Security for Hackers and Developers: Fuzzing

EXPLAINING FUZZ TESTING

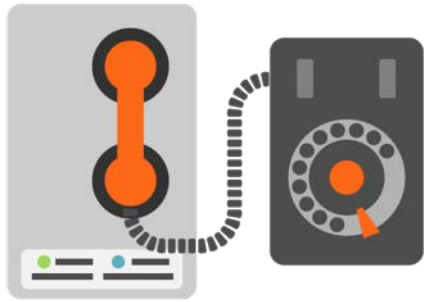
Dr. Jared DeMott
CTO AND FOUNDER

@jareddemott www.vdalabs.com

Overview

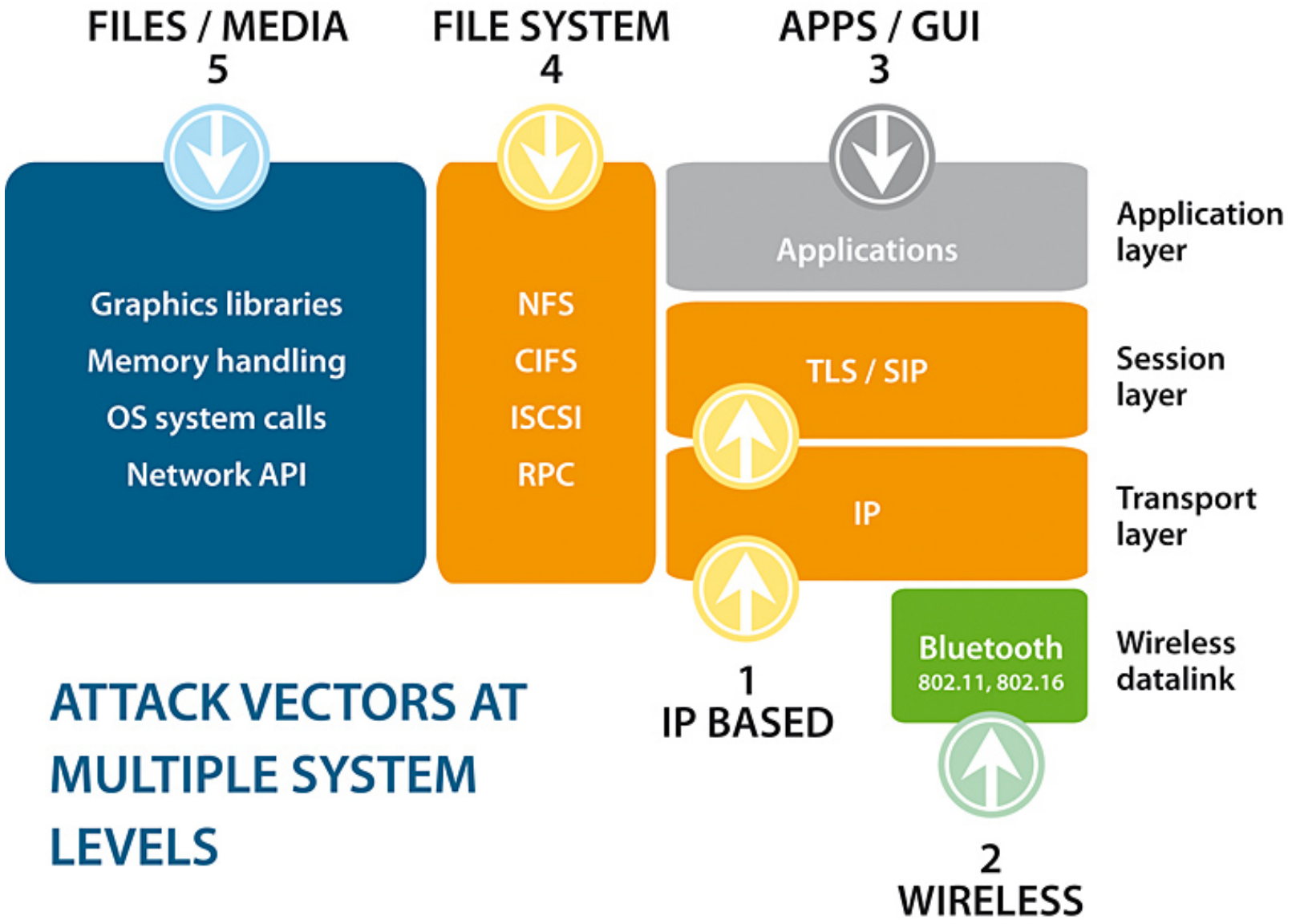
Why fuzzing?

- Define
  - Main types
- When effective
- Where it fits into SDL
- Target selection
- Engineering a fuzzer

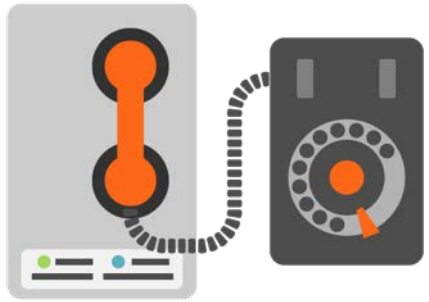


It was a dark and stormy night...

