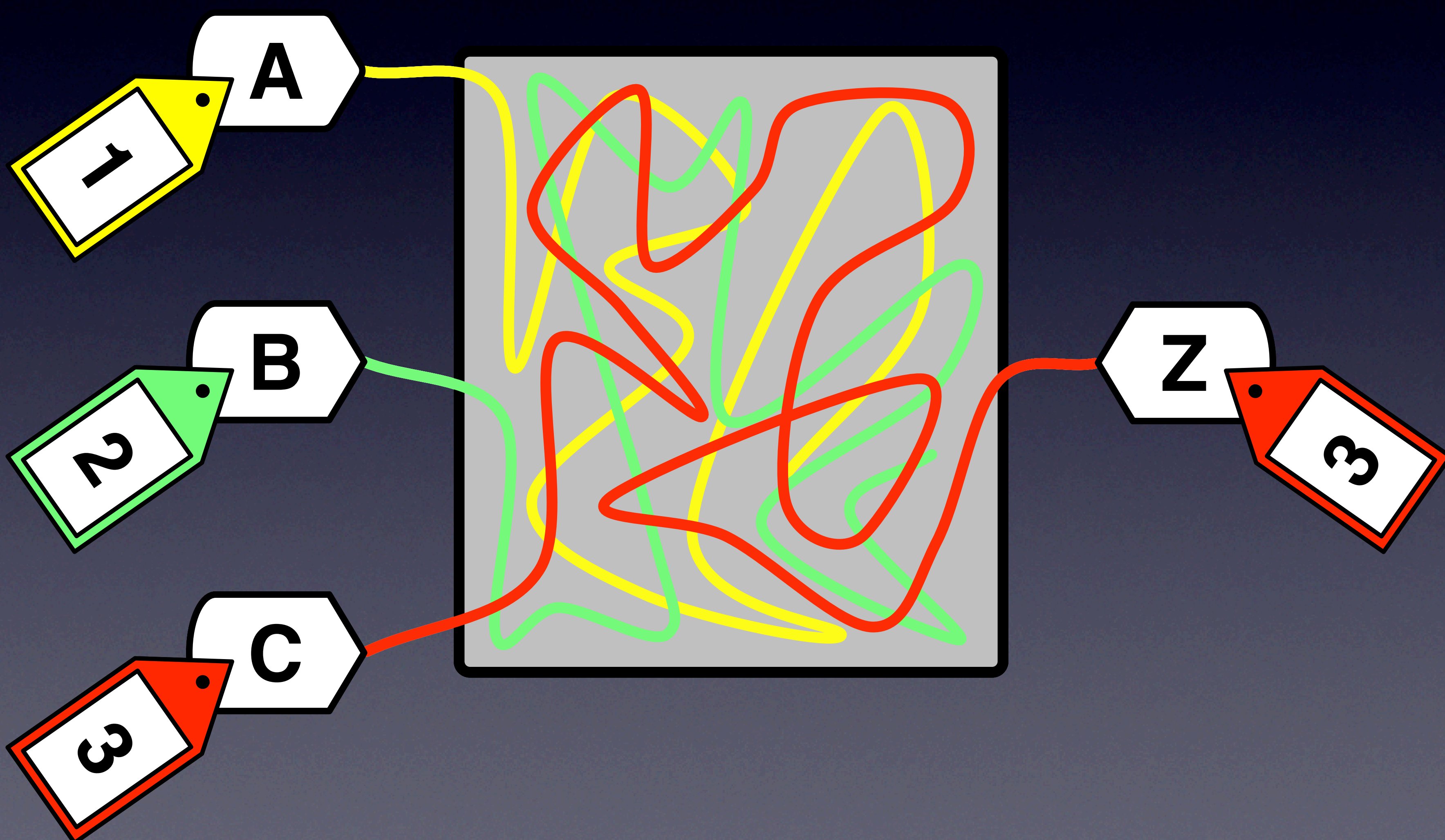
Dytan: A Generic Dynamic Taint Analysis Framework

James Clause, Wanchun (Paul) Li,
and Alessandro Orso
College of Computing
Georgia Institute of Technology

