From Penetrate and Patch to Building Security In

Michael Hicks
Professor of Computer Science
and the UofM Institute for Advanced Computer Studies (UMIACS)

Distinguished Scholar-Teacher talk
September 28, 2015

PL@UM
PROGRAMMING LANGUAGES UNIVERSITY OF MARYLAND

MARYLAND CYBERSECURITY CENTER

Security breaches

Just a few:

- **TJX** (2007) - 94 million records*
- **Adobe** (2013) - 150 million records, 38 million users
- **eBay** (2014) - 145 million records
- **Anthem** (2014) - Records of 80 million customers
- **Target** (2013) - 110 million records
- **Heartland** (2008) - 160 million records

**containing SSNs, credit card nums, other private info*

https://www.oneid.com/7-biggest-security-breaches-of-the-past-decade-2/

Defects and Vulnerabilities

- Many (if not all of) these breaches begin by exploiting a **vulnerability**

- This is a *security-relevant* **software defect** (bug) or **design flaw** that can be **exploited** to effect an undesired behavior

- The **use of software is growing**
  - So: more bugs and flaws
  - Especially in places that are new to using software

Google
2B LOC

Windows
50M LOC

…

…

SECTIONS HOME SEARCH The New York Times

MIDDLE EAST

Iran Fights Malware Attacking Computers

By DAVID E. SANGER SEPT. 25, 2010

Email

Share

Tweet

Save

More

WASHINGTON — The Iranian government agency that runs the country's nuclear facilities, including those the West suspects are part of a weapons program, has reported that its engineers are trying to protect their facilities from a sophisticated computer worm that has infected industrial plants across Iran.

The agency, the Atomic Energy Organization, did not specify whether the worm had already infected any of its nuclear facilities, including Natanz, the underground enrichment site that for several years has been a main target of American and Israeli covert programs.

But the announcement raised suspicions, and new questions, about the origins and target of the worm, Stuxnet, which computer experts say is a far cry from common computer malware that has affected the Internet for years. A worm is a self-replicating malware computer program. A virus is malware that infects its target by attaching itself to programs or documents.

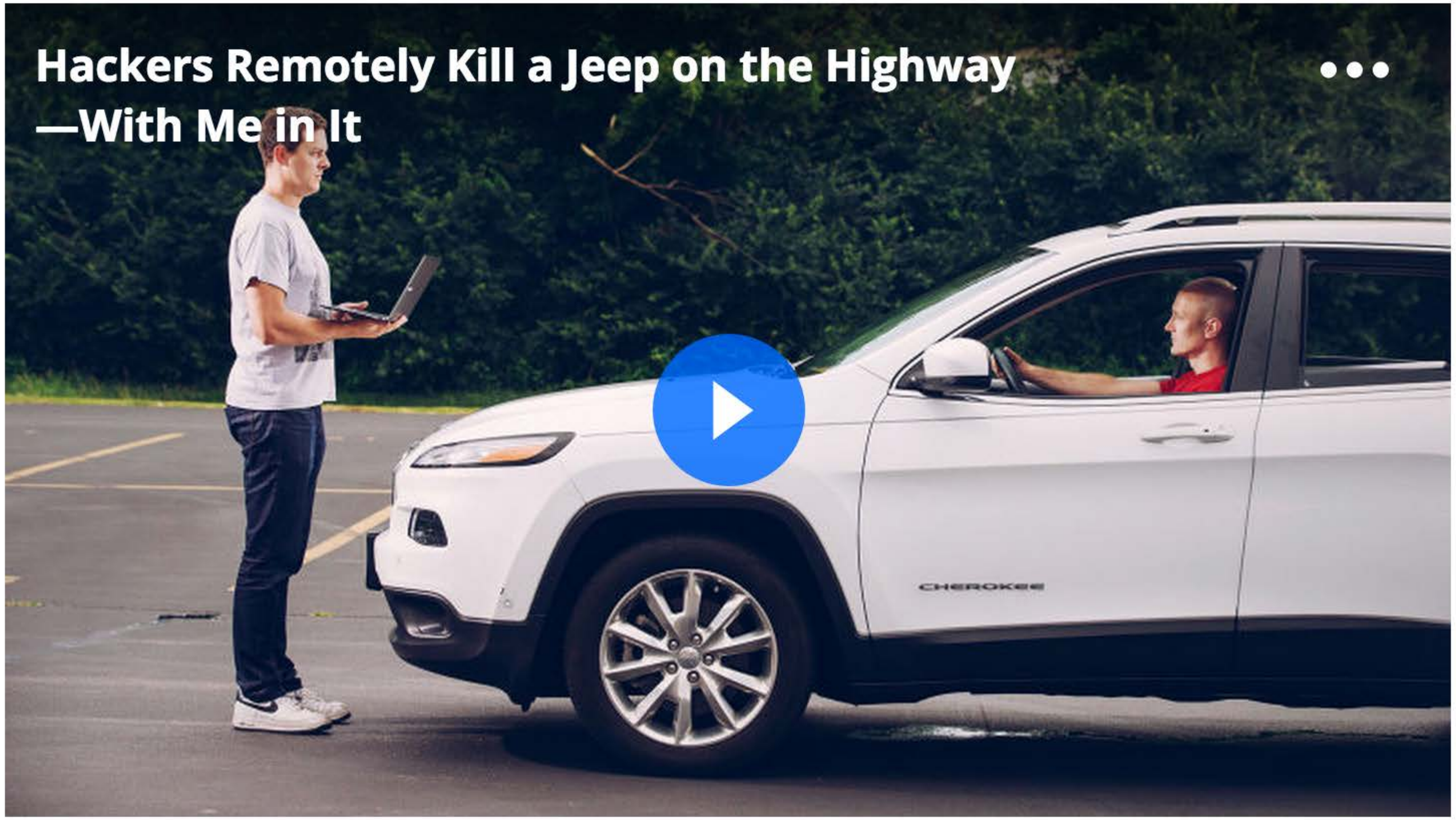
Stuxnet specifically targets … processes such as those used to control … **centrifuges for separating nuclear material**. Exploiting four zero-day flaws, **Stuxnet** functions by targeting machines using the Microsoft Windows operating system …, then seeking out Siemens Step7 software.

http://www.nytimes.com/2010/09/26/world/middleeast/26iran.html

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY 07.21.15 6:00 AM

HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY—WITH ME IN IT



I WAS DRIVING 70 mph on the edge of downtown St. Louis when the exploit began

The result of their work was a hacking technique—what the security industry calls a zero-day exploit—that can **target Jeep Cherokees and give the attacker wireless control**, via the Internet, to any of thousands of vehicles.

http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/

Considering **Correctness**

- **All software is buggy**, isn't it? Why not a problem from way back?
- A **normal user never sees most bugs**, or figures out how to **work around** them
- Therefore, **companies fix the most likely bugs**, to save money

Considering **Security**

Key difference:

An attacker is not a normal user!

- The attacker **will actively attempt to find defects**, using unusual interactions and features
  - A typical interaction with a bug results in a **crash**
  - An attacker will work to **exploit** the bug to do **much worse**, to achieve his goals

Cyber-defense?

