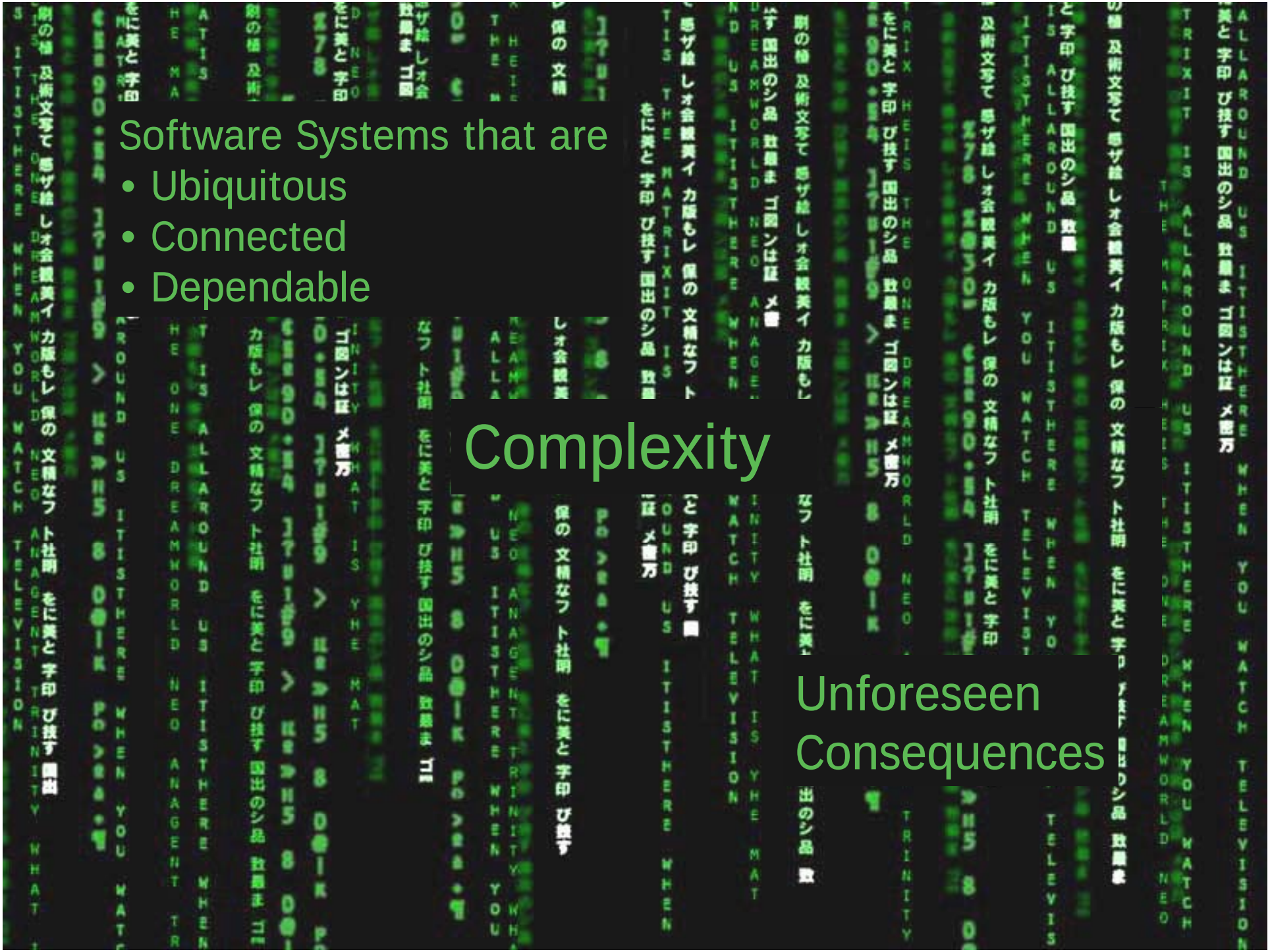
Secure Programming with Static Analysis

Jacob West
jacob@fortify.com

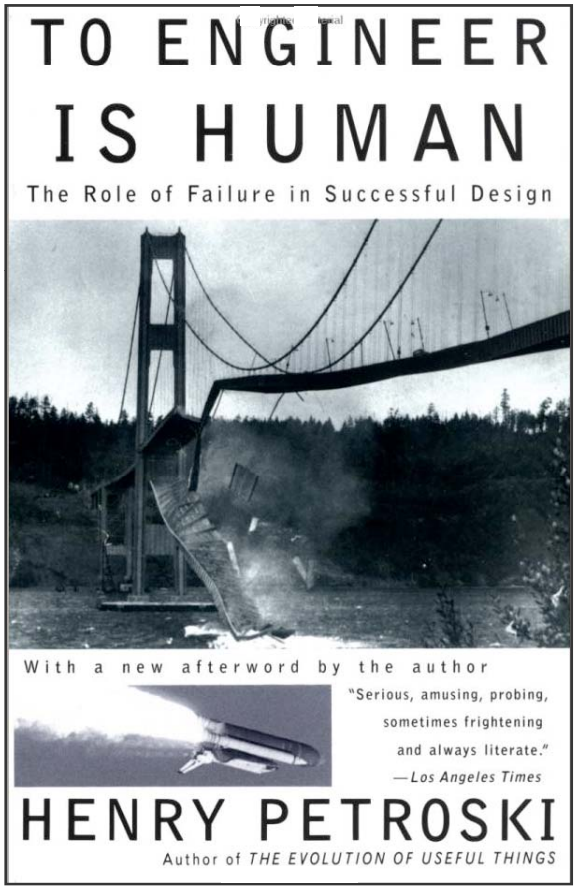

Software Systems that are
• Ubiquitous
• Connected
• Dependable
Complexity
Unforeseen
Consequences

Software Security Today

- The line between secure/insecure is often subtle
  - Many seemingly non-security decisions affect security
- Small problems can hurt a lot
- Smart people make dumb mistakes
  - As a group, programmers tend to make the same security mistakes over and over
- We need non-experts to get security right

Success is foreseeing failure.

– Henry Petroski



Non-functional Security Failures

Generic Mistakes

- Input validation
- Memory safety (buffer overflow)
- Handling errors and exceptions
- Maintaining privacy

Common Software Varieties

- Web applications
- Network services / SOA
- Privileged programs

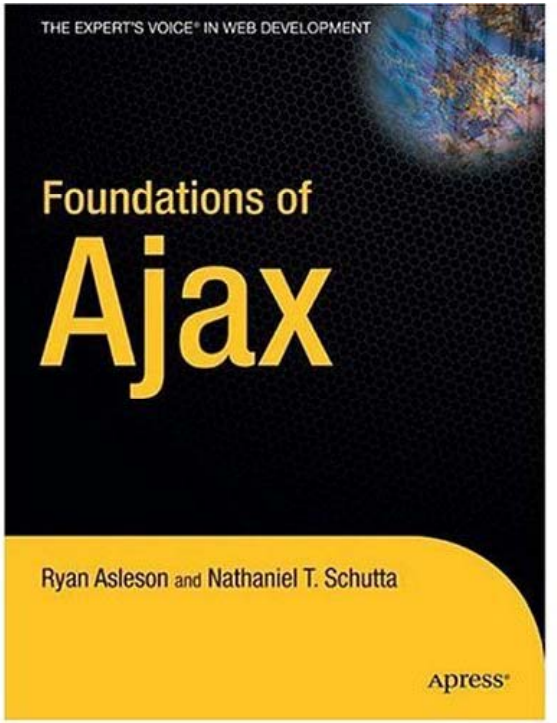
Buffer Overflow

MSDN sample code for function DirSpec:

```
int main(int argc, char *argv[]) {
  ...
  char DirSpec[MAX_PATH + 1];
  printf ("Target dir is %s.\n", argv[1]);
  strncpy (DirSpec, argv[1], strlen(argv[1])+1);
  ...
}
```

Cross-Site Scripting

```
<c:if
   test="${param.sayHello}">
   Hello ${param.name}!
</c:if>
```



"We never intended the code that's in there to actually be production-ready code"

- Ryan Asleson

FORTIFY SOFTWARE

Reliving Past Mistakes

- Cross-site scripting looks more and more like buffer overflow

Buffer Overflow

- Allows arbitrary code execution
- **Exploit is hard to write**
- Easy mistake to make in C/C++
- Well known problem for decades

Cross-site Scripting

- Allows arbitrary code execution
- **Exploit is easy to write**
- Easy mistake to make
- Well known problem for a decade

Wrong Answers

Try Harder

- Our people are smart and work hard.
- Just tell them to stop making mistakes.