Partially supported by:
NSF awards CCF-0541080 and CCR-0205422 to Georgia Tech,
DHS and US Air Force Contract No. FA8750-05-2-0214

Dynamic taint analysis

(aka dynamic information-flow analysis)



Dynamic tainting applications

Attack detection / prevention

Information policy enforcement

Testing

Data lifetime / scope

Dynamic tainting applications

Attack detection / prevention

Detect / prevent attacks such as SQL injection, buffer overruns, stack smashing, cross site scripting

e.g., Suh et al. 04, Newsome and Song 05, Halfond et al. 06, Kong et al. 06, Qin et al. 06

Testing

Data lifetime / scope

Dynamic tainting applications

Attack detection / prevention

Information policy enforcement

ensure classified information does not leak outside the system

e.g., Vachharajani et al. 04, McCamant and Ernst 06

Testing

Data lifetime / scope

Dynamic tainting applications

Attack detection / prevention

Information policy enforcement

Testing

Coverage metrics, test data generation heuristic, ...

e.g., Masri et al 05, Leek et al. 07

Data lifetime / scope

Dynamic tainting applications

Attack detection / prevention

Information policy enforcement

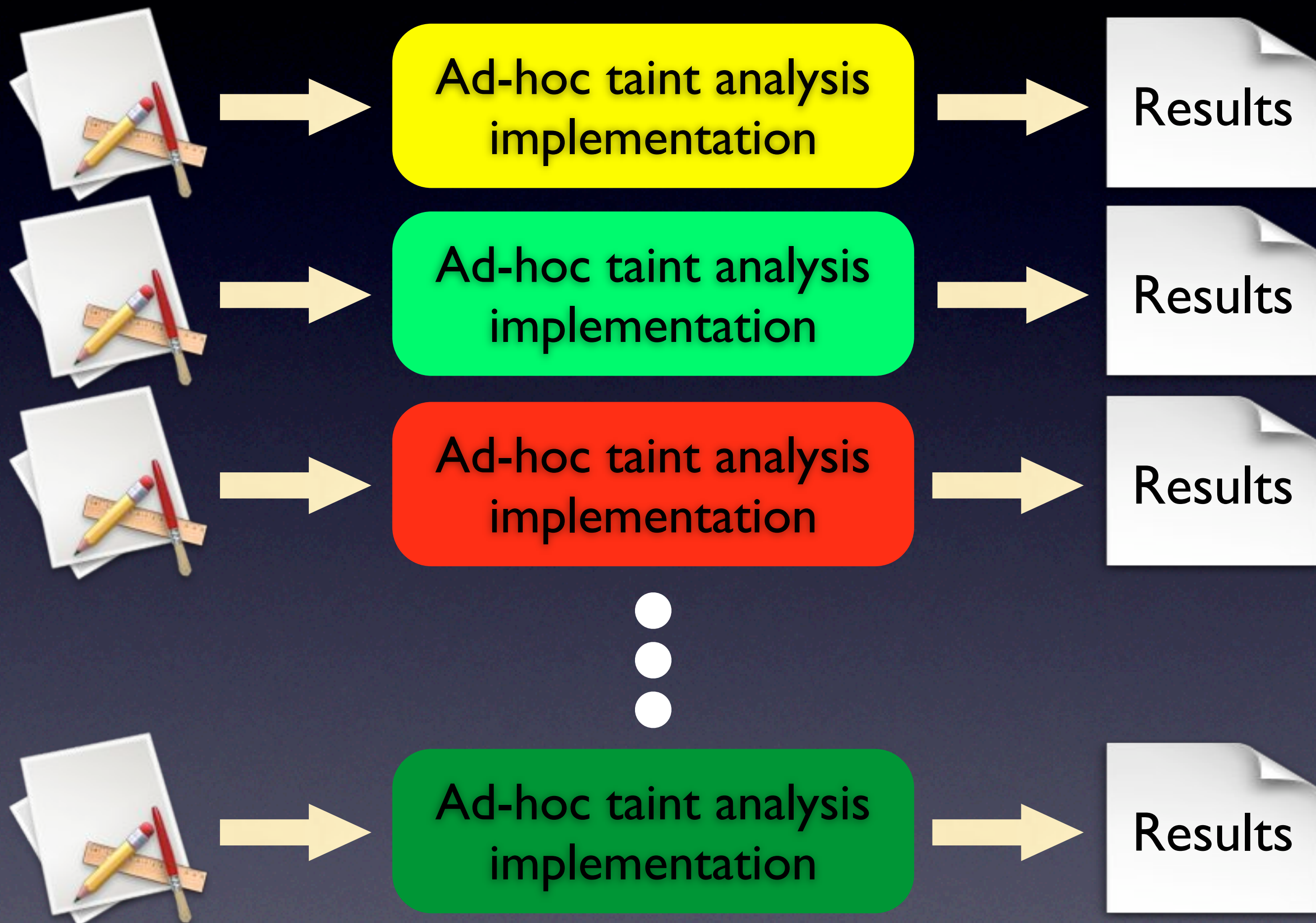
Testing

Data lifetime / scope

track how long sensitive data, such as passwords or account numbers, remain in the application

