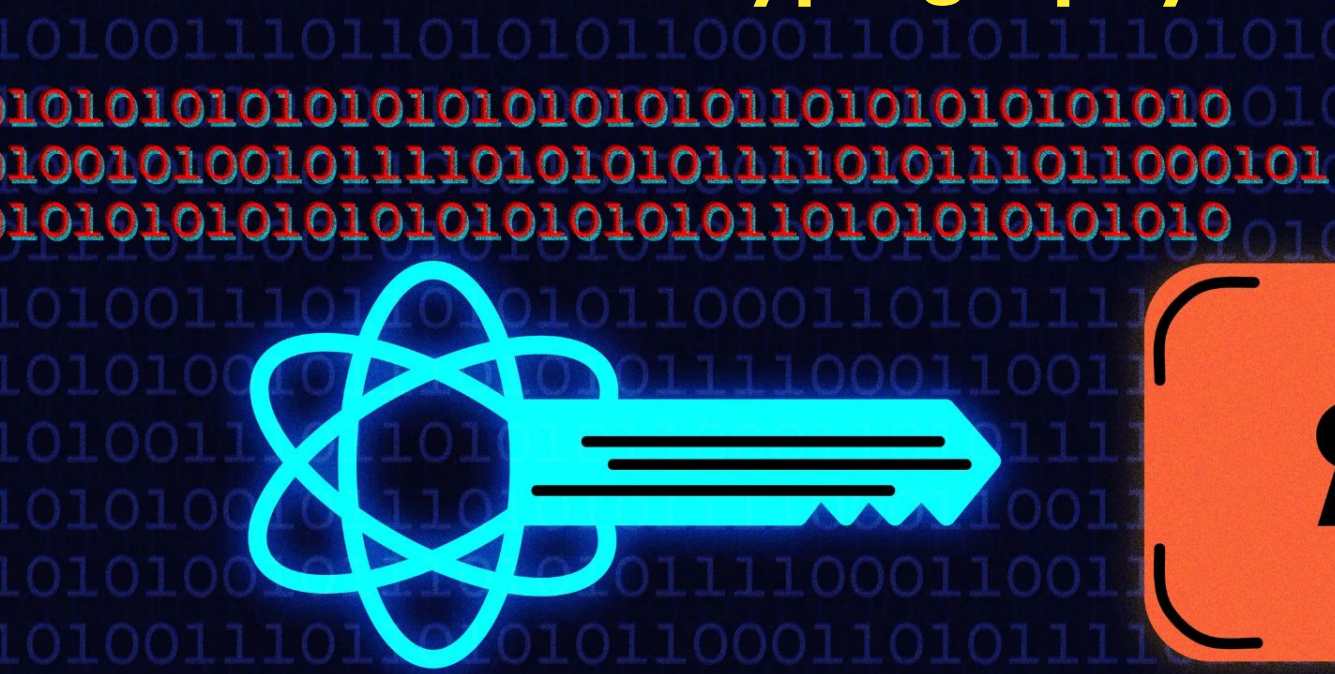

network monitoring,
intrusion detection, ...
attacks patterns can imitate
benign behavior
ALERT
Intrusion Detected

cryptography?



cryptography?

does not secure
attacks through encrypted connection,
traffic analysis, etc.




access control?

access control?

does not secure

- what service does w/ data it has accessed,
- data accessed outside access requests

information flow

Recall:

who has learned **what** about you?

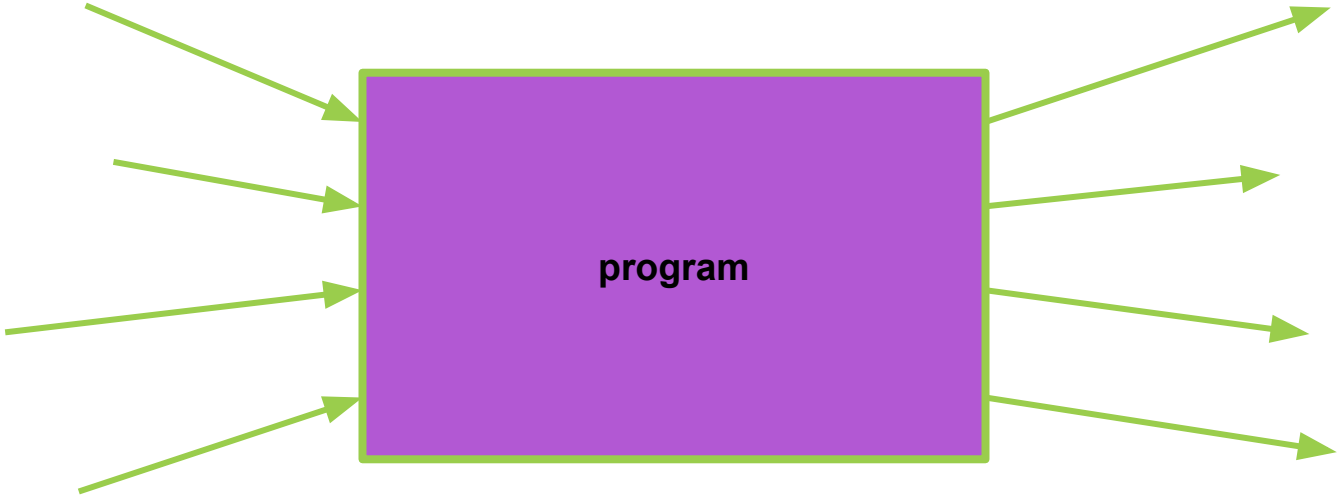
site's origin knows
date & time of page request,
IP addr, browser fingerprint, ...