- Not everyone is going to be a security expert.
- Getting security right requires feedback.

Fix It Later

- Code as usual.
- Build a better firewall (app firewall, intrusion detection, etc.)

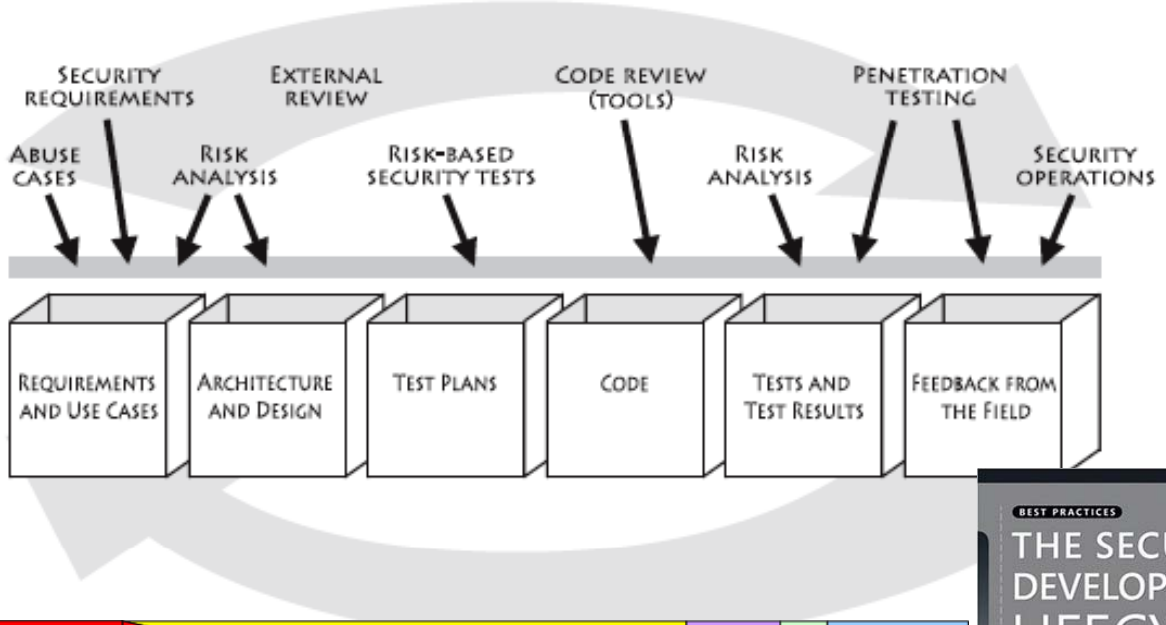
- More walls don't help when the software is meant to communicate.
- Security team can't keep up.

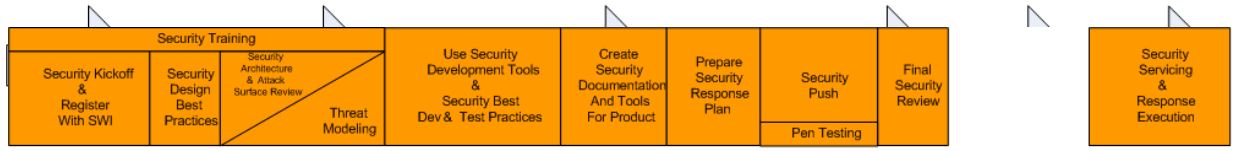
Test Your Way Out

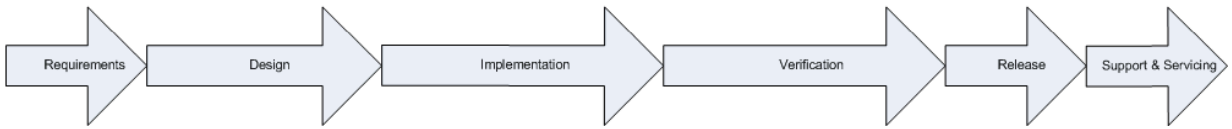
- Do a penetration test on the final version.
- Scramble to patch findings.

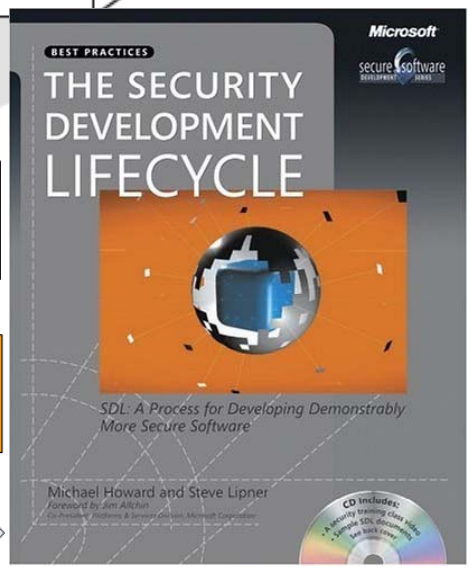
- Pen testing is good for demonstrating the problem.
- Doesn't work for the same reason you can't test quality in.