- Dr. Barton Miller
  - Madison, WI 1988
    - Modem - stumbled upon crash
      - 1990 they could crash or hang between 25-33% of the utility programs on the seven Unix variants



ATTACK VECTORS AT MULTIPLE SYSTEM LEVELS



Automated testing technique to find bugs in software

Focus on the *attack surface*

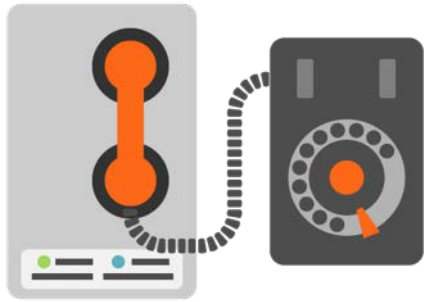
- Areas of code used to access or interface between systems

Focus on *boundary conditions*

- *Privilege and data*

20% being design issues
70% of modern security vulnerabilities are programming flaws
10% being configuration issues

Over 80% of communication software implementations were vulnerable to implementation-level security flaws
For example, 25 out of 30 Bluetooth implementations crashed when they were tested with Bluetooth fuzzing tools (2008)



Effective on C/C++

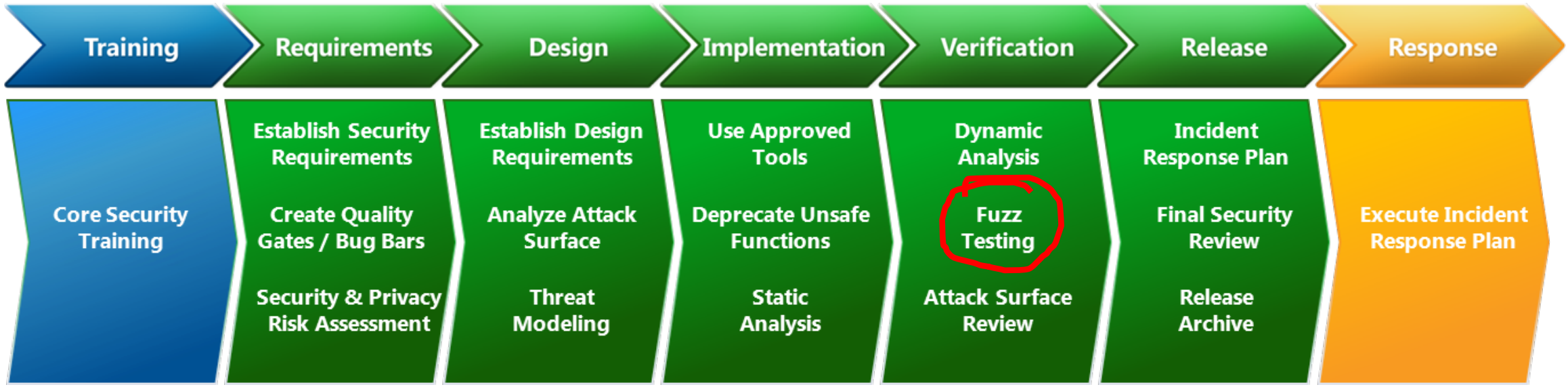
- Memory is unmanaged
- PC, embedded, telco, SCADA, etc.

