Quantitative Information Flow as network flow capacity

Stephen McCamant, Michael D. Ernst
MIT Computer Science and AI Lab

Conference on Programming Language Design and Implementation 2008.

Internet Security Seminar 2013

Lecturer : Dr. Tom Chothia

Presenter : Rajiv Ranjan Singh

The presentation outline

- Introduction
- Concept of Information Leakage
- Research findings
- Demo of *"flowcheck"* tool
- Results
- Discussion

INTRODUCTION

- Goal of Information Security
  - Safeguard secret data (maintain confidentiality)
  - Log / audit all possible information flow
- Challenges
  - How to measure information flow?
  - How to calculate safe threshold / bound for information flow

Introduction

There are some "Adversaries" we know about

- An active / passive attacker..
  - Remember eavesdropper "Eve"

OR

Something we don't know (or rather care about)

- Program giving out some useful information on the channel unknowingly

Information is flowing out unknowingly

Information Flow

- Cryptographic techniques can control the information released by the system
- What about information propagation?
- While encrypting the channel, we make sure cipher text is not easy to understand, but do we encrypt everything?
- What about little piece of unencrypted output? Does it say anything about what we are doing?

Information Flow

- Information flow is propagation of sensitive data in an unauthorised manner.

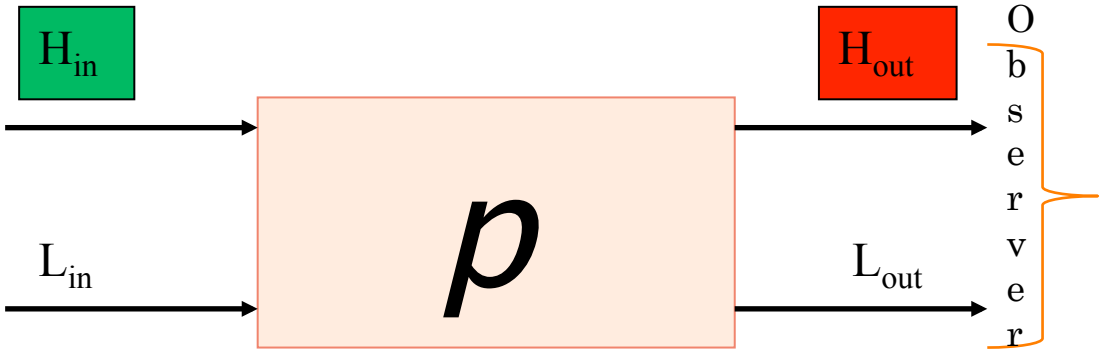
- So, we have two new classes of **"Adversaries"** now……

  - **The Program '*p*'**
    - It has direct access to all the sensitive information (secret)

  - **The Observer '*o*'**
    - It has direct access to only the output of *p* (public)

BEHAVIOUR OF THE PROGRAM

- We define two categories of information:
  - High (Secret) value variables - H
  - Low(Public) value variables - L