Security in the Development Lifecycle



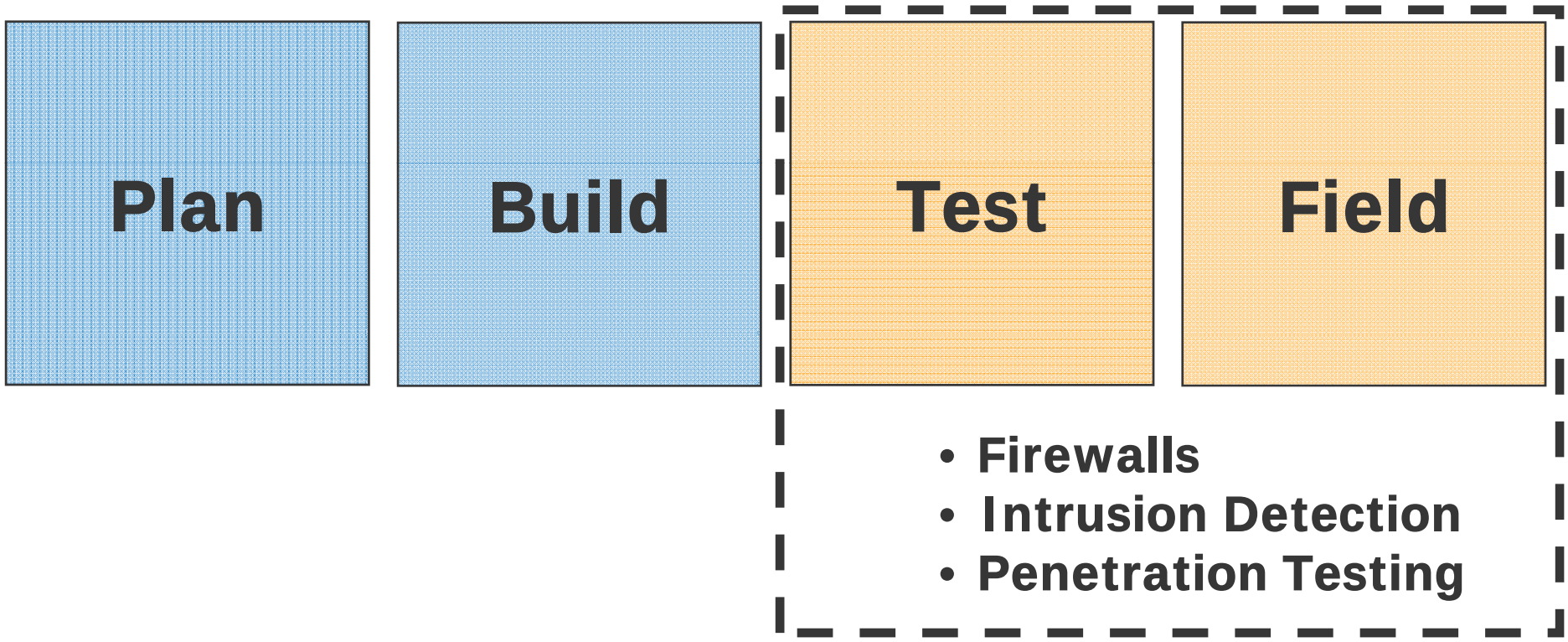






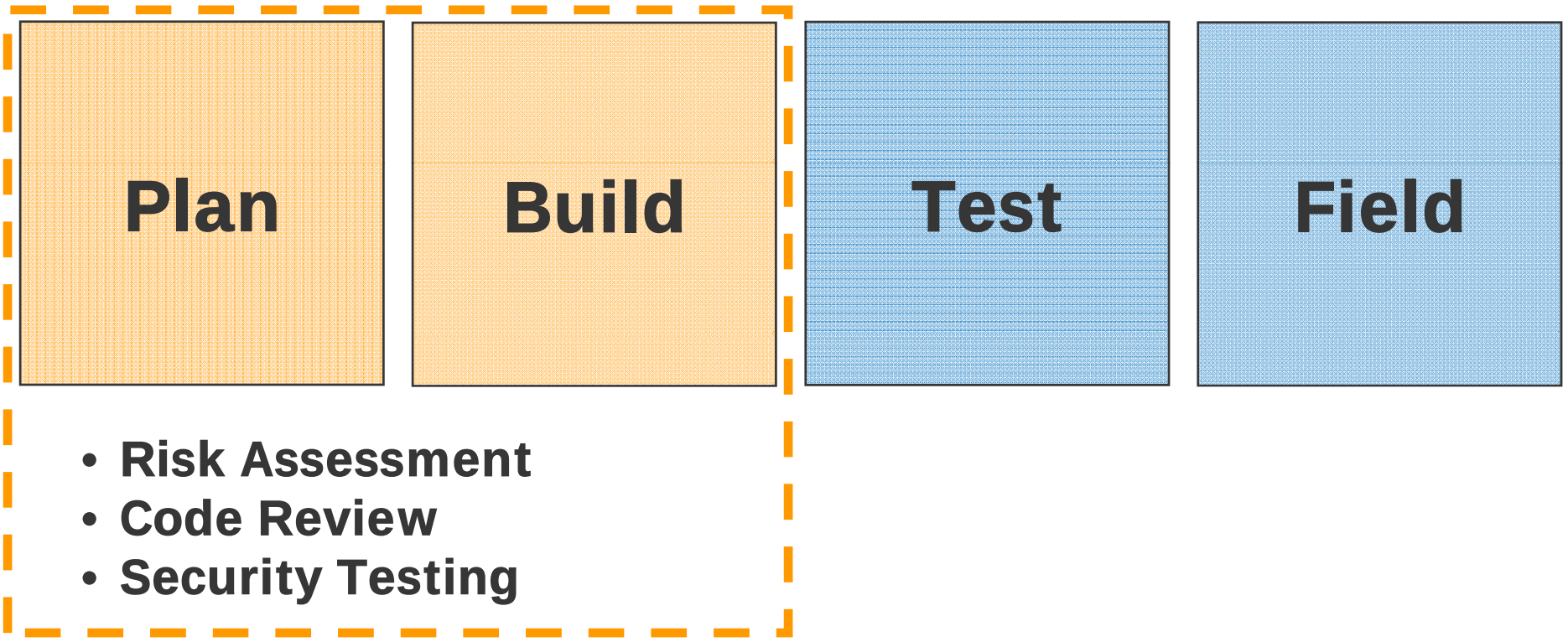
FORTIFY SOFTWARE

Security in the Development Lifecycle



FORTIFY SOFTWARE

Security in the Development Lifecycle



Overview

- Introduction
- Static Analysis: The Big Picture
- Inside a Static Analysis Tool
- Static Analysis in Practice
- What Next?
- Parting Thoughts

Static Analysis: The Big Picture

Static Analysis Defined

- Analyze code without executing it
- Able to contemplate many more possibilities than you could execute with conventional testing
- Doesn’t know what your code is supposed to do
- Must be told what to look for

FORTIFY SOFTWARE

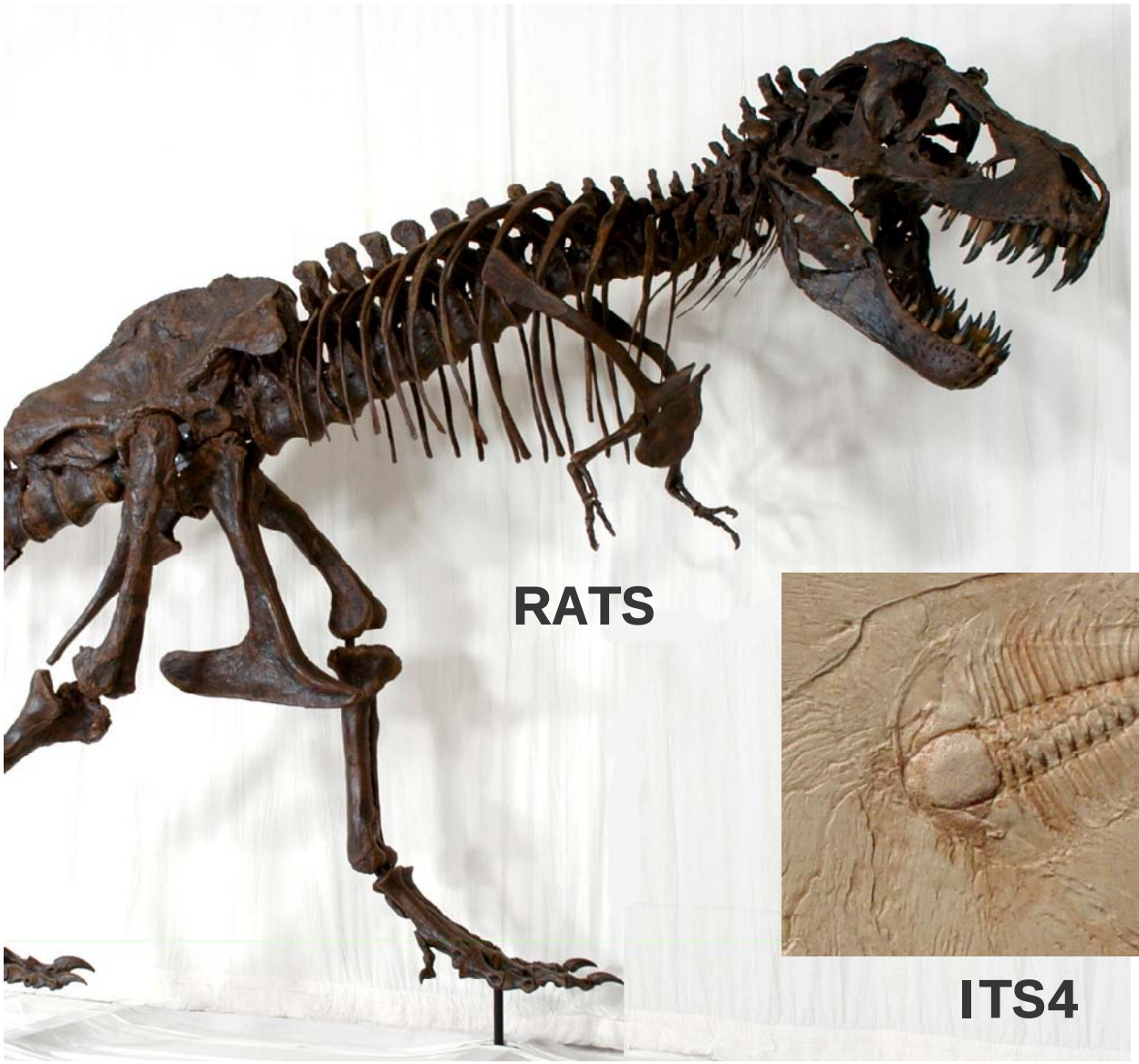
The Many Faces of Static Analysis

- Type checking
- Style checking
- Program understanding
- Program verification / Property checking
- Bug finding
- Security review