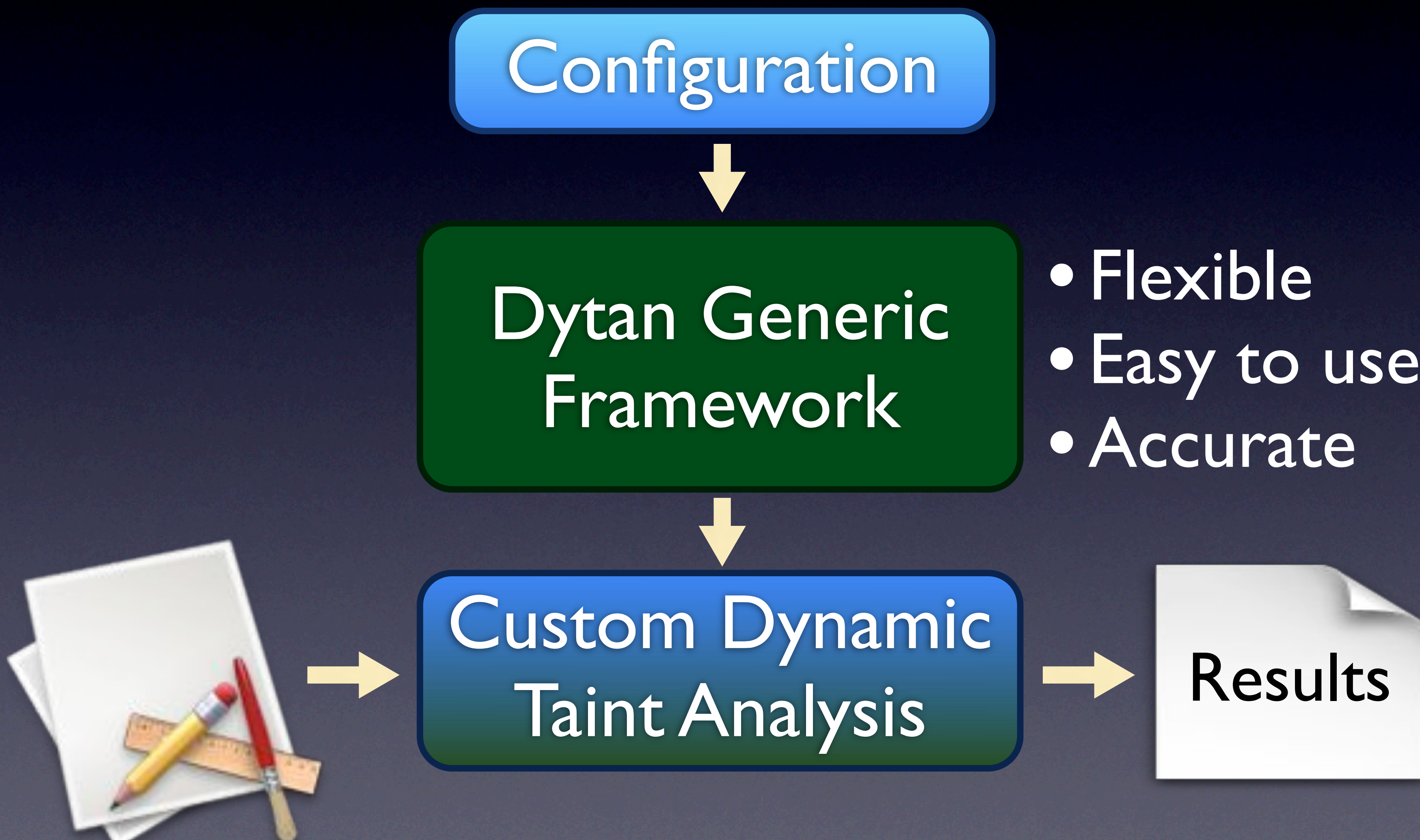
e.g., Chow et al. 04

Motivation



Ad-hoc taint analysis implementation
Results
Ad-hoc taint analysis implementation
Results
Ad-hoc taint analysis implementation
Results
Ad-hoc taint analysis implementation
Results

Motivation



Configuration
Dytan Generic Framework
• Flexible
• Easy to use
• Accurate
Custom Dynamic Taint Analysis
Results

Outline

- ✓ Motivation & overview
- Framework (Dytan)
  - flexibility
  - ease of use
  - accuracy
- Empirical evaluation
- Conclusions

Framework: flexibility

Configuration

Framework: flexibility

