



Packet Trace Manipulation Framework for Test Labs

Holger Dreger TU München

Andy Rupp Ruhr-Universität Bochum

Anja Feldmann TU München

Robin Sommer TU München



Motivation

- Deterministically test real network components
 - Use traces → fixed characteristics
 - Use synthetic traffic → limited variability
- Our approach
 - Use synthetically modified real traces
- Questions:
 - Still realistic?
 - What does "realistic" mean for a specific application?



Outline

- Application Scenario
- Trace Manipulation Operations
- Manipulation Framework
- Summary



Application Scenario

- Test NIDS: insert one trace (e.g. attack) into another (e.g. background traffic)
- Problem: Traces from different environments
- Problem: packet sequences that real endsystems would not produce: "artifacts"
 - Can be detected by protocol or statistical analysis
 - Example: Exceeding link bandwidth



Trace Manipulation - Basic Operations

- Split complex operation "insert" into basic operations
 - Understand "impact" of basic operations
- Adapt
 - Adapt packet headers to test-environment
- Merge
 - Merge one trace with another chronologically
- Scale
 - Compress or stretch a flow or trace
- Remove/Duplicate
 - Remove / duplicate single packets or flows
- Move
 - Displace single packets or flows



Artifacts caused by Basic Operations

■ Network layer

- Statistical characteristics e.g. pkts/s change
- No artifacts perceivable using protocol analysis

■ Transport layer

- TCP analysis usually reveals operations on single packets
- Possibly destroy TCP specific interaction like congestion control
- Stretching flows may result in packet gaps that are interpreted as loss by an actual end-system.



Artifacts caused by Basic Operations (2)

- Application Layer

- Merge does not cause artifacts if address spaces are disjoint
- All others: disturb relationship between flows
e.g. remove FTP control connection



Manipulation Framework

- Framework to build arbitrary filter networks
- Basic operations implemented as plug-ins
- Current plug-ins work on pcap Traces
- Configuration language uses plug-ins as primitives
- → flexible and extendable System



Example

```
Components {
  # Background Traffic Source
  Source bgt_source {
    type PCapFileSource;
    Config { pcap_src_list = "bgt_1"; }
  }
  # Attack Traffic Source
  Source attack_source {
    type PCapFileSource;
    Config {
      pcap_src_list = "iis_dt [2.0, 5.0]";
    }
  }
  IComponent spoofer {
    type IPSpoofers;
    Config {
      spoofing_rules =
        "dst port 80 -> src net 131.152.123.0/24
         dst host 134.96.223.242";
    }
  }
}
```



Summary

- Artifacts can not be avoided entirely
- Relevance of artifacts depends on application
- Flexible tool for complex yet well defined trace manipulations



Thank you

- Software prototype and plug-in package available at:

<http://www.net.in.tum.de/~rdc/>