Why Static Analysis is Good for Security

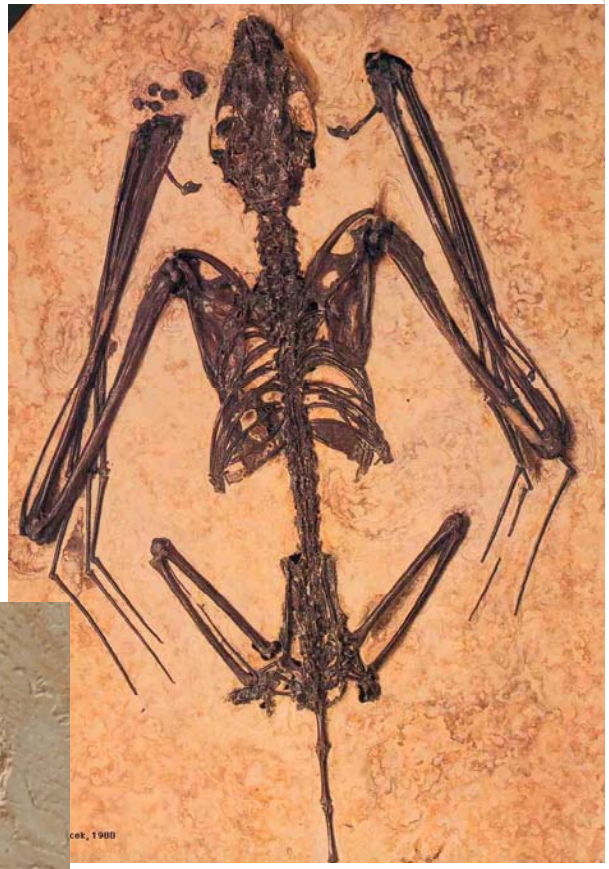
- Fast compared to manual code review
- Fast compared to testing
- Complete, consistent coverage
- Brings security knowledge with it
- Makes review process easier for non-experts

Prehistoric Static Analysis Tools



RATS

ITS4



Flawfinder

FORTIFY SOFTWARE

Prehistoric Static Analysis Tools

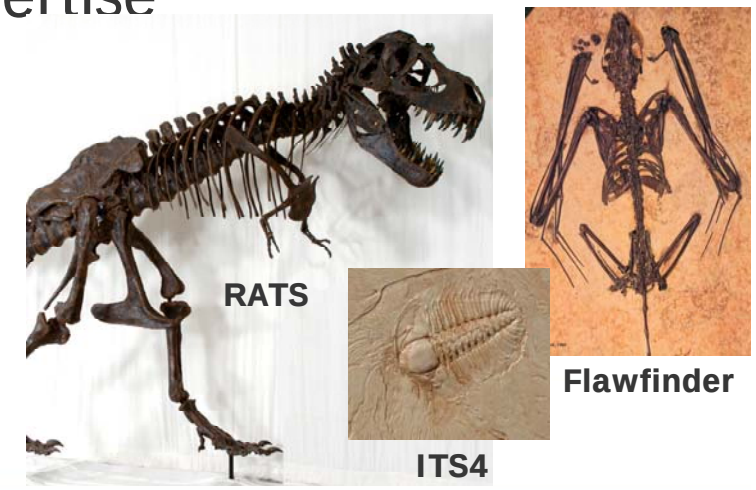
Glorified grep

(+) Good

- Help security experts audit code
- A place to collect info about bad coding practices

(-) Bad

- NOT BUG FINDERS
- Not helpful without security expertise



Advanced Static Analysis Tools: Prioritization

```
int main(int argc, char* argv[]) {
    char buf1[1024];
    char buf2[1024];
    char* shortString = "a short string";
    strcpy(buf1, shortString); /* eh. */
    strcpy(buf2, argv[0]);     /* !!! */
    ...
}
```

FORTIFY SOFTWARE

What You Won't Find

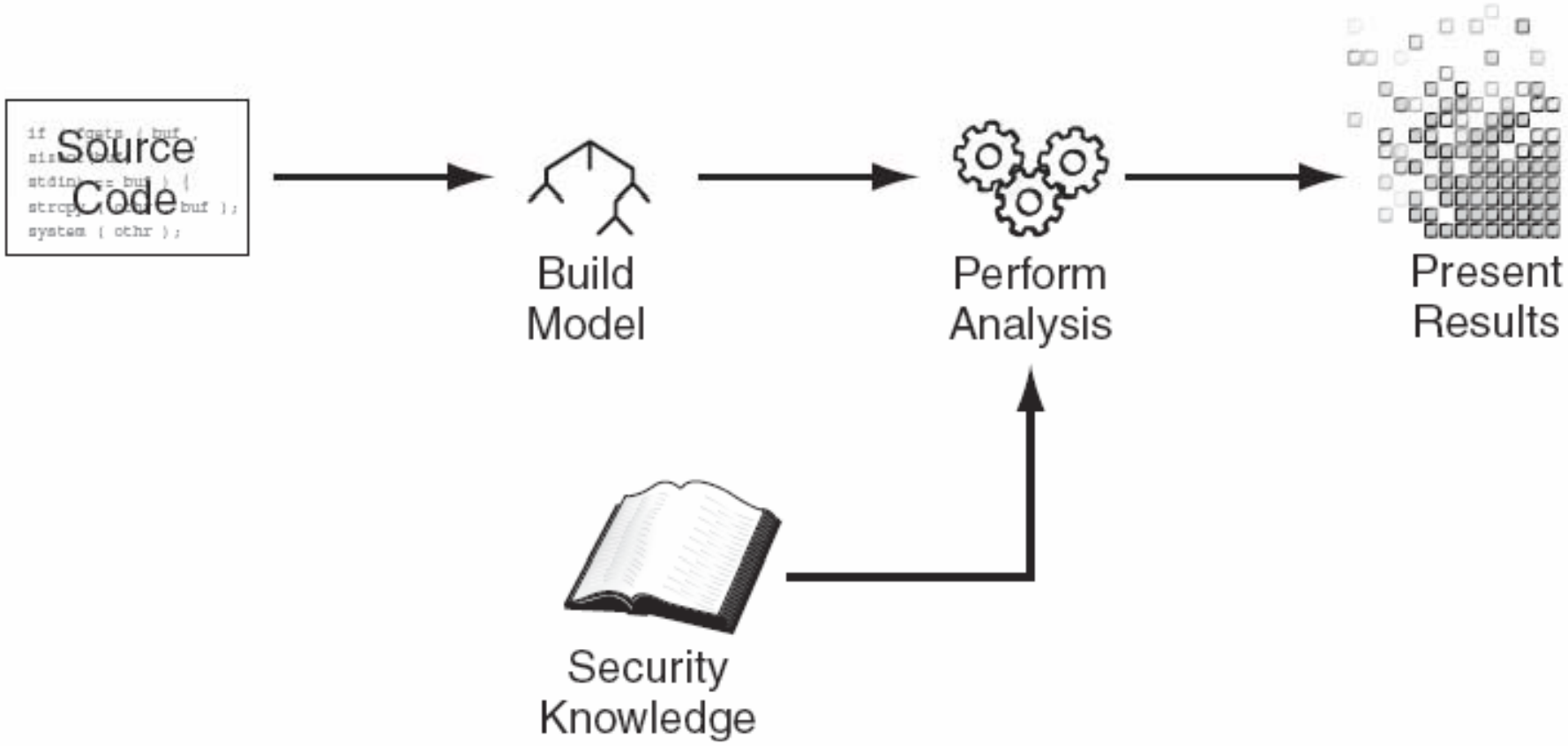
- Architecture errors
  - Microscope vs. telescope
- Bugs you're not looking for
  - Bug categories must be predefined
- System administration mistakes
- User mistakes

Security vs. Quality

- Bug finding tools focus on high confidence results
  - Bugs are cheap (plentiful)
  - Bug patterns, bug idioms
  - False alarms are killers
- Security tools focus on high risk results
  - More human input required
  - The bugs you miss are the killers