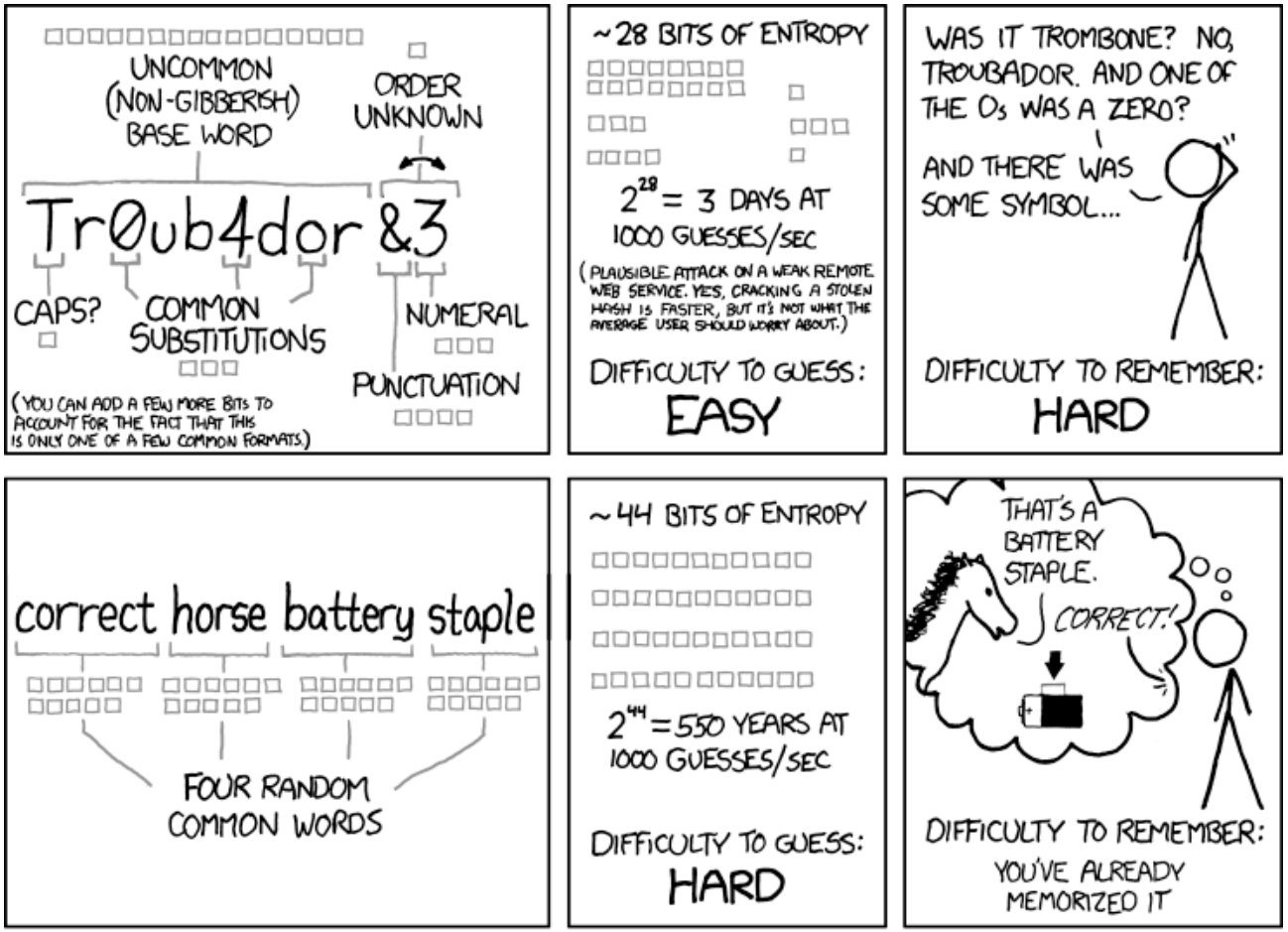


- Design goals
  - No need to worry about L_{in}
  - We have to make sure H_{out} is empty and
  - L_{out} does not say anything about H_{in}

A SECURE PROGRAM

- A Program p is secure iff its public observations(L_{out}) do not depend on any secret input(H_{in}).
- This is called Non-Interference.
- A completely Non-interfering program is secure, else it leaks some information
- Is secure program possible in reality???