Symantec

McAfee
An Intel Company

FireEye™
Next Generation Threat Protection

CISCO™

KASPERSKY lab

tenable®
network security

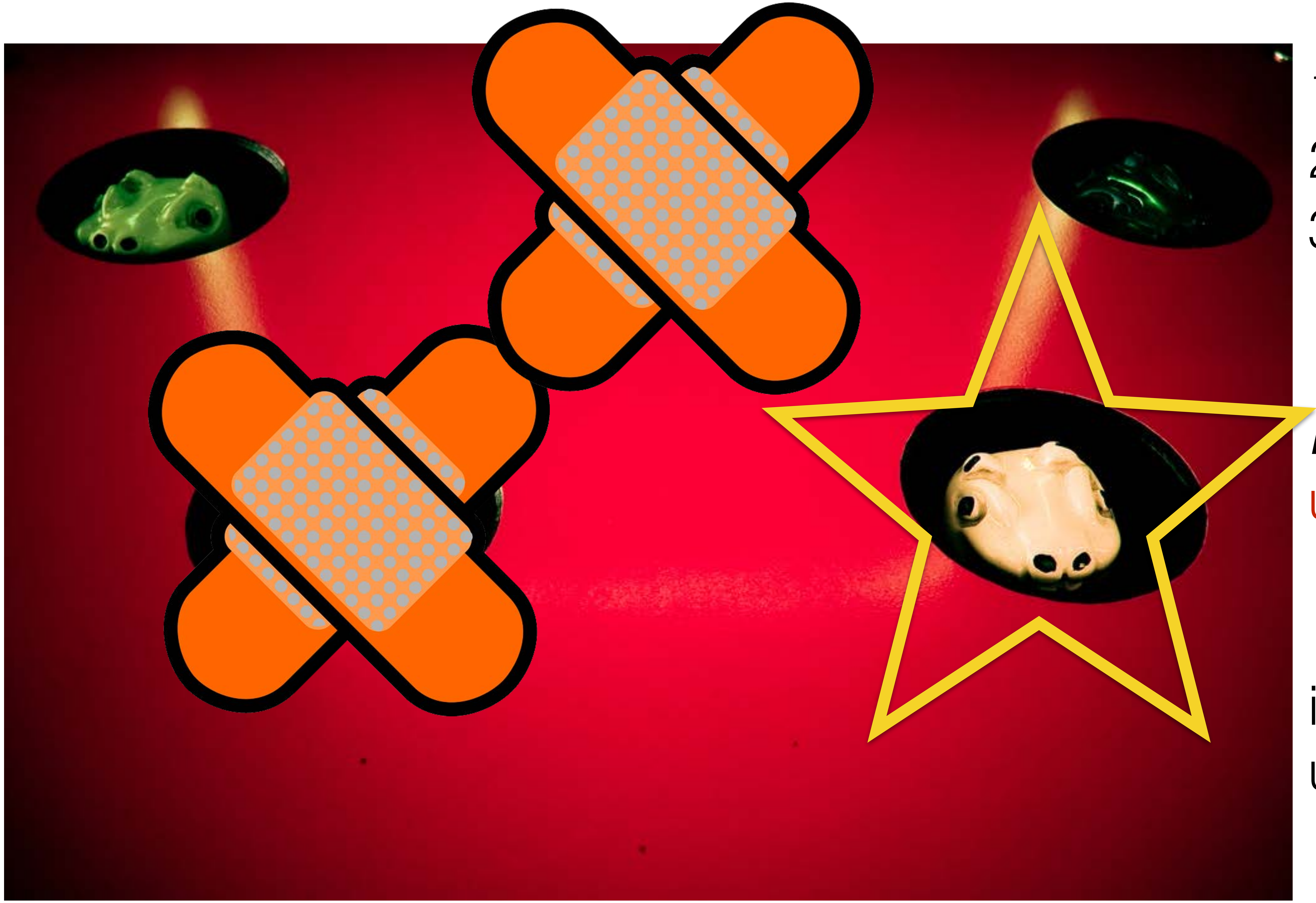
SNORT®

Cyber-defense?

Popular technologies such as **firewalls**, **anti-virus**, and **intrusion detection/prevention**, attempt to detect the attacks themselves.

But **new attacks** can be produced that avoid detection but **exploit the same vulnerabilities**

Penetrate and Patch



1. Find a vulnerability
2. Develop patch
3. Deploy patch (and detection signature)

But: Still vulnerable to undiscovered bugs

… and new bugs introduced by software upgrades

FireEye, Kaspersky hit with zero-day flaw claims

Researchers have disclosed severe security flaws within the firm's products over the holiday weekend.

By Charlie Osborne for Zero Day | September 8, 2015 -- 09:45 GMT (02:45 PDT) | Topic: Security

Researchers have revealed the existence of zero-day vulnerabilities within Kaspersky and FireEye's systems which could compromise customer safety.

Over the holiday weekend, security researcher Tavis Ormandy disclosed the existence of a vulnerability which impacts on Kaspersky products. Ormandy, known in the past for publicly revealing security flaws in Sophos and ESET antivirus products, said the vulnerability is "about as bad as it gets." In a tweet, the researcher said:

and **bugs in security products** themselves!

Security researcher Tavis Ormandy disclosed the existence of **a vulnerability which impacts on Kaspersky [security] products**.

Hermansen, [another researcher,] publicly disclosed a zero-day **vulnerability within cyberforensics firm FireEye's security product**, complete with proof-of-concept code.

http://www.zdnet.com/article/fireeye-kaspersky-hit-with-zero-day-flaw-claims/

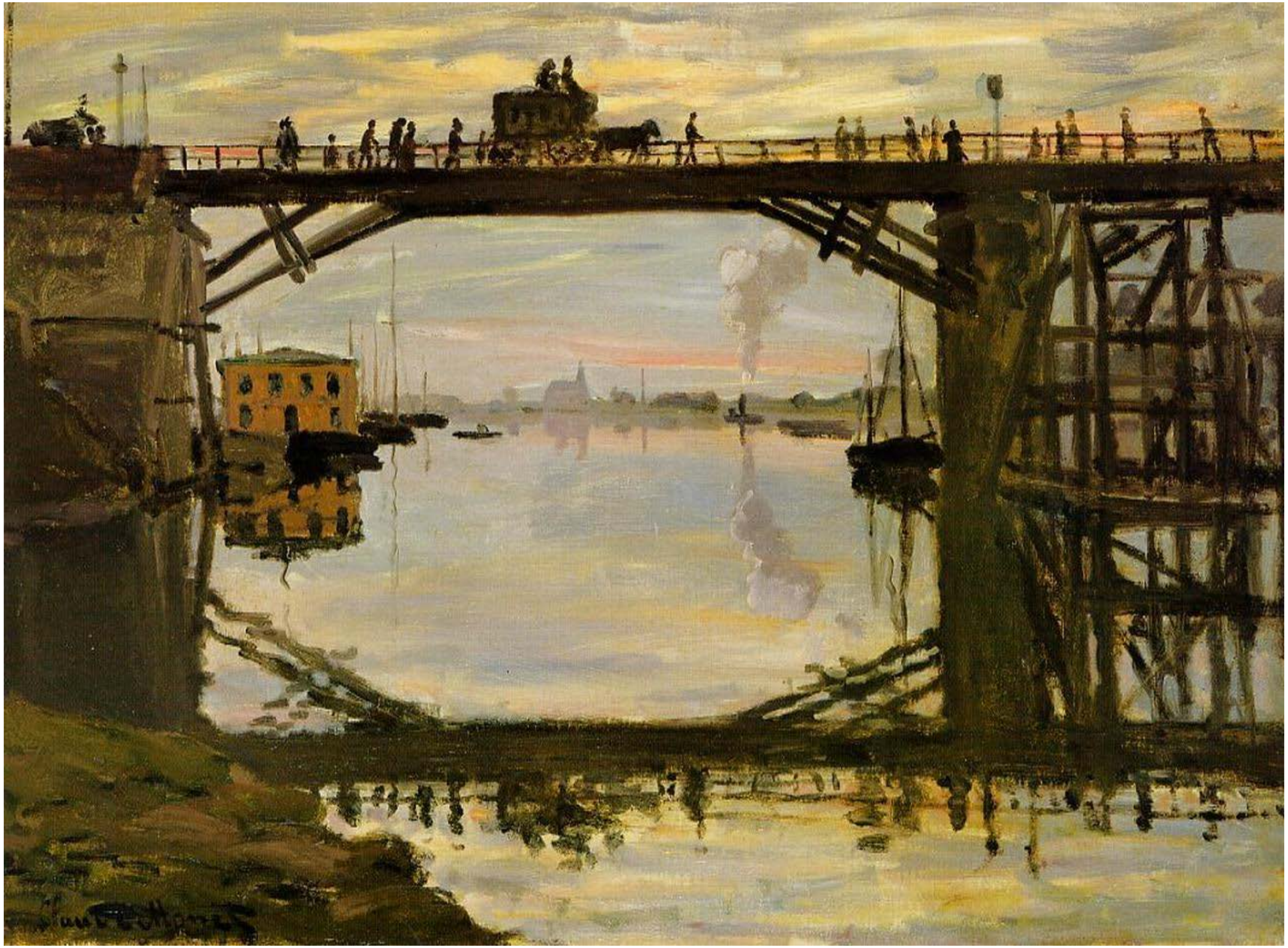
Building Security In

The long-term solution is to **prevent** all exploitable **bugs before deploying**

Avoid the holes to start with!