Inside a Static Analysis Tool

Under the Hood



Critical Attributes

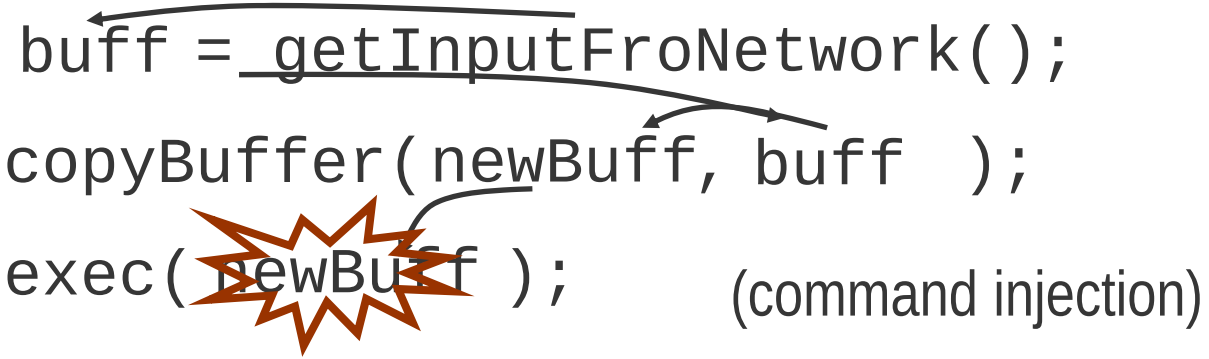
- Language support
  - Understands the relevant languages/dialects
- Analysis algorithms
  - Uses the right techniques to find and prioritize issues
- Capacity
  - Ability to gulp down millions of lines of code
- Rule set
  - Modeling rules, security properties
- Results management
  - Allow human to review results
  - Prioritization of issues
  - Control over what to report

Building a Model

- Front end looks a lot like a compiler
- Language support
  - One language/compiler is straightforward
  - Lots of combinations is harder
- Could analyze compiled code...
  - Everybody has the binary
  - No need to guess how the compiler works
  - No need for rules
- ...but
  - Decompilation can be difficult
  - Loss of context hurts. A lot.
  - Remediation requires mapping back to source anyway

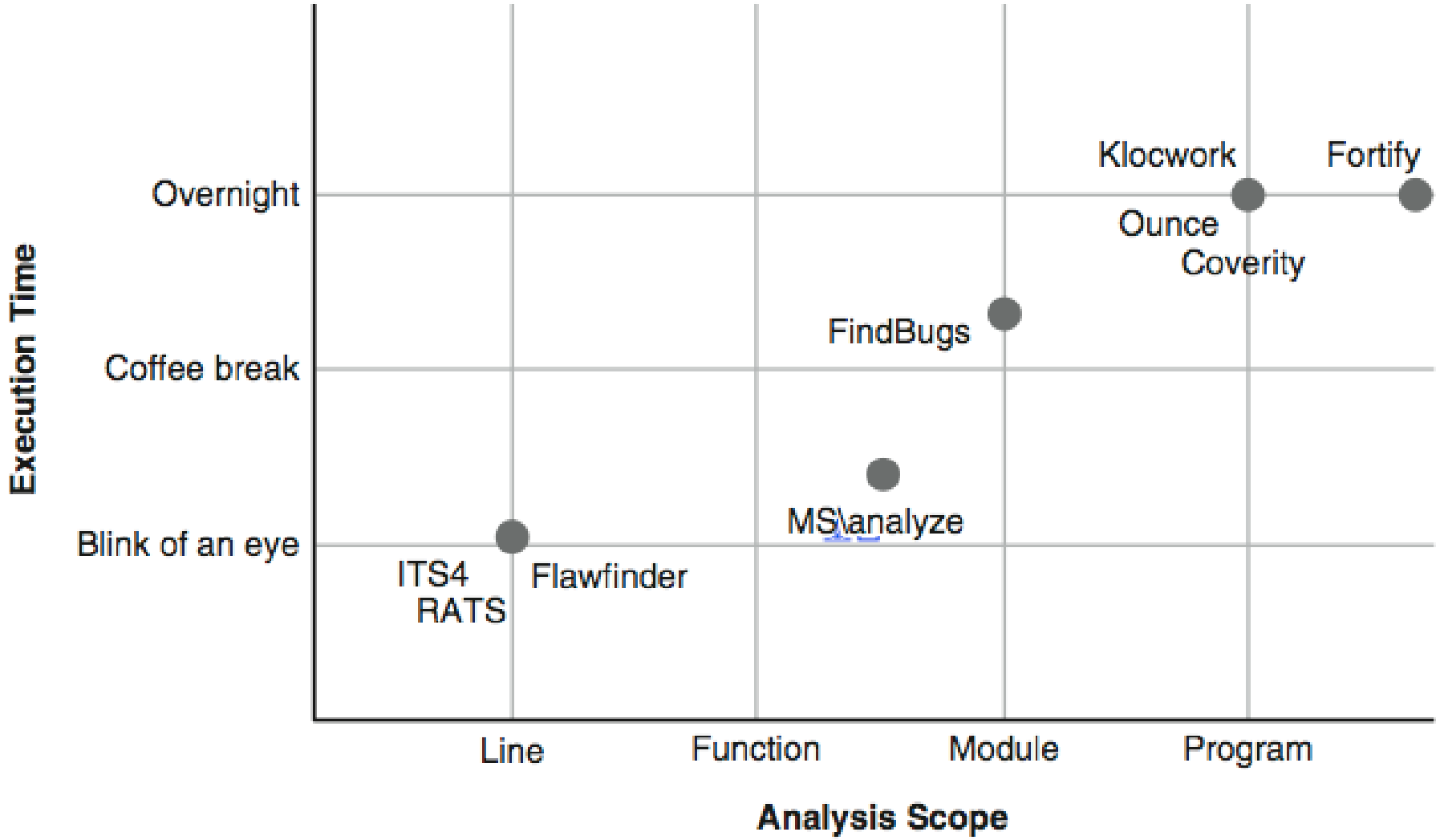
Analysis Techniques

- Taint propagation
  - Trace potentially tainted data through the program
  - Report locations where an attacker could take advantage of a vulnerable function or construct



- Many other approaches, no one right answer

Capacity: Scope vs. Performance



FORTIFY SOFTWARE

Only Two Ways to Go Wrong

- False positives
  - Incomplete/inaccurate model
  - Conservative analysis
- False negatives
  - Incomplete/inaccurate model
  - Missing rules
  - “Forgiving” analysis

Rules: Dataflow

- Specify
  - Security properties
  - Behavior of library code

```
buff = getInputFromNetwork();
copyBuffer(newBuff, buff);
exec(newBuff);
```

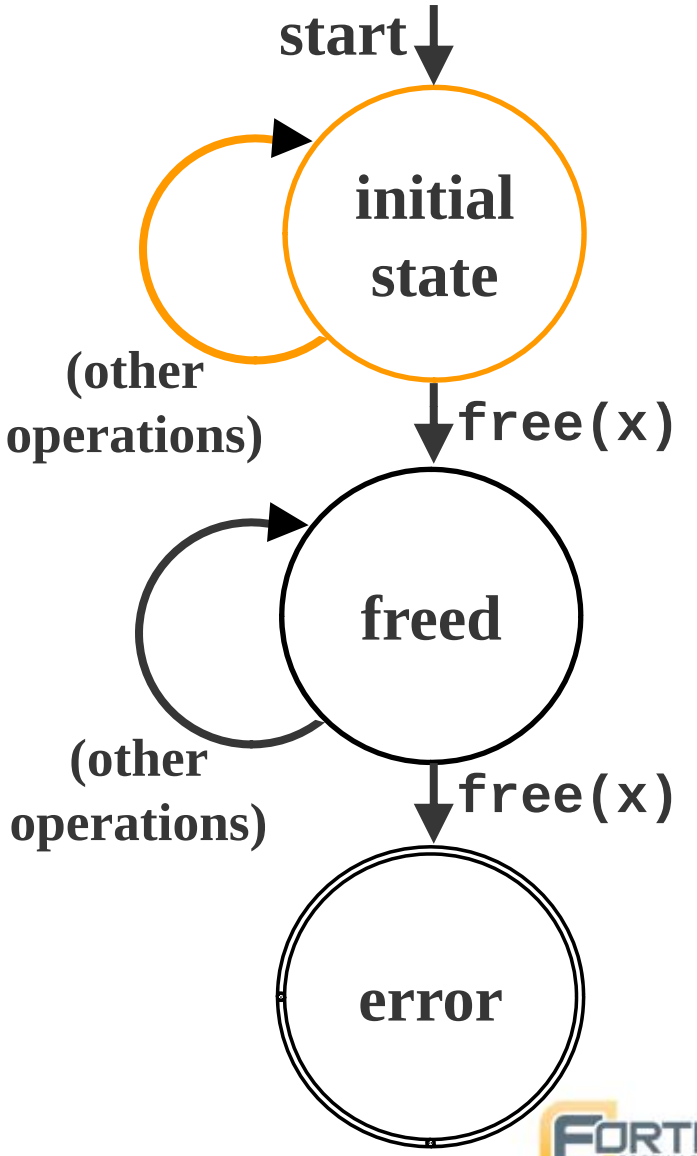
- Three rules to detect the command injection vulnerability