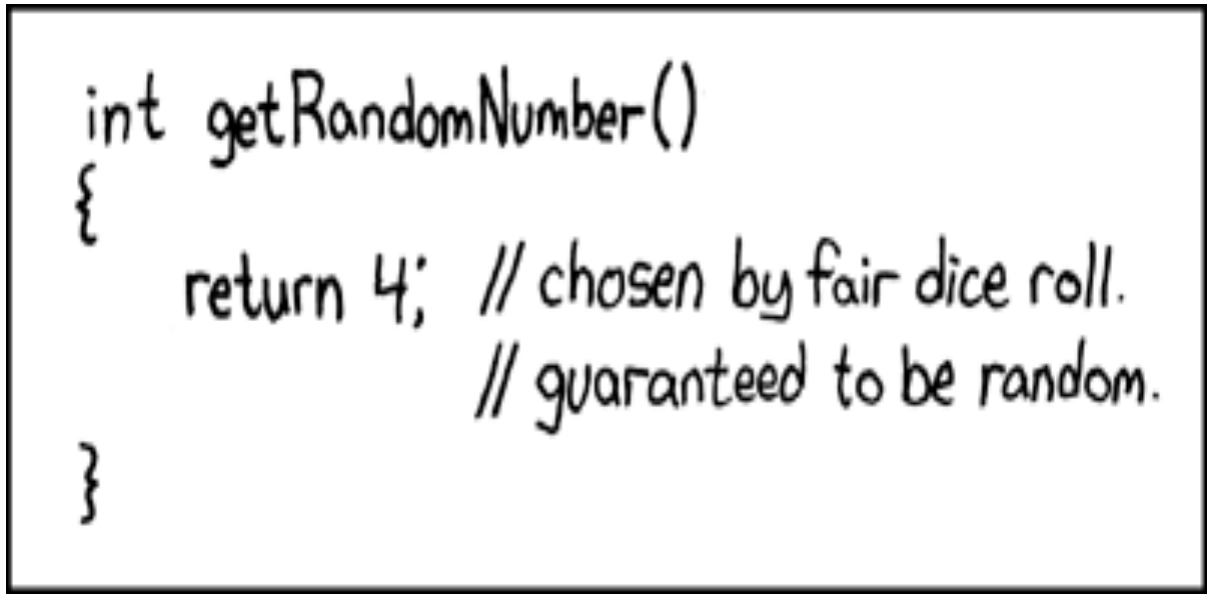
Program Claim



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

http://xkcd.com/936/

Program reality



http://xkcd.com/221/

Defining Information leakage

- The difference in the uncertainty about the secret h before and after observations O on the system:

$$H(h) - H(h \mid O) = I(h;O) \text{ (mutual information)}$$

Where, H(h) is uncertainty about h

- The correlation between secret h and observations O given L, a *measure of the information* h, O *share given* L

Information leakage example

- Consider a simple code

  if (pwd==guess) entry=1 else entry = 0

- Is this completely secure?
  - If entry=1, we know the password
  - If entry=0, we know it is not the password, eliminate one possibility

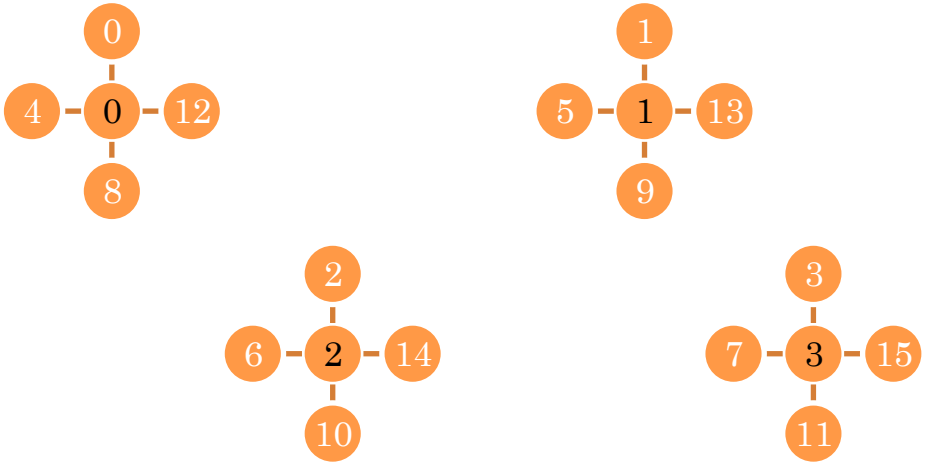
Information leakage example

- Another function

$$F(h) = h \bmod 4$$

- F(h) public, h secret (a 4-bit value 0000 to 1111)
- If output is 1, what is the input?
- Not difficult to guess
- 1,5,9 or 13

Quantifying Information leakage



- For 4 outputs , 16 possible inputs
  - P(before program execution) = 1/16
  - P(after program execution) = 1/4
- Why???
  - Because non-interference clause is violated.
- If Probability changes, program leaks information.
- So, most of the programs leak information.
- The question is …..how much??.

Quantifying Information leakage

- Use Shannon's Entropy formula

$$H = -\sum_{i=1}^{n} p_i . \log_2 p_i$$

- In information theory, **entropy** is the measurement of uncertainty in a random variable.

- For previous example
  - $p_i = ¼$ for i=0,1,2,3
  - $H = -\left(¼ . \log_2 ¼ \right) * 4 = 2$

- leaks 2 bits of information

QIF APPROACH IN PAST

- Most previous research on QIF focused on very small flows
- Previous approaches are based on concept of tainting (a tainted variable is probable to hold secret data).
  - A single tainted input can cause many later values to be tainted.
  - Making copies of secret data does not multiply the amount of secret information present.