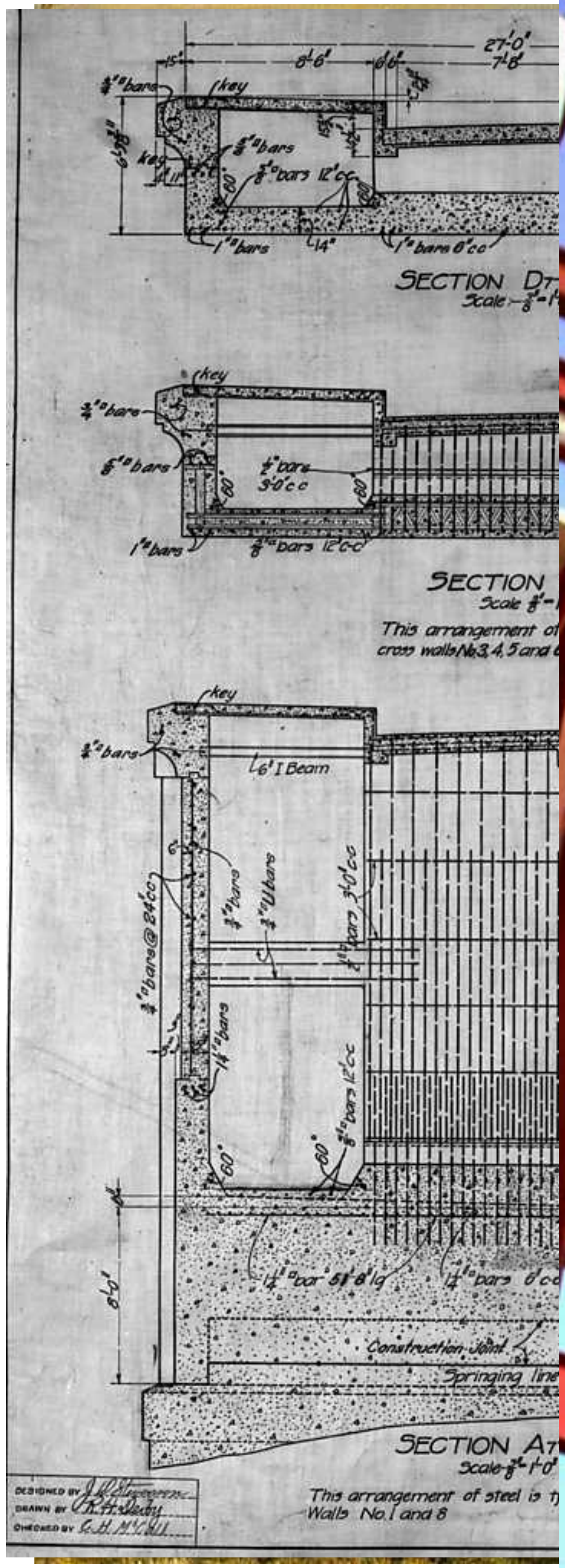
Analogy

- How do you **build a bridge** that stands up despite harsh conditions?
  - Heavy use
  - Earthquakes
  - Extreme weather
  - Etc.



Analogy

- Study the problem. Develop **the best**
  - **Methods**
  - **Materials**
  - **Tools**
- Then use them from Day 1!



Do not

- Use methods that **fail to incorporate larger lessons** (i.e., from past bridges built and past failures)
- **Use cheap materials** that are unresilient
- Use **unreliable tools** that produce inconsistent results
- Assume that you can do these things and everything will be OK (you can **just patch problems later**)

Unless you want your bridge to fail

Building Security In

- What about software?

Building Security In

- What about software?

Same idea: Security from Day 1

- Consider it in your **design**
- Use the **best tools and methods**
  - Best **programming languages**
  - Best **program development environment**
  - Best **testing and verification** methods

Building Security In

Why not done already?

- Ignorance
- Unproven/insufficient technology
- Concerns about cost
  - to change legacy programs
  - to (re)train staff in new process, technology, etc.

Some of my work

- Eliminating vulnerabilities at the outset with **better languages** and testing tools
    - Highlight: **Cyclone**: A safer “low level” programming language
- Focusing attention on building, not breaking
    - **Coursera on-line course** on software security
    - Build-it, Break-it, Fix-it **programming contest**

CYCLONE

coursera

BUILD BREAK FIX IT

From bugs to exploits

Software

- Software consists of **instructions** that tell a computer what to do
    - A **program** is a set of instructions to achieve a particular task
- Instructions are kept within the computer's **memory** when executed by the **processor**

Processor (CPU)

Data and Instructions

Memory (RAM)

Computing $R = X^Y$

- Goal: multiply X by itself a total of Y times

- Program: **R will contain the final result**
  - Use a **counter C** to track of the number of multiplications
  - Like counting on your fingers!

Computing $R = X^Y$

<u>Data</u>

$X = \boxed{3}$

$Y = \boxed{2}$

$C = \boxed{}$

$R = \boxed{}$

<u>Instructions</u>

Set R to 1
Set C to Y
Is $C \leq 0$ *?*
  If so, skip to the end

Set R to $X \cdot R$
Set C to C - 1
If $C > 0$ *repeat the above two instructions*

Computing $R = X^Y$

Data

$X = \boxed{3}$

$Y = \boxed{2}$

$C = \boxed{2}$

$R = \boxed{1}$

Instructions

Set R to 1
Set C to Y
Is C ≤ 0 ?
If so, skip to the end

Set R to X · R
Set C to C - 1
If C > 0 repeat the above two instructions

Computing $R = X^Y$

Data		Instructions
$X =$	3	*Set R to 1*
$Y =$	2	*Set C to Y*
$C =$	1	*Is $C \leq 0$?*
$R =$	3	*If so, skip to the end*

Set R to $X \cdot R$

Set C to C - 1

If $C > 0$ repeat the above two instructions

Computing $R = X^Y$

Data

$X =$ 3

$Y =$ 2

$C =$ 0

$R =$ 9

Instructions

Set R to 1
Set C to Y
Is C ≤ 0 ?
If so, skip to the end

Set R to X · R
Set C to C - 1
If C > 0 repeat the above two instructions
Done

Computing $R = X^Y$

```
exp:
      movl    $1, %eax        Set R to 1
      testl   %esi, %esi      Set C to Y
      jle     .L3             Is C ≤ 0 ?
.L6:                            If so, skip to the end
      imull   %edi, %eax
      subl    $1, %esi        Set R to X · R
      jne     .L6             Set C to C - 1
.L3:                          If C > 0 repeat the above two instructions
```

machine instructions

%edi = contains base value X
%esi = contains exponent Y and counter C
%eax = contains result R

