Protocol Implementation Testing:
Research Challenges & Opportunities

NISCC Workshop (28 Jan 2004)

Matthew Franz (mfranz@cisco.com)

http://www.cisco.com/go/ciag

Overview

Introduction and Problem Space

Technical and Organizational Challenges

Political and Cultural Challenges

What can be done?

Intro/Background

- I’m a researcher from Cisco’s Critical Infrastructure Assurance Group (CIAG) based in Austin, Texas

  From a small team that walks the fine line of being an externally facing group that conducts “vendor neutral security research” for a large network vendor

  Have a broad charter of improving CI network/computer security for the Critical Infrastructure, but work has focused within 3 technology areas: Internet infrastructure, control systems, and security testing/methodology

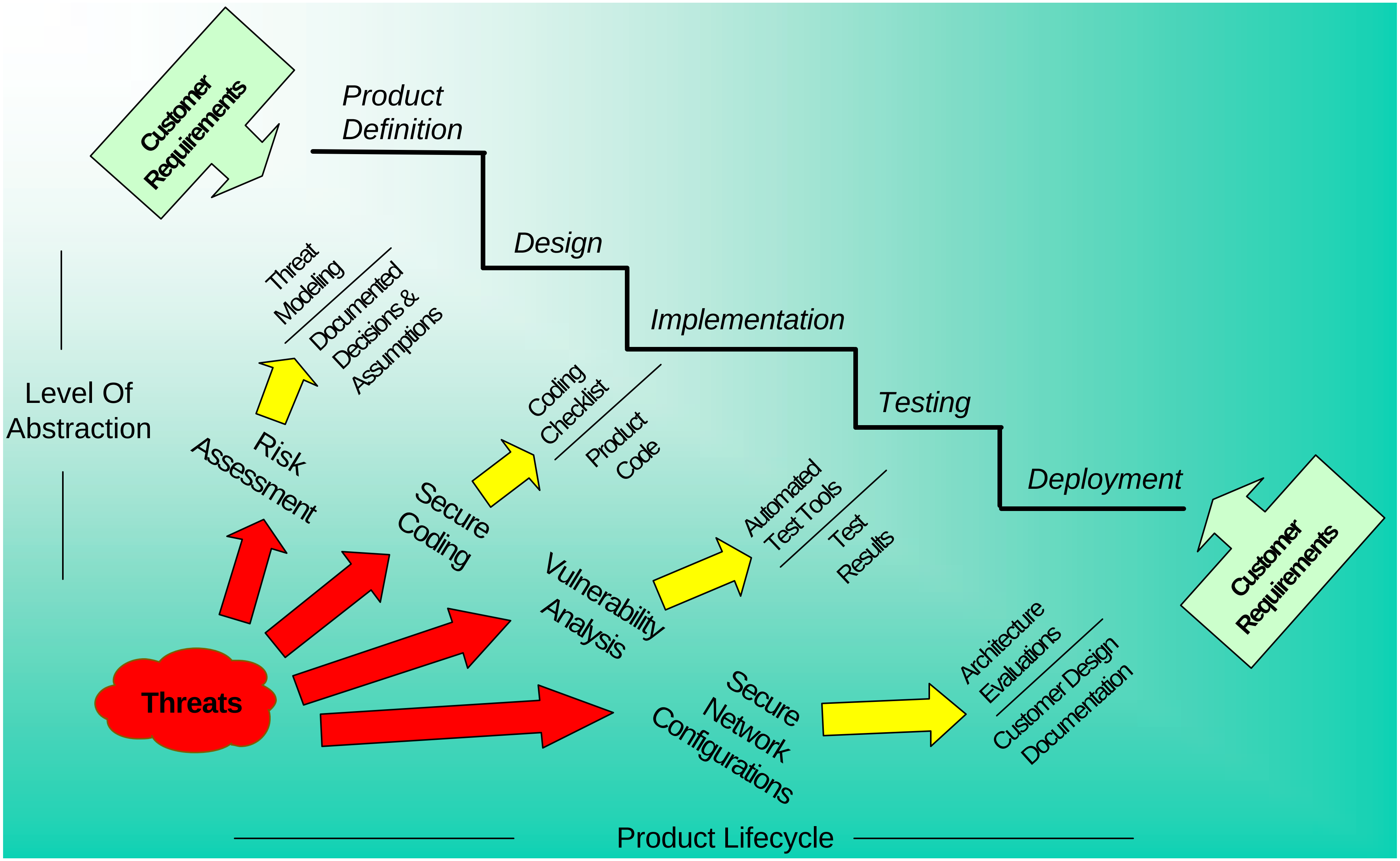
- My background

  Used to worked in small security test organization with challenging mission

  Participated in a number of cross-BU product security initiatives to integrate security into development cycle

  Designed/implemented a crude generic fuzzer in Python--mentioned in BGP prezos at BlackHat 2003 and NANOG

Securing the Development Process



Product Vulnerability/Threat Space

- **Fuzz-testing/boundary testing/protocol implementation testing is just one of several ways of attacking a device/protocol/application to discover implementation flaws**