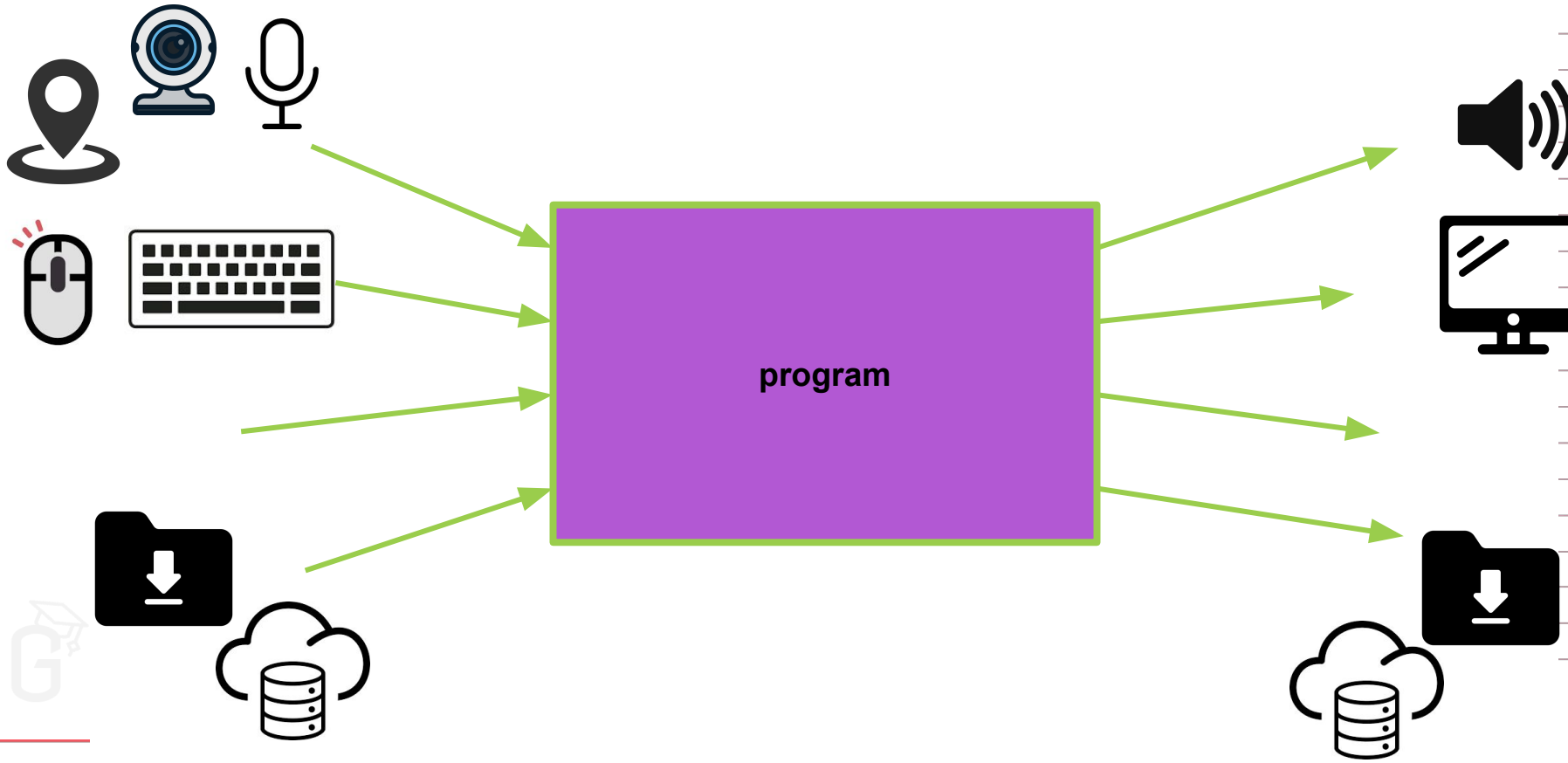
which animal we are
interested in

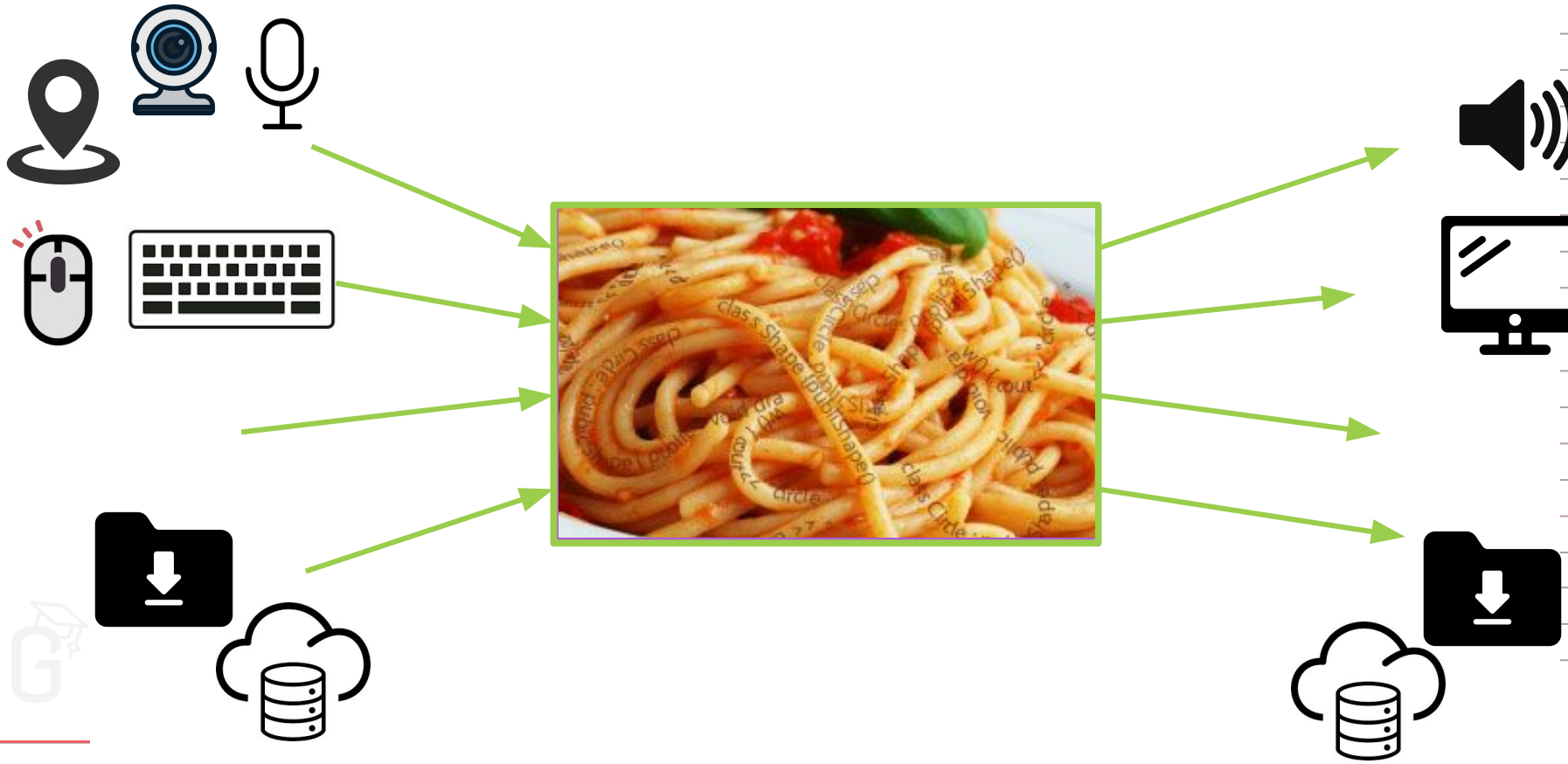
... and all our UI behavior
(scrolling, highlighting,
clicking, time spent, ...)

How does that happen?


program


program


program



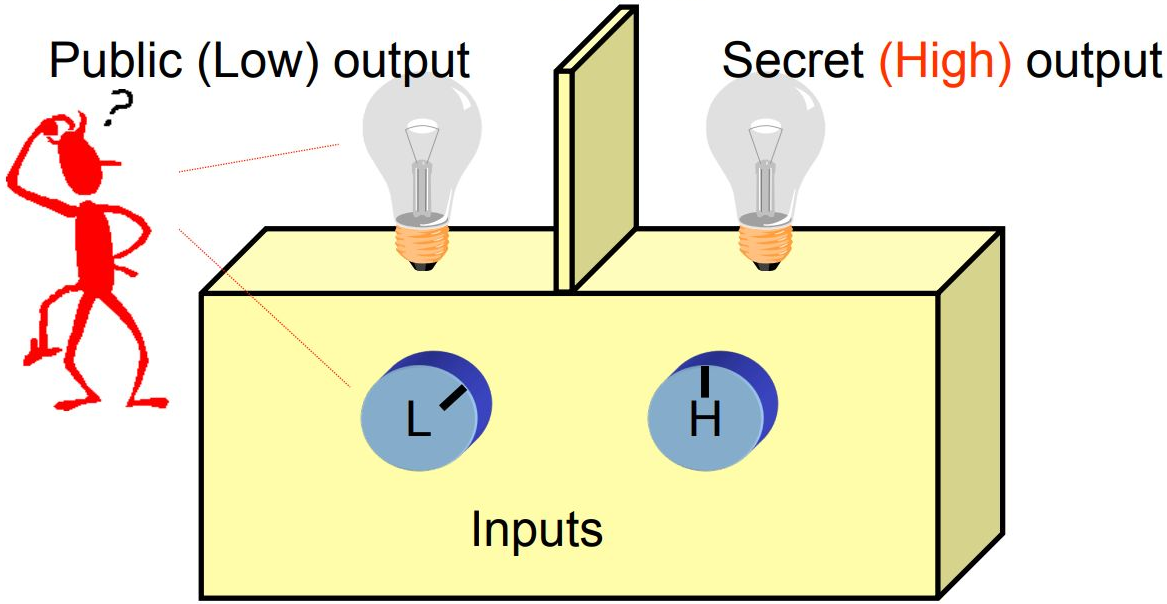
privacy concerns

only local attackers

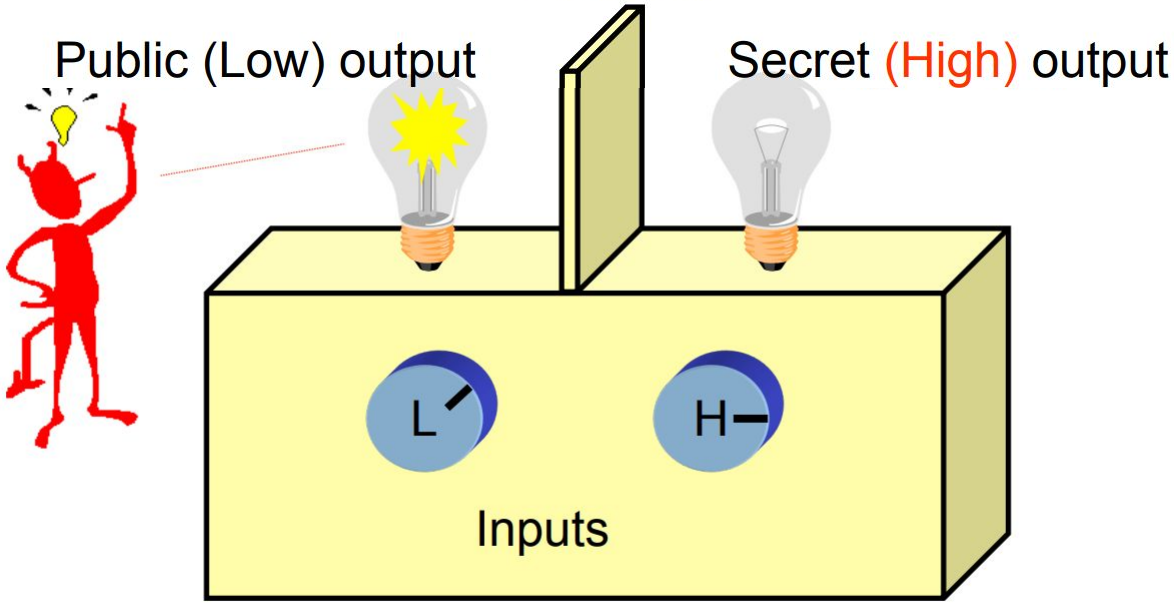
potentially *very* sensitive

remote attackers

When does information flow?



When does information flow?



information flows from secret input to public output

note that secret does not need to be "copied" to public for information to flow.

rather, information flows when *any* information about secret input manifests in public output

Information Flow Terminology

information source where information comes from (source of input)

information sink where information ends (destination of output)

endpoints on the spaghetti slide.

Information Flow Terminology

information source where information comes from (source of input)

information sink where information ends (destination of output)

information flow information transferred from one point to another

information leak unintended information flow

endpoints on the spaghetti slide.

examples at the beginning

Information Flow Terminology

information source where information comes from (source of input)

information sink where information ends (destination of output)

information flow information transferred from one point to another

information leak unintended information flow

side-channel information flows as a side-effect of how medium is used (unintended) circumvents logical protection measures in computer systems

covert channel side-channel intentionally utilized to leak information leak circumvents protection measures

endpoints on the spaghetti slide.

examples at the beginning

we'll see examples momentarily

side channels

- explicit & implicit flows
- termination & progress flows
- timing flows

Explicit Flows

explicit: `public := secret + 4;`

Explicit & implicit flows

explicit:

```
public := secret + 4;
```

implicit:

```
if (secret mod 2 == 1) {
    public := 1;
} else {
    public := 0;
}
```

Termination flows

termination: public := 0;

(& implicit) **while** (secret **mod** 2 == 1){public := 1};

leak: “whether or not the program *terminates*”.

1 bit leaked.

Termination flows

termination: **while** (secret **mod** 2 == 1) **{};**
public := 0;

no implicit flow

common assumption in program analysis: "all loops terminate"

leak: "whether or not the program *terminates*".

1 bit leaked.

Progress flows

progress: **while** (secret **mod** 2 == 1) **{};**

(& termination) **output** public 0;

leak: "whether or not the program *makes progress* on its output."

1 bit leaked.

Progress flows

progress:
(& implicit)

```
k := 0;
while (k <= secret) {
    output public k;
    k := k+1;
};
```


all of secret leaked.

Progress flows

progress:

```
k := 0;
while (k <= MAX) {
    output public k;
    if (secret<=k) {while True {};}
    k := k+1;
};
```

no implicit flow

all of secret leaked.

