A Linguistic Platform for Threat Development

Ben Kurtz

Imperfect Networks

Introduction

By applying programming language theory to the development of new networks attacks, we can create next-generation platforms capable of quickly handling arbitrary protocols and hardware, and exponentially reducing threat development time.

Overview

- Motivation
- Threat Testing
- Goals of a new system
- Applications of Programming Language Theory

Motivation

- We want to break stuff!
  - Easier
  - Faster
  - Better
- Minimize threat development turnaround time

Motivation (cont.)

Why do we want to break stuff?

- Network Equipment/Service Testing
- Malice

IDS systems eventually have to be tested…

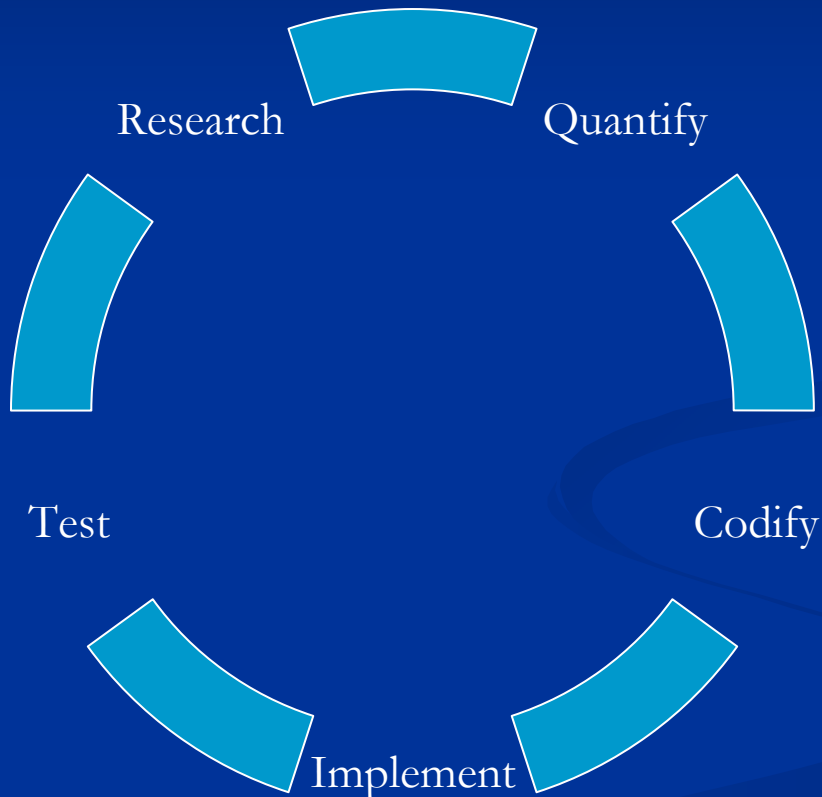
And they must be tested with REAL threats!

Motivation (cont.)

You can't have 0-day protection
without 0-day testing!

This requires same-day threat testing.

The Threat Testing Cycle



Window of Vulnerability

If it takes two weeks to complete the cycle,

Your Window of Vulnerability is two weeks!

Options for Improvement

- Option 1
  - Hire more people
  - Work in parallel
  - Still limited by current time-consuming tools
  - Window of Vulnerability remains the same
- Option 2
  - Automate the common, repetitive tasks associated with threat development.
  - Better tools
  - Only deal with the unique aspects of a threat.

So…

Hiring more people would be great!

But it's not going to happen.

We need better tools.

Threat Platforms

- We have some threat platforms available to us already…
  - Metasploit
  - Nessus
  - … Perl
- Versatility
- Speed
- Real, Live Threats

Design Goals

- One tool to generate all possible threats
- Platform/Target independence
- PCAP import
- Multi-source traffic playback
- Simulation and Testing
- Unified platform means unified reporting

Programming Language Theory

- Grammars
  - Rules that describe a language
  - Serve dual purposes:
    - Generation – to make valid expressions
    - Validation – to determine validity of an expression
- Extended Backus-Naur Form

Compilers

- Translates one language to another
- Stages of compilation
  - Lex – syntax
  - Parse – semantics
  - Intermediate Representation
  - Code Generation

Parser Generators

- Also called Compiler Compilers
- Often overlooked, but powerful
- PG's are compilers that can dynamically redefine the input and output grammars
- Specify each network protocol as a grammar, and feed it into a domain-specific parser generator for network traffic
- EBNF specification of protocols

EBNF

- A set of production rules for a grammar, including a starting rule.
- Each rule is comprised of:
  - Terminals – which are strings
  - Non-Terminals – which are pointers to other rules.
- Special symbols, similar to regular expressions
  - * operator – repeat 0 or more times
  - + operator – repeat 1 or more times
  - | operator – either/or

EBNF (cont.)

- Sample Grammar for Single Digit Addition:
  - S -> E
  - E -> E ‘+’ E | D
  - D -> ‘0’ | ‘1’ | ‘2’ | ‘3’ | ‘4’ | ‘5’ | ‘6’ | ‘7’ | etc.
- Valid strings:
  - 5
  - 7 + 8
  - 1 + 7 + 9

PLT (cont.)

- Normal Grammars
  - Simplest subset, Simple definitions
  - Have exactly one non-terminal per RHS of each rule
  - Simple parsing algorithms
- Bold statement: All network protocols can be defined with Normal Grammars!
- Erm… Mostly
- Checksums and Length Fields

Protocols in EBNF

- Each protocol is an ordered list of fields.
- Protocols that allow encapsulated protocols have a Payload, a special field akin to a non-terminal.
- Some protocols are more dynamic
- EBNF can handle dynamism (such as ICMP)

Ethernet Example

- Start -> ETH
- ETH -> srcMAC destMAC pktType Payload
- srcMAC -> MACAddress
- destMAC -> MACAddress
- pktType -> 2BytesHex
- Payload -> Any Other Protocol

Protocol Description

- Choosing the Format
- Writing the Grammar
  - RFC Block Diagrams
  - Each non-terminal is handled separately
- Handling Typed Data
- Accommodating PCAP Imports
- Type-Length-Value (TLV) Fields

Threat Description

- Format of Threats
- Metadata
- Named Variables
- Functions
- Lists
- Unrolling Ambiguous Iterative Behavior

Some Useful Functions

- Range
- Random
- Random String
- Homogenous String of Length X
- Checksums

Importing from PCAP

- Grammar-based PCAP decomposition
- Translating using Protocol Definitions
- Multi-sourced PCAP files
- Edit your imported PCAP for playback

Binding and Playback

- Pre-compilation of Threats (Threat Binding)
- Threat Engines
  - Functions
  - Checksums
  - Throughput
- Distributed design
- Multi-Source Traffic Playback

Conclusions

- Threat Development and Delivery Platforms based on Parser Generators have several advantages:
  - Speed of Development
  - Live Testing
  - PCAP Import and Playback
  - Platform and Protocol Independence

Q & A

At this time, I'd like
to open the floor up for
questions.