Detection-Based Cybersecurity: Specification-Based Detection

CS 459/559: Science of Cyber Security
16th Lecture

Instructor:

Guanhua Yan

Agenda

- ~~Quiz 1: September 29 (closed book)~~
- ~~Project 1 (offense): October 10~~
- Quiz 2: November 12
- Presentations: 11/17, 11/19, 11/24, 12/1, 12/3
- CTF competition: November 26
- Project 2 (defense): December 5
- Final report: December 15

Outline

- What is specification-based detection?
- Bro/Zeek
- CTF5G demo (by TA Srinidhi)

What is specification-based detection?

Specification-based detection

- **Specification-based detection** leverages the **specification** of a system, which describes the expected behavior of the system.

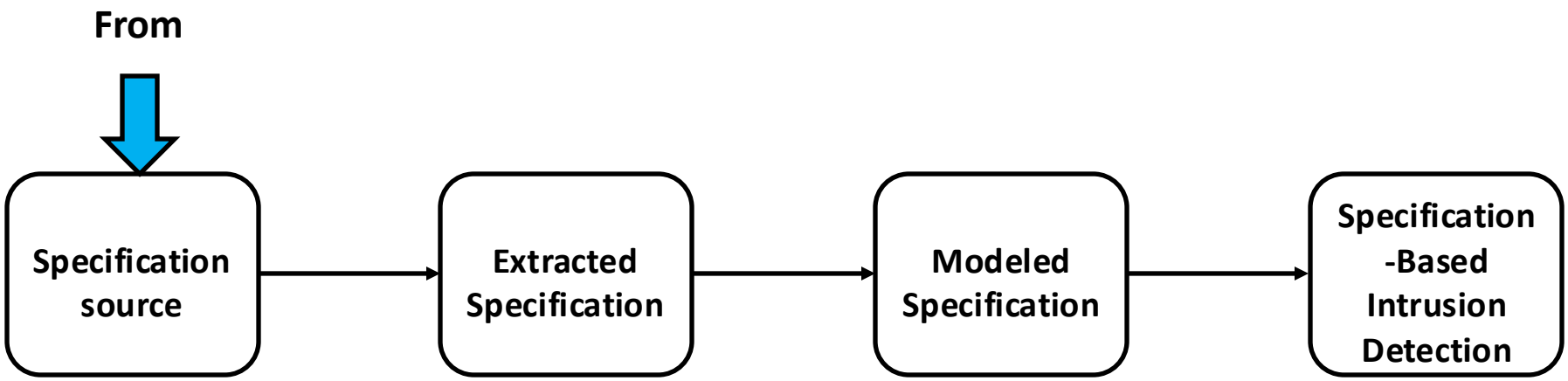
- Any **deviation** of the system operations from the defined correct behavior is **flagged as a security violation**.

Process of specification-based IDS

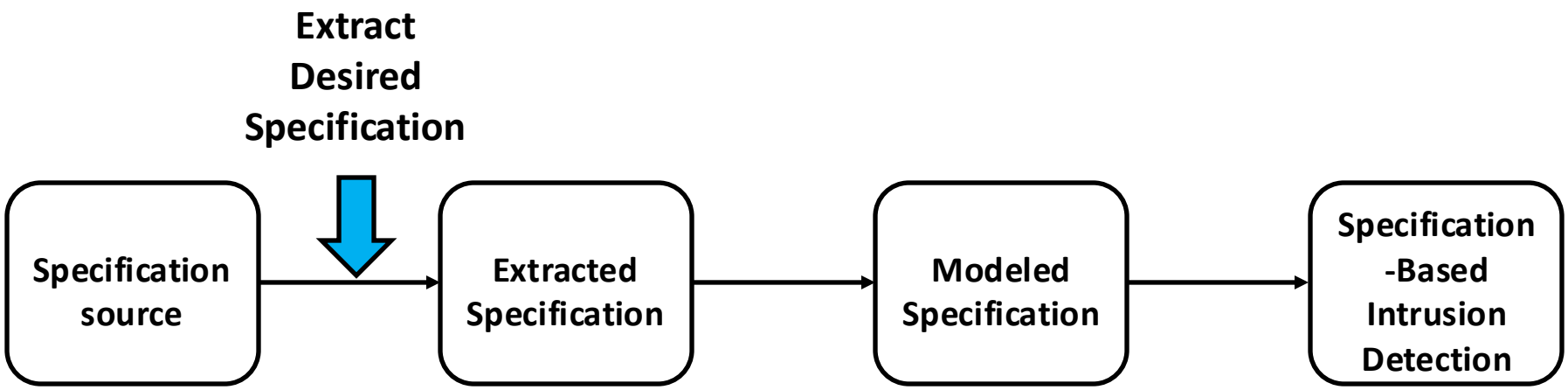
In general, the specification-based intrusion detection process involves the use of **a specification source** to extract the **expected behavior of a system**, which in turn is **modelled**.

A **detection mechanism** is then applied to the modelled specification for monitoring the system behavior for any deviation.

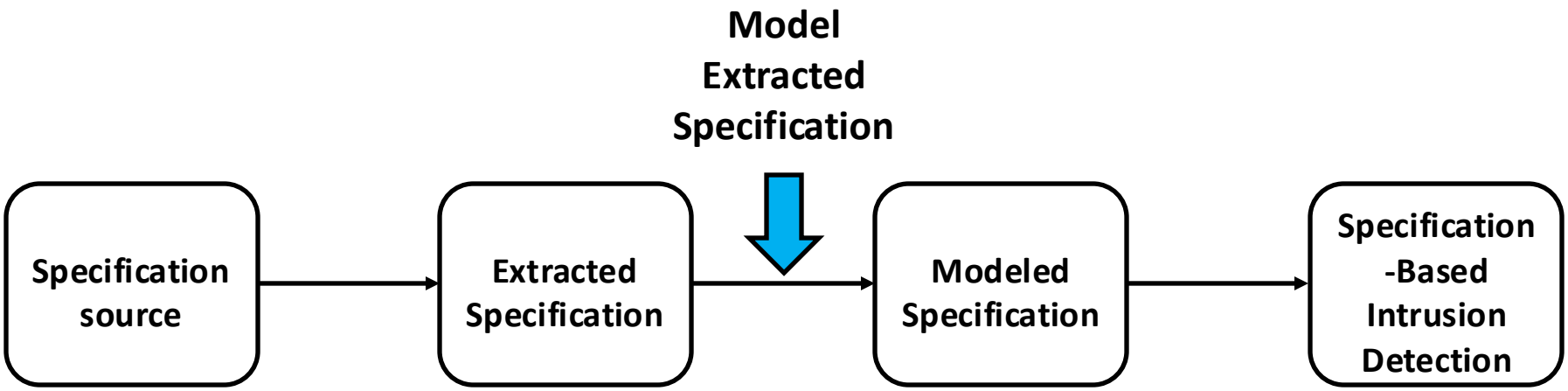
Workflow of specification-based detection