  - Reconnaissance
  - Sniffing and Replay
  - Spoofing (valid messages)
  - Flooding (valid/invalid messages)
  - Hijacking/Man in Middle
  - Malformed Messages
  - Out of Sequence Messages

Is this still the case? Was it?

"Malformed input" is a high-level type that covers illegally formatted input. This category is poorly understood and requires research (the PROTOS project has made great strides in certain subclasses of malformed input). Advisories rarely provide the detail to precisely understand how the input is malformed. For example, consider all the nmap/Spike scans that find *some* bug, but the bug is not diagnosed fully enough to determine the exact type of input that caused the problem

Steven M. Christey (BUGTRAQ, 26 November 2002)

A Fuzzy Chronology

1990,1995,2000 – University of Wisconsin, "An Empirical Study of the Reliability of UNIX Utilities" (Miller, Fredricksen, So)

1997 – Ballista (CMU)

1998 – IP Stack Integrity Checker (ISIC) by Mike Frantzen

2000 – SPIKE by Dave Aitel

2001 – OUSPG, "A Functional Method for Assessing Protocol Implementation Security" SNMP Test Cases

2002 – SSHredder, SIP Test Cases

2003 – More test cases, SMUDGE and PIF (not released)

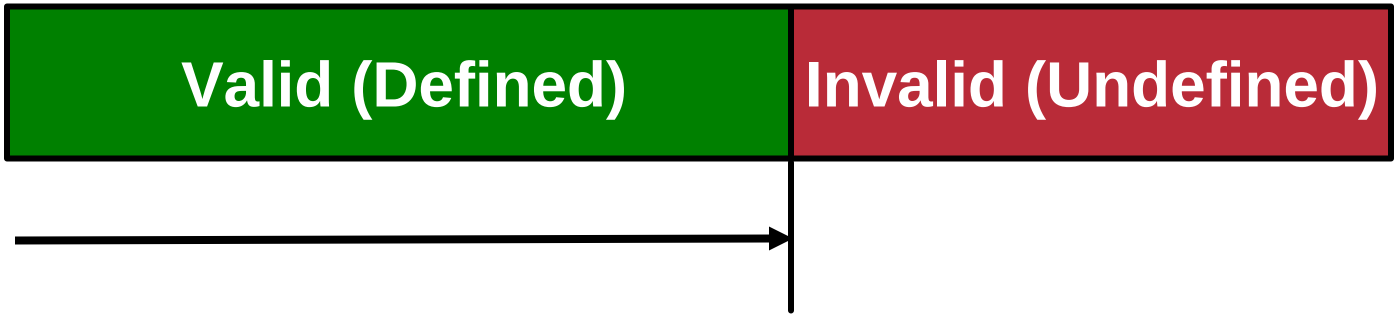
What did we get from ISIC?

