Experiences with Detecting Network Intruders Using Bro

Vern Paxson

AT&T Center for Internet Research at ICSI
International Computer Science Institute
and
Lawrence Berkeley National Laboratory
University of California
Berkeley, CA

vern@{aciri.org, ee.lbl.gov}

November 9, 2001

What can you learn watching a network link?

- Far and away, most traffic travels across the Internet unencrypted.
- Communication is <u>layered</u> with higher layers corresponding to greater semantic content.
- The entire communication between two hosts can be reassembled: individual *packets* (e.g., TCP/IP headers), application *connections* (TCP byte streams), user *sessions* (Web surfing).
- Appealing because it's *cheap* and gives broad coverage.

Problems with passive monitoring:

- Reactive, not proactive.
- Assumes network-oriented (often “external”) threat model.
- For high-speed links, monitor may not be able to keep up.
- Against a skilled opponent, there is a <u>fundamental</u> problem of evasion: confusing / manipulating the monitor.

Approaches to detecting attacks:

Signature-based looks for exact matches to known attacks.

Pro: good attack libraries, easy to understand its results.

Con: unable to detect new attacks, or even just variants.

Anomaly-based builds profile of “normal” use, flag deviations.

Pro: potentially detects wide range of attacks, including novel.

Con: potentially misses wide range of attacks, including known.

Con: can potentially be “trained” to accept attacks as normal.

Approaches to detecting attacks, con't:

BRO is "activity-based." Looks for patterns of activity that deviate from a site's *policy*. Also has some signature elements.

All intrusion detection systems suffer from the twin problems of *false positives* and *false negatives*.
These are not minor, but an Achilles heel.

BRO attempts to deal with false positives by providing a rich language for specifying a site's specific policy.

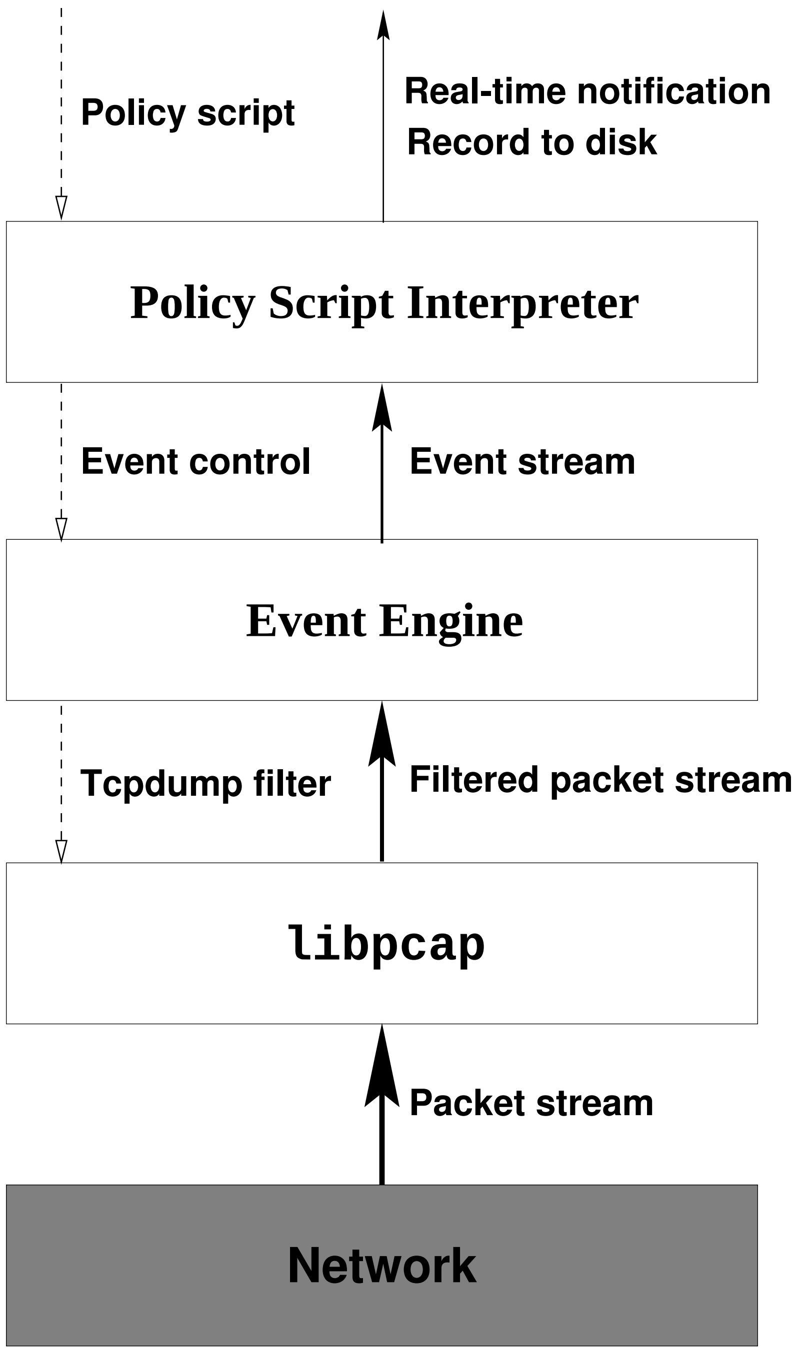
Some general considerations about the problem space:

Security is about *policy*.

The goal is risk management, not bulletproof protection.

<u>Scaling</u> works against us: as the volume of monitored traffic grows, so does its <u>diversity</u>.

$\Rightarrow$ NIDS research "in the lab" is often *far removed* from operational reality.


Policy script
Real-time notification
Record to disk
Policy Script Interpreter
Event control
Event stream
Event Engine
Tcpdump filter
Filtered packet stream
libpcap
Packet stream
Network

Event engine:

Event engine does generic (non-policy) analysis.

E.g. Connection-level: `connection_attempt`
`connection_finished`

E.g. Application-level: `ftp_request`, `pm_request_getport`,
`login_input_line`

E.g. Activity-level: `login_success`, `stepping_stone`,
`ssh_signature_found`

If you define a handler for a given event, it will be invoked any time the event occurs. Otherwise, event engine skips the work for detecting the event.

The Bro policy language:

Strongly typed $\Rightarrow$ catch errors at compile time.

Arithmetic types, `pattern`, `time`, `interval`, `port`, `addr`.

Records, associative tables & sets:

```
global ftp_sessions:  table[conn_id] of
                                   ftp_session_info
```

Strings are counted rather than NUL-terminated:

```
USER nice\0USER root
```

Analyzers:

For all TCP connections (via SYN/FIN/RST packets):

- start time, duration, service, addresses, sizes
- port, address scanning

Applications: DNS, FTP, HTTP, NTP, Finger, Portmapper, Ident.

Telnet and RLogin:

`login_successful`, `login_failure`

`activating_encryption`, `login_confused`

⇒ `login_input_line`, `login_output_line`

in first five months of operation,

120 UCB break-ins (60 root compromises)

Teeth:

“`rst`” terminates the local end of a TCP connection via RST packet(s). (Tricky for picky TCP stacks that insist on exact sequence numbers.)

“`drop-connectivity`” talks to border router, throws away given remote traffic: a *reactive firewall*.

Both invoked via `system()`, per arbitrary policy.

At LBL, 40–70 scans dropped each day.
Routers run with 500–2,000 ACL entries.
When scan blocking fails, mean time to break-in: <u>4 hours</u>.

Status:

- Operational since 1996.
- Two dozen monitor boxes deployed at LBL, UCB, ICSI.
- LBL boxes see about 1 billion packets/day ($\approx$ 11 Kpps).
  UCB: 5 billion.
- After filtering, LBL boxes see peaks of 2 Kpps.
- Connection logs: 0.5 GB/day LBL, 1.5 GB/day UCB.

Attacks on the monitor:

<u>Overload</u>: make it drop packets, sneak in.

Defense: leave doubt as to monitor's capabilities; shed load.

<u>Crash</u>: make it fault, or run out of resources.

Defense: watchdog; backup tracing;
resource management hooks.

<u>Discredit</u>: make it generate zillions of false alarms.