Programming Languages

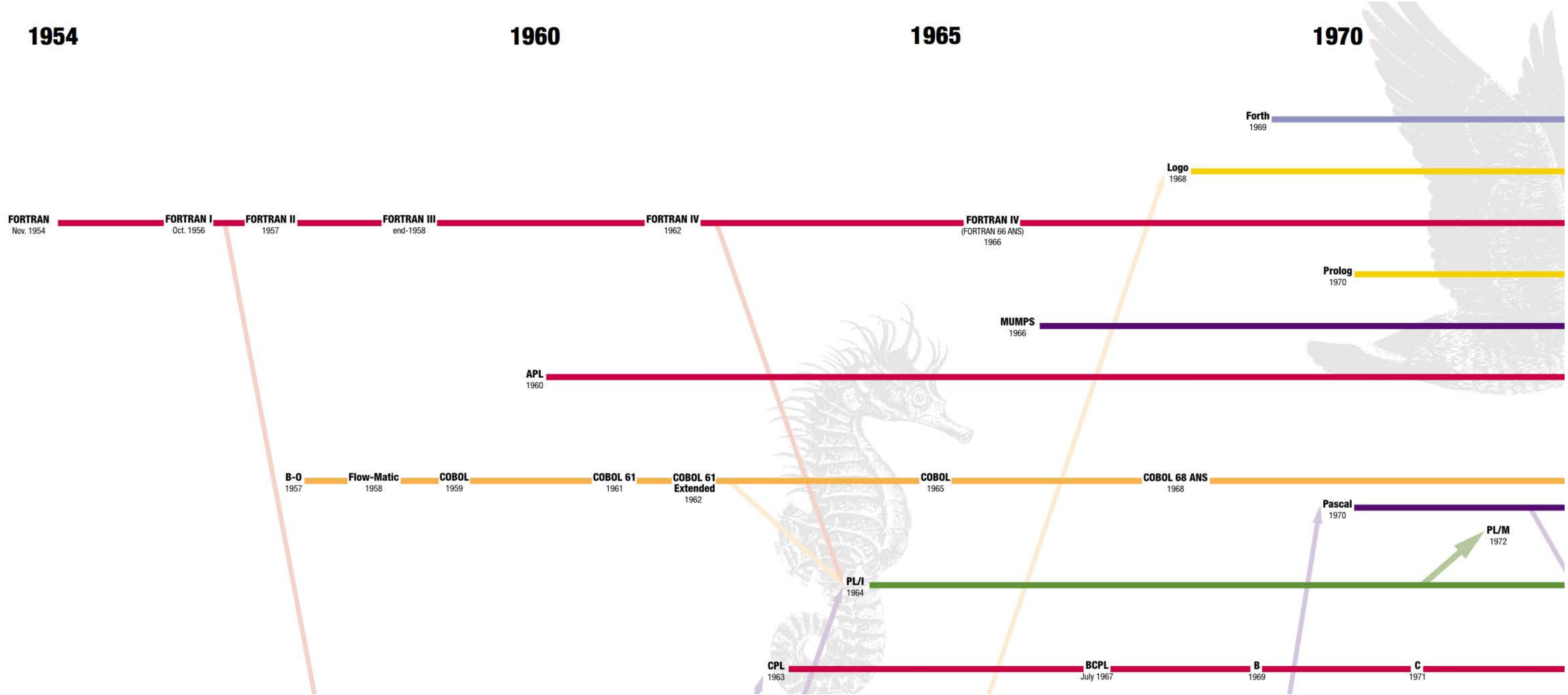
- **Many machine instructions for simple programs** - hard for humans to understand and maintain!

- **Programming languages** designed to **help**
  - *Higher level* - Closer to human language
  - First ones (e.g., FORTRAN) in the 1950's

- Programs are **translated** (aka ***compiled***) into machine instructions to be executed by the processor

- **Many languages developed in the last 60 years!**
  - Different languages have different strengths

Programming Languages



Programming Languages

Programming Languages

What is popular today?

Language Rank		Types	Spectrum Ranking
1.	Java		100.0
2.	C		99.2
3.	C++		95.5
4.	Python		93.4
5.	C#		92.2
6.	PHP		84.6
7.	Javascript		84.3
8.	Ruby		78.6
9.	R		74.0
10.	MATLAB		72.6

http://spectrum.ieee.org/static/interactive-the-top-programming-languages

Our program in the C language

```c
int exp(int x, int y) {
  int r = 1;
  while (y > 0) {
    r = r * x;
    y = y - 1;
  }
  return r;
}
```

In Java it would look much the same,
but that's not true in general

Our program in the **Python** language

```
def exp(x, y):
  r = 1
  while y > 0:
    r = r * x
    y = y - 1
  return r
```

Our program in the **OCaml** language

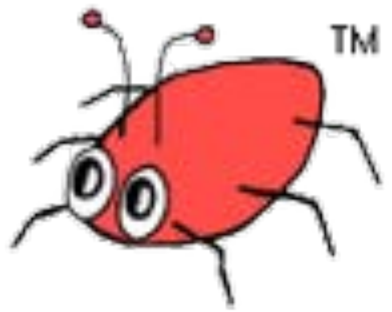
```
let rec exp x y =
  if y = 0 then
    1
  else
    x * exp x (y-1)
```

Our program in the **Prolog** language

```
exp(X,0,1) :- !.
exp(X,Y,R) :-
    Y1 is Y-1,
    exp(X,Y1,R1),
    R is X * R1.
```

Software flaws and defects

- Programmers make mistakes
- So software often has **defects** (aka **bugs**)




```
int exp(int x, int y) {
  int r = 1;
  while (y ≥ 0) {
    r = r * x;
    y = y - 1;
  }
  return r;
}
```

should be “greater than”
not “greater than or equal to”

Exploitable bugs

- Some **bugs** can be **exploited**
  - An attacker can control how the program runs so that any incorrect behavior serves the attacker

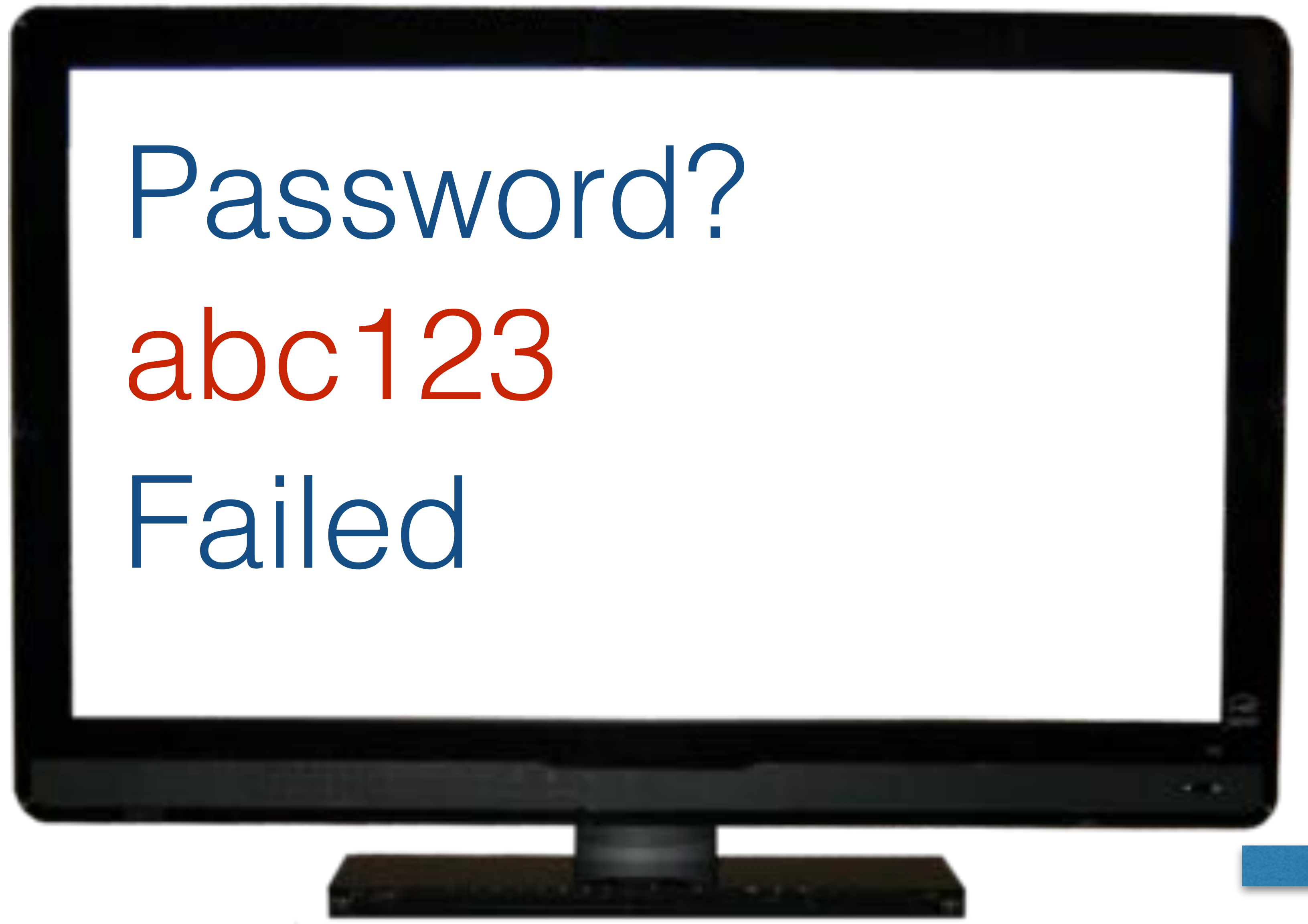
- **Many kinds of exploits** have been developed over time, with technical names like
  - **Buffer overflow**
  - Use after free
  - SQL injection
  - Command injection
  - Cross-site scripting
  - Cross-site request forgery
  - …