QIF APPROACH PRESENTED IN PAPER

- Present work : To measure information flow not using tainting but as a kind of network flow capacity.
  - We can model the possible information channels in an execution of a program as a network of limited-capacity pipes.
  - Then, the maximum rate at which fluid can flow through the network corresponds to the amount of secret information the execution can reveal.
- Expresses a confidential property as a limit on the number of bits that may be revealed
- Measures the bits a program reveals vs threshold.

MAX-FLOW MIN-CUT THEOREM

In optimization theory, the **max-flow min-cut theorem** states that:

In a flow network, the maximum amount of flow passing from the source to the sink is equal to the minimum capacity that when removed in a specific way from the network causes the situation that no flow can pass from the source to the sink.

The value of the maximum flow is* equal *to the capacity of the minimum cut.

Flow graph construction

- Edges represent values
  - capacities – no of bits of data they can hold.
- Nodes represent basic operations
  - in-degree of a node : the operation's arity
- A source node : all secret inputs
- A sink node : all public outputs
- Directed and acyclic graph
  - edges from older nodes to newer nodes

Dynamic maximum-flow analysis

Basic approach

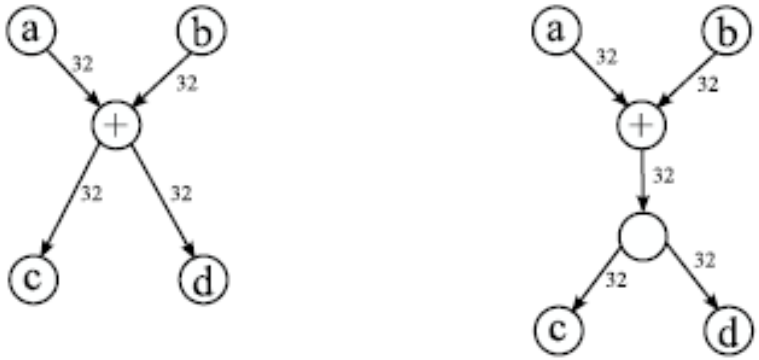


Figure 1. Two possible graphs representing the potential information flow in the expression `c = d = a + b`, where each variable is a 32-bit integer. The graph on the left permits 32 bits of information to flow from `a` to `c`, and a different 32 bits to flow from `b` to `d`. To avoid this, our tool uses the graph on the right.

- Edges represent values (bit capacity), Nodes represent basic operations.
- A source node represents all secret inputs
- A sink node represents all public outputs.

Dynamic maximum-flow analysis

Implicit flows

```
1 /* Print all the "."s or "?"s,
2    whichever is more common. */
3 void count_punct(char *buf) {
4     unsigned char num_dot = 0, num_qm = 0, num;
5     char common, *p;
6     ENTER_ENCLOSE(num_dot, num_qm);
7     while (p = buf; *p != '\0'; p++)
8         if (*p == '.')
9             num_dot++;
10        else if (*p == '?')
11            num_qm++;
12    LEAVE_ENCLOSE();
13    ENTER_ENCLOSE(common, num);
14    if (num_dot > num_qm) {
15        /* "."s were more common. */
16        common = '.'; num = num_dot;
17    } else {
18        /* "?"s were more common. */
19        common = '?'; num = num_qm;
20    }
21    LEAVE_ENCLOSE();
22    /* print "num" copies of "common". */
23    while (num--)
24        printf("%c", common);
25 }
```

Figure 2. C code to print all the occurrences of the most common punctuation character is a string. For instance, when run on its own source code, the program produces the output ".....". As detailed in Section 2.4, our tool reports that this execution reveals 9 bits of information about the input.