Timing flows

timing:

```
sleep secret;
output public 1;
```

no explicit flow
no implicit flow
no termination flow
no progress flow

all of secret leaked.

time the output

Information Flow, Summary

we saw various kinds of information flows (some over side-channels):

- explicit & implicit flows
- termination & progress flows
- timing flows

(there are other kinds of flows, out of scope of this module; see very end of slides).

with this, you can already now **look for information leaks in your programs**
(by means of manual inspection)

next: how do we write programs that exhibit secure information flow?
how do we write tools to check this for us?

secure information flow

begin with *domain modeling*; who is permitted to observe what

widely adopted in the corporate sector

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

levels constrain access: ensures *information does not flow to those not cleared for that level*.

enforces confidentiality and integrity. how:

i.e. that data is protected from unauthorized disclosure

i.e. that data is protected from unauthorized modification

"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

levels constrain access: ensures *information does not flow to those not cleared for that level*.

enforces confidentiality and integrity. how:

levels form a lattice.
lattice dictates which flows are permitted:

- "no read up"  (only read from below)
- "no write down" (only write upwards) 

"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

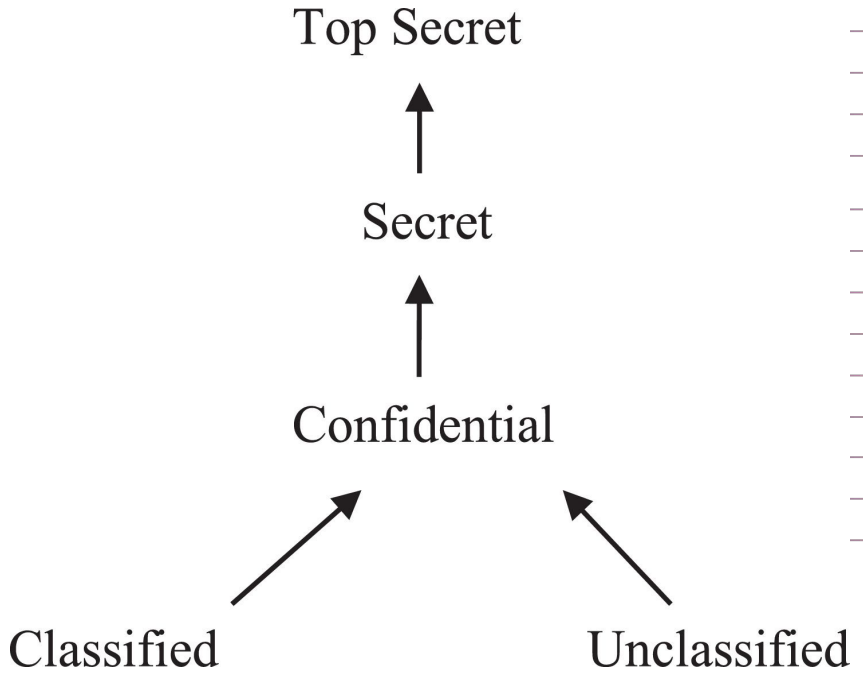
levels constrain access: ensures *information does not flow to those not cleared for that level*.

enforces confidentiality and integrity. how:

levels form a lattice.
lattice dictates which flows are permitted:

- "no read up" (only read from below)
- "no write down" (only write upwards)

Lattice (examples; you make your own)



"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

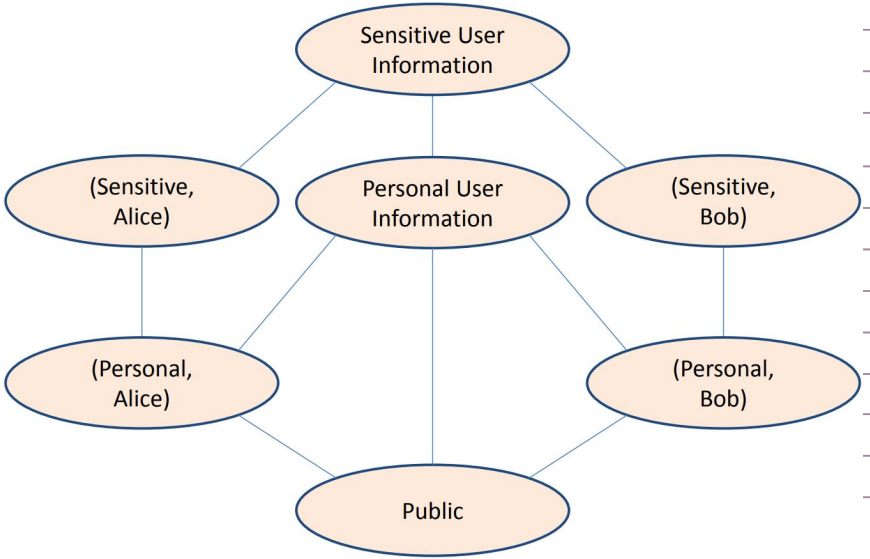
levels constrain access: ensures *information does not flow to those not cleared for that level*.

enforces confidentiality and integrity. how:

levels form a lattice.
lattice dictates which flows are permitted:

- "no read up" (only read from below)
- "no write down" (only write upwards)

Lattice (examples; you make your own)



"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

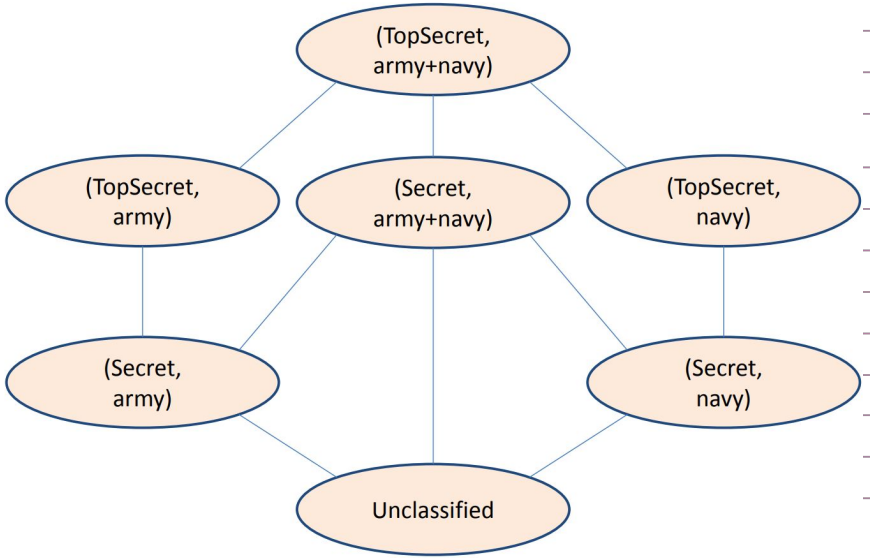
levels constrain access: ensures *information does not flow to those not cleared for that level*.

enforces confidentiality and integrity. how:

levels form a lattice.
lattice dictates which flows are permitted:

- "no read up" (only read from below)
- "no write down" (only write upwards)

Lattice (examples; you make your own)



"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

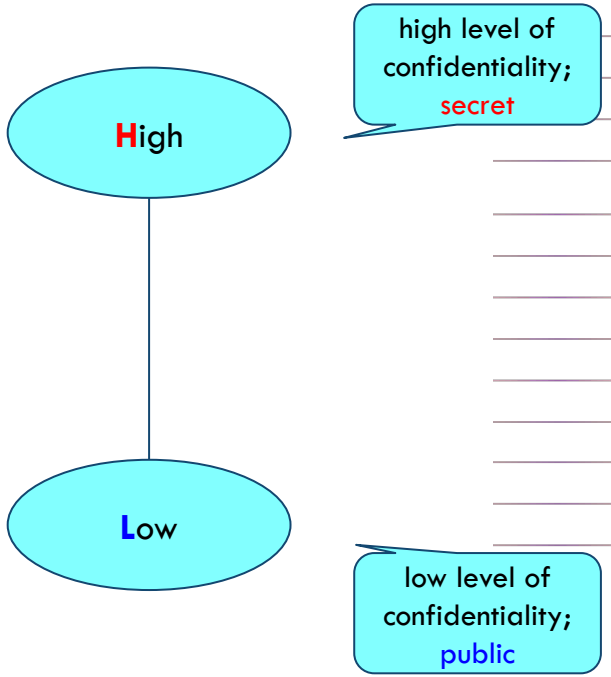
levels constrain access: ensures *information does not flow to those not cleared for that level*.

enforces confidentiality and integrity. how:

levels form a lattice.
lattice dictates which flows are permitted:

- "no read up" (only read from below)
- "no write down" (only write upwards)

Lattice (examples; you make your own)



"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Policy Specification (Multi-Level Security, aka. MLS)

Bell-LaPadula (+ Biba)

label sources and sinks with security levels.

levels constrain access: ensures *information does not flow to those not cleared for that level*.

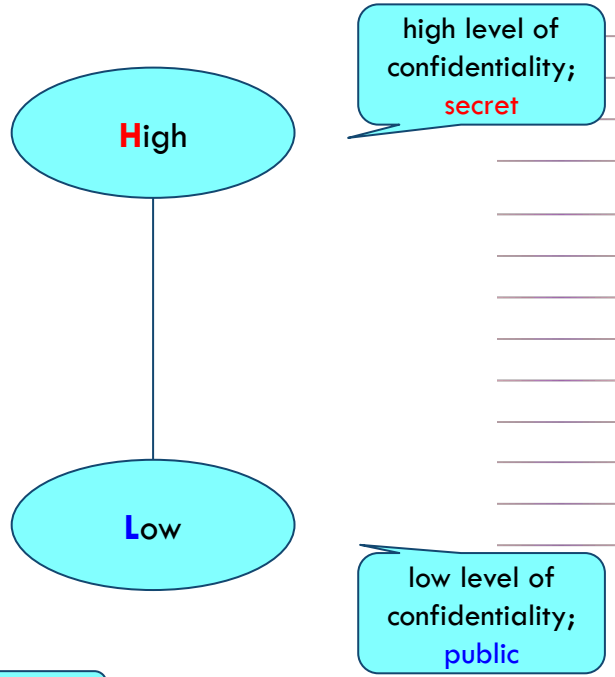
enforces confidentiality and integrity. how:

levels form a lattice.
lattice dictates which flows are permitted:

- "no read up" (only read from below)
- "no write down" (only write upwards)

"Secure Computer Systems", by DE Bell, LJ LaPadula, **1974**

Lattice (examples; you make your own)

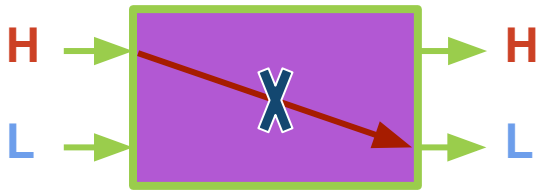


for simplicity, we will use this lattice in explanations and examples

Secure Information Flow

A program that has secure information flow has **no information leaks.**

This property is known as *noninterference*.



noninterference:
"H inputs do not interfere with L outputs."

Information-Flow Control

checking whether a program has secure information flow, is information-flow control.

prove absence of *undesired flows* of information in programs.