What is a buffer overflow?

- A buffer overflow is a dangerous bug that affects programs written in **C** and **C++**
- **Normally**, a program with this bug will simply **crash**
- But an **attacker** can alter the situations that cause the program to **do much worse**
  - **Steal** private information
  - **Corrupt** valuable information
  - **Run code** of the attacker's choice

Buffer overflows from 10,000 ft

- **Buffer** =
  - Block of memory associated with a variable
- **Overflow** =
  - Put more into the buffer than it can hold
- **Where does the overflowing data go?**



Normal interaction

Instructions

1. print “Password?” to the screen
2. read input into variable X
3. if X matches the password then log in
4. else print “Failed” to the screen

Data

$X = \boxed{\text{abc123}}$

Exploitation

Instructions

1. print “Password?” to the screen
2. read input into variable X
3. log in
4. else print “Failed” to the screen

Data

X = Overflow!!!!! 3.log in

Key idea

- The key feature of the buffer overflow attack is the attacker getting the application to **treat attacker-provided data as instructions (code)**

- This feature appears in many **other exploits** too

  - SQL injection treats data as **database queries**
  - Cross-site scripting treats data as **browser commands**
  - Command injection treats data as **operating system commands**
  - Etc.

Building security in

Stopping the attack

- **Buffer overflows** rely on the ability to **read or write outside the bounds of a buffer**

- **C and C++** programs expect the **programmer** to ensure this never happens
  - But humans (regularly) make mistakes!

- Other languages (like **Python, OCaml, Java**, etc.) ensure buffer sizes are respected
  - The **compiler** inserts checks at reads/writes
  - Such checks can halt the program
  - But will **prevent a bug from being exploited**

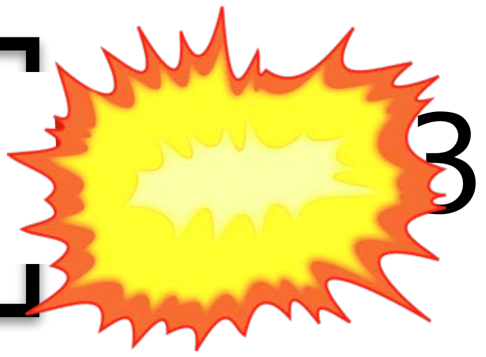
Preventing Exploitation

Instructions

1. print “Password?” to the screen
2. read input into variable X
3. if X matches the password then log in
4. else print “Failed” to the screen

Data

X = Overflow!



Program halted

So why use C and C++?

- Billions of lines of **existing C programs**
- Programmers are very **familiar** with C
- C gives you **fine control** over hardware resources
  - Very efficient
  - Great for writing “low level” programs

- **Best current advice**: Use other languages whenever you can, and use C and C++ when you must

- **Research** question: ***Can we do better?***

My Research

- **Cyclone** is a language with the **efficiency and control** of C but the **safety** of modern languages

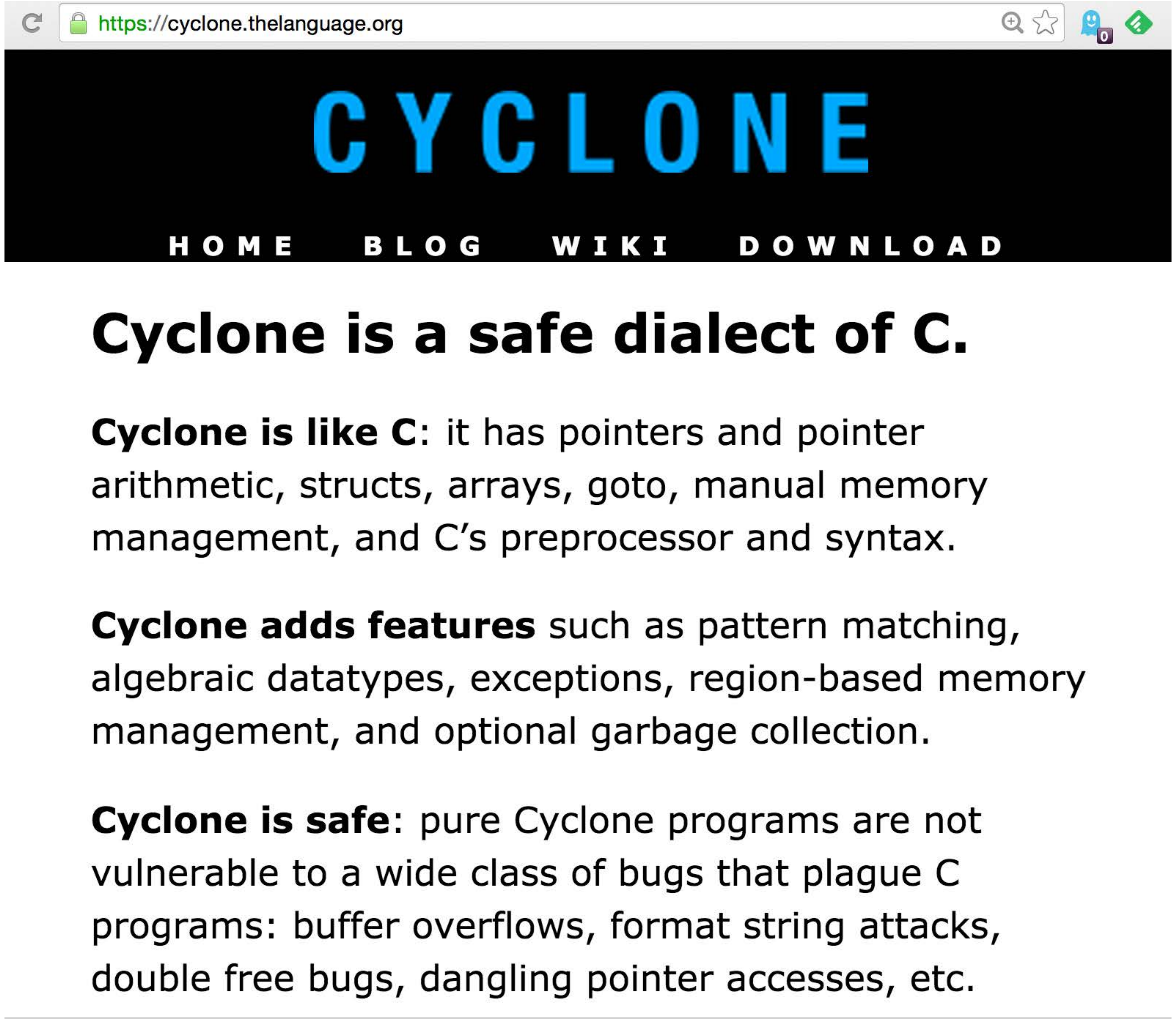
- Developed 2001 - 2006 in collaboration with researchers at Cornell, Harvard, Washington, and AT&T Labs Research