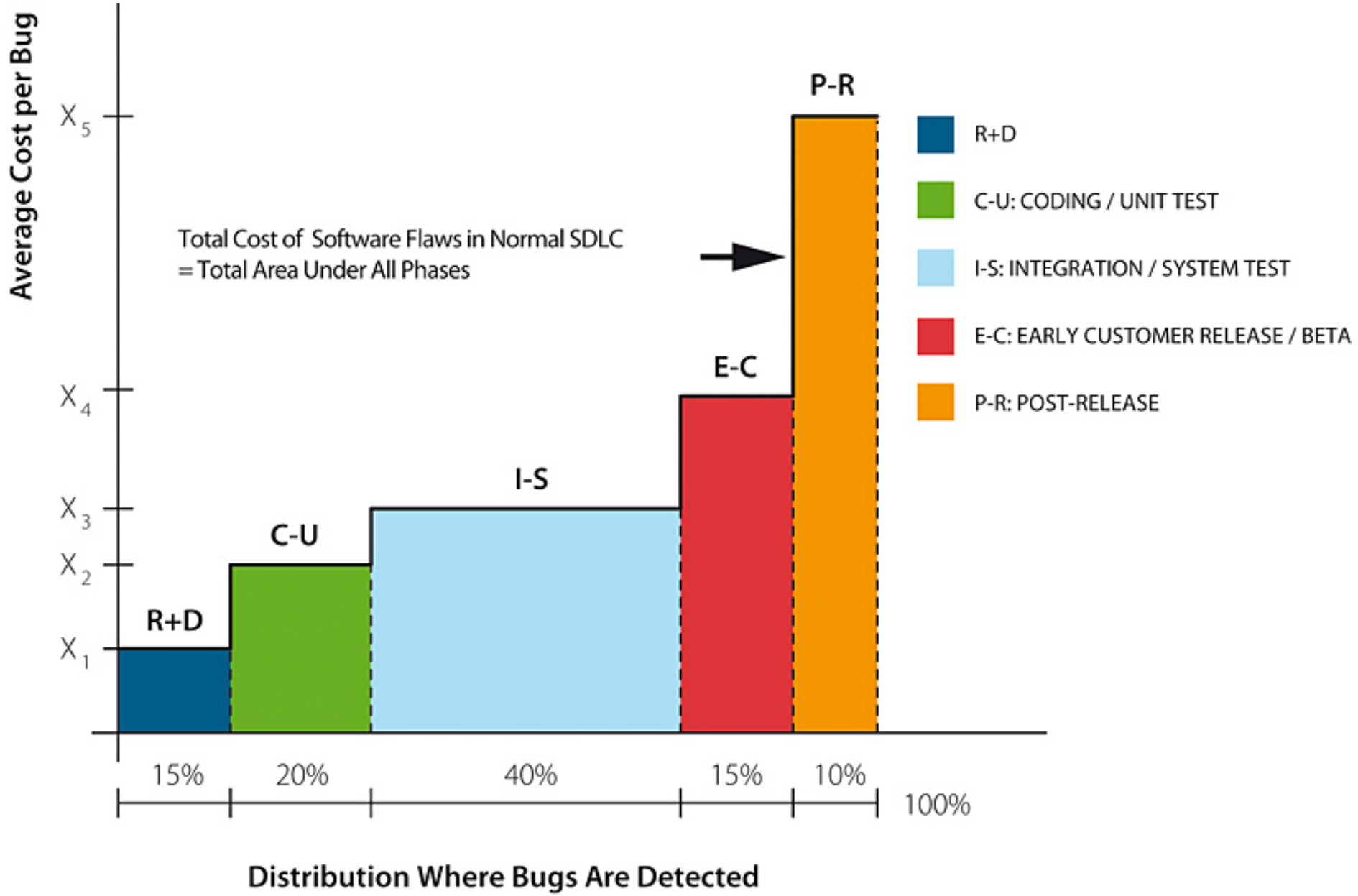
Also HLL interpreters are built with C/C++