- [app] does not leak
  [credit card nr] to [ad provider]

H L

confidentiality & integrity (duality)

program analysis & transformation

next slide

sidenote: noninterference not a property of individual executions; rather, noninterference is a property of all pairs of executions (at least) (a so-called hyperproperty). consequence: we cannot use testing to gain any significant assurance that a program is noninterfering.

Analysis

Transformation

Detect

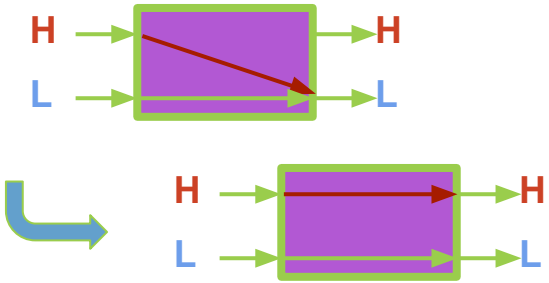
explicit: `avg := (height_a + height_b)/2;`

implicit:
```
if (cpr_nr_a mod 2 == 1) {
    is_male := 1;
} else {
    is_male := 0;
}
```

detected “statically”, at compile time

Repair

rewritten program: rejects leaks on **run-time.**



ways to do information-flow control

Analysis: Denning's Certification Method

"check":

- whilst within a branch on "e_B" ("`if`", "`while`"):

  raise pc by $lev(e_B)$

- when "$x := e_A$" is encountered:

  $lev(e_A) \sqsubseteq lev(x)$ ←explicit flow

  $lev(pc) \sqsubseteq lev(x)$ ←implicit flow

"*Certification of programs for secure information flow.*", by Dorothy E. Denning & Peter J. Denning, **1977**

check

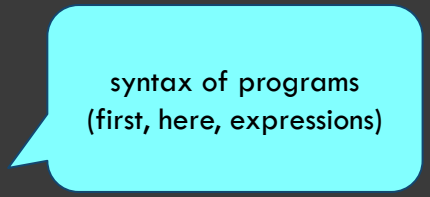
sample implementation of compile-time information-flow control

```haskell
-- variables
type Var = String

{- try it:
x_cpr_nr_alice = "cpr_nr_alice" :: Var
-}

-- expressions
data Exp
  = Val Val
  | Var Var
  | App BOp Exp Exp
  deriving (Show)

{- try it:
e_alice_is_female = App "%" (Var "cpr_nr_alice") (Val 2)
-}
```



```haskell
-- programs
data Prg
  = Skip
  | Assign Var Exp
  | Seq Prg Prg
  | If Exp Prg Prg
  | While Exp Prg
  deriving (Show)

{- try it:
p_assignments =
  Seq (
    Assign "x" (Val 4)
  ) (
    Assign "y" (Val 5)
  )
-}
```

```
check :: Lev -> Lab -> Prg -> Bool
check _ _ Skip = True
check pc g (Assign x e) = (levofexp g e) ⊑ (g!) x ∧ pc ⊑ (g!) x
check pc g (Seq p1 p2)  = check pc g p1 ∧ check pc g p2
check pc g (If e p1 p2) = check (pc ⊔ levofexp g e) g p1
                        ∧ check (pc ⊔ levofexp g e) g p2
check pc g (While e p)  = check (pc ⊔ levofexp g e) g p
```

(g!) x means look up the security level of x in g

pc is a security level

wrt. explicit & implicit flows (does not address termination, progress, timing)

theorem (soundness): given a g, it holds that, for any program p, check L g p ⇒ p is noninterfering

Transformation: Fenton's Data mark Machine

"check":

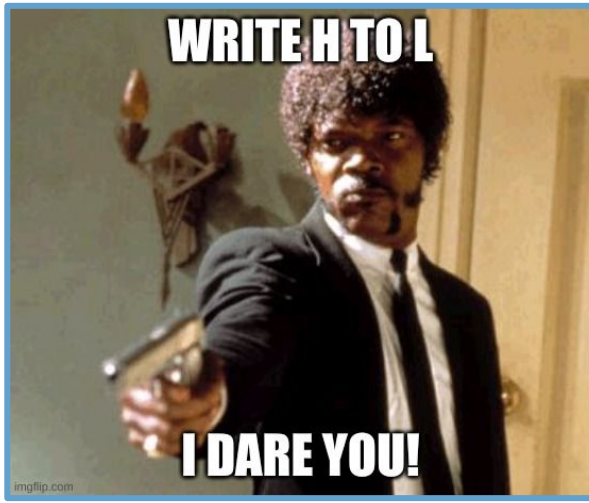
- whilst within a branch on "e_B" ("`if`", "`while`"):

  raise pc by $lev(e_B)$

- when "$x := e_A$" is encountered:

  $lev(e_A) \sqsubseteq lev(x)$ ⟵explicit flow

  $lev(pc) \sqsubseteq lev(x)$ ⟵implicit flow



"`monitor`": will do exactly the same thing, just on run-time (change "`eval`").

"Memoryless Subsystems", by Jeffrey S. Fenton, **1974**

monitor

sample implementation of run-time information-flow control

```
eval :: Prg -> Mem -> Mem
eval Skip           m = m
eval (Assign x e)   m = (m?) x v
  where v  = evalexp e m
eval (Seq p q) m = eval q m'
  where m' = eval p m
eval (If e p q)     m =
  if v == 0 then eval p m else eval q m
  where v  = evalexp e m
eval (While e p)    m =
  if v == 0
  then
```

memory is a function from variables to values

semantics of a program: function that maps an initial memory, to a final memory

```
monitor :: Lev -> Lab -> Prg -> Mem -> Mem
monitor _  _ Skip           m = m
monitor pc g (Assign x e)   m =
  if
    not $ (levofexp g e) ⊑ (g!) x
  then
    error $ "assigning " ++ (show e) ++ " to " ++ x
  else if
    not $ pc ⊑ (g!) x
  then
    error $ "assigning to " ++ x ++ " under pc=" ++ (show pc)
  else
    (m?) x v
  where v  = evalexp e m
monitor pc g (Seq p q) m = monitor pc g q m'
  where m' = monitor pc g p m
```

"if" and "while" on next slide

like previous semantics, but bookkeeps current pc, takes labeling of variables as argument, and generates error on infomration flow violation

```
monitor pc g (If e p q)     m =
  if v == 0
  then
    monitor (pc ⊔ l) g p m
  else
    monitor (pc ⊔ l) g q m
  where v = evalexp e m
        l = levofexp g e
monitor pc g (While e p)    m =
  if v == 0
  then
    m
  else
    monitor pc g (While e p) m'
  where v  = evalexp e m
        l  = levofexp g e
        m' = monitor (pc ⊔ l) g p m
```

wrt. explicit & implicit flows (does not address termination, progress, timing)

theorem (soundness): given a g, it holds that, for any program p, (monitor L g p) is noninterfering

Secure Information Flow, Summary

a program with secure information flow, is one that has no information leaks.
I.e. that is **noninterfering**.

to do information-flow control, you first model domain knowledge,
as a **lattice of of security levels**.

two kinds of tools that do information flow control:

- check: compile-time
- monitor: run-time, or source-to-source transformation

for code you write

for 3rd party code

you now know the inner workings of such tools. Can write your own.

next: example of such a tool.

writing software with secure information flow

explicit flow:

```
x := a;
```

secure?

depends on *security levels*

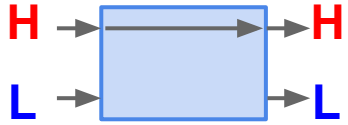
the variables are labeled with.

explicit flow:

x := a;

explicit flow:

x := a;



if

a : H

x : H

then

|- p.

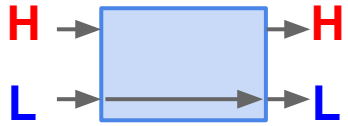
a “math” way to write

check L g p

levels of confidentiality

explicit flow:

x := a;



if

a : L

x : L

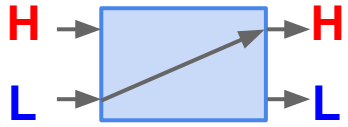
then

|- p.

levels of confidentiality

explicit flow:

x := a;



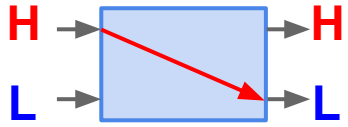
if
a : L
x : H
then
|- p

taking something public and classifying it

levels of confidentiality

explicit flow:

x := a;



if

a : H

x : L

then

p leaks information!