- Implicit data flow
  - Branches, arrays, pointers indirectly affect computation
  - Example: array[5] == 0 implies prior accesses did not access 5th index
- A sound graph should account for implicit flows
- Solution: add edges that represent all possible data flows

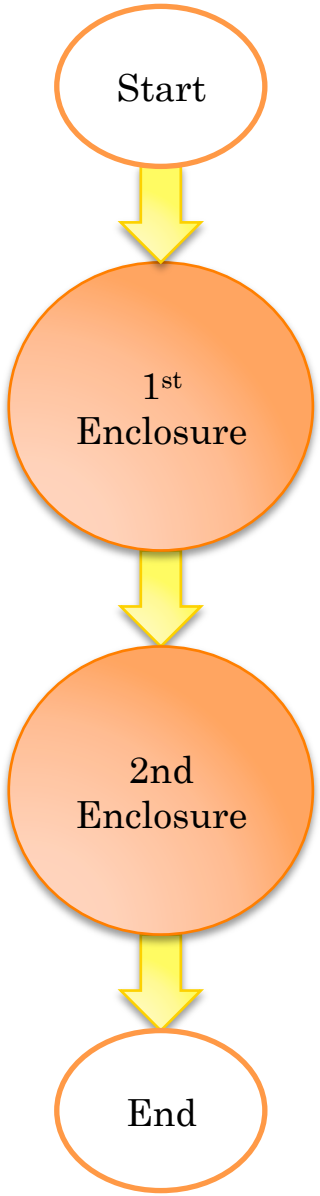
Dynamic maximum-flow analysis
Implicit flows

- Implicit flows are contained in an *enclosure* with defined inputs/outputs
- Our tool constructs a little complex graph.
- Each enclosure region has a distinguished node, and our tool adds edges from each implicit flow operation to that node, and then from that node to each output.
- We will see in next slide, how this is implemented on the C code we just saw.

Enclosure Example

```
1 /* Print all the "."s or "?"s,
2    whichever is more common. */
3 void count_punct(char *buf) {
4     unsigned char num_dot = 0, num_qm = 0, num;
5     char common, *p;
6     ENTER_ENCLOSE(num_dot, num_qm);
7     while (p = buf; *p != '\0'; p++)
8         if (*p == '.')
9             num_dot++;
10        else if (*p == '?')
11            num_qm++;
12    LEAVE_ENCLOSE();
13    ENTER_ENCLOSE(common, num);
14    if (num_dot > num_qm) {
15        /* "."s were more common. */
16        common = '.'; num = num_dot;
17    } else {
18        /* "?"s were more common. */
19        common = '?'; num = num_qm;
20    }
21    LEAVE_ENCLOSE();
22    /* print "num" copies of "common". */
23    while (num--)
24        printf("%c", common);
25 }
```

Figure 2. C code to print all the occurrences of the most common punctuation character is a string. For instance, when run on its own source code, the program produces the output ".....". As detailed in Section 2.4, our tool reports that this execution reveals 9 bits of information about the input.



Dynamic maximum-flow analysis
Implicit flows

- Previous function counts the '.' and '?' in a string
- Prints more common of two as many times as it appeared.
- Source code contains 8 periods and 4 question marks
- Our tool measures leakage as 9 bits of secret input: 1 bit from branch and 8 bits from the count.
- The corresponding minimum cut consists of two edges
  - Comparison between num_dot and num_qm on line 14 (capacity 1 bit)
  - Value of num after the second enclosure region on line 21 (capacity 8 bits).

Dynamic maximum-flow analysis
Implicit flows

- ENTER ENCLOSE and LEAVE ENCLOSE improve the precision of the results.
- without them, a leak of 1 bit each time a value from the input buffer was compared to a constant, 1855 in total.
- The maximum flow computation also improves precision.
- Without it, simple tainting would give a bound of 64 bits.