Defense: supple alarm filtering, rule editing.

<u>Subterfuge</u>: fool the monitor.

Example: how to pull out strings from TCP session?

The problem of evasion:

How to detect particular text (e.g., “`USER root`”) in a packet?

Easiest: scan for the text in each packet.

No good: text might span multiple packets.

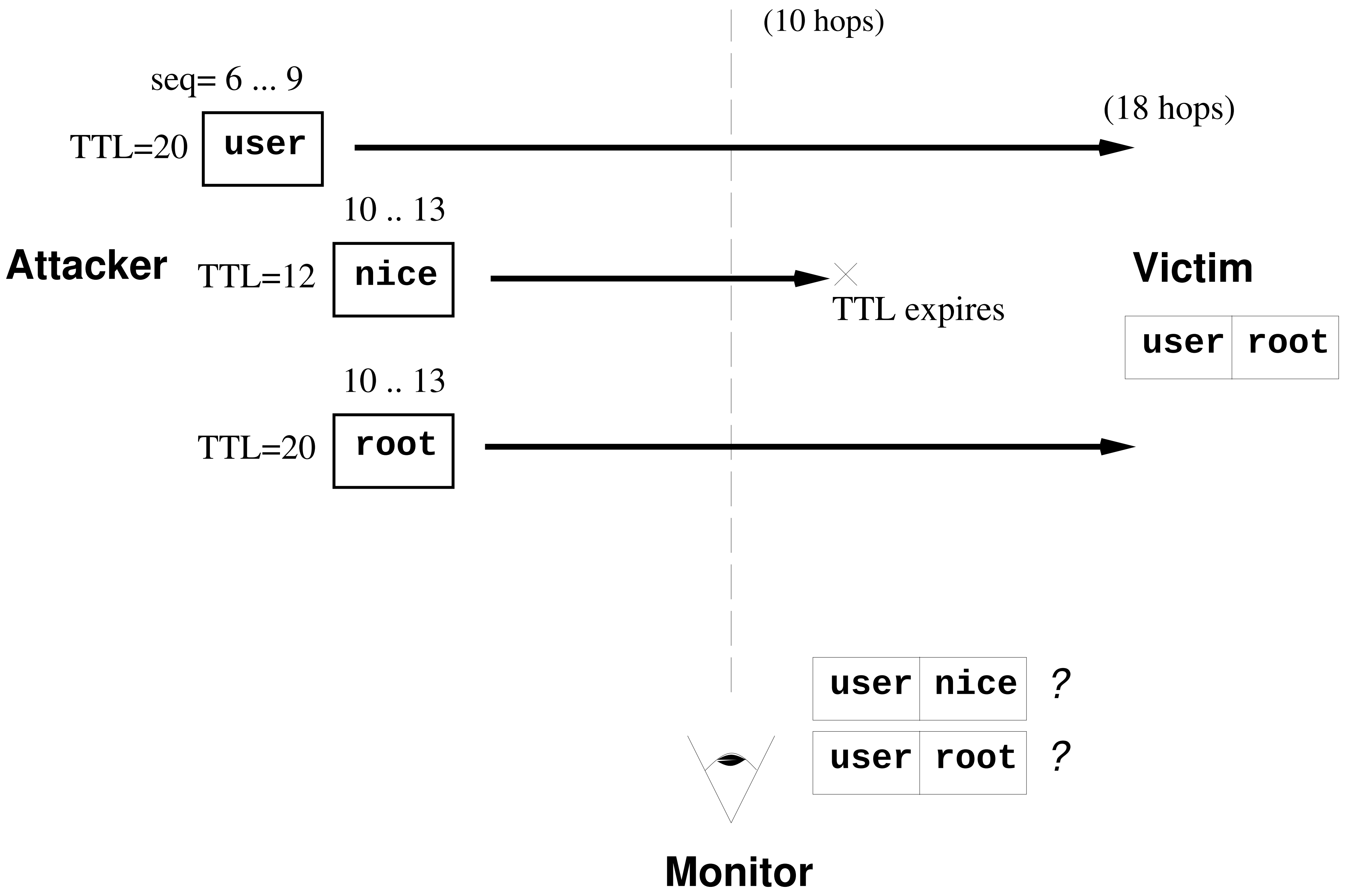
(Major inadequacy of *stateless* NIDS’s.)