- Can sometimes call directly to native

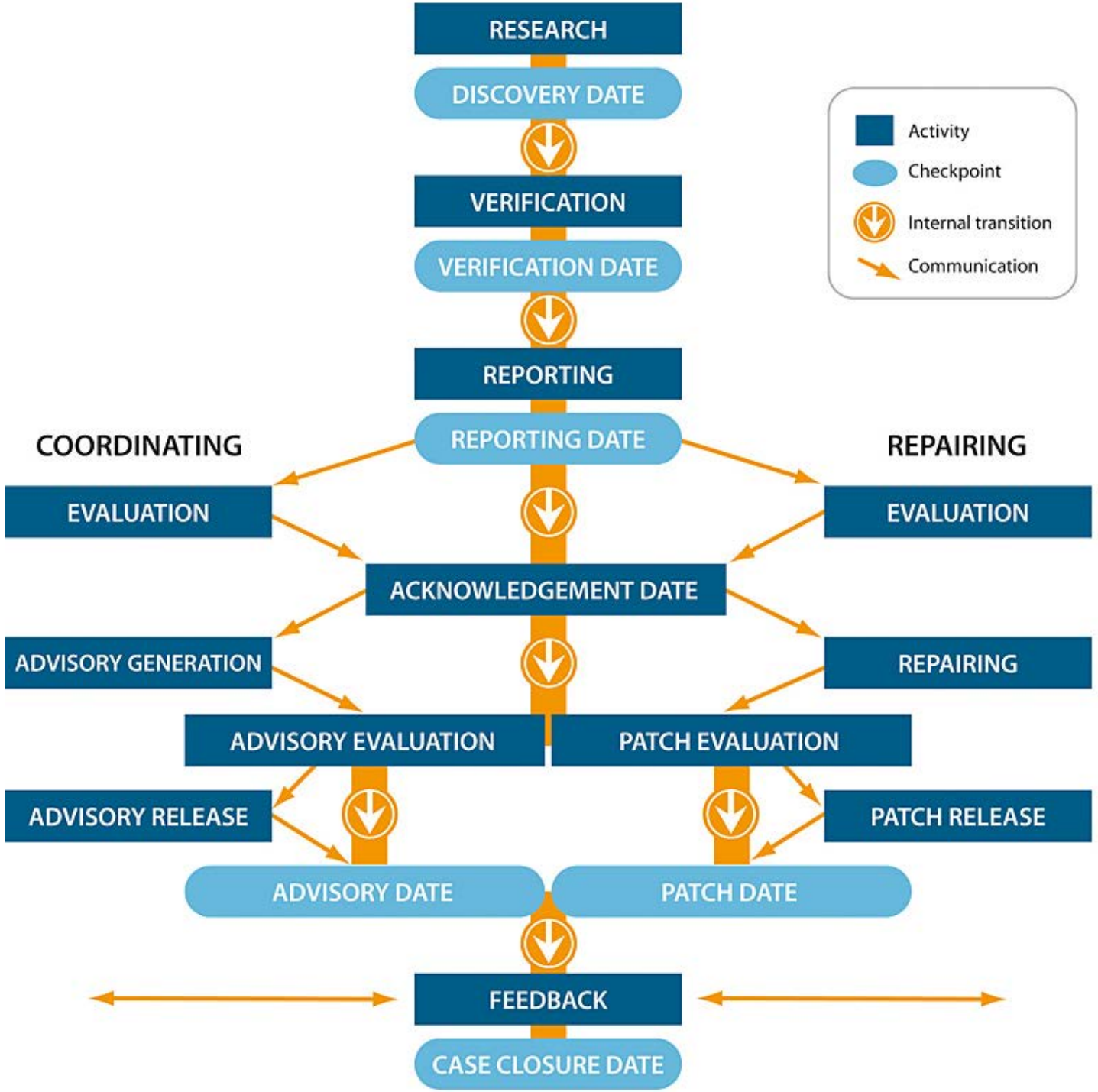
Could be used for Web Apps/APIs

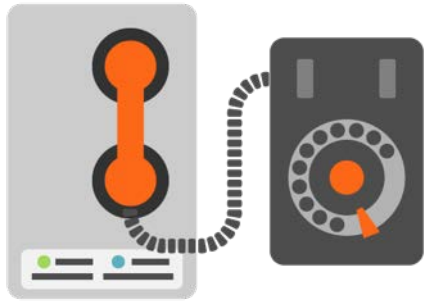
- Looking for different exceptions typically though


Training
Requirements
Design
Implementation
Verification
Release
Response
Core Security Training
Establish Security Requirements
Create Quality Gates / Bug Bars
Security & Privacy Risk Assessment
Establish Design Requirements
Analyze Attack Surface
Threat Modeling
Use Approved Tools
Deprecate Unsafe Functions
Static Analysis
Dynamic Analysis
Fuzz Testing
Attack Surface Review
Incident Response Plan
Final Security Review
Release Archive
Execute Incident Response Plan


Average Cost per Bug
X5
X4
X3
X2
X1
Total Cost of Software Flaws in Normal SDLC
= Total Area Under All Phases
P-R
E-C
I-S
C-U
R+D
15%
20%
40%
15%
10%
100%
Distribution Where Bugs Are Detected
R+D
C-U: CODING / UNIT TEST
I-S: INTEGRATION / SYSTEM TEST
E-C: EARLY CUSTOMER RELEASE / BETA
P-R: POST-RELEASE

Patch Tuesday is Expensive





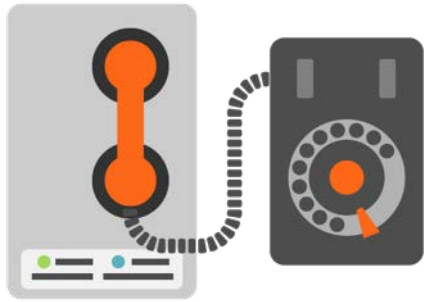
“Complete” software testing is an NP-Hard problem

- Training issue too

Take a program that can parse a one megabyte file

- How many combinations of bits?

Fuzzing hits many cases, quickly, and comes up with combinations a tester might not manually consider important



Approach

- Two main types of fuzzing
  - Mutation and Generation
  - Also called dumb and intelligent
  - “Most” of the data needs to be delivered correctly for fuzzing to work well

Data

- Files, APIs, network, reg keys, etc.
  - Integers, strings, etc.

FUZZER

SUT

GET / HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive

VALID request

VALID response

HTTP/1.1 200 OK
Date:Tue, 21 Nov 2006 19:48:44 GMT
Server:Apache/2.0.53 (Fedora)
Accept-Ranges:bytes
Transfer-Encoding:chunked
Content-Type:text/html; charset=iso-8859-1
Connection:Close

GET aAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaA HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive

ANOMALOUS request

ERROR response

HTTP/1.1 404 Not Found
Date:Tue, 21 Nov 2006 19:54:30 GMT
Server:Apache/2.0.53 (Fedora)
Content-Length:290
Content-Type:text/html; charset=iso-8859-1
Connection:Keep-Alive

HTTP Request
Hardware address length sent as 16 instead of 6 (correct for ethernet)

ANOMALY sent

ANOMALOUS response

RETURNS
Server host name of $$$$$$$$$$$$$$$$$$$$$
Memory addresses filled with $$$$$$$$ $$$$$$$$

GET http://[?aAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaAaA::0] HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Connection: Keep-Alive

ANOMALY sent

CRASH (no response)

<========[NO RESPONSE]=========

input

ANOMALIES

expose

VULNERABILITIES

FIELD LEVEL
overflows,
integer anomalies

STRUCTURAL
underflows, repetition
of elements, unexpected
elements

SEQUENCE LEVEL
out of sequence omitted/
unexpected repetition/
spamming

WHAT FUZZING FINDS

crashes

denial of service (DOS)