1) **getInputFromNetwork()** postcondition:
   return value is tainted
2) **copyBuffer(arg1, arg2)** postcondition:
   arg1 array values set to arg2 array values
3) **exec(arg)** precondition:
   arg must not be tainted

Rules: Control Flow

- Look for dangerous sequences
- Example: Double-free vulnerability

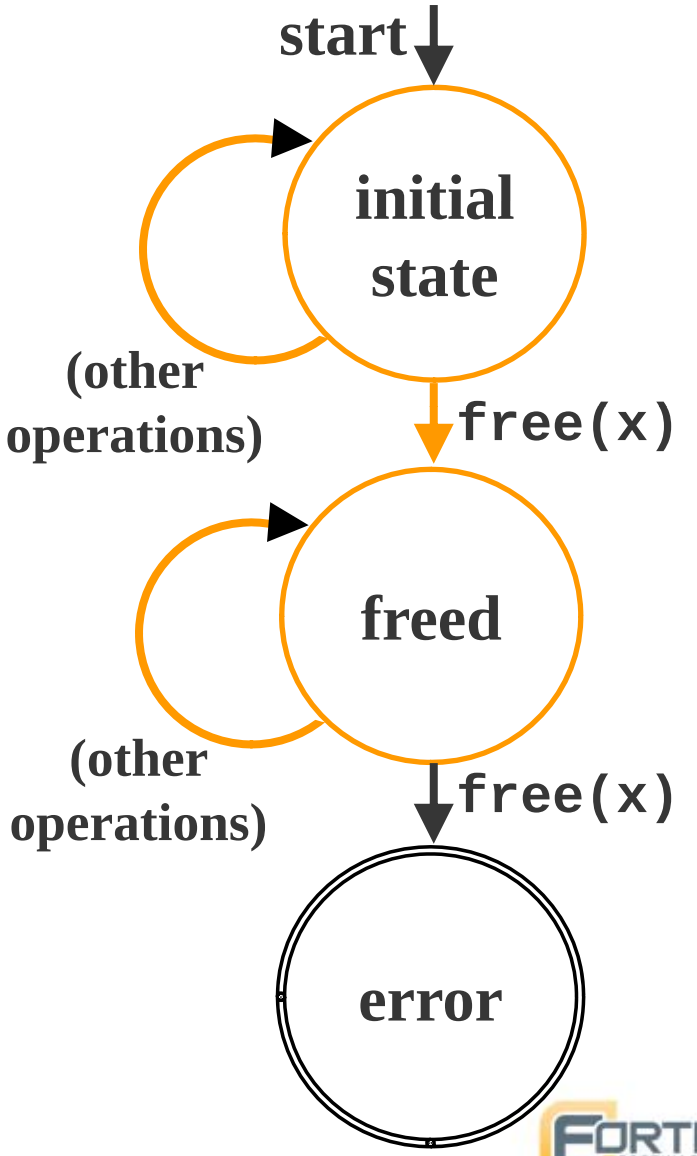
```
while ((node = *ref) != NULL) {
  *ref = node->next;
  free(node);
  if (!unchain(ref)) {
    break;
  }
}
if (node != 0) {
  free(node);
  return UNCHAIN_FAIL;
}
```



Rules: Control Flow

- Look for dangerous sequences
- Example: Double-free vulnerability

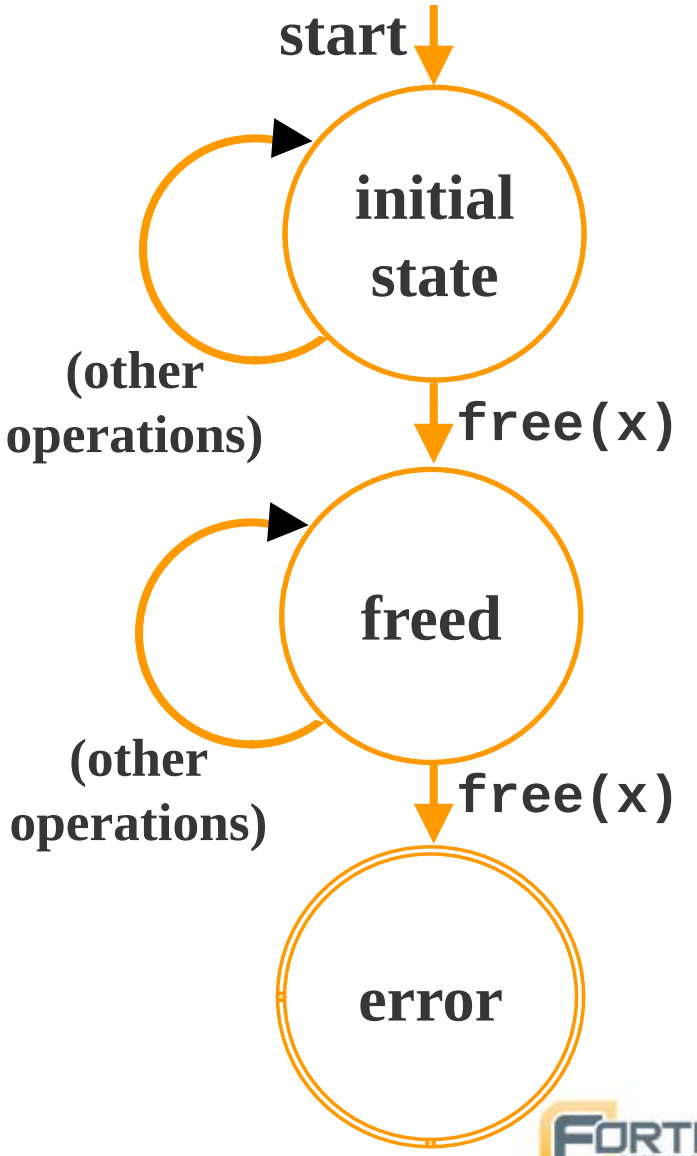
```
while ((node = *ref) != NULL) {
  *ref = node->next;
  free(node);
  if (!unchain(ref)) {
    break;
  }
}
if (node != 0) {
  free(node);
  return UNCHAIN_FAIL;
}
```



Rules: Control Flow

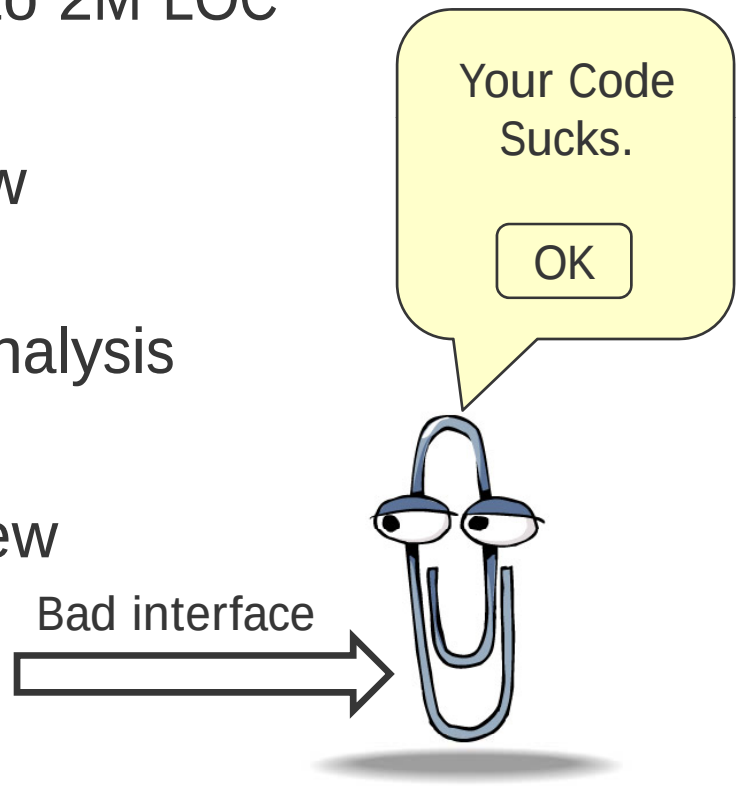
- Look for dangerous sequences
- Example: Double-free vulnerability

```
while ((node = *ref) != NULL) {
  *ref = node->next;
  free(node);
  if (!unchain(ref)) {
    break;
  }
}
if (node != 0) {
  free(node);
  return UNCHAIN_FAIL;
}
```



