ÉCOLE POLYTECHNIQUE MONTRÉAL

SOCCER LAB
SOftware Cost-effective Change and Evolution Research Lab

Search Based Software Testing for Software Security: Breaking Code to Make it Safer

Search Based SECurity (SBSec)

Giuliano (Giulio) Antoniol, Canada Research Chair Tier I in Software Change and Evolution
The Software Cost-effective Change and Evolution Laboratory
Ecole Polytechnique de Montreal
antoniol@ieee.org

April 1, 2009 - Denver, Colorado, USA

Outline

- Motivations
- Difficult and easy and problems
- Challenging and open problems

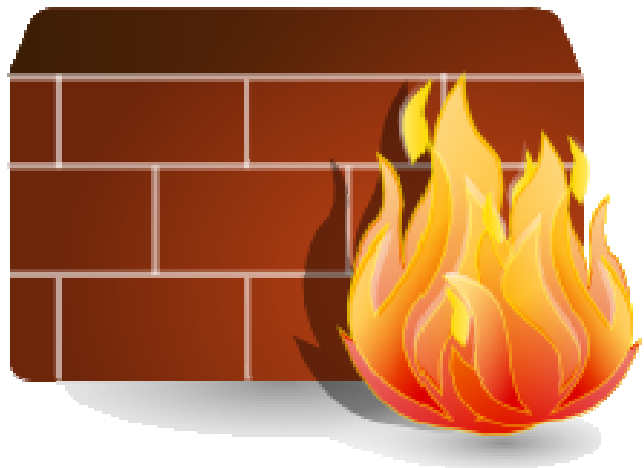
Computerized Systems

- Security and dependability is often a must
- Mitre Corporation definition: ***A vulnerability is mistake in software that can be directly used by a hacker to gain access to a system or network***
- Vulnerability
  - Jeopardize information confidentiality
    - Break into a system and stole information
  - Lead to financial losses
    - Use stolen identity to access financial data
  - Threats to humans

Vulnerability

- Does not require that a successful attack be carried out

- A vulnerable network application plus a firewall may be impossible to exploit … but … the application remains vulnerable though the system is not



Well … My Point …

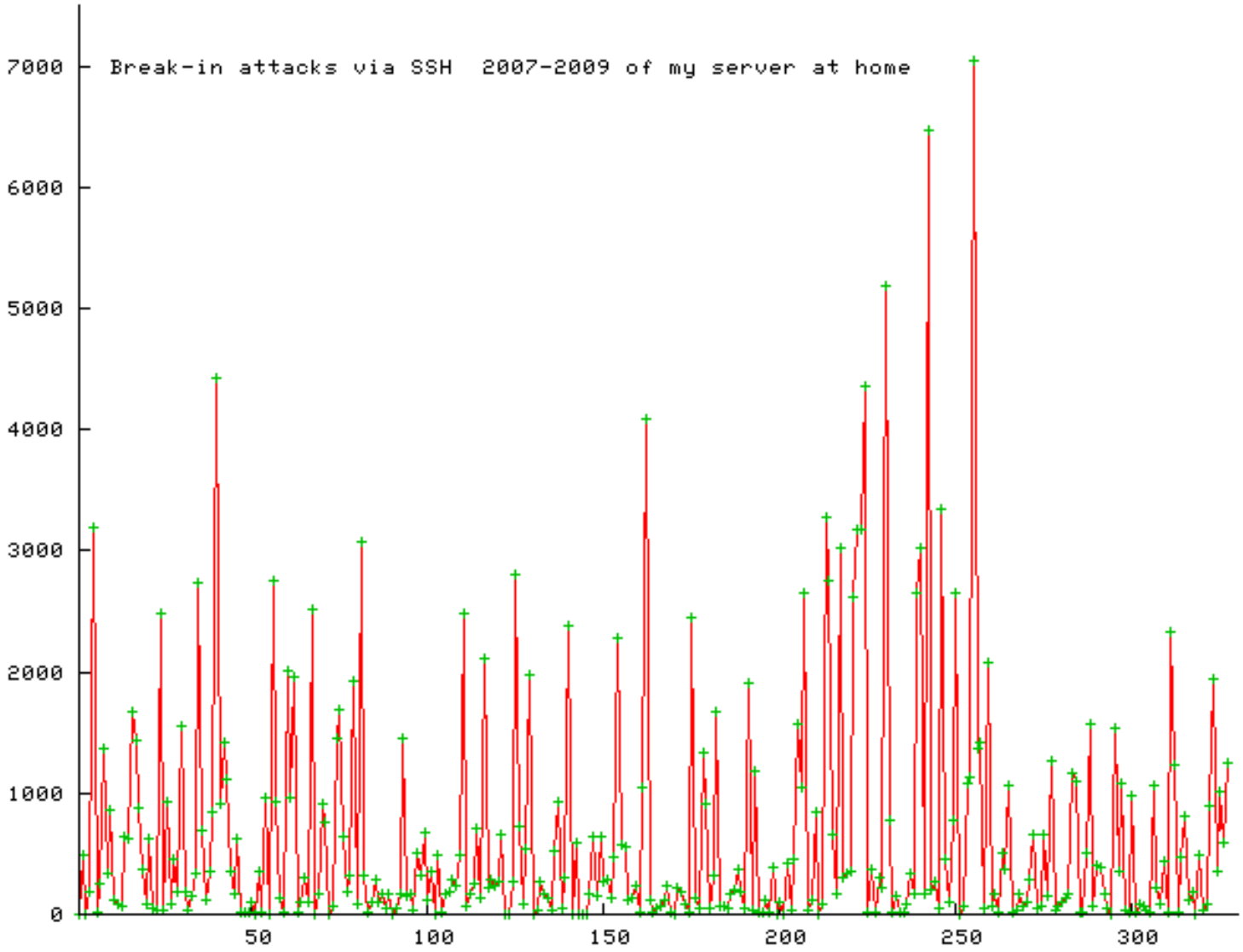
- Software Industry create and evolves complex computerized systems
- Software "engineering"
  - the design and manufacture of complex products
  - the application of science and mathematics by which the properties of matter and the sources of energy in nature are made useful to people
- Industry mostly promoted certain aspect of quality for example dependability
- We are (still) unable to provide effective and automated tools to ensure vulnerability removal … but we are not alone …