taking something secret
and publishing it

levels of confidentiality

levels of confidentiality

explicit flow:

```
x := a;
```

implicit flow:

```
if a {
  x := 1;
} else {
  x := 0;
}
```

explicit flow:

```
x := a;
```

levels of confidentiality

implicit flow:

```
if a {
  x := 1;
} else {
  x := 0;
}
```

if

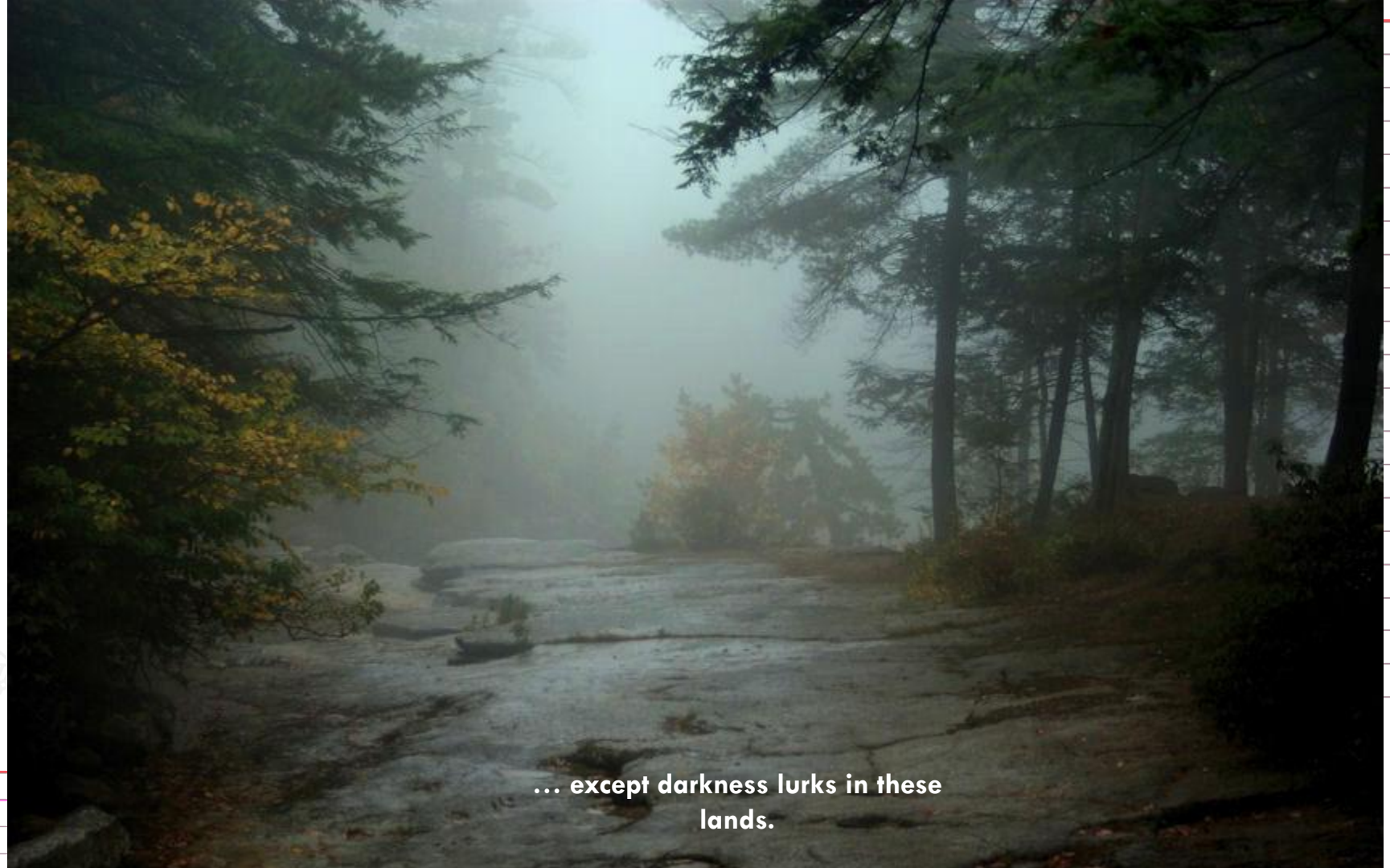
a : H

x : L

then

p leaks information!

and we have tools with which to
statically or dynamically ensure security of programs.
all is well...


… except darkness lurks in these
lands.

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

age range for dating

https://xkcd.com/314/

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

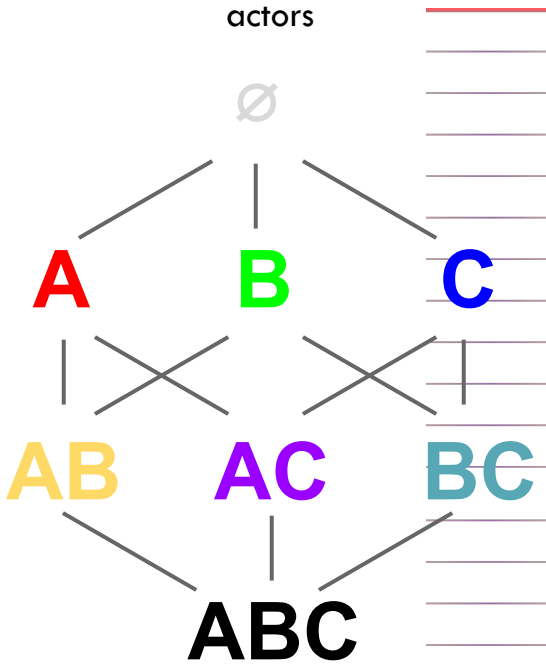
actors
(principals, users)

A B C

x intended for **A**,
y intended for **C**.
how to proceed?

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

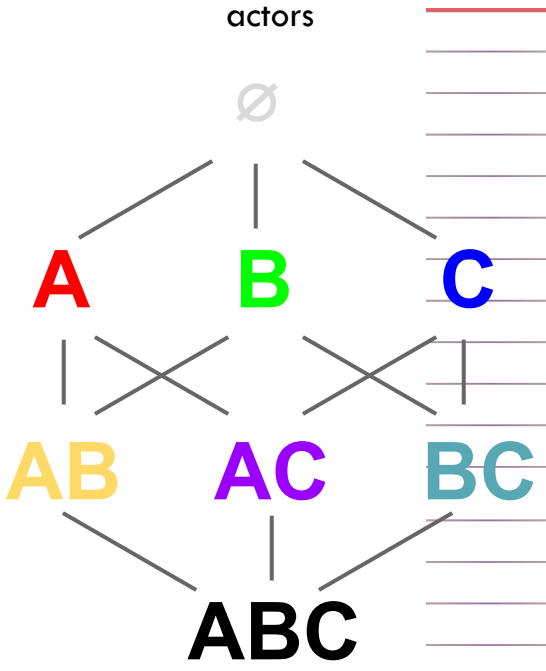


x intended for **A**,
y intended for **C**.
"push down" level of x.

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

spot the leak?

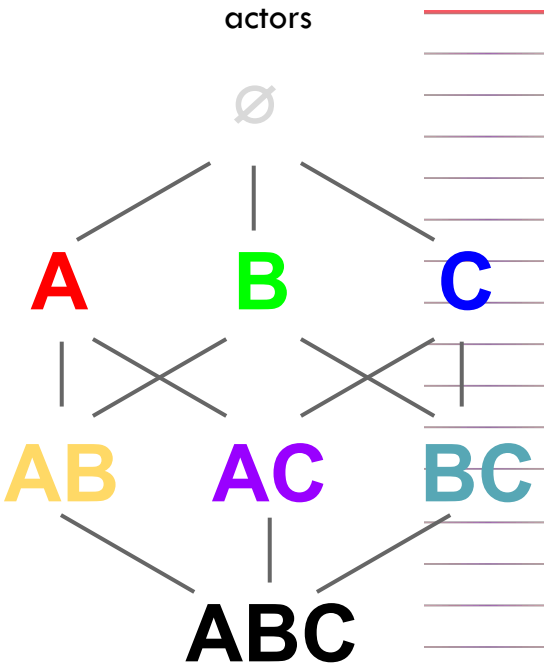


x intended for **A**,
y intended for **C**.

Slightly Larger Example

leak!

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

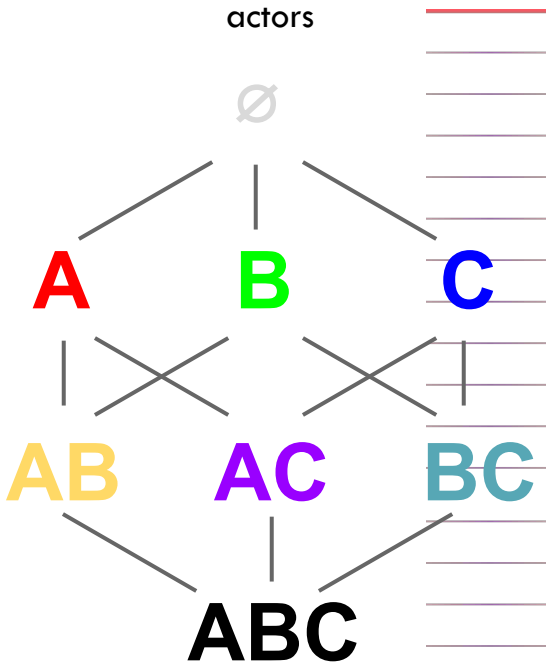


x intended for **A**,
y intended for **C**.

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

OK

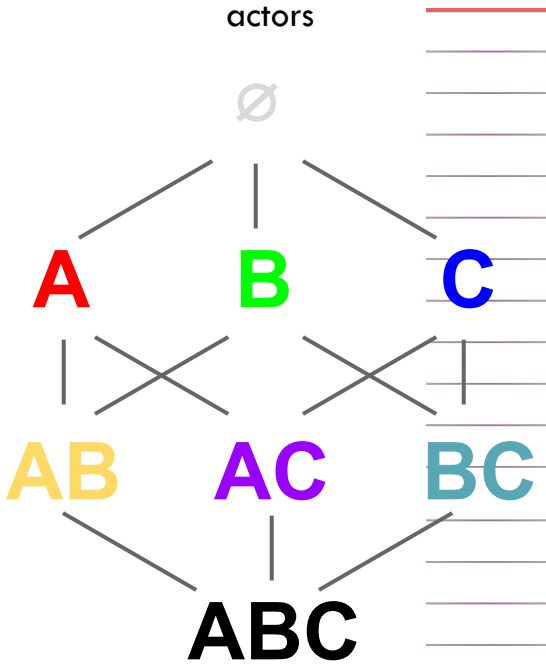


x intended for **A**,
y intended for **C**.
"push down" level of y.

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

spot the leak?

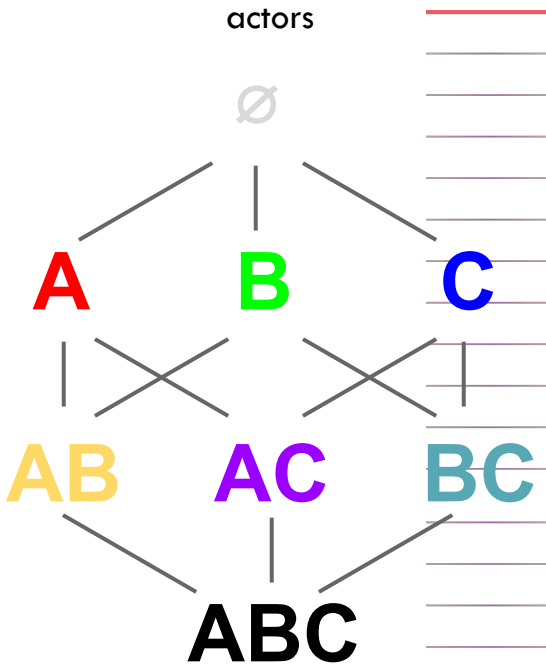


x intended for **A**,
y intended for **C**.

Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

leak!