Framework: flexibility

Which data to tag, and how to tag it

Framework: flexibility

How tags should be propagated at runtime

Framework: flexibility

Where and how tags should be checked

Taint sources

What to tag

Identify what program data should be assigned tags

- Variables (local or global)
- Function parameters
- Function return values
- Data from an input stream
  network, filesystem, keyboard, ...
- Specific input stream
  141.195.121.134:80, a.txt,...

How to tag

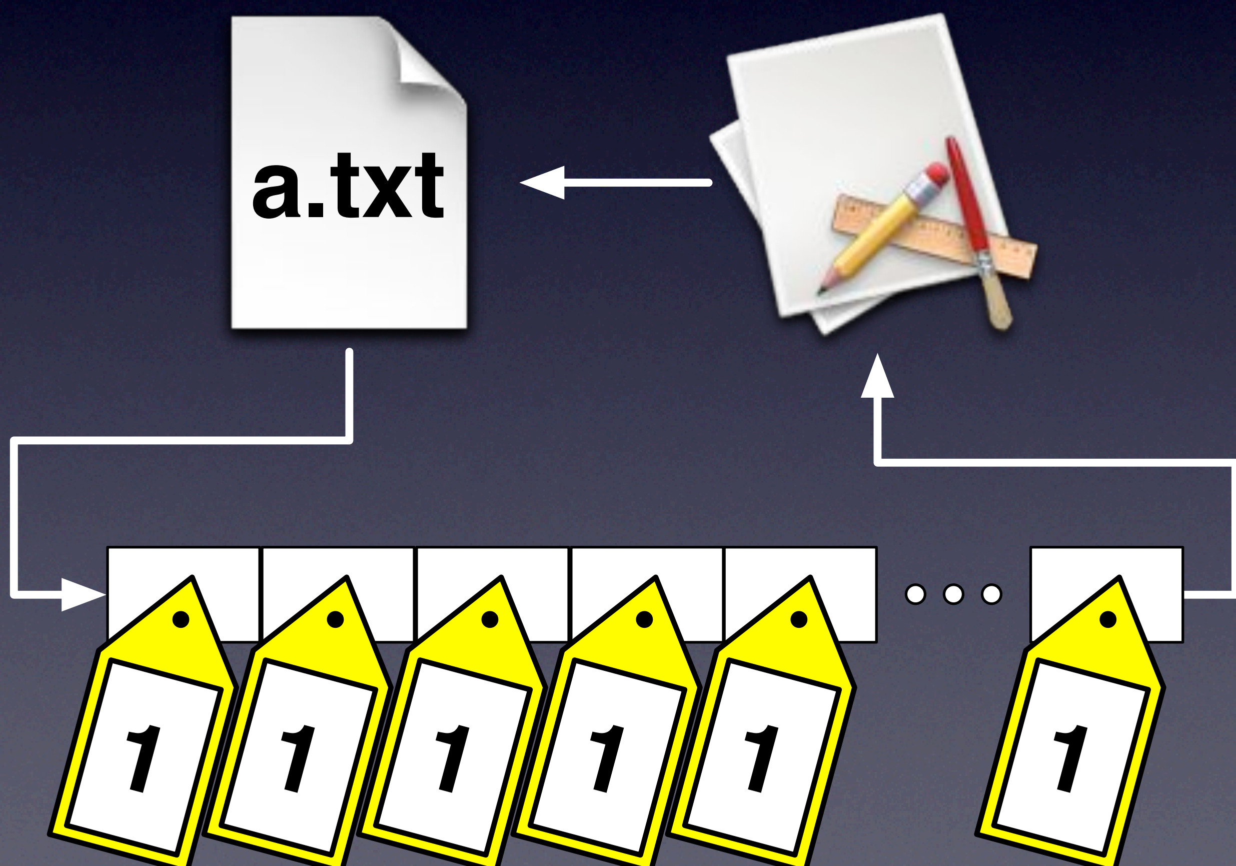
Describe how tags should be assigned for identified data