Car Industry – Quality and Design

Even military products are not 100% dependable or reliable …
cars … are dependable … but not vulnerability free … they can be easily stolen …

Searching for the Culprits …

Clear and Present Danger (SSH)



April 1, 2009 - Denver, Colorado, USA

It is not 100% true …

- That secure software is difficult to produce
- That secure computerized system are very expensive
- That software is hardly ever secure
- That network applications can be easily attacked

➔ Pessimism gave people the wrong state of mind!

Two (almost) bullet proof and very very secure computerized systems

April 1, 2009 - Denver, Colorado, USA

Does this look familiar?

=

Class diagram of a system:

5370 Classes
55390 Relations
19430 Files

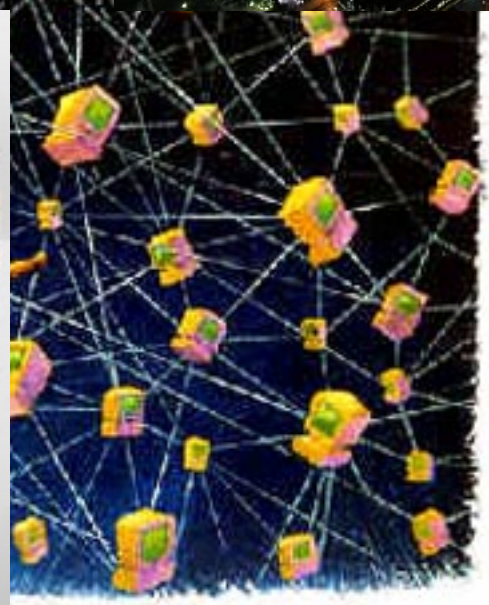
… Overall …

Size, complexity and degree of interconnection

promote software vulnerabilities

very often size does indeed matter …

We heavily depend on computerized systems



Mitre and SANS Recent Report

Out of the 25 most dangerous programming errors:

- CWE-20 - Improper input validation
- CWE-89 - Failure to preserve SQL query structure
- CWE-119 - Failure to constrain operations within the bounds of a memory buffer
- CWE-285 - Improper access control
- We have a huge quantity of software that is vulnerability prone!