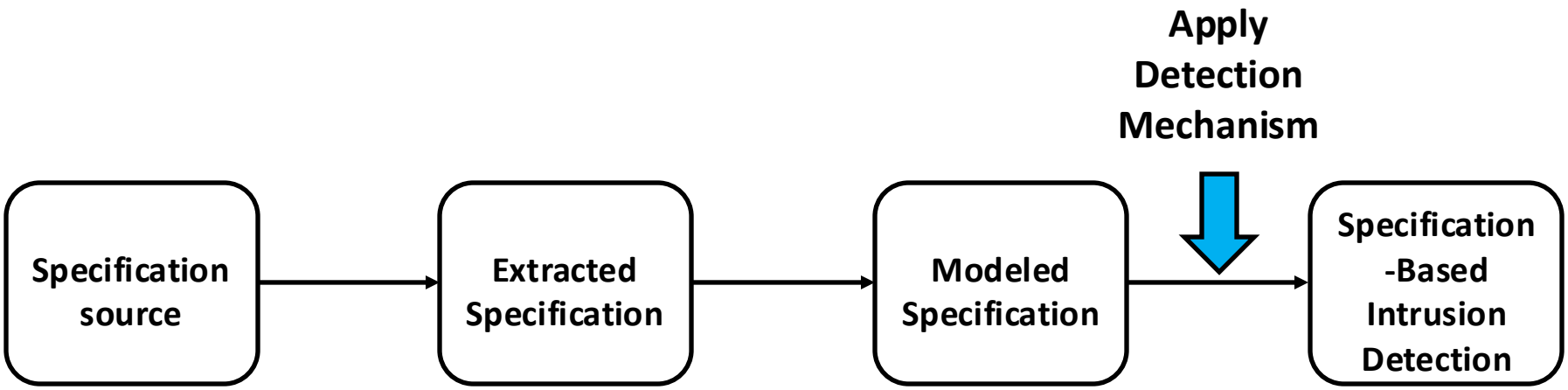
Workflow of specification-based detection



Workflow of specification-based detection



Workflow of specification-based detection



Why specification-based IDS?

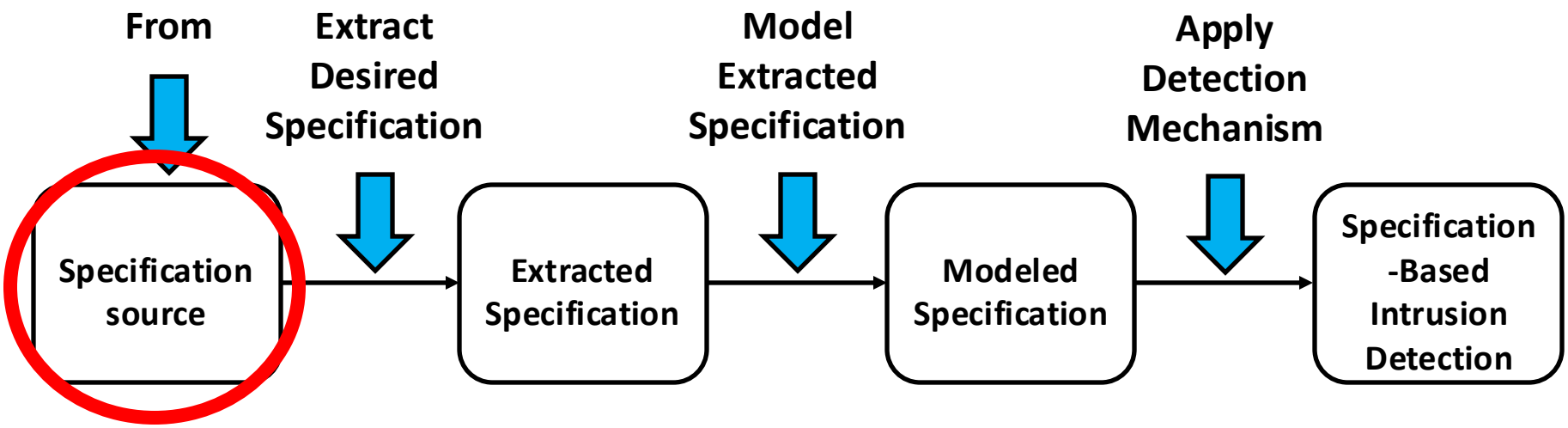
- Even though **anomaly detection** is able to detect novel attacks, it suffers from a high rate of false alarm because unseen legitimate system behaviors are classified as anomalies.

- **Signature-based detection (sometimes also called misuse detection)**, on the other hand, does not generate false alarms but it is unable to detect novel attacks.

Advantages of specification-based IDS

- Specification-based detection combines the advantages of both approaches.
  - **Its false positive rate is similar to misuse detection** as it does not generate false alarms when unusual system behaviors are discovered.
  - **Like anomaly detection, specification-based intrusion detection is able to detect novel attacks** because it detects attacks as deviations from the defined correct system behaviors.

Workflow of specification-based detection



Specification source

- Specification source refers to how to obtain the correct system behavior.

- Three major specification sources of specification-based intrusion detection techniques:
  - Protocol specification
  - Reference model
  - Observed behavior from target system

Protocol specification

- Protocol specification is a formal document that defines the expected behavior of a system.

- **Protocols considered in specification-based IDS:**
  - AODV (Ad hoc On-demand Multipath Distance Vector)
  - IEEE 802.11 protocol and EAP (Extensible Authentication Protocol)
  - DNP3 (Distributed Network Protocol 3): a set of communications protocols used between components in process automation systems
  - C12.22 standard protocol (American National Standard for Protocol Specification for Interfacing to Data Communication Networks)
  - CAN (Controller Area Network)

Reference model

- **A reference model is an abstract framework that includes a minimal set of concepts, axioms (rules) and relationships and it is independent of standards, technologies or other concrete characteristics.**