- Single tag
- One tag per source
- Multiple tags per source
- ...

Taint sources

What to tag: a.txt

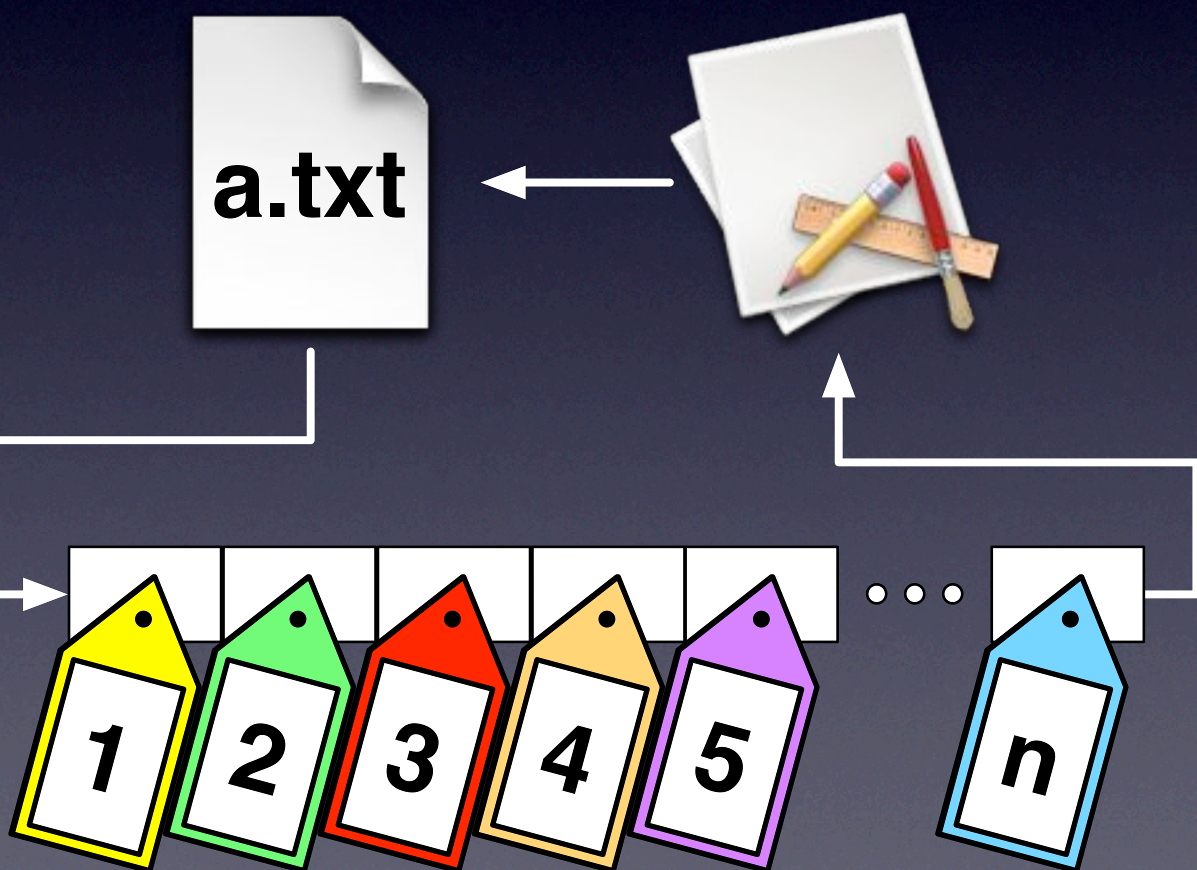
How to tag: single tag



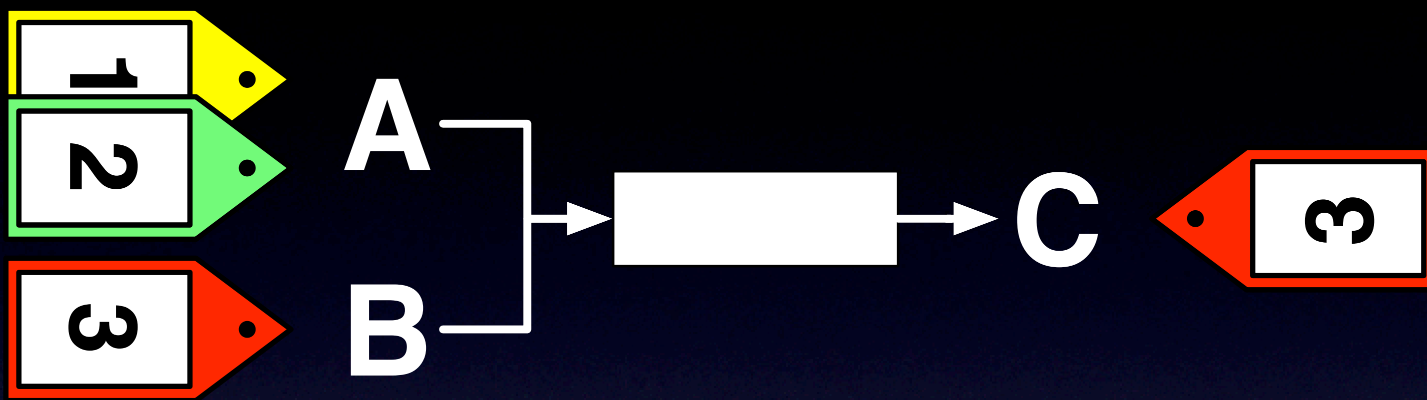
Taint sources

What to tag: a.txt

How to tag: multiple tags



Propagation policy



Affecting data

Data that affects the outcome of a statement through

- Data dependencies
- Control dependencies

A policy can consider both or only data dependencies

Mapping function

Define how tags associated with affecting data should be combined

- Union
- Max
- ...

Propagation policy

```
if(X) {
    C = A + B;
}
```

<u>Affecting data:</u>

- data dependence
- control dependence

<u>Mapping function:</u>

- union
- max

Propagation policy

```
if(X) {
    C = A + B;
}
```

Affecting data:

✔ data dependence

control dependence

Mapping function:

✔ union

max

Propagation policy

```
if(X) {
    C = A + B;
}
```

Affecting data:

- ✔ data dependence
- ✔ control dependence

Mapping function:

- union
- ✔ max

Taint Sinks

Where to check

Location in the program to perform a check

- Function entry / exit
- Statement type
- Specific program point

What to check

The data whose tags should be checked

- Variables
- Function parameters
- Function return value

How to check

Set of conditions to check and a set of actions to perform if the conditions are not met.

- validate presence of tags (exit or log)
- ensure absence of tags (exit or log)
- ...