- Several contemporary efforts

ccured

SoftBound

CETS

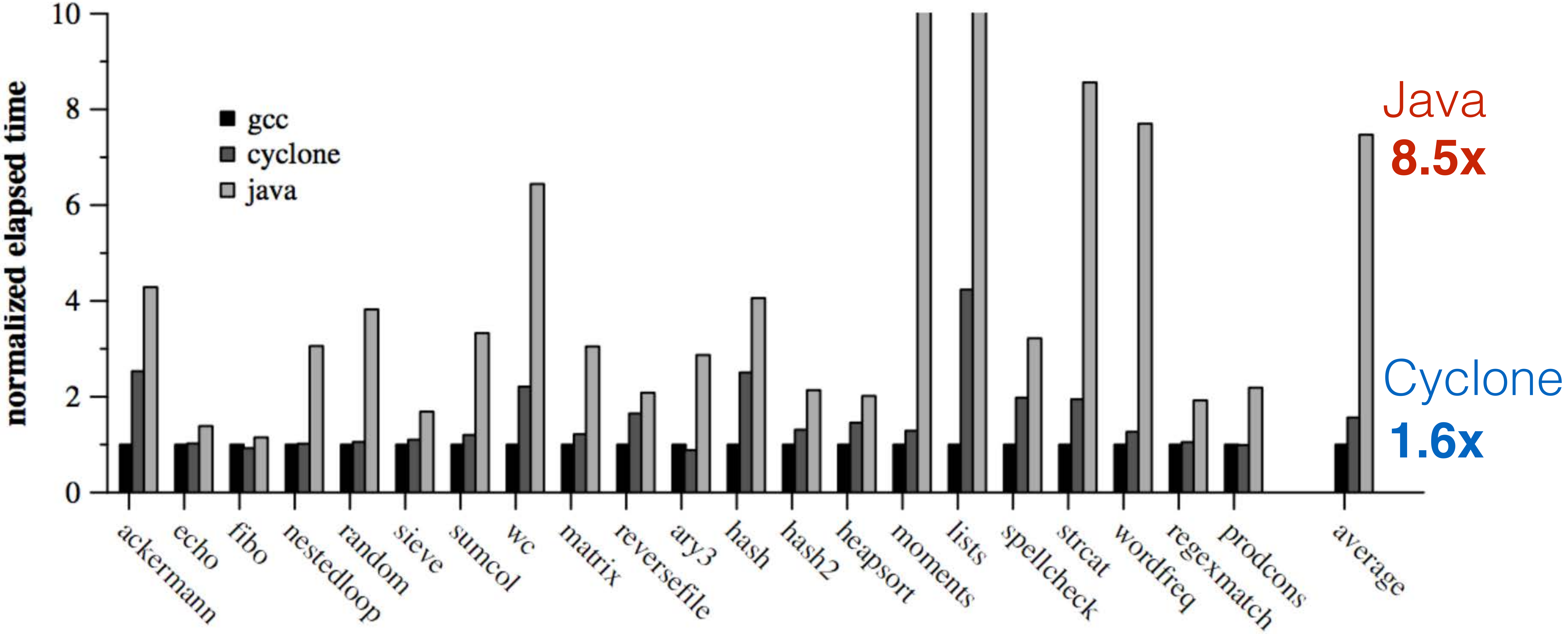


Science of language design

How do we know if Cyclone meets its goals?

- Formalize it mathematically, and **prove that its programs are secure**
- Show that it can be used to **write useful programs**
  - Choose them from relevant benchmarks and domains
  - And attempt to measure the difficulty of writing these programs
- Show that Cyclone **programs perform well**

Performance comparison



Translated the C programs to Cyclone; changed only 5-15% of the program

Performance comparison

Test	C		Cyclone GC		Cyclone Manual	
	Time	Mem	Time	Mem	Time	Mem
Epic	0.70	12.5M	1.11 (1.61)	22.3M (1.78)	1.11 (1.61)	12.5M (1.0)
KissFFT	1.33	394K	1.40 (1.05)	708K (1.80)	1.41 (1.06)	392K (0.99)
Betaftpd	4.00	6.2K	4.00 (1.0)	192K (30.1)	4.00 (1.0)	8.2K (1.32)
Cfrac	8.75	284K	15.23 (1.74)	1.44M (5.19)	14.53 (1.66)	706K (2.49)
8139too	334	27.7K			333(0.99)	31.8K (1.14)

- Programmers can **tune performance while retaining safety**
- **Space usage is much closer to C**'s when using these features (and far better than typical modern languages)

Takeaway

Cyclone addresses several of the reasons people use inadequate methods:

- Ignorance
- **Unproven/insufficient technology**
- Concerns about cost
  - to change **legacy code**
  - to (re)**train staff**

- By staying close to C, Cyclone provides a path from legacy code to something safer, while addressing technical and non-technical concerns

Impact

- Cyclone was a research language - its influence (and that of related efforts) is on modern language and system design.

- The **Rust language** from Mozilla borrows many of the memory management features from Cyclone

  https://www.rust-lang.org/

- Coming soon:
  - **Intel MPX** hardware: support to make checking faster

    https://software.intel.com/en-us/blogs/2013/07/22/intel-memory-protection-extensions-intel-mpx-support-in-the-gnu-toolchain

  - **Safe C extension** to LLVM, being developed by **Microsoft Research**

Engendering and Evaluating the Build-it Mentality

Cybersecurity: White hat, Black Hat

Build it

- **Design** and **implement** computer systems in a way that **prevents** security defects

Break it

- **Find** defects that constitute **vulnerabilities** and **exploit** them

Black Hat

Problem**: Too much emphasis on **breaking**, not **building

Break it

- **Find** defects that constitute **vulnerabilities** and **exploit** them

DEFCON CTF, Collegiate Cyber defense challenge (CCDC), Pwn to Own, …

Our proposed remedy

BUILD
BREAK
FIX
IT

(BIBIFI)

A new kind of security contest:
rewards breaking **and building**

Scoring System

- **Build-it Score**
  - **Gains** points for **good performance**
  - **Gains** points for implementing **optional features**
  - **Loses** points for *unique* **bugs found**
    - More points for (obviously) security-relevant bugs
    - Fixing bugs helps show that multiple test cases might be tickling the same bug, thus reducing the penalty for those test cases
- **Break-it Score**
  - **Gains** points for **unique bugs** found (scaled by how many other teams found the same bug)
- **Winners in both categories**

Educational Experiment

- This contest aims to educate its participants, but it has a broader agenda too

Show what works!