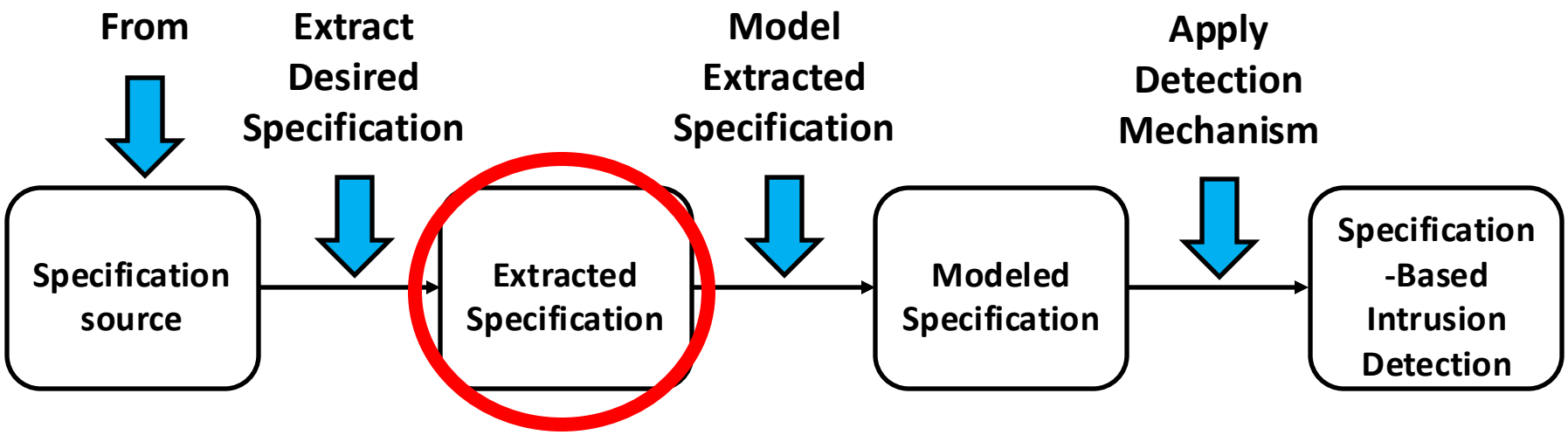
- **Examples:**
  - Reference model of a modern electrical grid
  - Reference model of unmanned air vehicles
  - Reference model of medical cyber-physical system

Observed behavior from target system

- The target system is monitored during its normal operation and such knowledge is used to specify its correct behavior.

- Examples:
  - Audit logs from electric power systems
  - SCADA (Supervisory Control and Data Acquisition) system, which is a computerized system that monitors, controls, and analyzes industrial processes and machines.

Workflow of specification-based detection

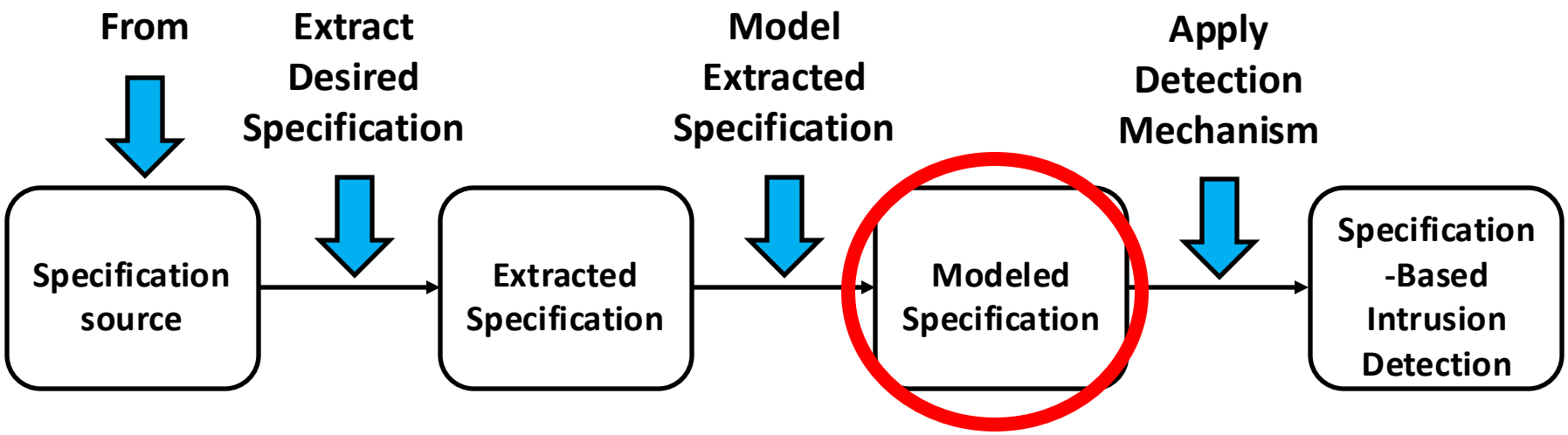


Specification extraction

- **Manual:** manual extraction of the correct system behavior from the specification source
  - Expensive and tedious
  - Error prone

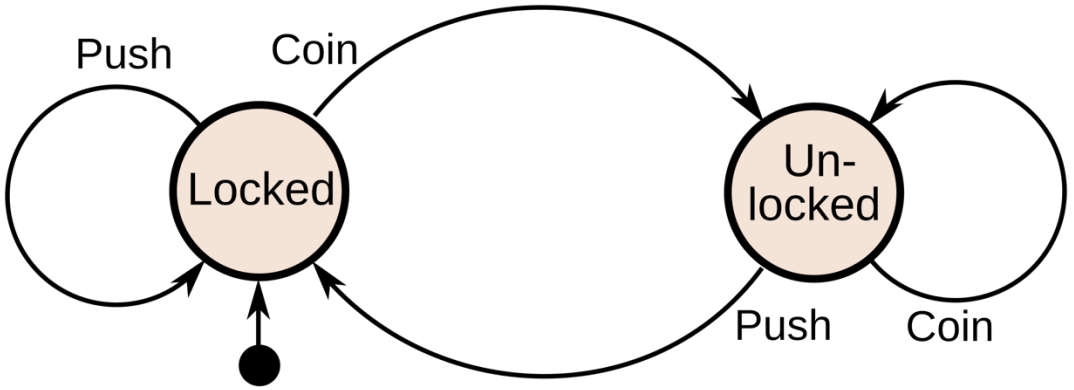
- **Automated**
  - Automated extraction of behavior rules of IoT device using the operational profile
  - NLP (Natural Language Processing)-based