Taint Sinks

```
cmd = read(file);
args = read(socket);
cmd = trim(cmd + args);
...
tok[] = parse(cmd);
exec(tok[0], tok[1]);
```

Taint Sinks

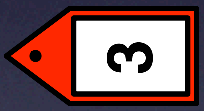
```
cmd = read(file);
args = read(socket);
cmd = trim(cmd + args);
...
tok[] = parse(cmd);
exec(tok[0], tok[1]);
```

<u>Where / what to check:</u>

function: exec, param: 0

<u>How to check:</u>

validate presence of: 

validate absence of: 

<u>Result:</u>

Taint Sinks

```
cmd = read(file);
args = read(socket);
cmd = trim(cmd + args);
...
tok[] = parse(cmd);
exec(tok[0], tok[1]);
```

<u>Where / what to check:</u>

function: exec, param: 0

<u>How to check:</u>

validate presence of: 2

validate absence of: 3

<u>Result:</u> ✗

Framework: ease of use

Provide two ways to configure the framework

- Basic
  - Select sources, propagation policies, and sinks from a set of predefined options
  - XML based configuration
- Advanced
  - Suitable for more esoteric applications
  - Extend OO implementation

Framework: accuracy

- Dytan operates at the binary level
  - consider the actual program semantics
  - transparently handle libraries
- Dytan accounts for both data- and control-flow dependencies

Framework: accuracy

The most common source of inaccuracy is incorrectly identifying the information produced and consumed by a statement

Two common examples:

- Implicit operands

  `add %eax,  %ebx  // A = A +  B`

  produced: `%eax`, **`%eflags`**

- Address Generators

  `add %eax, [%ebx] // A = A + *B`

  consumed: `%eax`, `[%ebx]`, **`%ebx`**

Outline

- ✓ Motivation & overview
- ✓ Framework
  - ✓ flexibility
  - ✓ ease of use
  - ✓ accuracy
- Empirical evaluation
- Conclusions

Empirical evaluation

- RQ1: Can Dytan be used to (easily) implement existing dynamic taint analyses?
- RQ2: How do inaccurate propagation policies affect the analysis results?
- In addition: discussion on performance

RQ1: flexibility

Goal: show that Dytan can be used to (easily) implement existing dynamic taint analyses

- Selected two techniques:
  - Overwrite attack detection [Qin et al. 04]
  - SQL injection detection [Halfond et al. 06]
- Used Dytan to re-implement both techniques
  - Measure implementation time
  - Validate against the original implementation

RQ1: results

- Implementation time:
    - Overwrite attack detection: < 1 hour
    - SQL injection detection: < 1 day