- Many **ideas** for **improving computer security**
  - But few of these have been put to a **scientific test**

- This **contest** sets up an **experiment**
  - Independent variables are the **choices you make** when you develop, or when you hunt for bugs
  - The dependent variable is the **final outcome**
  - Science: <u>**Which choices correlate with success?**</u>

May-June 2015 Contest

- **98 registered teams**
  - Teams ranged in size from 1-5 (median 2)
- **79 teams made a build-it submission**
  - **62** teams' submissions **qualified**
- **66 teams made a break-it submission**
  - 9128 non-unique correctness bugs
  - 36 unique confidentiality bugs
  - 40 unique integrity bugs

Build-it Winners




1st prize: Team **JavaTheHut**

Break-it Winners

1st prize: Team **Black_Horse**

2nd prize: Team **Tosca**

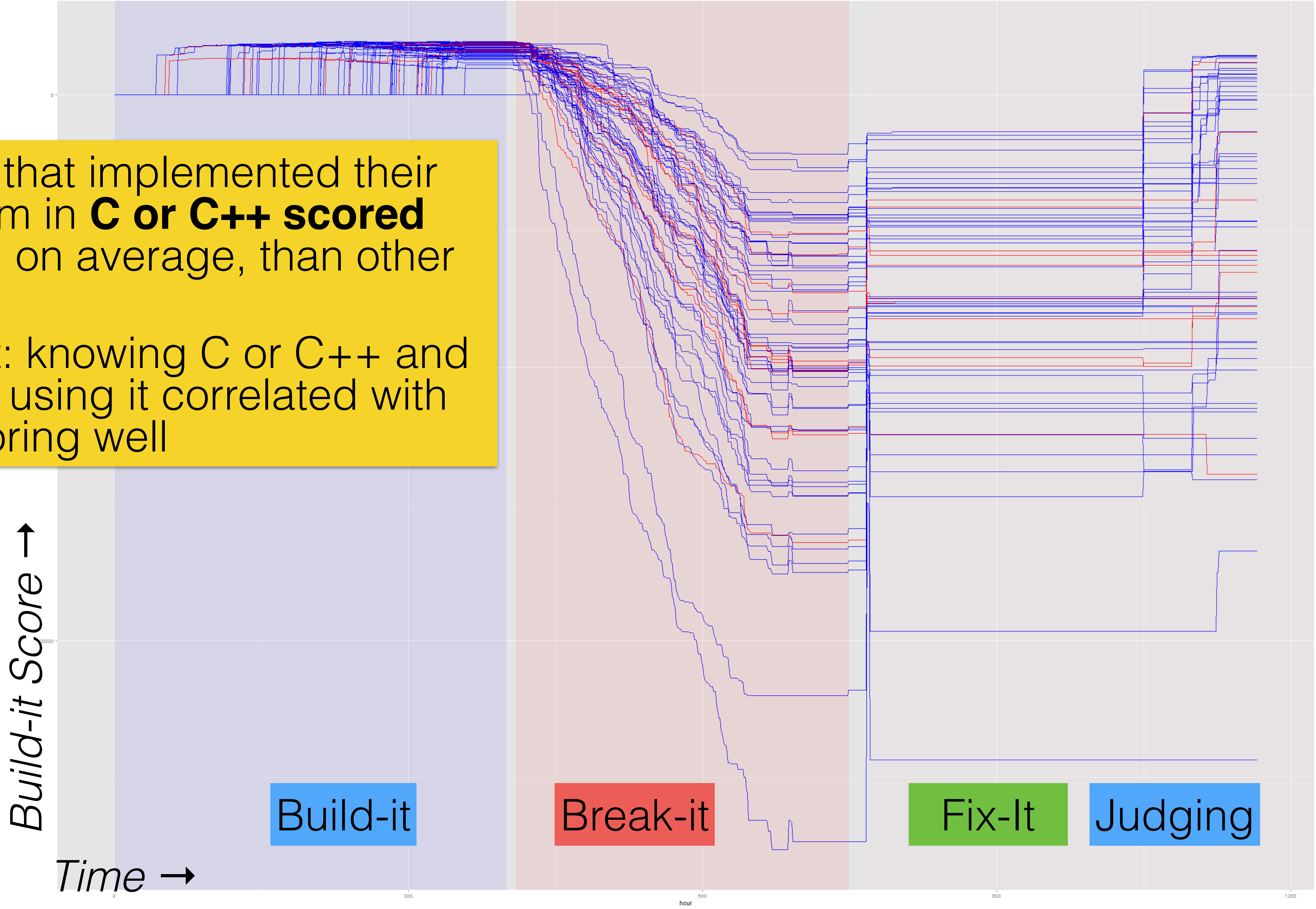
Language choices

- **Many languages** used
  - C, C++
  - C#, Java, Scala
  - Python, Perl
  - Bash
  - Javascript
  - Visual Basic
  - F#, OCaml
  - PHP

- **Python most popular**, followed by **Java**, **C**, **C++**
  - Seems to follow general popularity trends
  - Winners used Java

Teams that implemented their program in **C or C++ scored worse**, on average, than other teams

- But: knowing C or C++ and not using it correlated with scoring well



Contest promise

Recall the reasons people use inadequate methods, once again:

- **Ignorance**
- **Unproven/**insufficient **technology**
- **Concerns about cost**
  - to change legacy programs
  - to (re)train staff

- BIBIFI hopes to educate students, and provide evidence for what works
  - More data gathering and analysis in progress

Outreach and Education

PL Research

- My efforts occur within a broad research community considering how **programming languages (PL)** can improve the quality of software

- How? By developing
  - Novel **programming languages** or **constructs**
  - Advanced **programming tools** and **techniques**
  - **Mathematical methods** for understanding software
    - To **prove** that it satisfies desirable security properties
  - And more …

- Lots of really fantastic work happening

PL @UM

PROGRAMMING LANGUAGES UNIVERSITY OF MARYLAND

Blogging

- In June 2014 I started blogging about the great work being done in programming languages
  - Tutorials, interviews, cross-disciplinary connections, more

- Since then, about 45 posts, 180,000 page views (most popular post received 30K views).

  http://www.pl-enthusiast.net/2015/06/02/the-pl-enthusiast-turns-one/

http://www.pl-enthusiast.net/

MOOCs

- In November 2014 I started teaching an on-line course on software security
  - Some of the course slides in this presentation
  - It has been offered 4 times, with 93,332 learners enrolled, and 3,034 who have completed the course.

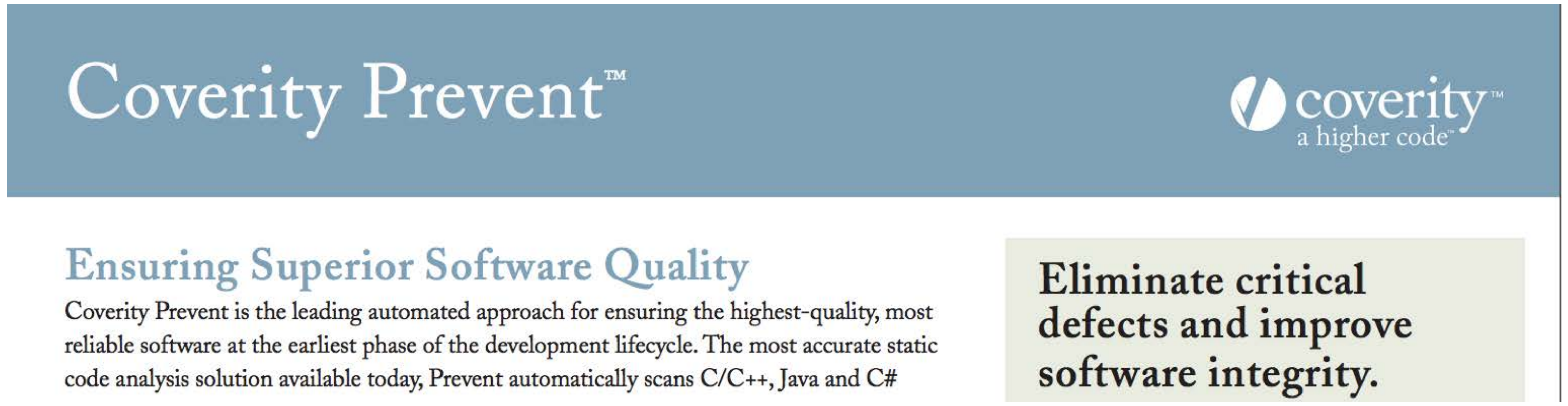
- Since May 2015, I have hosted the Coursera “Capstone” project using the BIBIFI contest

Looking ahead

- Things are getting **better**
  - Many software systems that were previously vulnerable to attack are finally becoming more secure
  - Researchers and practitioners are creating better technology and getting the word out about building software to be more secure

- But they are also getting **worse**
  - The consequences of a mistake are higher
  - New domains for software sometimes result in repeating the mistakes of the past