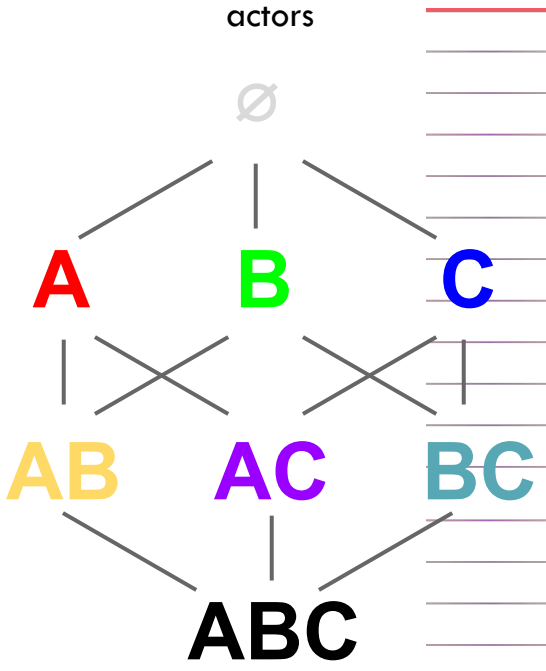


x intended for **A**,
y intended for **C**.

Slightly Larger Example

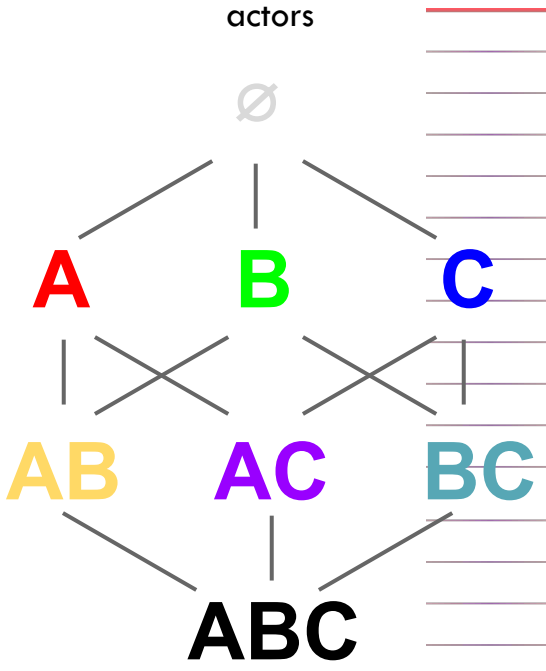
```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```

OK



Slightly Larger Example

```
if |a-b| <= a/2 + 7 {
  x := 1;
} else {
  x := 0;
}
y := c >= a & c >= b;
```



every program **can** be securely labeled.
but we **may not like** those labelings.
we only want to reveal whether certain birthdays are within certain intervals...

but a program, `checked` or `monitored` with this labeling, can leak **all** of b, **anywhere**!!!
(the labeling alone thus doesn't reassure us much…)

"0-bit Leak" policy rarely reasonable

enforcements too coarse/restrictive (all/nothing)

some programs leak **by design.**

how do you secure such programs?

- login interface (success/failure of attempt)
- average salary of workers
- online shopping (ratings, suggested items)

some programs have a **dynamic** security policy

- social network (view/post permissions)
- (un)shared files, ...

"0-bit Leak" policy rarely reasonable

enforcements too coarse/restrictive (all/nothing)

some programs leak **by design.**

- login interf tempt)
- average sa
- online shop items)

how do you secure such programs?

How to **conditionally allow leaks**? And,
How to **specify a policy that changes,**
to e.g.

- declare **leakage intent,** or
- add/remove users?

How is such a policy enforced **statically**?

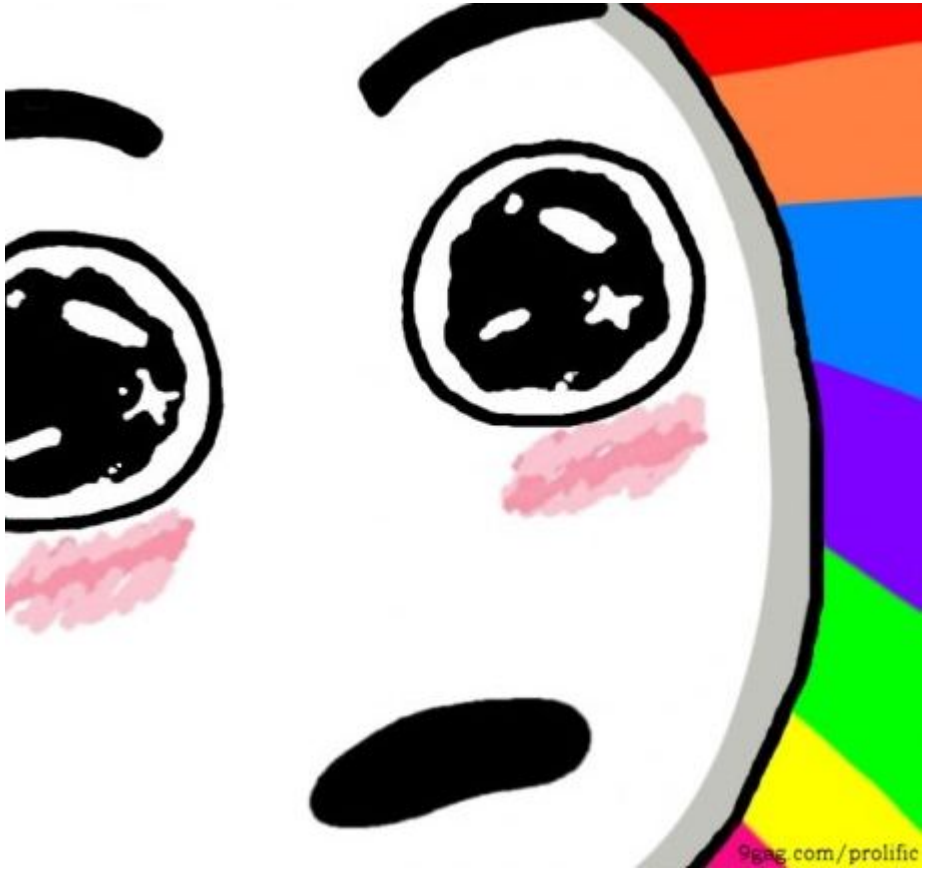
some program ty policy

- social network (view/post permissions)
- (un)shared files, ...

Paragon

Paragon

Paragon



Paragon

About

Paragon is a language extension to the programming language Java that enables practical programming with information flow controls.

Download

To compile and run a Paragon program you need the Paragon compiler as well as a collection of interface files and the Paragon run time environment. Select the right version:

Paragon 0.1 Supports explicit actors (as in the POPL '10 paper).
Paragon 0.2 Supports objects as actors (as in the APLAS '13 paper).

				Interface files	Runtime
Paragon 0.1	32 bit	64 bit	64 bit	source libPI.zip	libs
Paragon 0.2	32 bit	64 bit	64 bit	source libPI.zip	paragon_rt.jar

Paragon 0.2 can also be installed from Hackage: `cabal install paragon`

> Usage instructions

Tutorials

Get started with Paragon via our online interactive tutorial, or take a look at some of our case studies.

- Tutorial: interactive or as pdf.
- Case study: Sealed Bid Auction.
- Case study: Social Network.
- Case study: Mental Poker.
- Case study: ParaJPMail.

Publications

Selected publications:

- **Paragon for Practical Programming with Information-Flow Control** Niklas Broberg, Bart van Delft and David Sands *Asian Symposium on Programming Languages and Systems (APLAS) 2013, Springer* [.pdf] [Technial Report] Introduction of the Paragon programming language.
- **Paralocks - role-based information flow control and beyond** Niklas Broberg and David Sands *POPL'10, Proceedings of the 37th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* [.pdf] Description of the Paralocks language used for specifying information flow policies.

Other publications:

- **A Datalog Semantics for Paralocks** Bart van Delft, Niklas Broberg and David Sands

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
        ?high boolean mySecret;
        ?low  boolean myPublic;
        public void m() {
                mySecret = myPublic;
        }
}
```

levels of confidentiality


```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
      ?high boolean mySecret;
      ?low  boolean myPublic;
      public void m() {
            mySecret = myPublic;
      }
}
```

reading this yields value of confidentiality level high.
writing to it has no other side-effects.

(should really be ?!high, or just high)

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level *written to* as a side-effect during invocation.
treat field read/write as get/set method call.

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
      ?high boolean mySecret;
      ?low  boolean myPublic;
      public void m() {
            mySecret = myPublic;
      }
}
```

reading this yields value of confidentiality level high.
writing to it has no other side-effects.

(should really be ?!high, or just high)

OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level *written to* as a side-effect during invocation.
treat field read/write as get/set method call.

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
      ?high boolean mySecret;
      ?low  boolean myPublic;
      public void m() {
            myPublic = false;
      }
}
```

Why is this program rejected?

NOT OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
      ?high boolean mySecret;
      ?low  boolean myPublic;
      public void m() {
            myPublic = false;
      }
}
```

write effect mismatch.

This is the technical reason; why needed? (hint: pc)

NOT OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
      ?high boolean mySecret;
      ?low  boolean myPublic;
      public void m() {
            myPublic = false;
      }
}
```

write effect mismatch.

calling this method
in a high branch
leaks otherwise

What is the fix?

NOT OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
      ?high boolean mySecret;
      ?low  boolean myPublic;
      !low public void m() {
            myPublic = false;
      }
}
```

OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.


```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
        ?high boolean mySecret;
        ?low  boolean myPublic;
        !low public void m() {
                myPublic = mySecret;
        }
}
```

explicit flow

NOT OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.

```
import static HighLow.*; // Importing defs of high and low
public class MyClass {
        ?high boolean mySecret;
        ?low  boolean myPublic;
        !low public void m() {
                if (mySecret) {
                        myPublic = true;
                } else {
                        myPublic = false;
                }
        }
}
```

implicit flow

NOT OK

each method has a:

? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*

! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.

treat field read/write as get/set method call.


```
import tutorial.HighLow.*;
import tutorial.UntrustedTrusted.*;

public class MyClass {
    ?(high + untrusted) String myDiary;
    ?(high + trusted)   String myPassword;

    public !(high + untrusted) void savePassword(){
       myDiary += myPassword;
    }
}
```

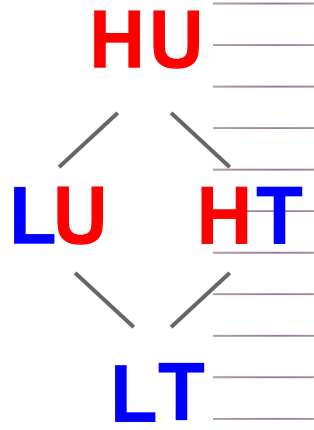
greatest lower bound;
easily *build new policies*
from old ones.

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.