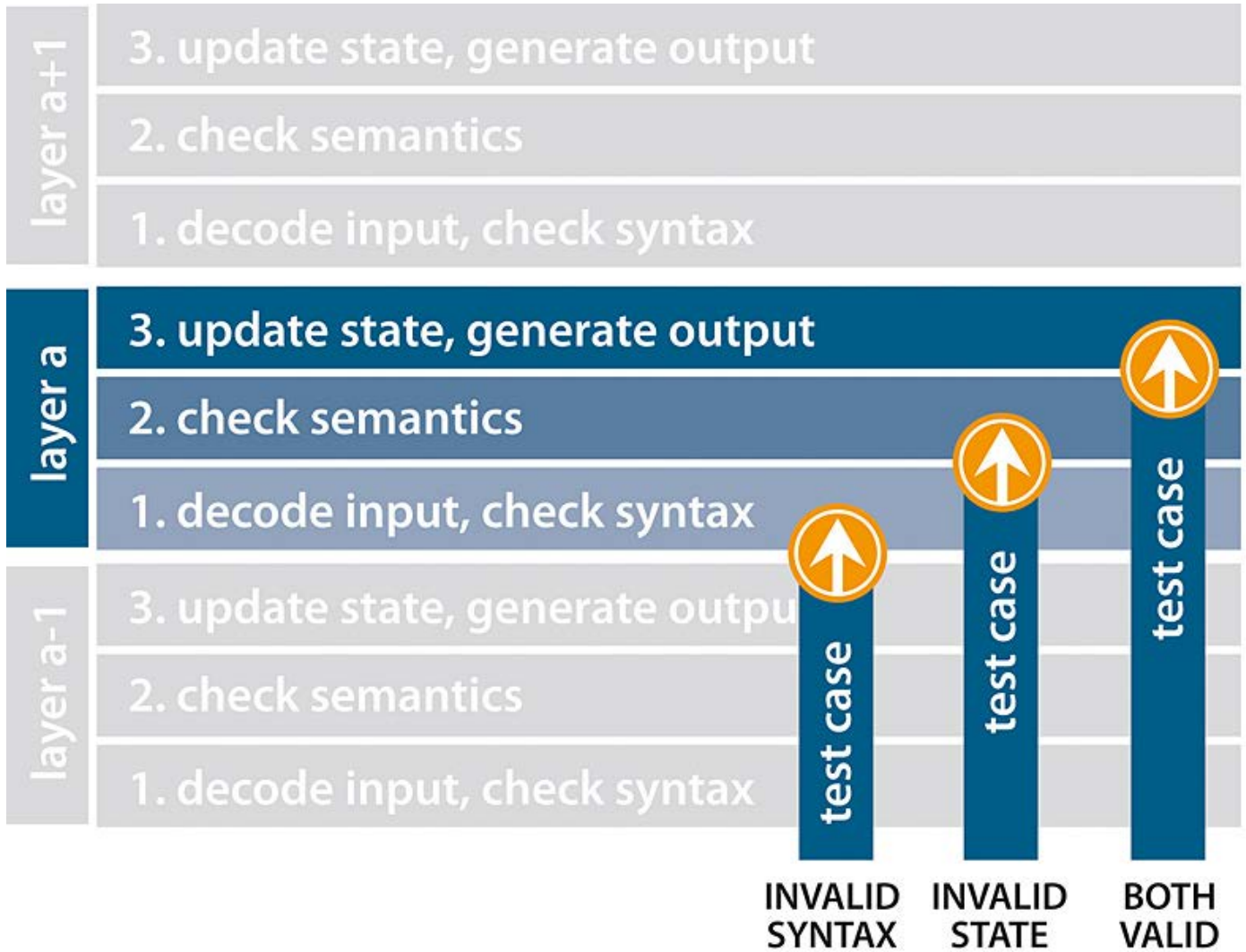
security exposures

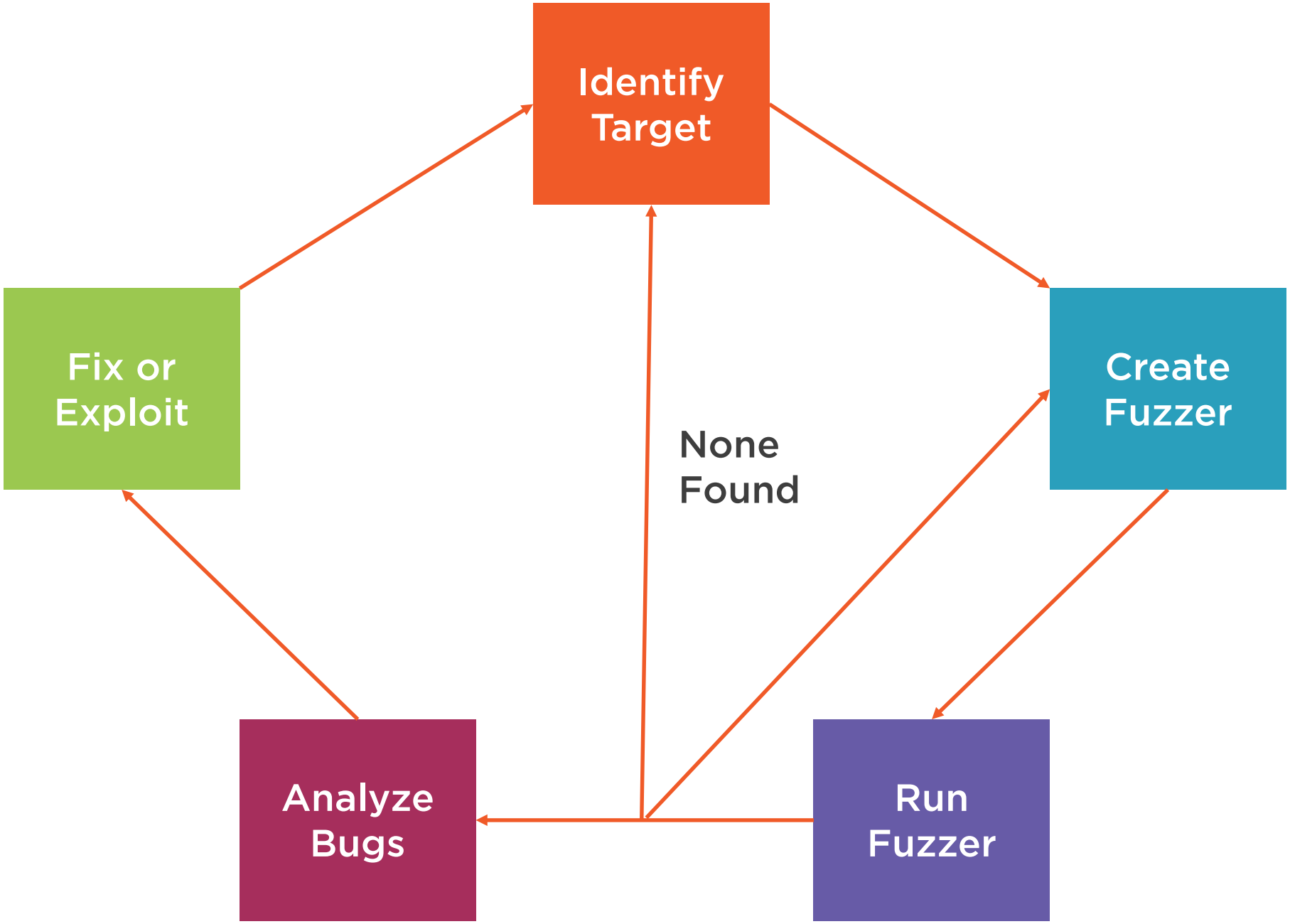
performance degration

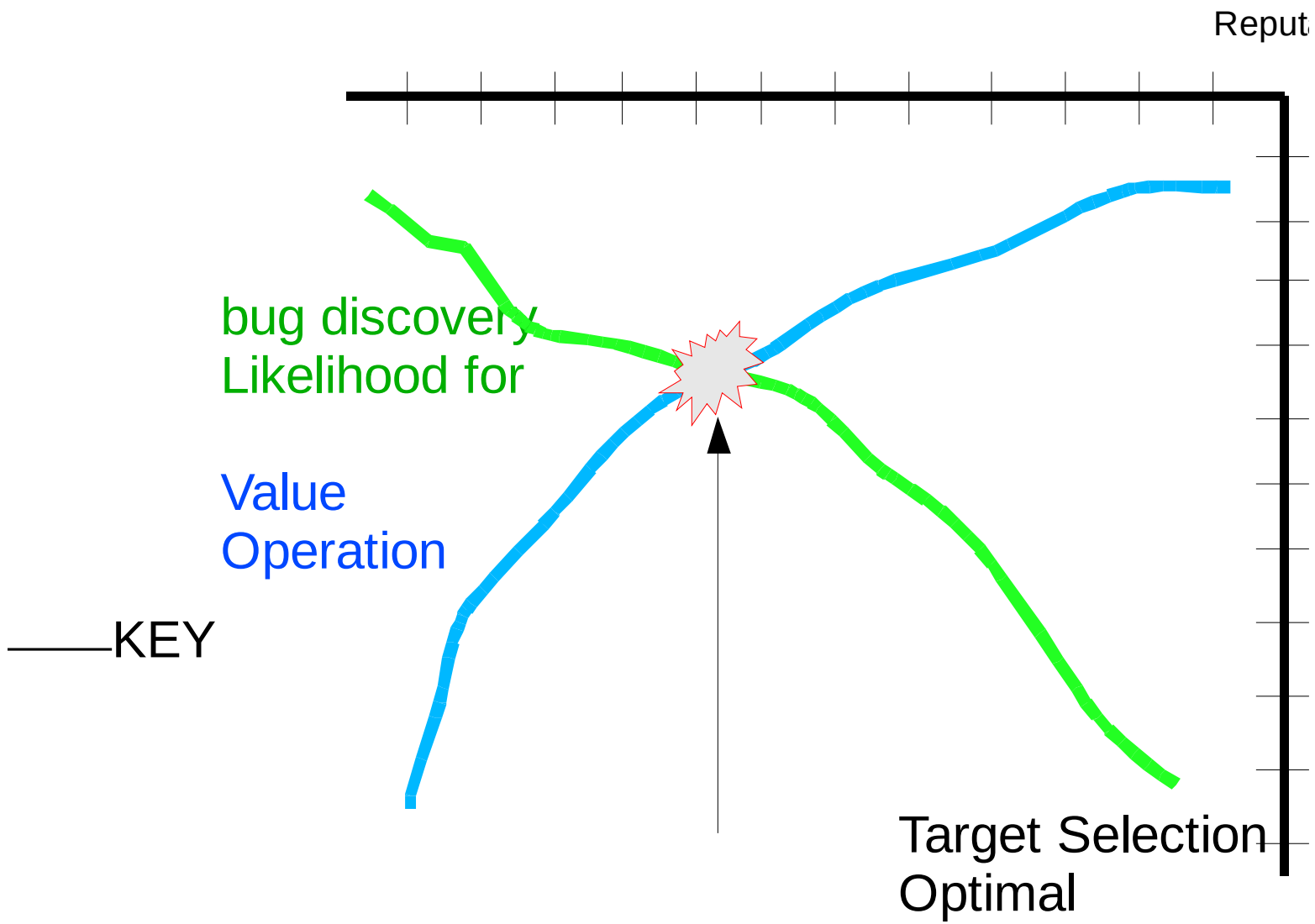
slow responses

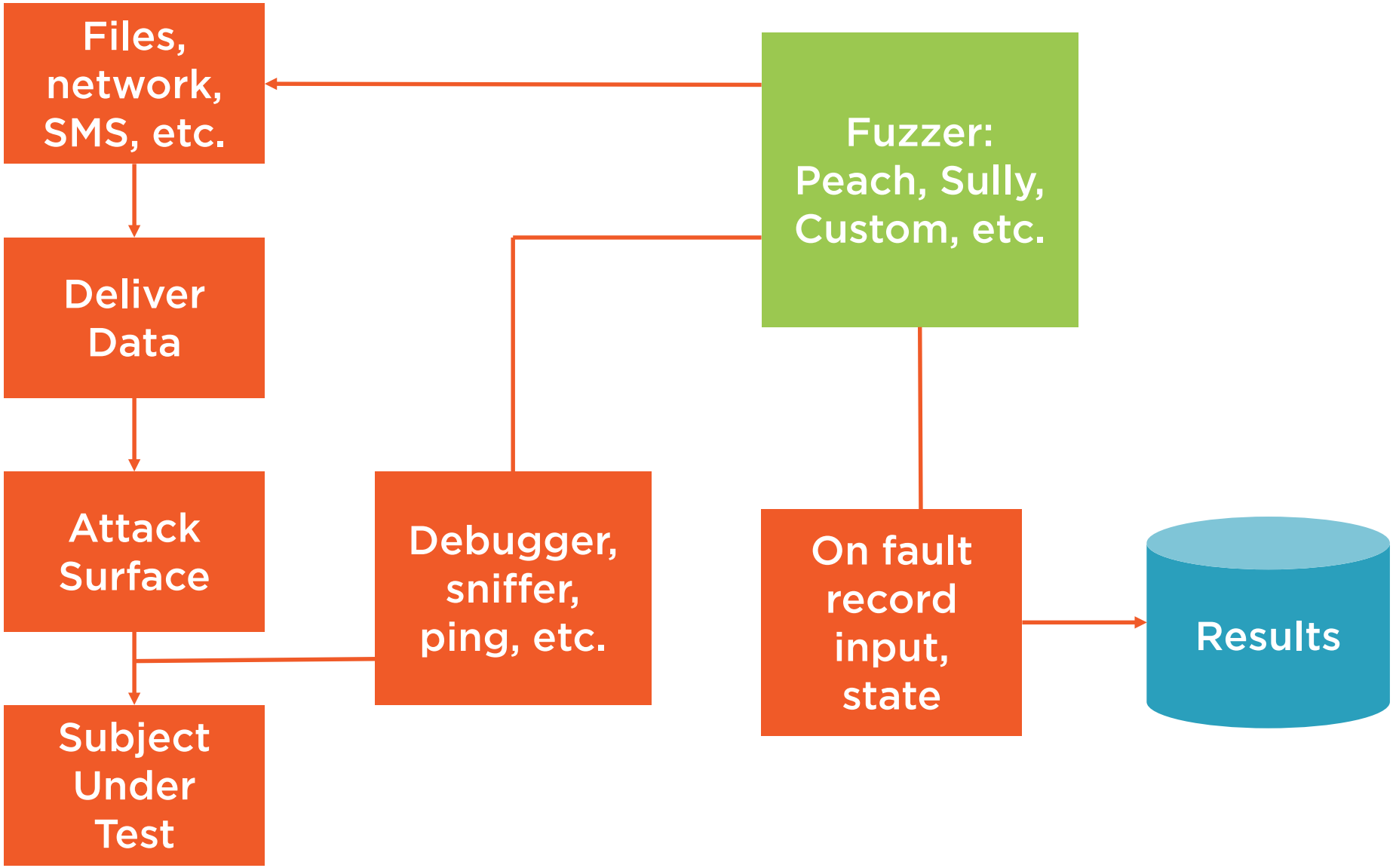
trashing

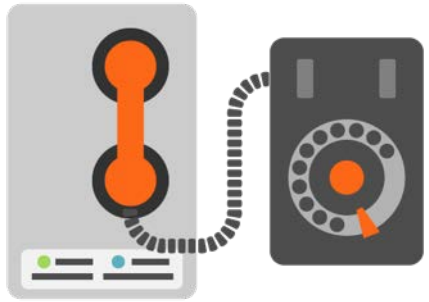
anomalous


layer a+1
3. update state, generate output
2. check semantics
1. decode input, check syntax
layer a
3. update state, generate output
2. check semantics
1. decode input, check syntax
layer a-1
3. update state, generate outpu
2. check semantics
1. decode input, check syntax
test case
test case
test case
INVALID SYNTAX
INVALID STATE
BOTH VALID


Identify Target
Create Fuzzer
Run Fuzzer
Analyze Bugs