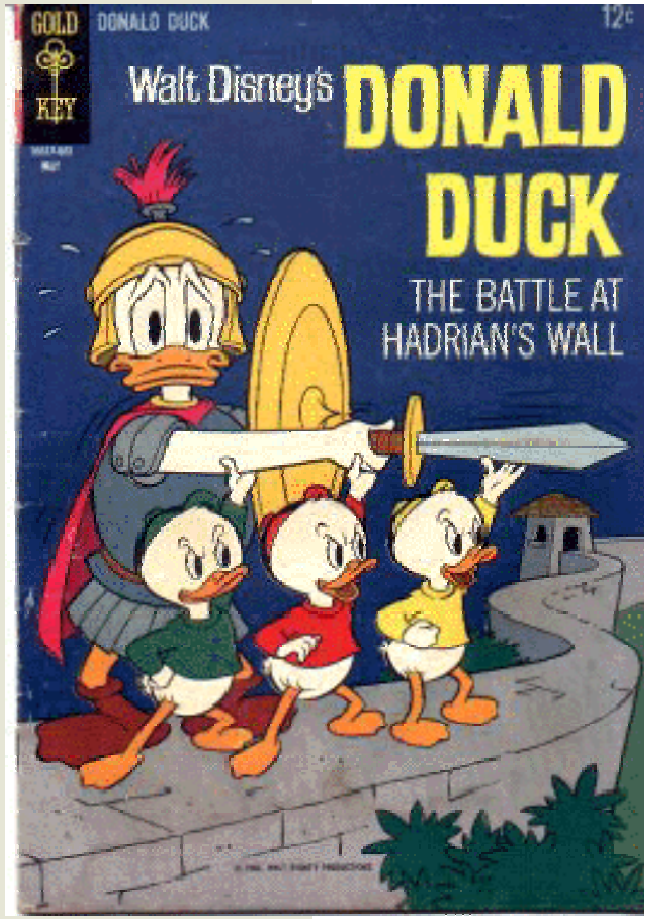
A 2008 SOCCER Lab Student Servlet

```
String username = req.getParameter("Username");
String userpass = req.getParameter("Userpass");
String query =
   new String("Select * from users where user_name = '"    + username
     + "' and user_passwd='"      + userpass + "'" );
   /* data base connection code  and  query execution */
 try {
   ...
  stmt = conn.createStatement();
  rs = stmt.executeQuery(query);
  if (rs.next())
        return true;
  ... } catch(Exception e) {
```

What the Servlet Tells us

- Defensive programming
- Security oriented design
- Awareness of risk
- Seems not be part of "that" student culture
- Developing robust software resilient to attack and vulnerability free is not part of many curricula

Testing The Defense Against Vulnerabilities



Pythia - 1400 B.C. to A.D. 381

The testing dilemma we need Pythia …

Vulnerability detection: it is often very easy to know when something goes wrong!

We may not need Pythia

A Caveat – No Vulnerability doesn't Imply Security

Security is also largely a social process.

No matter how robust and resilient a system is it may be highly vulnerable

Penetration testing attempts also to account for social engineering

ıdo, USA

SBSec Two Types of Problems

- **Difficult problems – e.g., password cracking**

- **Easy problems – e.g., buffer overflow**

We need guidance toward the most promising search regions

We need to encode domain and problem knowledge but this can be a challenging task