Workflow of specification-based detection

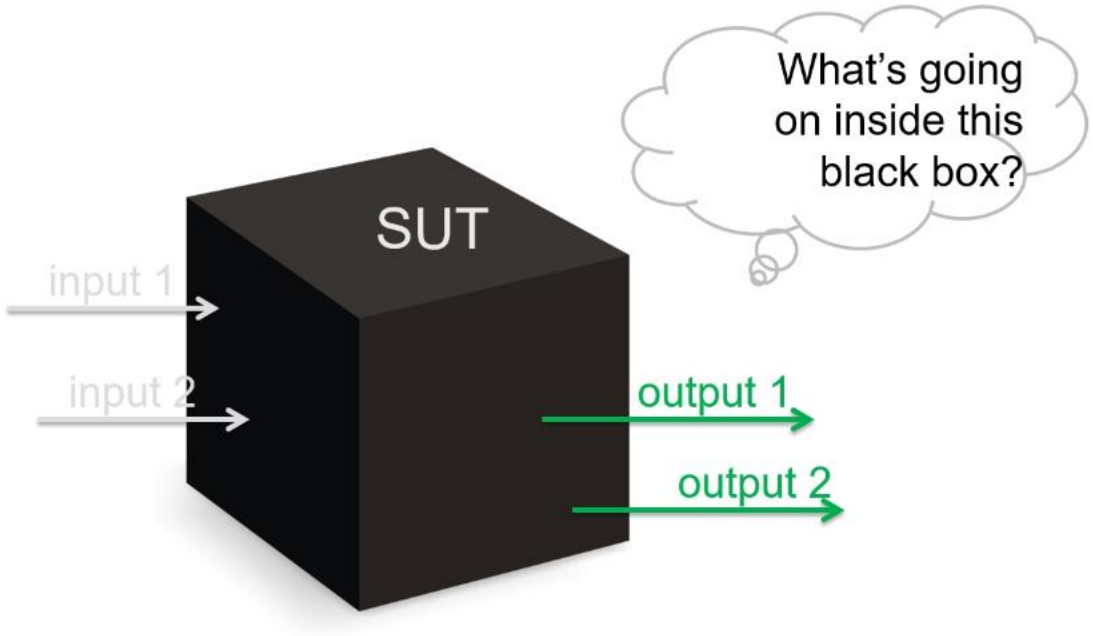


(1) State-based modeling

- Finite state machines (FSMs) are commonly used in formal protocol specifications.
- FSM is an abstract machine that can be in exactly one of a finite number of states at any given time.
- FSM: DFA (Deterministic Finite Automata) vs. NFA (Nondeterministic Finite Automata)



FSM inference

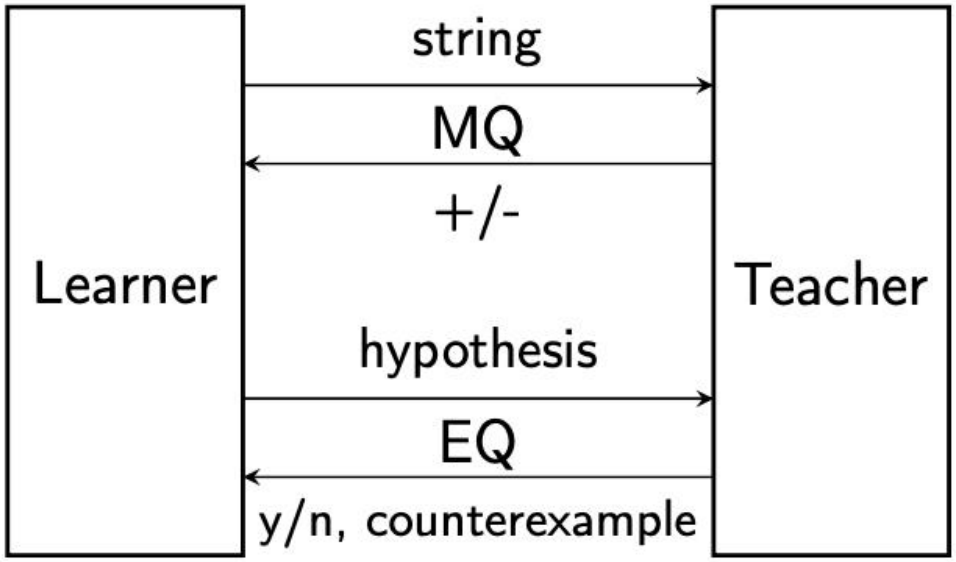


We assume SUT behaves deterministically and can be reset.

SUT: System Under Test

Minimally adequate teacher

Learning framework proposed by Angluin (1987):



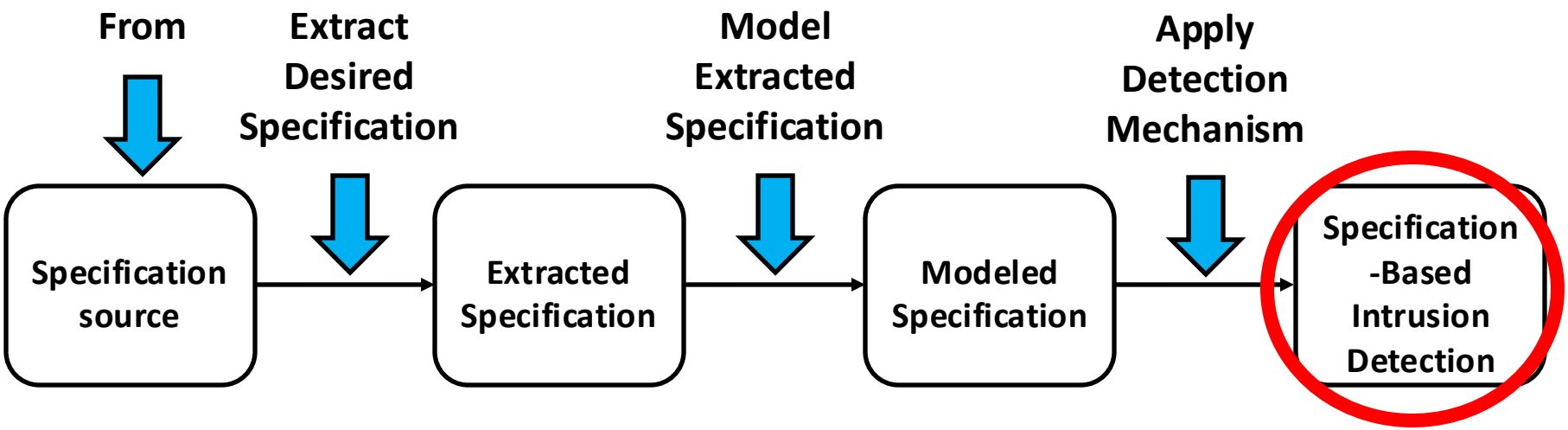
Learner asks membership queries and equivalence queries

(2) Specific specification languages

- **Bro, now known as Zeek, is a popular specification-based intrusion detection system.**
  - It has its own policy scripts

- NS-2/3, an open- source event-driven simulator, can be used for modelling the dynamic nature of communication networks

Workflow of specification-based detection



Detection methods

- **State-based approach:** The desired state of the system is defined using the specification source that have been extracted and modelled. The goal of the detection mechanism is to detect any deviation from the desired state.

- **Transition-based approach**: The detection of malicious behavior can also be accomplished by monitoring the transition between states.

- **Trace-based approach**: **Attacks are detected by monitoring whether the execution traces of the target system deviates from the specification.**

Detection methods

- **State-based approach:** The desired state of the system is defined using the specification source that have been extracted and modelled. The goal of the detection mechanism is to detect any deviation from the desired state.

- **Transition-based approach**: The detection of malicious behavior can also be accomplished by monitoring the transition between states.

- **Trace-based approach**: **Attacks are detected by monitoring whether the execution traces of the target system deviates from the specification.**

Detection methods

- **State-based approach:** The desired state of the system is defined using the specification source that have been extracted and modelled. The goal of the detection mechanism is to detect any deviation from the desired state.

- **Transition-based approach**: The detection of malicious behavior can also be accomplished by monitoring the transition between states.