FLOWCHECK TOOL

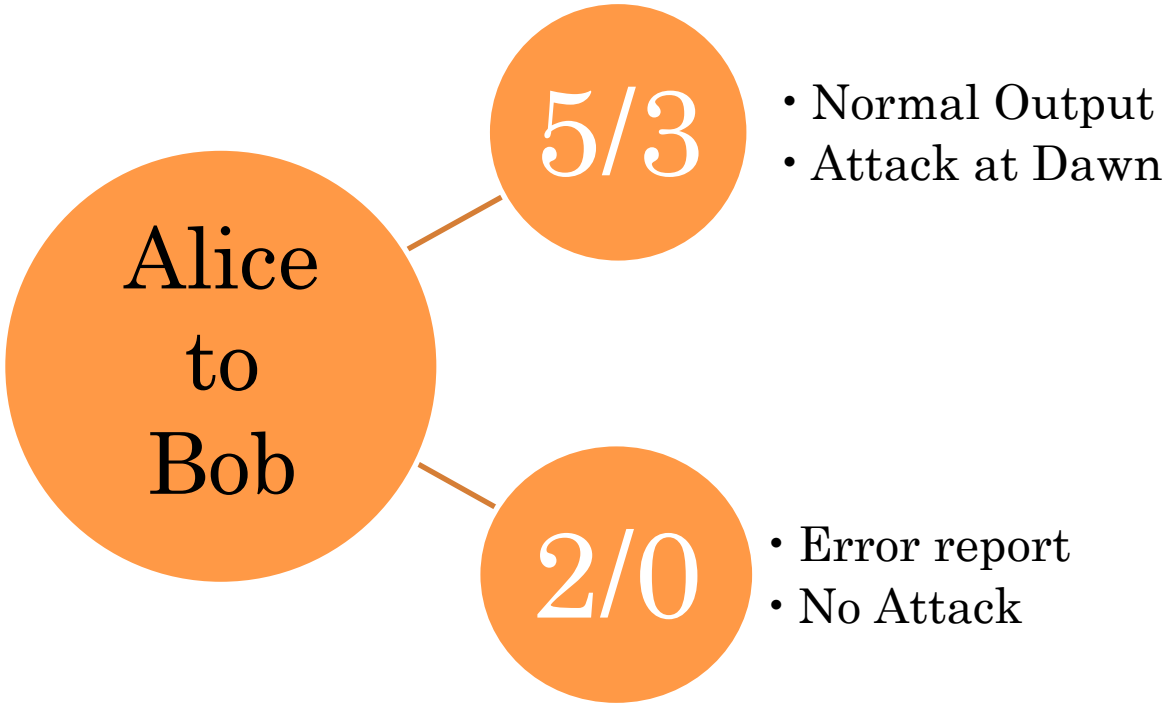
- **Flowcheck: A Valgrind tool for QIF**
  - The implementation of QIF is a dynamic analysis tool called Flowcheck.
  - Flowcheck is a security testing tool.
  - Run the program under its supervision, with certain inputs marked as secret.
  - Produces an upper-bound estimate of the amount of information about those secret inputs revealed by the program's outputs.
  - Flowcheck is based on the Valgrind framework for dynamic analysis tools.

DEMO….

SOUNDNESS AND CONSISTENCY

GENERAL ATTACK MODEL

Adversary(Alice) uses a program to communicate a secret message to a confederate(Bob) via a program c=a/b



What is the channel capacity in this case? (32 bit or 1 bit)

The Channel capacity is determined by the total number of different public outputs the program can produce.

Soundness and consistency general attack model

- An output of 0 means 32 bit?
  - A zero (0) may have multiple representations.
- In the example:
  - '0' → "Attack at dawn
  - '1' → "No Attack"
- For each input i ∈ I sent by Alice
- Tool reports information-flow bound k(i)
- Tool is *sound* iff there is also a code c where for each message i, Alice and Bob could have communicated i using exactly k(i) bits
- Using the division program, it would be sound for the tool to report a bound of 1 bit.

CHECKING A FLOW BOUND

- Computing a minimum cut
  - Discover maximum flow with a minimum cut and verify over many runs.

- Tainting base checking
  - If any other tainted bits reach the output or an implicit flow operation, they are conservatively counted in the location reported.