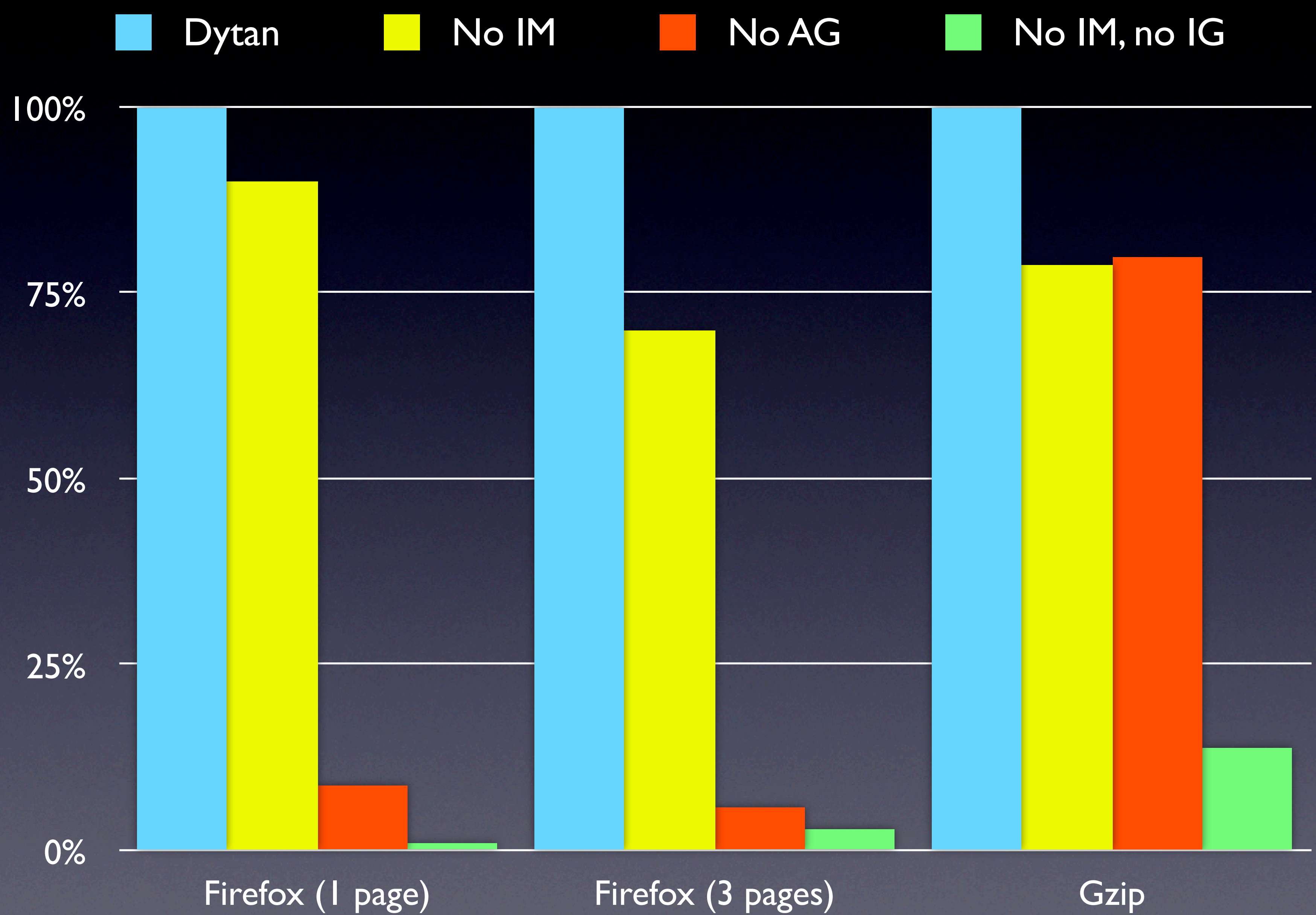
- Comparison with original implementations:
    - Successfully stopped same attacks as the original implementations

RQ2: accuracy impact

Goal: measure the effect of inaccurate propagation policies on analysis results

- Selected two subjects:
  - Gzip (75kb w/o libraries)
  - Firefox (850kb w/o libraries)
- Use Dytan to taint program inputs and measure the amount of heap data tainted at program exit
- Compare Dytan against inaccurate policies
  - no implicit operands (no IM)
  - no address generators (no AG)
  - no implicit operands, no address generators (no IM, no AG)

RQ2: results


Dytan
No IM
No AG
No IM, no IG
100%
75%
50%
25%
0%
Firefox (1 page)
Firefox (3 pages)
Gzip

Performance

- Measured for **gzip**:

  ≈30x for data flow

  ≈50x for data and control flow

- High overhead, but...
  - In line with existing implementations
  - Designed for experimentation
    - Favors flexibility over performance
  - Implementation can be further optimized

Related work

- Existing dynamic tainting approaches [Suh et al. 04, Newsome and Song 05, Halfond et al. 06, Kong et al. 06, ...]
  - Ad-hoc
- Other dynamic taint analysis frameworks [Xu et al. 06 and Lam and Chiueh 06]
  - Focused on security applications
    - Single taint mark
    - No control-flow propagation
  - Operate at the source code level

Conclusions

- Dytan
  - a general framework for dynamic tainting
  - allows for instantiating and experimenting with different dynamic taint analysis approaches
- Initial evaluation
  - flexible
  - easy to use
  - accurate

Future directions

- Tool release (documentation, code cleanup)

  http://www.cc.gatech.edu/~clause/dytan/

  (pre-release on request)
- Optimization (general and specific)
- Applications
  - Memory protection
  - Debugging

Questions?

http://www.cc.gatech.edu/~clause/dytan/