Displaying Results

- Must convince programmer that there's a bug in the code
- Different interfaces for different scenarios:
  - Security auditor parachutes in to 2M LOC
  - Programmer reviews own code
  - Programmers share code review responsibilities
- Interface is just as important as analysis
- Don't show same bad result twice
- Try this at home: Java Open Review
  http://opensource.fortify.com



FORTIFY SOFTWARE

Static Analysis in Practice

Two Ways to Use the Tools

- Analyze completed programs
  - Fancy penetration test. Bleah.
  - Results can be overwhelming
  - Most people have to start here
  - Good motivator

- Analyze as you write code
  - Run as part of build
  - Nightly/weekly/milestone
  - Fix as you go

Typical Objections and Their True Meanings

Objection	Translation
“It takes too long to run.”	“I think security is optional, so I don’t want to do it.”
“It has too many false positives.”	“I think security is optional, so I don’t want to do it.”
“It doesn’t fit with the way I work.”	“I think security is optional, so I don’t want to do it.”

Adopting a Static Analysis Tool

1) Some culture change required
   - More than just another tool
   - Often carries the banner for software security program
   - Pitfall: the tool doesn't solve the problem by itself
2) Define the playing field
   - Choose specific objectives
   - Build a gate
3) Do training up front
   - Software security training is paramount
   - Tool training is helpful too

Adopting a Static Analysis Tool

4) Start small
   - Do a pilot rollout to a friendly dev group
   - Build on your success

5) Go for the throat
   - Tools detect lots of stuff. **Turn most of it off.**
   - Focus on easy-to-understand, highly relevant problems.

6) Appoint a champion
   - Make sure there is a point person on the dev team
   - Choose a developer who knows a little about everything

Adopting a Static Analysis Tool

7) Measure the outcome
   - Keep track of tool findings
   - Keep track of outcome (issues fixed)

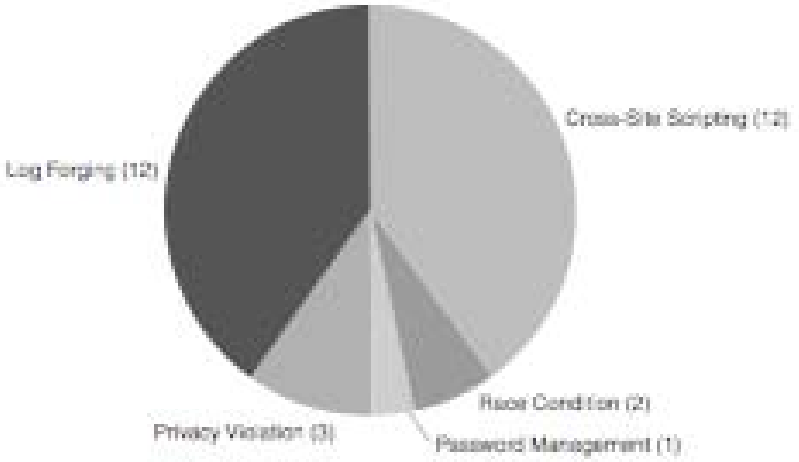
8) Make it your own
   - Investigate customization
   - Map tool against internal security standards.
   - Best case scenario is cyclic:
     - The tool reinforces coding guidelines
     - Coding guidelines are written with automated checking in mind

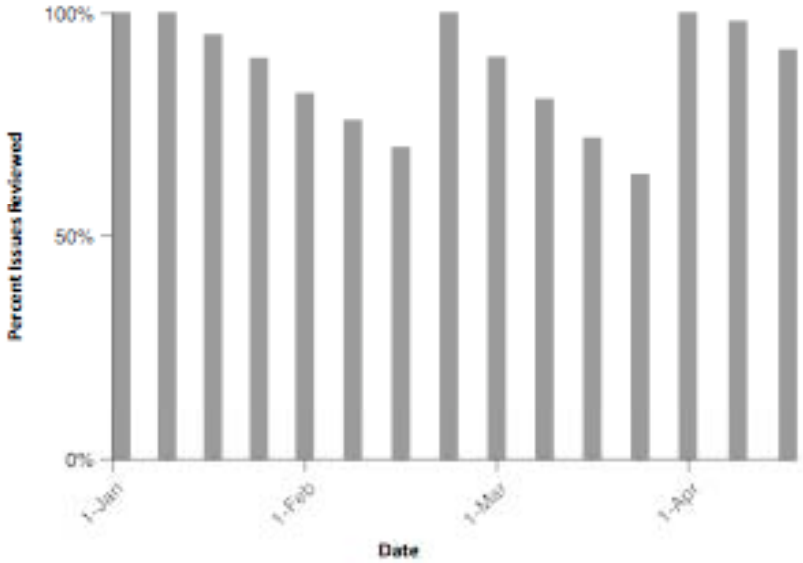
9) The first time around is the worst
   - Budget 2x typical cycle cost
   - Typical numbers: 10% of time for security,
     20% for the first time

FORTIFY SOFTWARE

Metrics

- ?? Defect Density → Vulnerability Density ??
- NOT A GOOD RISK BAROMETER
- Good for answering questions such as
  - Which bugs do we write most often?
  - How much remediation effort is required?





What Next?

Seven Pernicious Kingdoms

- Catalog, define, and categorize common mistakes
- http://www.fortify.com/vulncat

- Input validation and representation
- API abuse
- Security features
- Time and state
- Error handling
- Code quality
- Encapsulation
- * Environment

FORTIFY SOFTWARE

Security Testing

- Most widely used security testing techniques are about controllability
  - Fuzzing (random input)
  - Shooting dirty data (input that often causes trouble)
- A different take: improve observability
  - Instrument code to observe runtime behavior:

    Fortify Tracer
- Benefits
  - Security-oriented code coverage
  - Vastly improved error reporting
  - Finds more bugs
- Uses rule set from static analysis tool!