```
import tutorial.HighLow.*;
import tutorial.UntrustedTrusted.*;

public class MyClass {
    ?(high + untrusted) String myDiary;
    ?(high + trusted)   String myPassword;

    public !(high + untrusted) void savePassword(){
      myDiary += myPassword;
    }
}
```



OK

each method has a:
? : **read effect;** [default: "**L**"]
(highest) confidentiality level
of value *returned*
! : **write effect;** [default: "**H**"]
(lowest) confidentiality level
written to as a side-effect during invocation.
treat field read/write as get/set method call.

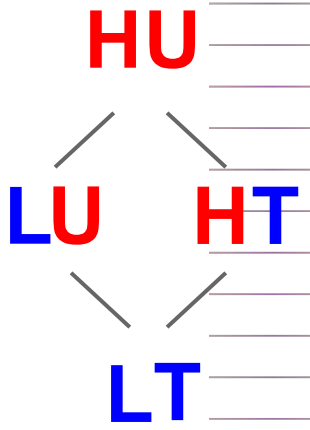
```
import tutorial.HighLow.*;
import tutorial.UntrustedTrusted.*;

public class MyClass {
    ?(high + trusted)   String myDiary;
    ?(high + trusted)   String myPassword;
    ?(low  + untrusted) String fire;

    public !(high + trusted) void savePassword(){
      myDiary += myPassword;
    }
    public !(high + trusted) void burnDiary(){
      myDiary =  fire;
    }
}
```

NOT OK

policy prevents diary from being burned.



Integrity

```
public class HighLow {
    private static final Object lowObserver;
    private static final Object highObserver;

    public static final policy high = { highObserver : };
    public static final policy low  = { lowObserver :
                                      ; highObserver : };
}
```

actor; just an object, representing an observer of information.
a *basic building block* of policies.

high can be observed by highObserver,
under **all** circumstances.

We can define our own policy from scratch.

```
public class KeySeller {
    public static lock Paid;
    public final Object customer = new Object();
    ?{customer: }     String customerData;
    ?{customer: Paid} String softwareKey;

    public void processPayment() {
        // customer pays for item
        if (paymentSuccessful) { open Paid; } else {...}
    }
    public !{customer: Paid} void buyKey(){
        processPayment();
        if (Paid) { customerData = softwareKey; } else {...}
    }
}
```

flow lock; just a boolean.
a *basic building block*
of *dynamics* in policies,
defining **policy state.**

customer can read softwareKey when Paid is open.

runtime testing of a lock!

```
public class KeySeller {
    public static lock Paid;
    public final Object customer = new Object();
    ?{customer: }     String customerData;
    ?{customer: Paid} String softwareKey;

    public void processPayment() {
        // customer pays for item
        if (paymentSuccessful) { open Paid; } else {...}
    }
    public !{customer: Paid} void buyKey(){
        processPayment();
        if (Paid) { customerData = softwareKey; } else {...}
    }
}
```

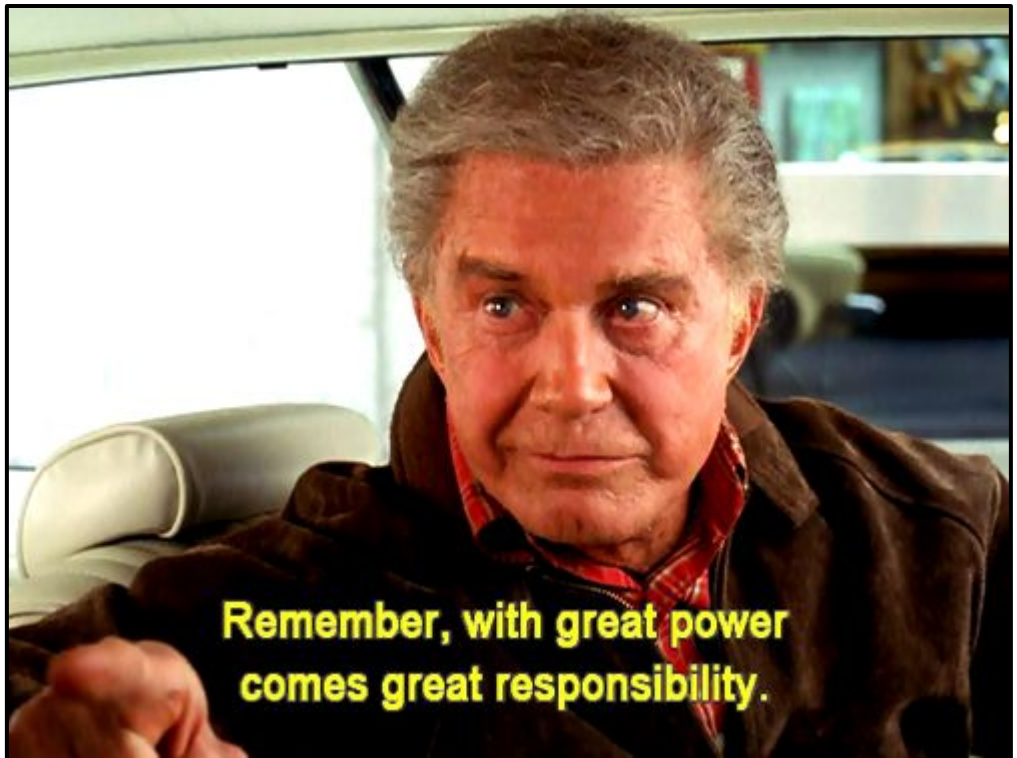
Here programmer declares **leakage intent** (here in a manner similar to an *escape hatch*), thus **declassifying** the key!

statically!!!

only checks for explicit & implicit flows

Paragon ensures the **only** leaks that occur are those **declared intentional** by the programmer!

Paragon ensures the **only** leaks that occur are those **declared intentional** by the programmer!



Paragon ensures the **only** leaks that occur are those **declared intentional** by the programmer!

"With great power comes great responsibility."

- understand the **implications** of a leak
  it's tempting to declassify everything to pass analysis.
  but then there's no security guarantee...
- can be hard when policy is dynamic
- **even harder** when information can leak through the current state of the policy.

Paragon tracks this!

```
policy workData =
   { Manager m :
   ; (Manager m) Employee e : GivesPermissions(m, e)
   ; (Manager m) Employee e : IsBoss(m), WorksFor(e, m) };
```

relational lock!

who has access to a workData?

an **actor** of **type** Manager

any e that has been given permission by an m

{ elton : ; Employee e : ReadsFor(e, elton) }

want to support runtime changing permissions.

{ elton : ; Employee e : ReadsFor(e, elton) }

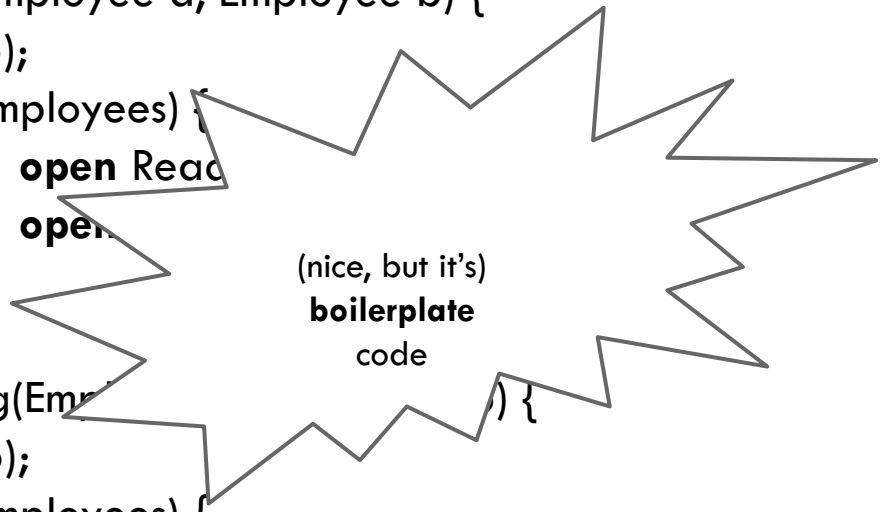
want to support runtime changing permissions.

```
void allowReading(Employee a, Employee b) {
  open ReadsFor(a, b);
  for (Employee e : employees) {
    if (ReadsFor(b, e))  open ReadsFor(a, e);
    if (ReadsFor(e, a))  open ReadsFor(e, b);
  }
}
void disallowReading(Employee a, Employee b) {
  close ReadsFor(a, b);
  for (Employee e : employees) {
    if (ReadsFor(b, e))  close ReadsFor(a, e);
  }
}
```

{ elton : ; Employee e : ReadsFor(e, elton) }

want to support runtime changing permissions.

```
void allowReading(Employee a, Employee b) {
  open ReadsFor(a, b);
  for (Employee e : employees) {
    if (ReadsFor(b, e))  open Read
    if (ReadsFor(e, a))  open
  }
}
void disallowReading(Em                    ) {
  close ReadsFor(a, b);
  for (Employee e : employees) {
    if (ReadsFor(b, e))  close ReadsFor(a, e);
  }
}
```



{ elton : ; Employee e : ReadsFor(e, elton) }

want to support runtime changing permissions.

lock ReadsFor(Employee, Employee)
{ (Employee a) ReadsFor(a,a) :
; (Employee a, b, c) ReadsFor(a,c) : ReadsFor(a,b), ReadsFor(b,c) };

implicitly opened locks

now, instead of

allowReading and disallowReading,

we just open / close ReadsFor, and

transitive permissions update **automatically!**

Tip of the iceberg

- sane default labels
  - low read effect, high write effect
  - **label polymorphism** in method parameters!
- exception handling
- encapsulation: hide information-flow policy behind an interface; detect when invoking code tries to violate the policy through the interface.
- Java-Paragon bridge; .pi file summarizing information flows in a Java file
  - used to port Java standard library to Paragon!

e.g. access control in Android

No

- concurrency (yet)
- dynamic loading

Language is still maturing

- bugs

concurrency woes

- race conditions
- randomness
- synchronous I/O

sleep secret
out public 1

← **in parallel with** →

sleep 100
out public 0

no explicit flow
no implicit flow
no termination flow
no progress flow
has timing flow

processes race on public
which public output occurs first depends on
which process reaches output statement first
(which depends on secret)

no explicit flow
no implicit flow
no termination flow
no progress flow
no timing flow

secret = 3: 1 *can* be the first output on L.

secret = 500: 1 *cannot* be the first output on L;
0 will be.

Thus, then multiple processes are made part of one system, a **timing flow** can turn into an **explicit flow**

works even if processes communicate over shared memory

sleep secret
x := 1

← **in parallel with** →

sleep 100
out public x

no explicit flow
no implicit flow
no termination flow
no progress flow
has timing flow

processes race on public
which public output occurs first depends on
which process reaches output statement first
(which depends on secret)

no explicit flow
no implicit flow
no termination flow
no progress flow
no timing flow

secret = 3: 1 *can* be the first output on L.

secret = 500: 1 *cannot* be the first output on L;
0 will be.

Thus, then multiple processes are made part of one system, a **timing flow** can turn into an **explicit flow**