Fix or Exploit
None Found


Reput
bug discovery
Likelihood for
Value
Operation
KEY
Target Selection
Optimal

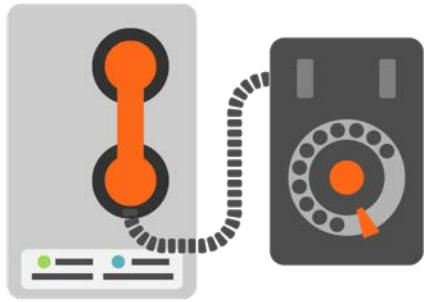

Files, network, SMS, etc.
Deliver Data
Attack Surface
Subject Under Test
Debugger, sniffer, ping, etc.
Fuzzer: Peach, Sully, Custom, etc.
On fault record input, state
Results



Mature Fuzzer

- Accurate data model
- Anomaly library
- Delivery harness
- Monitoring
- Reporting

```
"/.:/"  + "A"*5000 + "\x00\x00",
"/.../" + "A"*5000 + "\x00\x00",
"/.../.../.../.../.../.../.../.../.../.../",
"/../../../../../../../../../../../../etc/passwd",
"/../../../../../../../../../../../../boot.ini",
"..:..:..:..:..:..:..:..:..:..:..:..:..:..:",
"\\\\*",
"\\\\?\\",
"/\\" * 5000,
"/." * 5000,
"!@#$%%^#$%#$@#$%$$@#$%^^**(()",
"%01%02%03%04%0a%0d%0aADSF",
"%01%02%03@%04%0a%0d%0aADSF",
"/%00/",
"%00/",
"%00",
"%u0000",
    # format strings.
    "%n"     * 100,
    "%n"     * 500,
    "\"%n\"" * 500,
    "%s"     * 100,
    "%s"     * 500,
    "\"%s\"" * 500,
    # command injection.
    "|touch /tmp/SULLEY",
    ";touch /tmp/SULLEY;",
    "|notepad",
    ";notepad;",
    "\nnotepad\n",
    # SQL injection.
    "1;SELECT%20*",
    "'sqlattempt1",
    "(sqlattempt2)",
    "OR%201=1",
# some binary strings.
"\xde\xad\xbe\xef",
"\xde\xad\xbe\xef" * 10,
"\xde\xad\xbe\xef" * 100,
"\xde\xad\xbe\xef" * 1000,
"\xde\xad\xbe\xef" * 10000,
"\x00"             * 1000,
# miscellaneous.
"\r\n" * 100,
"<>" * 500,     # sendmail crackaddr
```

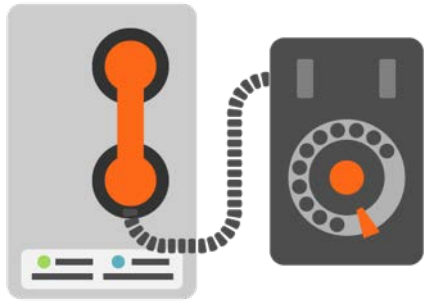


Intelligent fuzzing

- Timing, state, and format of data
  - How is it going to be parsed?

Fuzzing a binary protocol

- [Type][Len][Variable]
  - Length should “usually” be correct
  - Type should be as many as possible
  - Variable
    - Often replacement from the anomaly library



Calculate runtime

- Elements * Anomalies * Time/Run = Total Run Time
  - 20 * 1000 * 10sec = 55.5hrs
    - Can decrease with more computers

Summary

Fuzzing is an important part of native testing

Requires engineering work

Next we look deeper at mutation