- Libnet based suite of tools that can find a lot of problems in a lot of products
  - High rate of packet generation, injection
  - Invalid TCP/IP Options
  - Bad Checksums
- Basic UDP message fuzzing (udpsic 500)
- Basic IP Protocol Fuzzing (isic -P 0x51)
- Basic Ethernet (esic –p 0x1111)
- Needle in a haystack—hard to find root cause[s]
  - Was it flooding, a single packet, or multiple packets?
  - Apart from initial seed, there is no control over messages that get generated

Simple Approach to Test Case Generation

- The deeper into the message we are able to inject invalid data, the greater confidence we have in the implementation will properly process malicious input
- Goal is to find mishandling of truncated messages, incorrect length values, and illegal type codes which can lead to “unstable operation” protocol implementations

Message/Packet Depth



Valid (Defined)

Invalid (Undefined)

Example 1: Crude IKE fuzz using udpsic

```
Internet Security Association and Key Management Protocol
    Initiator cookie: 0xA5671784D770C760
    Responder cookie: 0x039210C4D8255AD0
    Next payload: Private USE (179)
    Version: 15.14
    Exchange type: DOI Specific Use (67)
    Flags
        .... ...0 = No encryption
        .... ..0. = No commit
        .... .0.. = No authentication
    Message ID: 0x2D89E563
    Length: 999130073
```

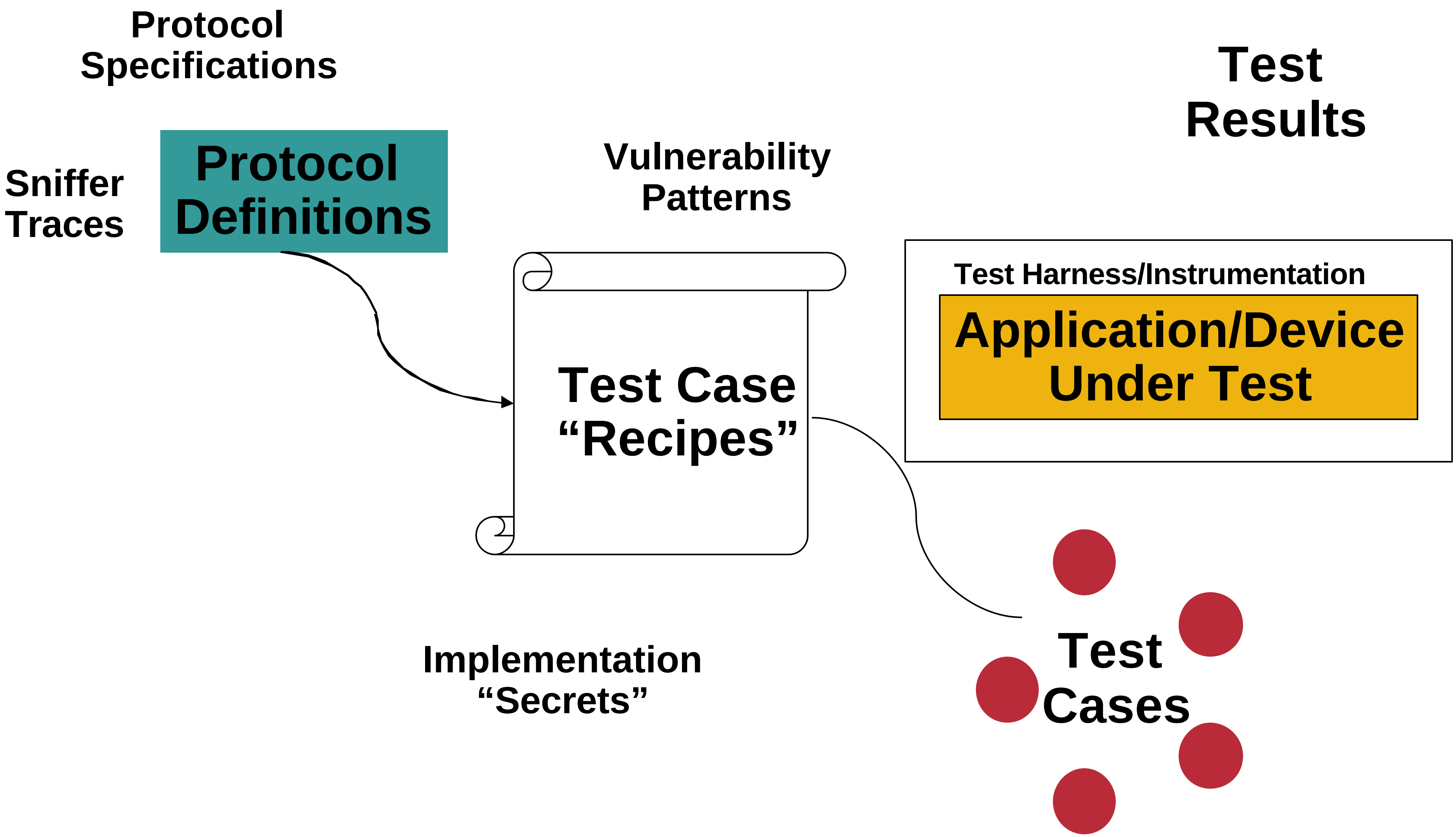
Majority of implementations would not process this message due to invalid cookie next payload, and exchange type values