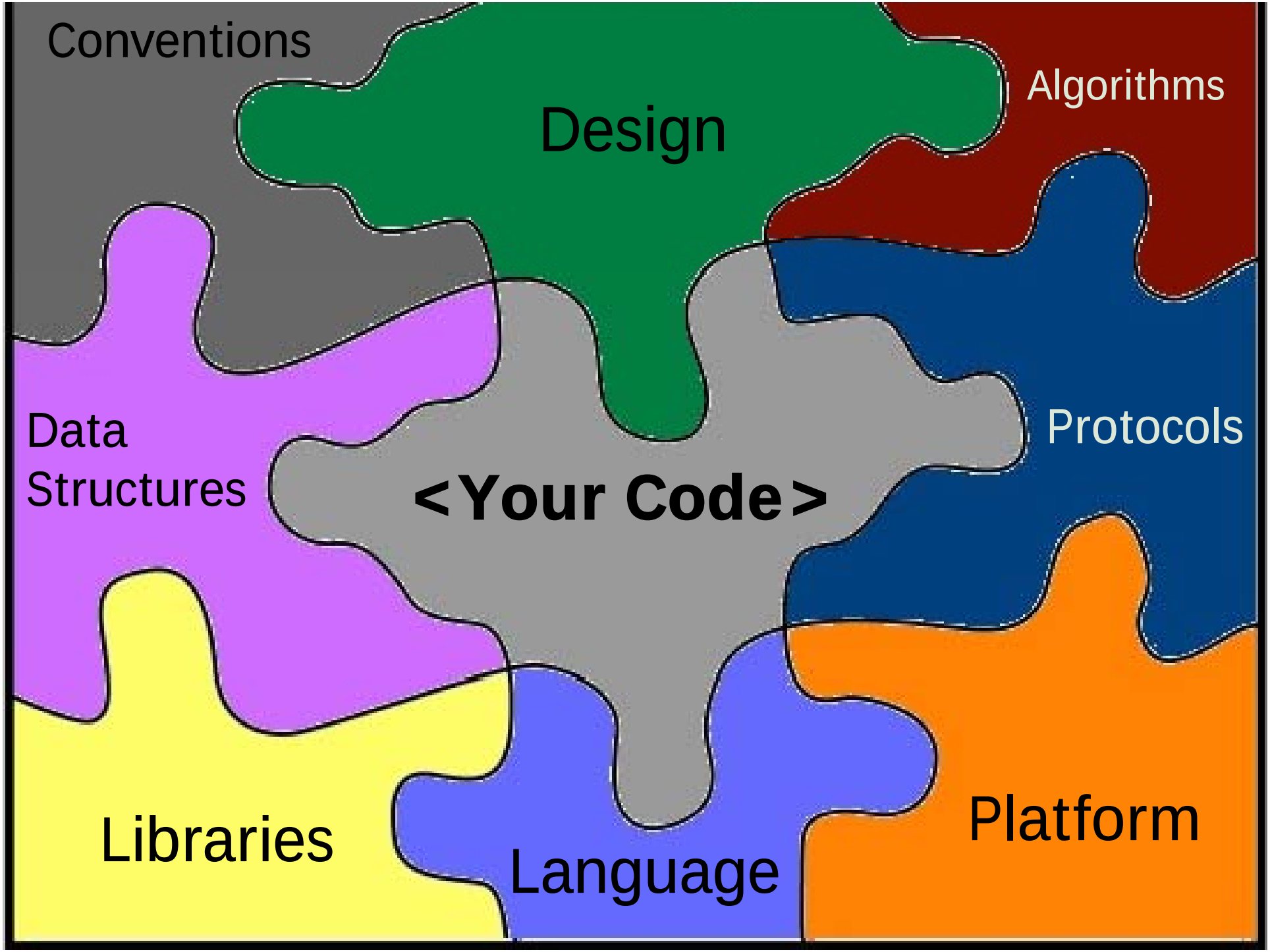
FORTIFY SOFTWARE

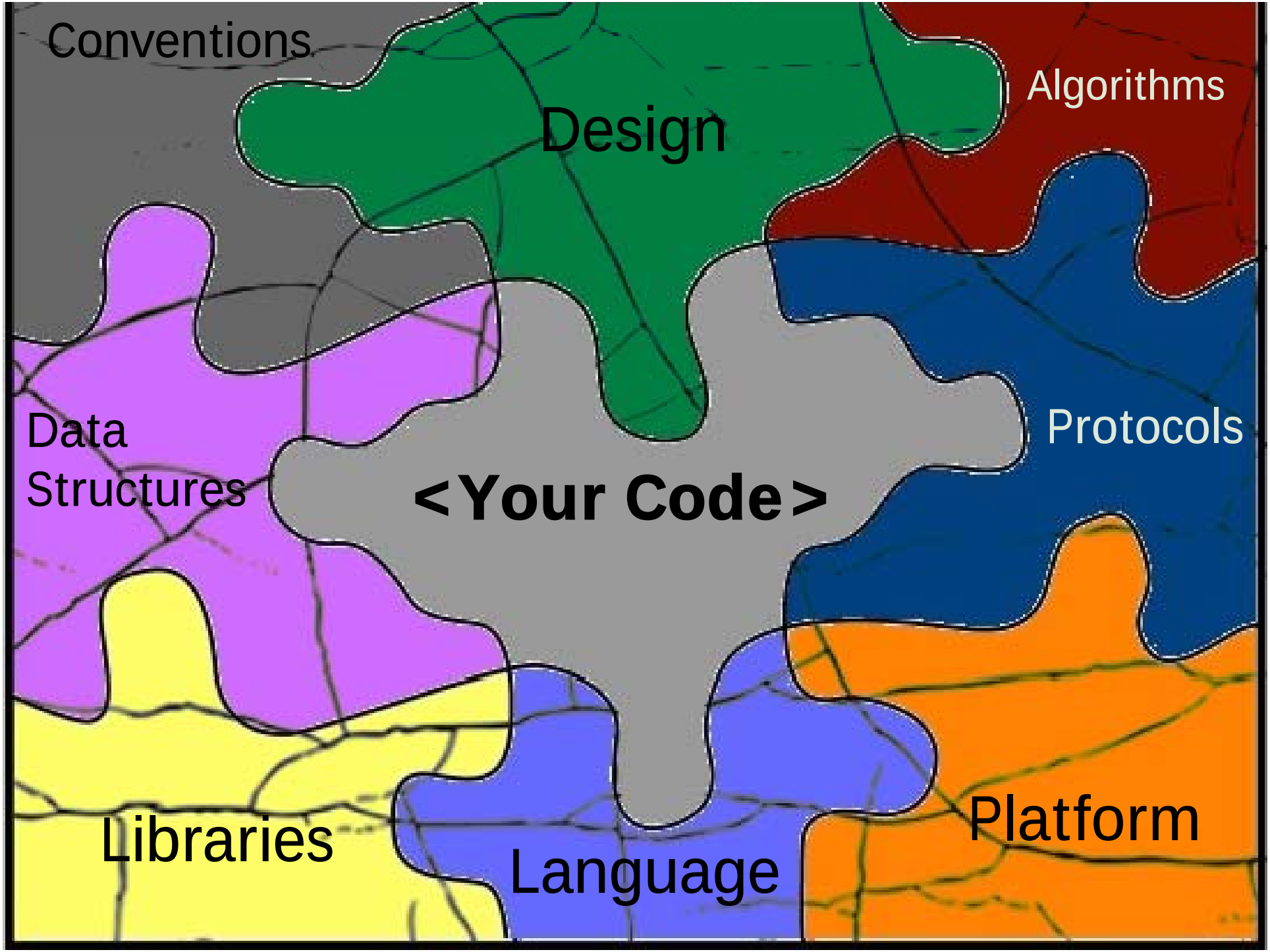
Detecting Attacks at Runtime

- If you can find bugs, can you fix them?
- Instrument program, watch it run:

  Fortify Defender
- More context than external systems
- Flexible response: log, block, etc
- Low performance overhead is a must
- Potential to detect misuse in addition to bugs

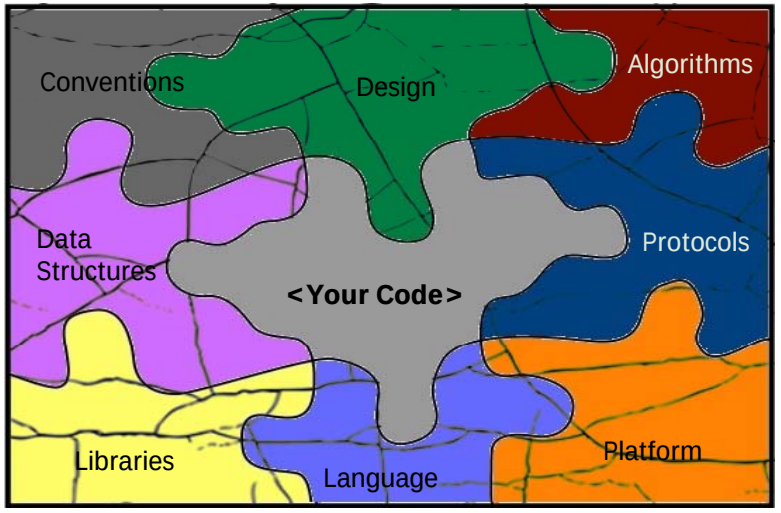
Parting Thoughts


Conventions
Design
Algorithms
Data
Structures
<Your Code>
Protocols
Libraries
Language
Platform


Conventions
Design
Algorithms
Data
Structures
<Your Code>
Protocols
Libraries
Language
Platform

The Buck Stops With Your Code

- **Security problems everywhere you look**
  - Languages, libraries, frameworks, etc.
- **Right answer**
  - Better languages, libraries, frameworks, etc.
- **Realistic answer**
  - Build secure programs out of insecure pieces



Summary

- Mistakes happen. Plan for them.
- Security is now part of programming
- For code auditors: tools make code review efficient
- For programmers: tools bring security expertise
- Critical components of a good tool:
  - Algorithm
  - Rules
  - Interface
  - Adoption Plan

Brian Chess
brian@fortify.com

Jacob West
jacob@fortify.com