Password Cracking

First Password Cracking Registered Event …

Ali Baba and the

Forty Thieves

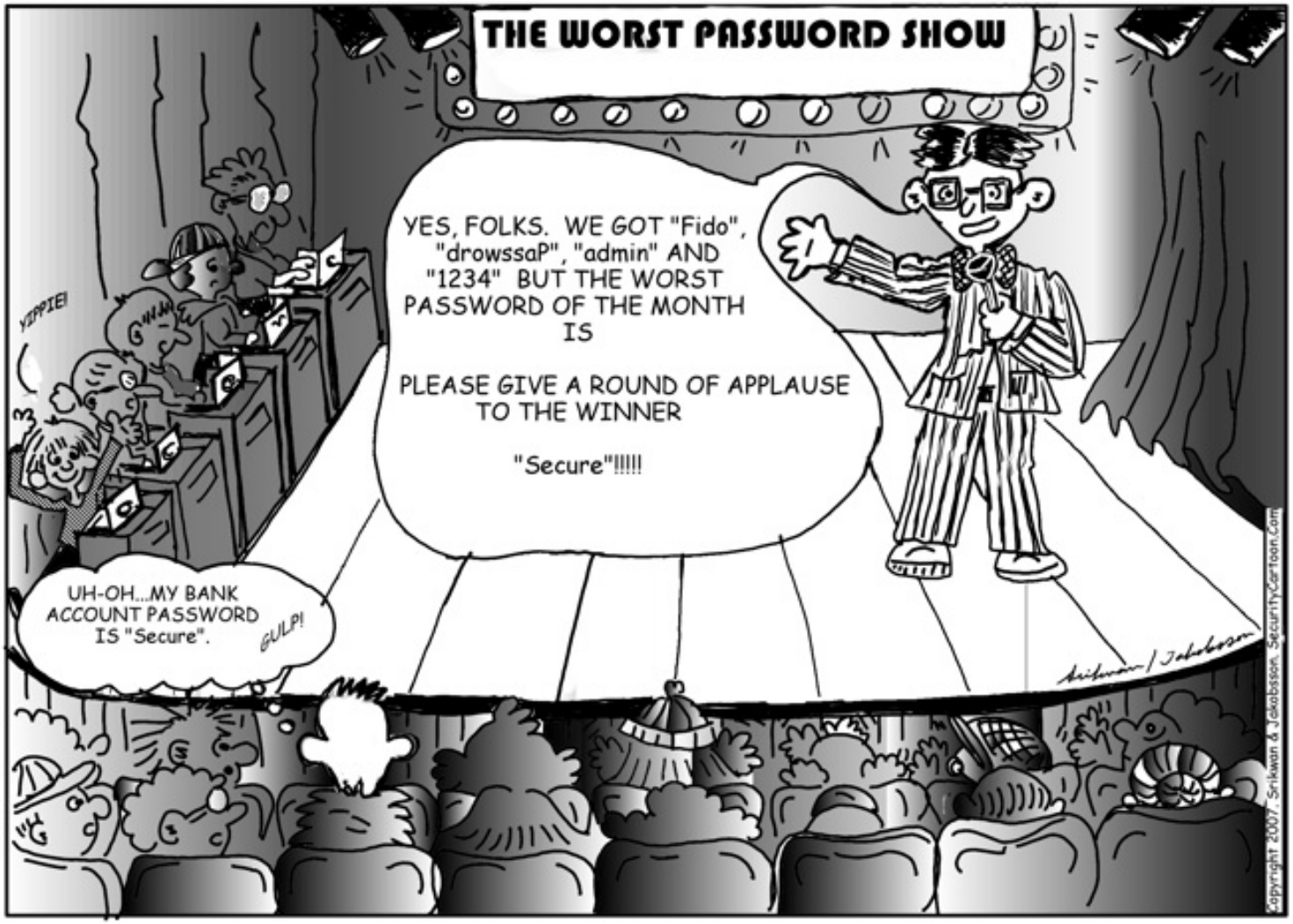
Ali Baba by Maxfield Parrish (1909).

April 1, 2009 - Denver, Colorado, USA

Unix/Linux DES Encrypted Password

Flat landscape thus transform the problem into an easier one

Password Selection is a Social Process



April 1, 2009 - Denver, Colorado, USA

We Should Chose Safe Passwords …

"What do you mean Rumpelstiltskin is too long for a password?!"

Reprinted from The Funny Times / PO Box 18530 / Cleveland Heights, OH 44118
phone: (216) 371-8600 / e-mail: ft@funnytimes.com

Customize John the Ripper

- People from a certain area, culture, company can use a set of common rules
- Encode password forming rules as the population to evolve
- Reward rules cracking more passwords
- … what else can we do … well … easier …

Go to Any Soft. Eng. Conference

POP3 protocol: username and password ….

Buffer Overflow/Overrun

Buffer Overflow/Overrun - (CWE-119, CWE-20)

- Improper user input validation
- Improper buffer boundary validation
- Static tools exist but false alarms (30% - 100%)
- SBST/SBSec generate buffer overrun reveling data

Buffer Overflow

- Unwanted condition about 50% of detected vulnerabilities
- Unsafe condition: BUFFER OVERRUN
  - crash is not required
- Implicit safety constraints

Risky instructions

Deepest statement

```
1: int main (int argc, char **argv){
2:              char buff[5];
3:              int how_many = atoi(argv[1]), p=0, w=0;
4:              strcpy(buff,argv[0]);
5:              if (argc==3){
6:                              for (p=0; p< how_many; p++)
7:                                              strcat(buff,argv[2]);
8:              } else if (argc>3){
9:                              for (p=0; p< how_many; p++)
10:                                             for (w=2; w< argc; w++)
11:                                                             strcat(buff,argv[w]);
                }
12:             printf("Buffer: %s\n",buff);
13:             exit(0);
    }
```