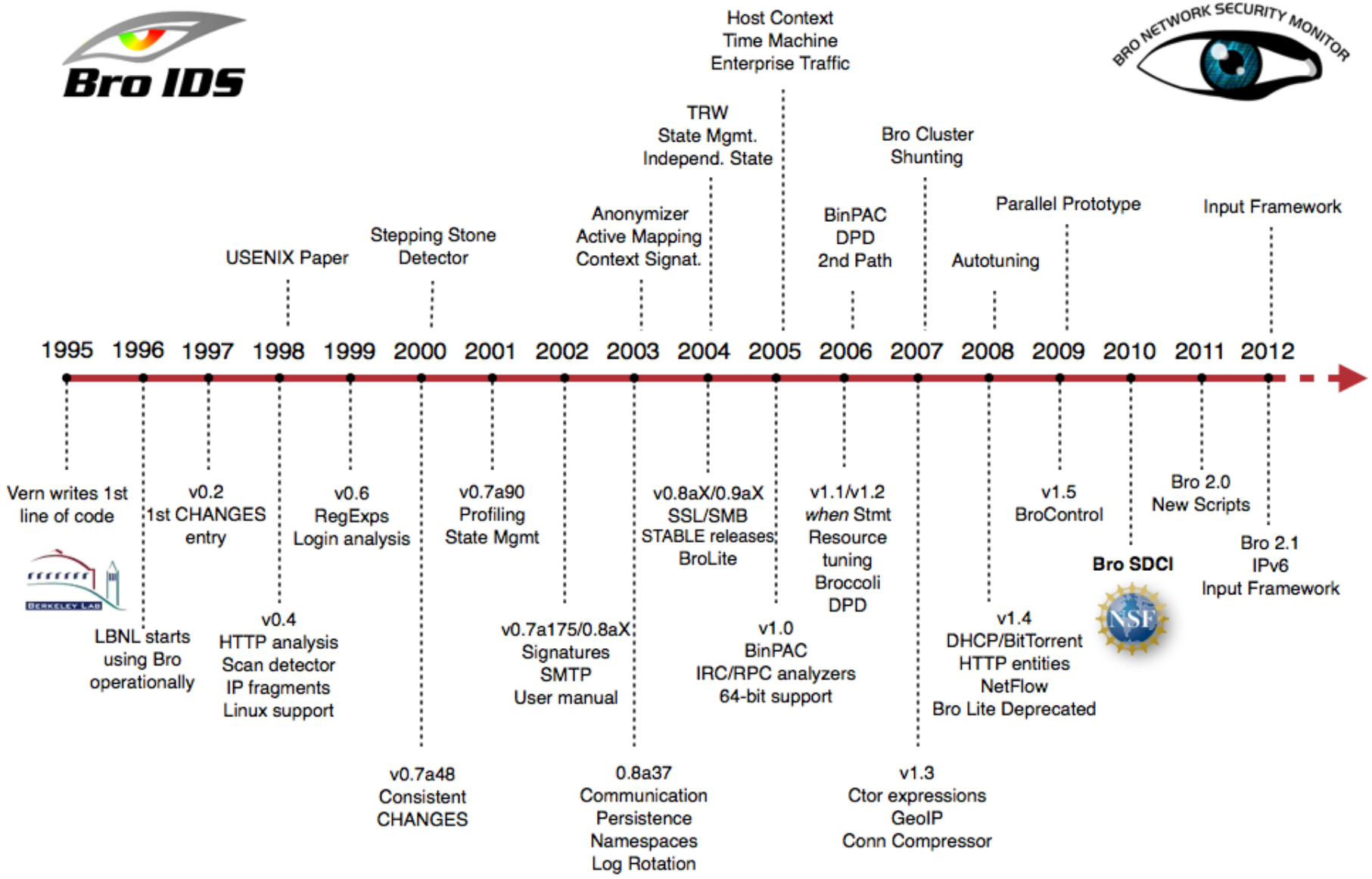
- **Trace-based approach**: **Attacks are detected by monitoring whether the execution traces of the target system deviates from the specification.**

The Bro Network Intrusion Detection System

Robin Sommer
Lawrence Berkeley National Laboratory

rsommer@lbl.gov
http://www.icir.org

UC Computing Services Conference 2007


Bro IDS
BRO NETWORK SECURITY MONITOR
Host Context
Time Machine
Enterprise Traffic
TRW
State Mgmt.
Independ. State
Bro Cluster
Shunting
Anonymizer
Active Mapping
Context Signat.
BinPAC
DPD
2nd Path
Parallel Prototype
Input Framework
USENIX Paper
Stepping Stone
Detector
Autotuning
1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
Vern writes 1st line of code
LBNL starts using Bro operationally
v0.2
1st CHANGES entry
v0.4
HTTP analysis
Scan detector
IP fragments
Linux support
v0.6
RegExps
Login analysis
v0.7a48
Consistent
CHANGES
v0.7a90
Profiling
State Mgmt
v0.7a175/0.8aX:
Signatures
SMTP
User manual
0.8a37
Communication
Persistence
Namespaces
Log Rotation
v0.8aX/0.9aX
SSL/SMB
STABLE releases
BroLite
v1.0
BinPAC
IRC/RPC analyzers
64-bit support
v1.1/v1.2
when Stmt
Resource
tuning
Broccoli
DPD
v1.3
Ctor expressions
GeoIP
Conn Compressor
v1.4
DHCP/BitTorrent
HTTP entities
NetFlow
Bro Lite Deprecated
v1.5
BroControl
Bro SDCI
NSF
Bro 2.0
New Scripts
Bro 2.1
IPv6
Input Framework

"Who's Using It?"

Installations across the US

Universities
Research Labs
Supercomputing Centers
Government Organizations
Fortune 50 Enterprises

Examples

Lawrence Berkeley National Lab
National Center for Supercomputing Applications
Indiana University
General Electric
Mozilla Corporation
... and many more sites I can't talk about.