*There is **more work to do**!*

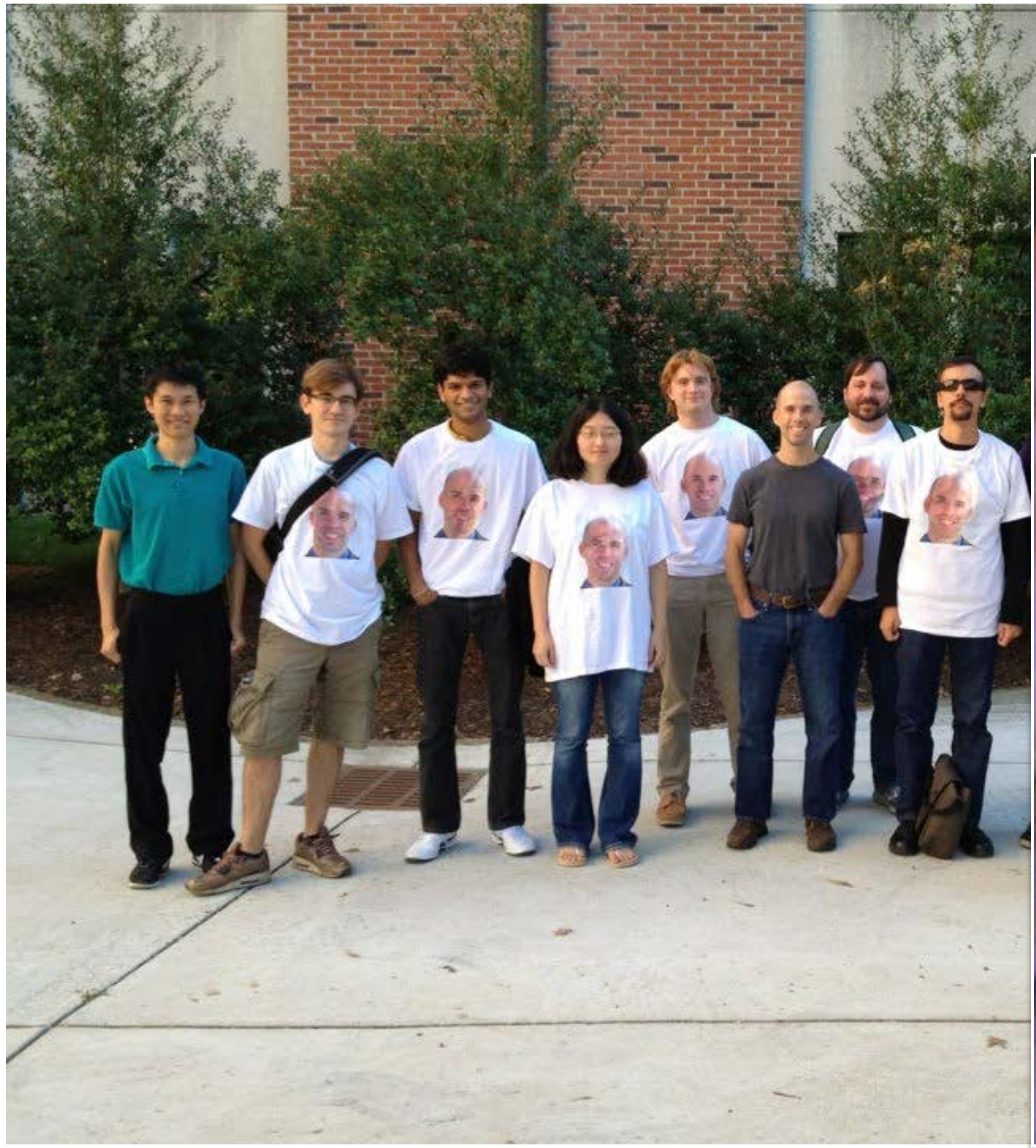


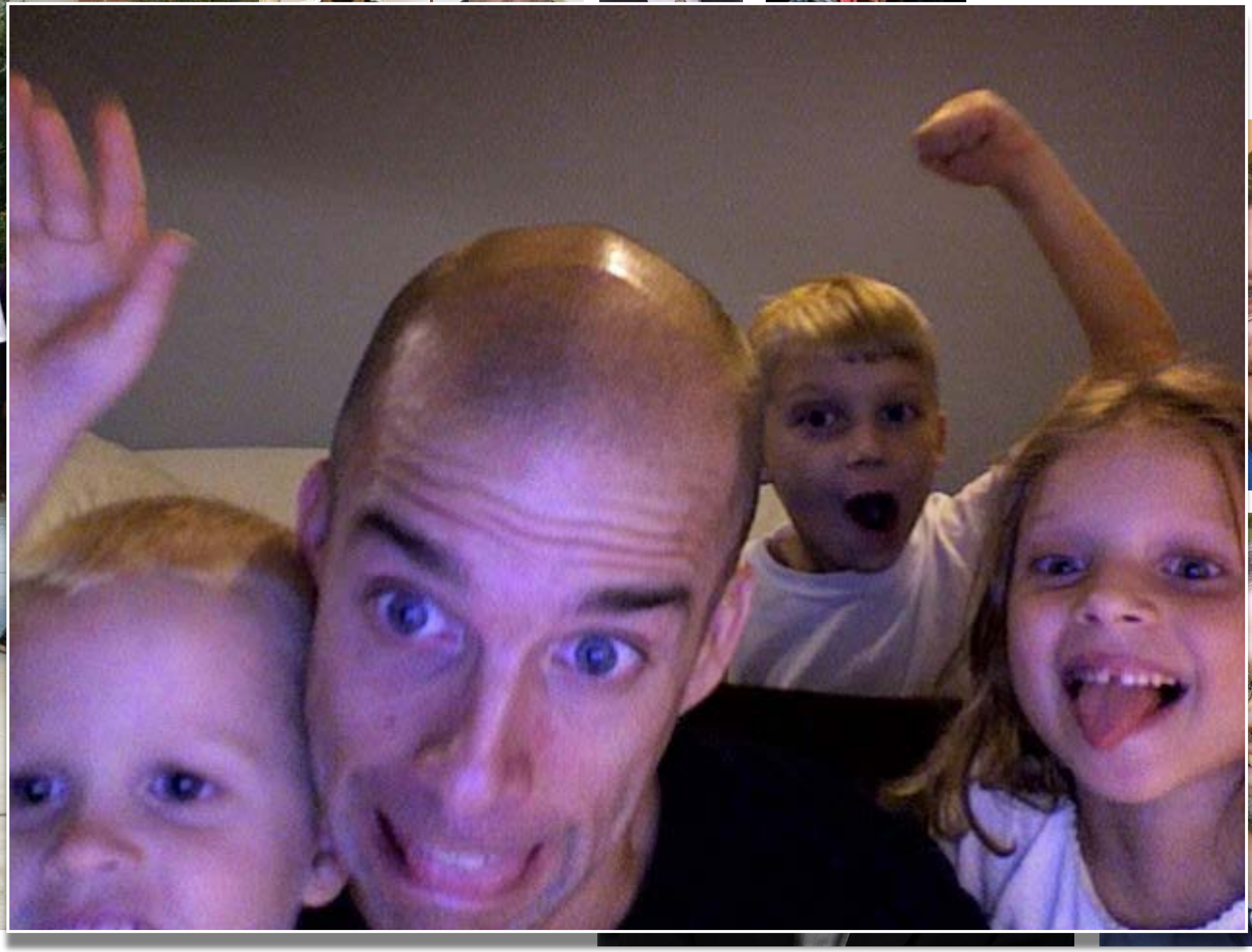
…

University of Maryland 18 56
FindBugs™
because it's easy

Many thanks!

- Students and post-docs,
- Collaborators and mentors,
- Family





Summary

- We need to make building software more like building bridges
  - No more penetrate and patch
  - Consistent consideration of quality goals, including security, from day 1
    - Using the best methods, tools, programming languages, etc.

- Academics, researchers, practitioners all have a role to play