BASIC Fitness Function

The terms of a fitness function:

1. Statement Coverage (sc)
    1. code coverage contributes to increase the likelihood that an exception is raised
2. Vulnerable Statement Coverage(vcov)
    1. quickly drives towards buffer overflow detection
3. Max nesting(nesting)
    1. target deeper statements
4. Distance from buffer limit

$$Fitness(g) = w_1\, sc_g + w_2 \log(\mathrm{k}_g) v\,\mathrm{cov}_g + w_3\, nesting_g + w_4 \max_i \left[\min_j \left(L_{ij}(g) - SB_i\right)\right]$$

This is good

- But …
  - It really favor crashes …
  - An overrun is just enough for an unsafe programming pattern …
  - It is not amenable for complete automation ….
    - Weights are tuned by hand ....

- A minor change DYNAMIC WEIGTHS

$$Fitness(g) = w_1\, sc_g + w_2 \log(\mathrm{k_g}) v \,\mathrm{cov}_g + w_3\, nesting_g + w_4 \max_{\mathrm{i}} \left[\min_j \left(L_{ij}(g)\right)\right]$$

The Linear Problem

$$\max_{w_1,w_2,w_3,w_4} \left\{ \sum_{g \in Population} Fitness(g) \right\}$$

subject to

$$0 \le w_i \le 1 \qquad i = 1,2,3,4$$

$$w_1 + w_2 + w_3 + w_4 = 1$$

$$\forall g \in Population:$$

$$Fitness(g) \ge 0$$

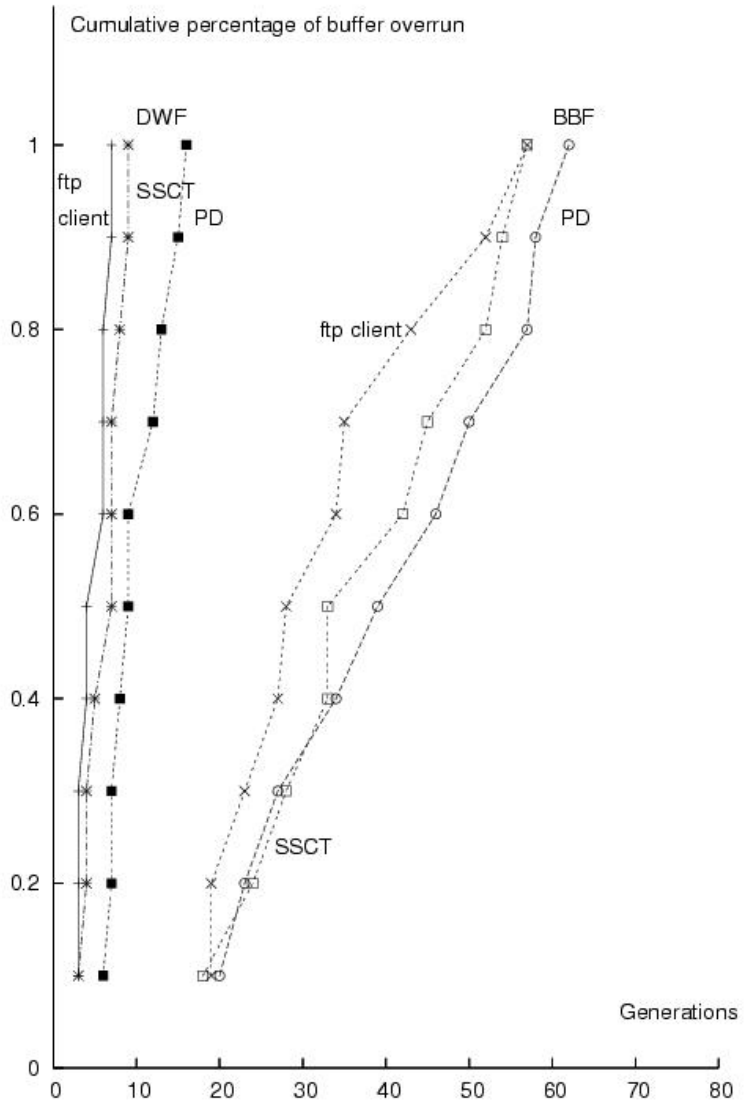
$$w_1 sc_g \le w_2 \log(k_g) v \cov_g$$

$$w_2 \log(k_g) v \cov_g \le w_3 nesting_g$$

$$w_3 nesting_g \le w_4 \max_i \{\min_j L_{i,j}(g)\}$$

Buffer Overrun Detection

- **Application:**
  - A suite of open source C functions such as FTP clients
- **Right**
  - Buffer overrun percentage versus number of generations
    - Weights tuned by hand
- **Left**
  - Completely automated system results for the fame functions