Fully integrated into *Security Onion*

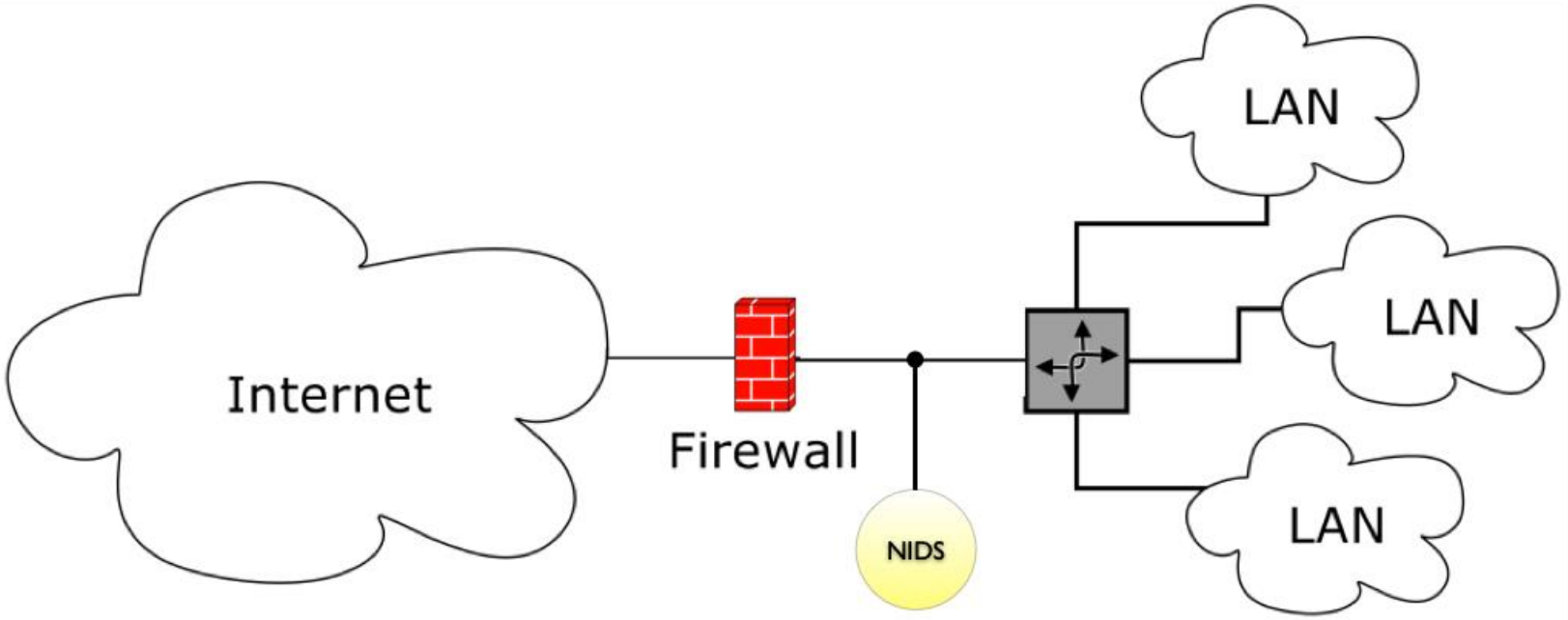
Popular security-oriented Linux distribution

Community

50/90/150/185 attendees at BroCon '12/'13/'14/'15
110 organizations at BroCon '14
~4,000 Twitter followers
~1000 mailing list subscribers
~100 users average on IRC channel
10,000+ downloads / version from 150 countries

Design of the Bro NIDS

Network Intrusion Detection



System Philosophy

- Bro provides a real-time network analysis framework
  - Primary a network intrusion detection system (NIDS)
  - However it is also used for pure traffic analysis
- Focus is on
  - Application-level semantic analysis (rather than analyzing individual packets)
  - Tracking information over time
- Strong separation of mechanism and policy
  - The core of the system is policy-neutral (no notion "good" or "bad")
  - User provides local site policy

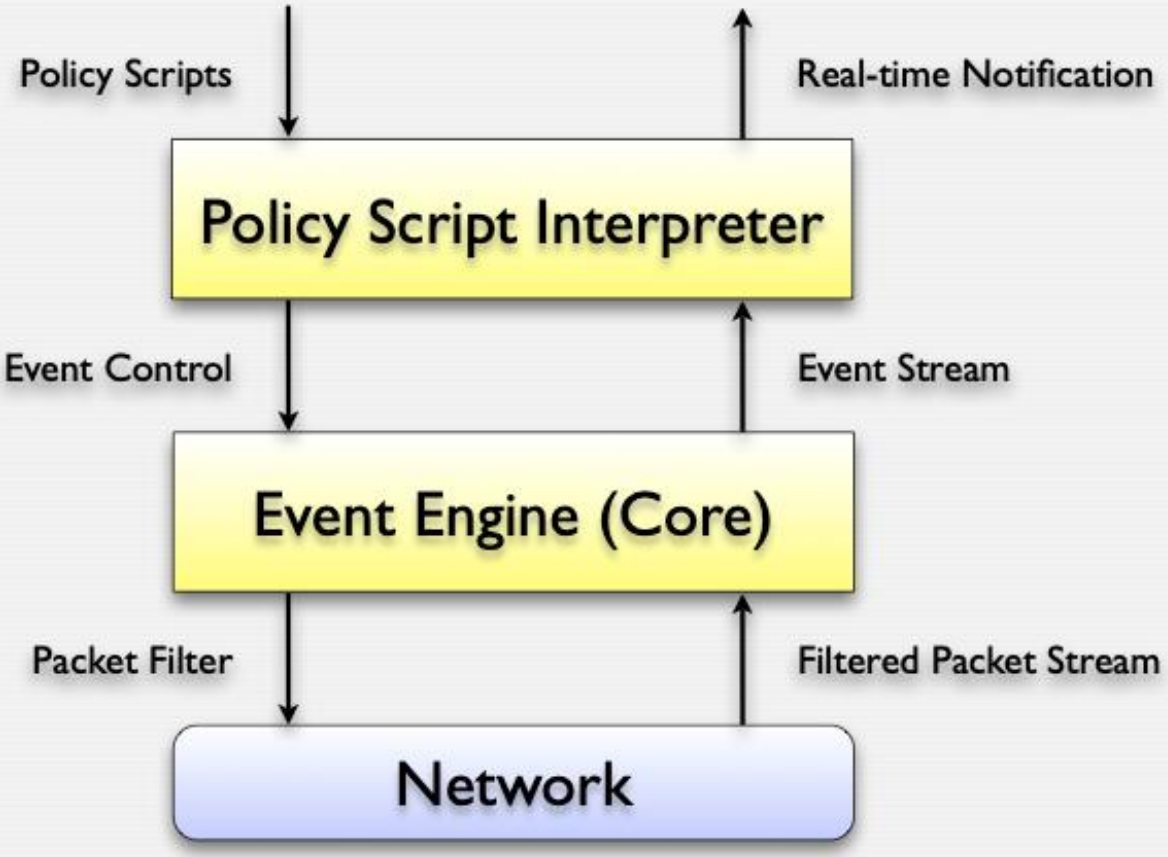
System Philosophy (2)

- Operators *program* their policy
  - Not really meaningful to talk about what Bro detects "by default"
- Analysis model is *not* signature matching
  - Bro is fundamentally different from, e.g., Snort (though it *can* do signatures as well)
- Analysis model is *not* anomaly detection
  - Though it does support such approaches (and others) in principle
- System thoroughly logs all activity
  - It does not just alert
  - Logs are invaluable for forensics