```
if secret mod 2 == 0 { out public 0 }
else { while true {} }
```

← in → parallel with

```
sleep 100
out public 1
```

no explicit flow
no implicit flow
no termination flow
has progress flow
no timing flow

processes race on public
which public output occurs first depends on
which process reaches output statement first
(which depends on secret)

no explicit flow
no implicit flow
no termination flow
no progress flow
no timing flow

secret = 0: 0 *can* be the first output on L.

secret = 1: 0 *cannot* be the first output on L;
1 will be.

Thus, then multiple processes are made part of one system, a **progress flow** can turn into an **explicit flow**

will also work for shared memory concurrency

random bit, either 0 or 1

x = random(0,1)

skip

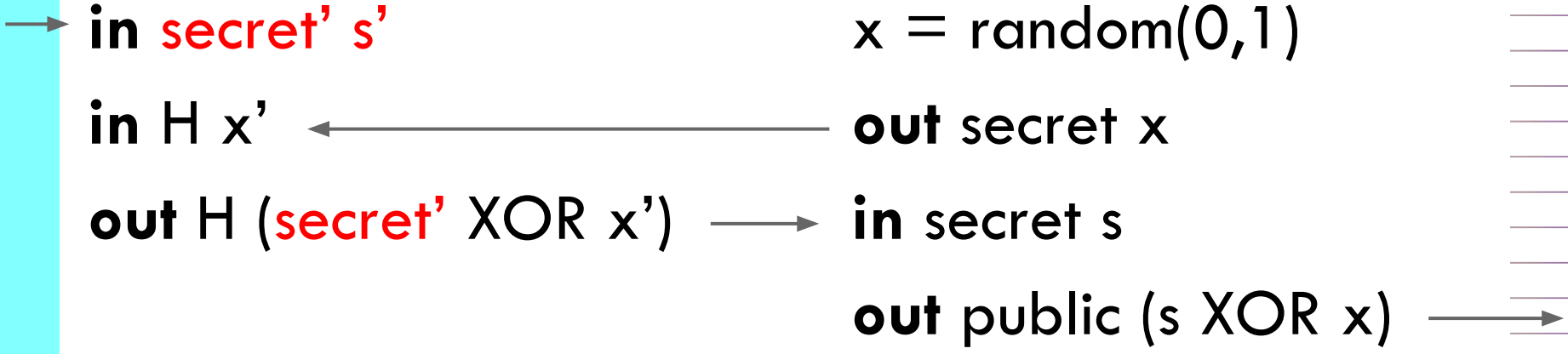
receive 1 bit

in secret s

out public (s XOR x)

1-bit encryption

```
x = random(0,1)
out secret x
in secret s
out public (s XOR x)
```



$$
\begin{aligned}
s \text{ XOR } x &= \text{secret'} \text{ XOR } x' \text{ XOR } x \\
&= \text{secret'} \text{ XOR } x \text{ XOR } x \\
&= \text{secret'}
\end{aligned}
$$

must exercise great care wrt.
how processes disclose randomness.
(left process has no leaks, but it inadvertently,
or maliciously, creates an unfortunate
dependency in the right process)

in secret_0 h

out public 0

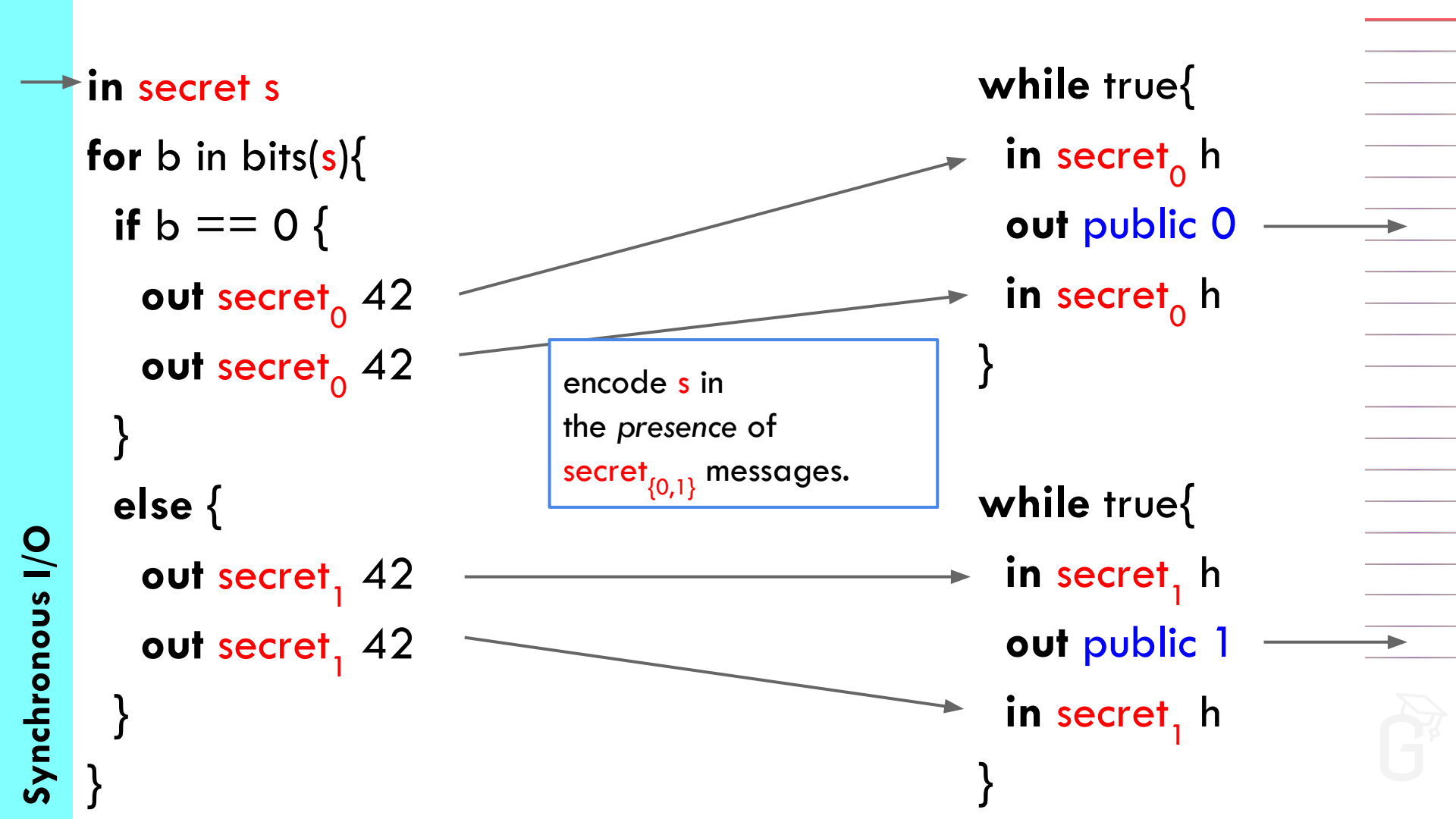
in secret_0 h

Synchronous I/O

```
in secret s
for b in bits(s){
  if b == 0 {
    out secret0 42
    out secret0 42
  }
  else {
    out secret1 42
    out secret1 42
  }
}
```

```
while true{
  in secret0 h
  out public 0
  in secret0 h
}

while true{
  in secret1 h
  out public 1
  in secret1 h
}
```


in secret s
for b in bits(s){
if b == 0 {
out secret0 42
out secret0 42
}
else {
out secret1 42
out secret1 42
}
}
while true{
in secret0 h
out public 0
in secret0 h
}
while true{
in secret1 h
out public 1
in secret1 h
}
encode s in
the presence of
secret{0,1} messages.

in secret h ;

out public 0

willingness of environment to provide **secret** input *interferes* with presence of **public** output.

out secret 0 ;

out public 0

willingness of environment to *receive* **secret** output interferes with presence of **public** output.

processes communicating synchronously exert control over each other in unintuitive ways;
an **output will behave like an input** (flow-wise).

Information Flow Control, Summary

in simple scenarios, we can specify a flow policy as a lattice, and use classic compile-time or runtime enforcement to detect/prevent information leaks.

in practice, simple scenarios are rare. some apps leak by design. dynamic policies needed.

Paragon: Java, extended with way to express and enforce dynamic policies.

bleeding edge (tool is not ready for adoption). even it has shortcomings: leaks that arise due to concurrency issues (no information flow tool help).

you now know enough to use tools for detecting/preventing simple leaks, and to manually inspect code to detect/prevent even devious leaks.

Things to consider when picking the right tool:

- **platform**
  3rd party JS in browser vs. your own Paragon
- **security concerns**
  integrity needed? secure through transformation?
- **threat**
  what attacks are we considering?
- **trust**
  did we create the code? the policy? the enforcement? the OS and hardware?

Things to consider when picking the right tool:

- platform

  3rd party JS in browser vs. your own Paragon
- security concerns

  integrity needed? secure through transformation?
- threat

  what attacks are we co
- trust

  did we create the code
  the OS and hardware?

we can be “forceful” when handling 3rd party non-safety-critical code.

we **implicitly** trust the tools we use.

- Paragon
- Jif, LIO, ifc-ts, …

all tools make assumptions on the platform; some attacks fall out of scope, e.g. concurrency, side-channels in hardware, …

FIND OUT MORE

https://gamess.dk/

GameSS

WHO IS BEHIND GameSS

Partners behind the project

SDU

DTU

IT UNIVERSITY OF CPH

AALBORG UNIVERSITET

Happy Z

Collaborators

ERHVERVSAKADEMI AARHUS

kea
KØBENHAVNS ERHVERVSAKADEMI

ERHVERVSAKADEMI

UCN

Erhvervsakademi og Professionshøjskole

Supported by

Uddannelses- og Forskningsministeriet

Ministry of Higher Education and Science Denmark