An Engineering Problem

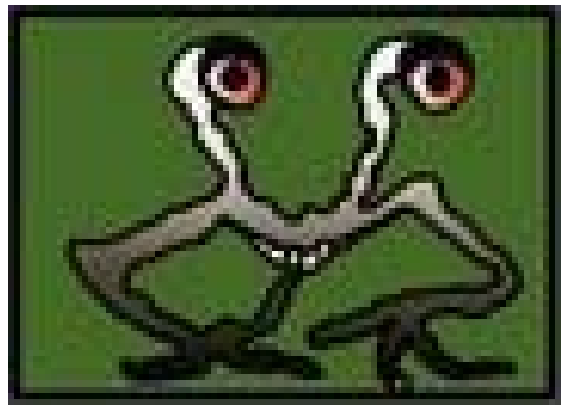
- A simple change dramatically improves problem formulation
- DWF four to five time faster
  - BBF is way simpler to implement
- BBF never takes longer than ten minutes
- However, DWF eliminates the need to manually tune parameters …
- An expert and a novice tester will obtain the same results with very similar efforts when testing the same code

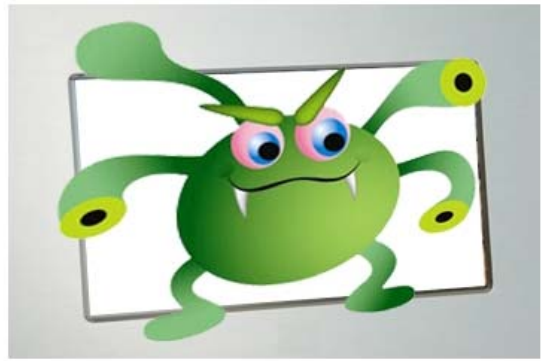
Our Big Challenge

- Reformulate vulnerability detection as SBST problems
  - Domain and problem knowledge representation
  - Landscape shape
  - Technology required to deal with software artifacts
  - Applicability to a large class of software (buffer overflow versus PDF vulnerability)

Open Challenges …

- What to break
- System wide test
- Data types
- When enough is enough
- Exception raising and singularities
- The best searching strategy
- Vulnerability testing environments
- … and many more …





First Rule – There is no rule

- We should move away from source code
- Binary is as good as source code plus sometimes no have source code is available
- Hackers use binary code instrumentation, binary debuggers, …
- Why don’t we do the same ?

What to Break

- Any executable artifact should be the target of SBSec testing
- Libraries (static and dynamic) should also be subject or our attention (CVE-2006-3459 libTIFF vulnerability)
- Applications talk via protocol test for application robustness breaking protocol rules
- Virtually no work has addressed the problem of generating test input data starting from binaries and only a few target protocols

What to Break (Cont'd)

- How do we glue together in an efficient way absolute or symbolic debuggers, instrumentation, and emulation tools ?
- Is there a real difference between working with code source or binaries ?
- A protocol can be modeled via grammars, what is the most efficient way to test applications via protocol stress testing?

SBSec and System Wide Test

- Hackers probe a system
- Ultimately it is the system that has to be resilient to attacks
- Low risk vulnerabilities in several components may result in a major threat
- Scalability is indeed an issue

System Wide Test (Cont'd)

- Civil engineering test each component
- Civil engineering also test the final artifact … well … not always in the right way …
- We are still at the component level this is good but it is not the problem …
- Tacoma bridge as a whole collapsed due to wind effect!

System Wide Test (Cont)

- What is the model of a system?
- How do we represent vulnerability goals?
- Testing each single vulnerable statement may not be viable in large systems
- How can we avoid flat landscapes?