Target Environments

- Bro is specifically well-suited for scientific environments
  - Extremely useful in networks with liberal ("default allow") policies
  - High-performance on commodity hardware
  - Supports intrusion prevention schemes
  - Open-source (BSD license)
  - Developed at LBNL & the International Computer Science Institute
- It does however require some effort to use effectively
  - Pretty complex, script-based system
  - Requires understanding of the network
  - No GUI, just ASCII logs
  - Only partially documented
  - Lacking resources to fully polish the system
- Development is primarily driven by *research*
  - However, our focus is operational use; we invest much time into "practical" issues
  - Want to bridge gap between research and operational deployment

Architecture



Event-Engine

- Event-engine is written in C++
- Performs *policy-neutral* analysis
  - Turns low-level activity into high-level events
  - Examples: `connection_established, http_request`
  - Events are annoated with context (e.g., IP addresses, URL)
- Contains *analyzers* for >30 protocols, including
  - ARP, IP, ICMP, TCP, UDP
  - DCE-RPC, DNS, FTP, Finger, Gnutella, HTTP, IRC, Ident, NCP, NFS, NTP, NetBIOS, POP3, Portmapper, RPC, Rsh, Rlogin, SMB, SMTP, SSH, SSL, SunRPC, Telnet
- Analyzers generate ~300 events ...

Policy Scripts

- Scripts process event stream, incorporating ...
  - ... context from past events
  - ... site's local security policy
- Scripts take actions
  - Recording activity to disk
  - Generating alerts via syslog or mail
  - Executing program as a form of response

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Script Example: Tracking SSH Hosts

```
global ssh_hosts: set[addr];

event connection_established(c: connection)
    {
    local responder = c$id$resp_h; # Responder's address
    local service = c$id$resp_p;   # Responder's port

    if ( service != 22/tcp )
        return; # Not SSH.

    if ( responder in ssh_hosts )
        return; # We already know this one.

    add ssh_hosts[responder]; # Found a new host.
    print "New SSH host found", responder;
    }
```

Expressing Policy

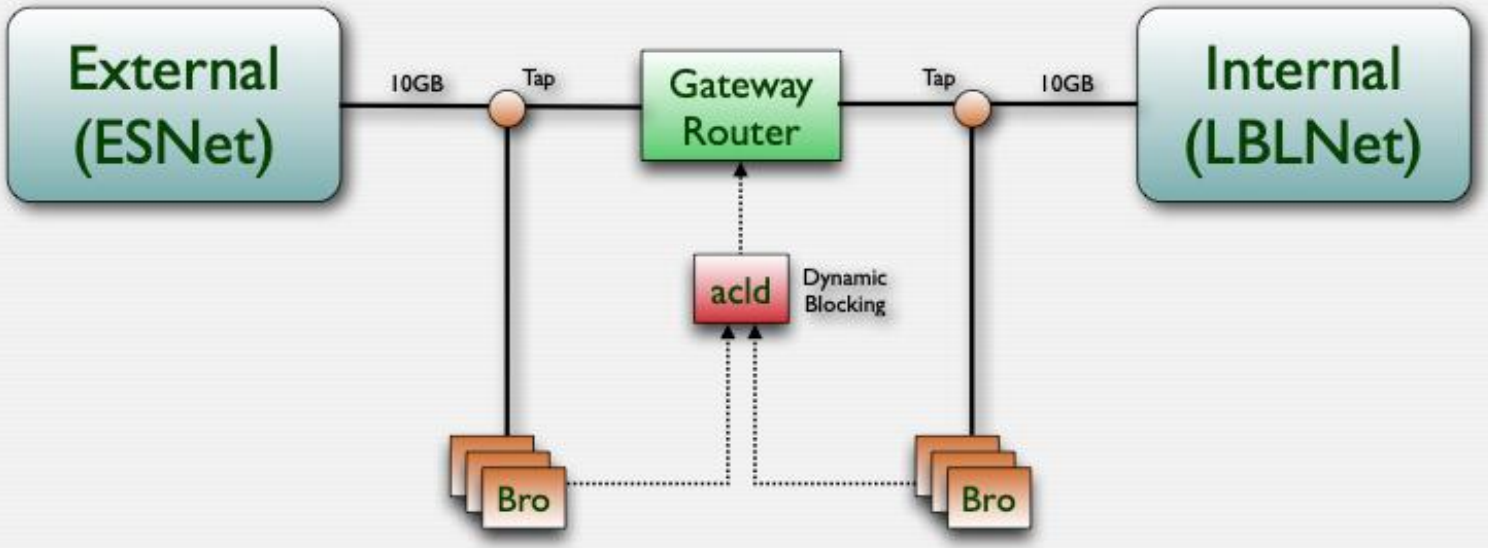
- Scripts are written in custom, domain-specific language
  - Bro ships with 20K+ lines of script code
  - Default scripts detect attacks & log activity extensively
- Language is
  - Procedural
  - Event-based
  - Strongly typed
  - Rich in types
    - Usual script-language types, such as tables and sets
    - Domain-specific types, such as addresses, ports, subnets
  - Supporting state management (expiration, timers, etc.)
  - Supporting communication with other Bro instances

LBNL's Bro Installation

Bro at the Lawrence Berkeley Lab

- LBNL has been using Bro for >10 years
- Uses Bro to monitor its 10 Gbps Internet uplink
- Bro is one of the main components of lab's security
  - Several Bro boxes for different tasks
  - Bro automatically *blocks* attackers (~4000/day, mainly scanners)