Example 1: Smarter IKE Fuzz with *PIF*

```
Internet Security Association and Key Management Protocol
        Initiator cookie: 0xC73BDEFD478C8A8E
        Responder cookie: 0x0000000000000000
        Next payload: Security Association (1)
        Version: 1.0
        Exchange type: Identity Protection (Main Mode) (2)
        Flags
            .... ...0 = No encryption
            .... ..0. = No commit
            .... .0.. = No authentication
        Message ID: 0x00000000
        Length: 2618069175
        Not enough room in payload for all transforms
```

Up until the length field, this is a valid message

A Simple Multi-protocol Fuzzer



Protocol Specifications

Sniffer Traces

Protocol Definitions

Vulnerability Patterns

Test Case “Recipes”

Implementation “Secrets”

Test Results

Test Harness/Instrumentation

Application/Device Under Test

Test Cases

We know implementations are broken, so what?

- How do we start defining coverage or completeness?
  - Number of test cases
  - Packet "Depth" and "Breadth" (field values)
  - Algorithms used to generate malformed payloads
  - Richness of vulnerability "primitives"
- What else?

Does fuzz-testing really scale?

- Assuming we have decent tool suite, what are the multipliers?

  Test Cases to Inject

  Thousands to Hundreds of Thousands

  Images/Applications to be tested

  Dozens to Hundreds

  Possible Configurations of Application/Device being tested

  Several to Dozens all of which may impact test results

- Other Issues

  Who decides which protocols need testing?

  Who tests? How often?

  Who defines the protocol grammar? Who builds test cases?

  Does the cost make sense?

Sins of the Fuzzer: Caught Between Fear and Greed

- Who are the players?
  - "Pure" Academic Researchers
  - Various Ad Hoc/Open Source Efforts
  - Application Security Vendors
  - Product Vendors?
- Collaborative research may be even trickier than vulnerability disclosure
  - How to you make responsible decisions based on vulnerability potential?
  - To really tackle the coverage issue, don't we need the source of the applications/implementations under test?
  - We need more than just test-cases and test-results, but this compromises the business model of application security/tool vendors
  - Pervasive attitude of "do we really want to go there?"

Baby Steps Forward

- Protocol implementation testing is still a niche research area, although there is lots of ad hoc activity (and confusion)
- OUSPG work was foundational, but writing fuzzing tools has become a cottage industry

    Need a common definition or taxonomy?

    Analysis of different approaches—are we all really talking about the same thing?

    Are protocol grammars created equal?

    What approaches apply to all protocols and which are protocol specific?

    How do we compare the results of multiple fuzzers?

- Measurement/automation is a different set of problem