Data Types

- Numeric data types are well understood and reasonably well handled

- How about
  - Strings
    - Encrypted information
    - Natural language
  - Pointers
  - Dynamic structures (e.g., lists, trees …)
  - Objects

Data Types (Cont'd)

- Canada has two official languages and EU joins 27 plus countries
- SQL injection works on strings following SQL grammar + user supplied information
  - User can supply info in any known language
- SQL tables names may or may not be based on English grammar

Data Types (Cont'd)

- How do we represent and manipulate linguistic knowledge?
  - Traditional grammars or context sensitive grammars?
  - Probabilistic grammars?
  - N-gram models?
- Does knowing that an application process French strings change the similarity of two strings with respect to German or Spanish?
- If we do not incorporate linguistic knowledge into our search, how better will it perform wrt. random search?

Data Types (Cont'd)

- If strings represent
  - Ciphered text
  - Pdf documents
  - SSL or 802.1x certificates
  - Any complex information
- Does it really worth investing in SBSec or will it be better to resort on developers and experts?
- Where is the engineering trade-off, the cost-benefit boundary?

When Enough is Enough

- White box or mutation testing have clear, well-understood measures of how close we are to the end
- We may resort on empirical study to bridge the gap coverage versus fault revealing power
- Statistical testing builds model to predict the number of remaining faults
- Capture recapture models estimate remaining hidden defects

When Enough is Enough (Cont'd)

- Can we provide some upper bound of vulnerabilities still there?
- Is there a way to quantify the risk of a vulnerability slipping into production code when halting test generation at any given point in time?
- After 24 hours my SBSec algorithm was running what is the improvement over the first 12 hours?

Exception Raising

- This is in-between an easy task and a very difficult one
- Re-use John Clark's idea to explicitly represent exceptions
  - division by zero, out of boundary, null pointer, ..
- Newly introduced "if(s)" goes along with exception handling conditions already there
- Now we have a branch coverage problem

Exception Raising (Cont'd)

- Condition can be very complex!
  - Can we simplify via program transformation complex conditions into sub- conditions easier to attain?

- There could be as many conditions as there are divisions, array access …
  - How can we avoid the explosion of computational time?

Exception Raising (Cont'd)

- DOS
  - Bandwidth …
  - Memory and disk space
  - Cpu time – temporal testing
- Can we reuse penetration testing strategies and what infrastructure do we need to avoid flooding our intranet?
  - In vitro studies or virtualization infrastructure?
- Should we adapt syntax based testing + robustness testing aka high load plus wrong data?

Quoting Charles Darwin

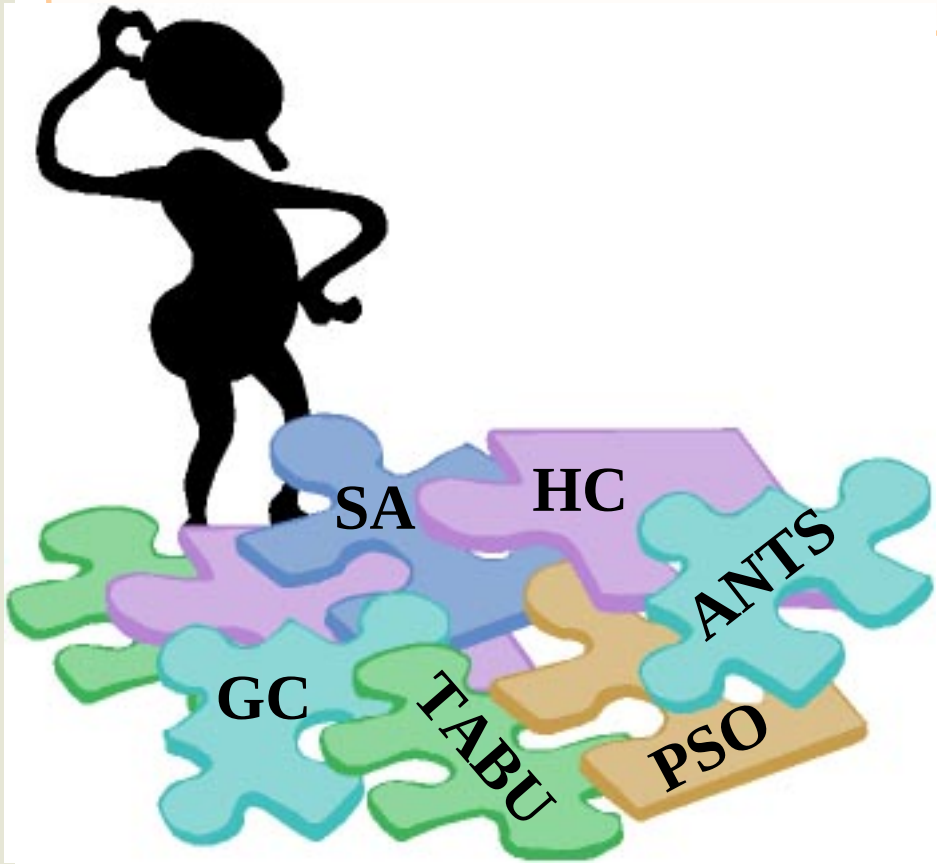
It is not the strongest of the species that survives, nor the most intelligent, but the one that is most responsive to change