LBNL Monitoring Setup



Recent Developments (1)
The Bro Cluster

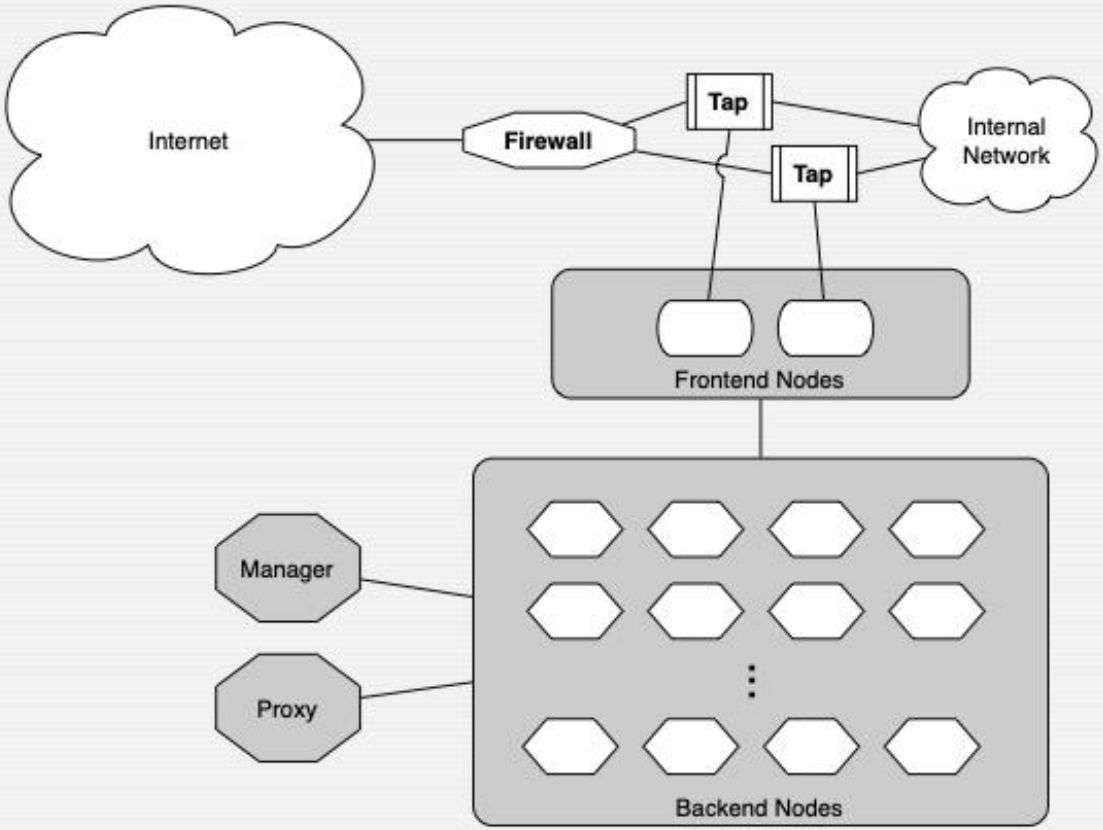
Motivation

- NIDSs reach their limits on commodity hardware
  - Keep needing to do *more analysis* on *more data* at *higher speeds*
  - Analysis gets richer over timer, as attacks get more sophisticated
  - However, single CPU performance is not growing anymore the way it used to
  - Single NIDS instance (Snort, Bro) cannot cope with >=1Gbps links
- Key to overcome current limits is *parallel analysis*
  - Volume is high but composed of many *independent* tasks
  - Need to exploit parallelism to cope with load

The Bro Cluster

- Load-balancing approach: use many boxes instead of one
- Most NIDS provide support for multi-system setups
- However instances tend to work independent
  - Central manager collects alerts of independent NIDS instances
  - Aggregates results instead of correlating analysis
- The Bro cluster works *transparently* like a single NIDS
  - Gives same results as single NIDS would if it could analyze all traffic
  - No loss in detection accuracy
  - Scalable to large number of nodes
  - Single system for user interface (log aggregation, configuration changes)

Architecture



Front-Ends

- Distribute traffic to back-ends by rewriting MACs
  - In software via Click (open-source "modular router")
  - In hardware via Force-10's P10 (prototype in collaboration with F10)
- Fault-tolerance
  - Easy to retarget traffic if a back-end node fails
- Per connection-hashing
  - Either 4-tuple (addrs,ports) or 2-tuple (addrs)
  - MD5 mod n, ADD mod n

Back-ends

- Running Bro as their analysis engine
- Bro provides extensive communication facilities
  - Independent state framework
  - Sharing of *low-level* state
  - Script-layer variables can be *synchronized*
- Basic approach: pick state to be synchronized
  - A few subtleties needed to be solved
- Central manager
  - Collects output of all instances
  - Raises alerts
  - Provides dynamic reconfiguration facilities

Recent Developments (2)
Dynamic Protocol Detection

Port-based Protocol Analysis

- Bro has lots of application-layer analyzers
- But which protocol does a connection use?
- Traditionally NIDS rely on ports
  - Port 80? Oh, that's HTTP.
- Obviously deficient in two ways
  - There's non-HTTP traffic on port 80 (firewalls tend to open this port...)
  - There's HTTP on ports other than port 80
- Particularly problematic for security monitoring
  - Want to know if somebody avoids the well-known port

Port-independent Analysis

- Look at the *payload* to see what is, e.g., HTTP
- Analyzers already know how a protocol looks like
  - Leverage existing protocol analyzers
  - Let each analyzer *try to parse* the payload
    - If it succeeds, great!
    - If not, then it's actually another protocol
- Ideal setting: *for every connection, try all analyzers*
- However, performance is prohibitive
  - Can't parse 10000s of connections in parallel with all analyzers

Making it realistic ...

- Bro uses byte patterns to *prefilter* connections
  - An HTTP signature looks for *potential* uses of HTTP
  - Then the HTTP analyzer verifies by trying to parse the payload
  - Signatures can be loose because false positives are inexpensive (no alerts!)
- Other NIDS often ship with protocol signatures
  - These directly generate alerts (image reporting all non-80 HTTP conns)
  - These do not trigger protocol-layer semantic analysis (e.g., extracting URLs)
- In Bro, a match triggers further analysis
- Main internal concept: analyzer trees
  - Each connection is associated with an analyzer tree

Finding Bots

- IRC-based bots are a prevalent problem
  - Infected client machines accept commands from their "master"
  - Often IRC-based but not on port 6667
- Just detecting IRC connections not sufficient
  - Often there is legimitate IRC on ports other than 6667
- DPD allows to analyze all IRC sessions *semantically*
  - Looks for typical patterns in NICK and TOPIC
  - Reports if it find IRC sessions showing both such NICKs and TOPICs
- Very reliable detection of bots
  - Munich universities use it to actively block internal bots automatically

DPD: Summary & Outlook

- Port-independent protocol analysis
  - Idea is straight-forward, but Bro is the only system which does it
- Bro now has a very generic analyzer framework
  - Allows arbitrary changes to analyzer setup during lifetime of connection
  - Is not restricted to any particular approach for protocol detection
- Main performance impact: need to examine *all* packets
  - Well, that's pretty hard to avoid
- Potential extensions
  - More protocol-detection heuristics (e.g., statistical approaches)
  - Analyze tunnels by pipelining analyzers (e.g., to look inside SSL)
  - Hardware support for pre-filtering (e.g., on-NIC filtering)

Conclusion & Outlook

Summary

- Bro is likely one of the most powerful NIDS available
  - Open-source and runs on commodity hardware
  - While primarily a research system, it is well suited for operational use
  - One of the main components of LBNL's network security monitoring
- Largest problem: Documentation is lacking ...
  - Especially for new features ...
- Still, there is quite a bit of information available:
  - The Bro Wiki contains quite a bit of information.
  - Much of the new functionality is introduced in papers. See `www.bro-ids.org` for a list.
  - The most up-to-date documentation is still the CHANGES file ...
- The mailing list is open for all kinds of questions
  - From beginner to expert.


zeek.org
Register now for ZeekWeek! OCTOBER 12-14 | Austin, TX
Register now
zeek
Get Zeek
Documentation
Community
Events
Blog
About
zeek
An Open Source Network Security Monitoring Tool
Zeek (formerly Bro) is the world's leading platform for network security monitoring.
Flexible, open source, and powered by defenders.
Get Zeek
ZEEK AND YE SHALL FIND

CTF5G Demo (By TA Srinidhi)

End of Lecture 16