CHECKING A FLOW BOUND

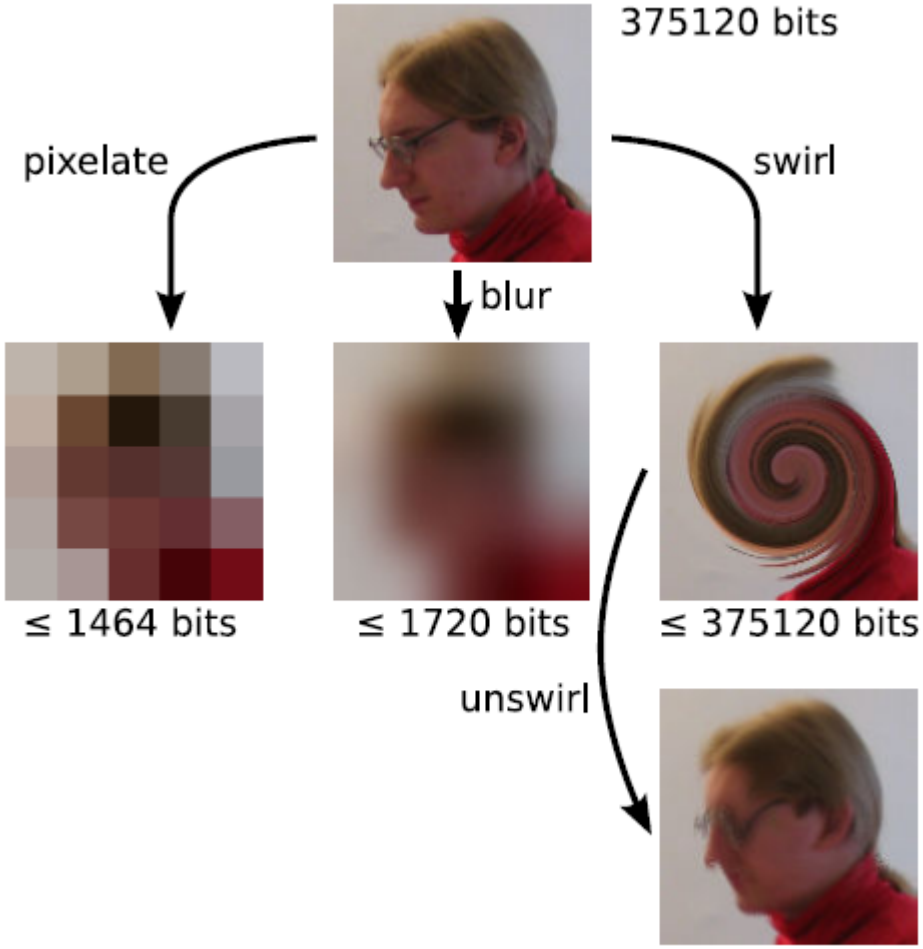
- Output comparison checking
  - Run two copies of a program in lockstep, one which initially has access to the secret input, and the other which operates on a non-sensitive input.
  - If the programs produce the same output, then the data that the second program received from the first at the cuts is the only secret information needed to produce the output, and the flow policy is satisfied.
  - If the outputs diverge, then another flow is present and execution should be terminated.

CASE STUDIES

Program	KLOC	# of libraries	secret data
KBattleship	6.6	37	ship locations
OpenSSH client	65	13	authentication key
ImageMagick	290	20	original image details
OpenGroupware.org	550	34	schedule details
X server	440	11	displayed text

Figure 4. Summary of the programs examined in the case studies of Section 8. The program sizes, measured in thousands of lines of code (KLOC), include blank lines and comments, but do not include binary libraries (3rd column, measured with `ldd`) that were included in the analysis but not directly involved with the security policy.

CASE STUDIES
IMAGEMAGICK



- Image transformations vary in how much information they preserve.
- Pixelating (left) or blurring (middle) the original image (top, 375120 bits), reveals only 1464 or 1720 bits.
- Information revealed by a twisting transformation (right) is 375120 bits.

SUMMARY

- Presented new approach for calculating information flow in a program
- Maximum flow is a more precise graph model of information propagation than reachability
- Checked model on variety of real applications

Conclusion

Accurately quantifying information leak is very important

Information leakage can be costly

- Legal and Financial impacts
  - Insecure code may lead to litigations
- Image and credibility damage
  - Leaked information by secure software may prove counterproductive to business
- Loss of competitive advantage
  - Intellectual/strategic information may be revealed inadvertently

Related work

- jif tool
  - java information flow for information flow control and access control
  - http://www.cs.cornell.edu/jif/
- jpf-qif
  - Quantitative Information Flow Analysis for Java Bytecode
  - http://babelfish.arc.nasa.gov/trac/jpf/wiki/summer-projects/2012-qif
  - http://www.eecs.qmul.ac.uk/~qsp30/
- AQUA tool
  - QIF checking via static analysis
  - https://qmro.qmul.ac.uk/jspui/bitstream/123456789/1260/1/HEUSSERAutomatingQuantitative2011.pdf

THANK YOU !!!

Questions/ Comments??

http://xkcd.com/525/