April 1, 2009 - Denver, Colorado, USA

Adaptability

- Intrusion Detection Systems (IDS) help to increase security
- New types of attacks are badly managed by IDS!
- Can we conceive adaptable SBSec vulnerability detection strategies and-or develop vulnerability detection frameworks?
- Can we optimize our approaches so that they run around-the-clock to evaluate impact of changes?
  - scenario: the sysadmin tag some data, these data are reused
- Can we really make IDS rules dynamically adapted?

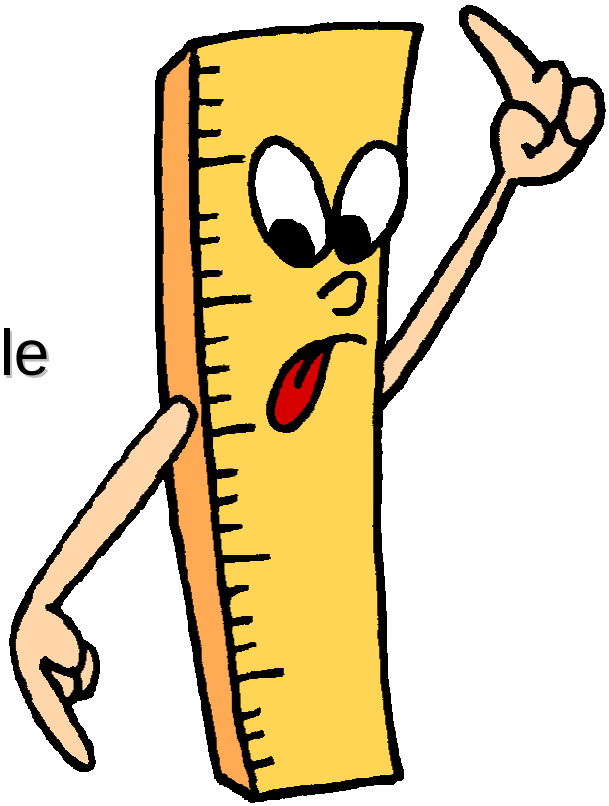
The Best Searching Strategy



Plus many other and multi-dimensional algorithm variants (e.g., NSGA/NSGA II)

A Common Yardstick

- A common empirical framework
- Methodology, approaches, measures
- Data sets created by test people
  - space.c or similar program
  - SIR repository
- MIT Lincoln LAB



Benchmarks … Requirements

- Support for multiple types of vulnerabilities
- Independence of methodology
- Ground Truth
- Scalability testing

Environments and Eclipse

- Why don’t we use existing platforms and frameworks to implement and distribute our tools?
- Eclipse can help implementing and deploying prototypes
- Java metaheuristic libraries are available or not difficult to code

And … Don't Forget Your Mom …

If you're not familiar with The Mom Test, it's basically a quick way to see if your site is so simple that your mom can use it.

A variant that can also reveal results fast for you is the "Five second usability test", which is kind of similar, but gives you other results.

[Jesper Rønn-Jensen, front-end Web developer, usability specialist at Capgemini Denmark]

We Have to Remember …

… that managers, programmers are not researchers …

… when I was a manager my time was planned and spent differently … my time was money ...

… no one will invest time if the benefits are unclear and substantially greater of the costs …

Conclusion

- SBST can play a major role
  - See the European projects
- Important contributions have been published
  - See the CREST Database
- Much more can be foresee
- There is an huge amount of software that need to be tested against vulnerabilities
- As customers of computerized system, we rely on them as society and also end users
  - Firewall, IDS, operating systems, e-mail
- SBSec can make the difference