Okay, then: remember text scanned in last packet.

No good: out-of-order delivery.

Okay, then: fully reassemble the byte stream.

Tricky. Costs state . . . and *still* evadeable.


(10 hops)
seq= 6 ... 9
TTL=20
user
(18 hops)
10 .. 13
Attacker
TTL=12
nice
TTL expires
Victim
user
root
10 .. 13
TTL=20
root
user
nice
?
user
root
?
Monitor

Crud seen on a DMZ:

Storms of 10,000+ FIN or RST packets, due to TCP bugs.

Storms due to foggy days.

Private addresses leaking out.

Legitimate tiny fragments.

Fragments with DF (`Don’t Fragment`) set.

Overlapping fragments.

TCPs that acknowledge data that was never sent (!).

TCPs that *retransmit different data than sent in the first place*.

The problem of evasion, con't:

- Many evasions look anomalous, but when monitoring a high volume traffic stream, you already see *lots of anomalous-but-benign junk.*
- Problem is pervasive. Dozens of evasions identified in the literature.
- Evasion "toolkits" have been available for a number of years. (Not yet widely used.)
- Likely arms race.

Possible ways to resist evasion:

- Exercise great care when developing IDS, attending to all possible exceptional events.
- Deploy sensors on the end hosts.
- Bifurcating analysis. E.g., for “rob<*DEL*><*BS*><*BS*>ot”, examine directly; as “ro<*BS*><*BS*>ot”; and as “root”.
- Understand network and end-host details sufficiently well to resolve ambiguities.
- Deploy a traffic “normalizer” (or “scrubber”) to remove ambiguities from traffic stream.

Detecting “stepping stones”:

Internet attacks invariably do not come from the attacker’s own personal machine, but from a *stepping-stone*: an intermediary previously compromised.

Furthermore, usually it is a *chain* of stepping stones.
Manually tracing attacker back across the chain is virtually impossible.

So: want to detect that a connection going into a site is closely related to one going out of the site.

Detecting stepping stones, con’t:

Approach #1 (Staniford/Heberlein, 1994):

Look for similar text content.

For each connection, generate a <u>thumbprint</u> (24 bytes) summarizing per-minute character frequencies.

Connections with similar thumbprints become likely stepping stone candidates.

Detecting stepping stones, con't:

Approach #2 (USAF, 1994):

- Break-in to upstream attack site.
- Rummage through its logs.
- Find out from where it was originally compromised.
- <u>Repeat</u>.

Detecting stepping stones, con't:

Approach #3 (Zhang/Paxson, 2000):

Leverage unique on/off pattern of user login sessions.

Look for connections that end <u>idle periods</u> at the same time.

Two idle periods correlated if ending time differ by $\leq \sigma$ sec.

If enough idle periods coincide $\Rightarrow$ stepping stone pair.

For $A \to B \to C$ stepping stone, just 2 correlations suffices.

(For $A \to B \to \ldots \to C \to D$, 4 suffices.)

Detecting stepping stones, con't:

$\Rightarrow$ Works very well, *even for encrypted traffic*.

But: easy to evade, if attacker cognizant of algorithm.

And: also turns out there are frequent *legit* stepping stones.

Detecting backdoors:

“Backdoor”: a service installed on a compromised machine to allow the attacker to surreptitously return.

How to find access to these against sea of background traffic?

General algorithm for interactive traffic (Zhang/Paxson 2000):

- look for frequent small packets
- look for small packets with large interarrivals

Detecting backdoors, con't:

Protocol-specific: SSH, Rlogin, Telnet, FTP.

Algorithms also amenable to filtering for large perf. gain:

e.g.,

```
tcp[(tcp[12]>>2):4] = 0x5353482D and
    (tcp[((tcp[12]>>2)+4):2] = 0x312E or
     tcp[((tcp[12]>>2)+4):2] = 0x322E)
```

Detecting backdoors, con't:

Plus: a hack for detecting some *root* backdoors ("`# `").

$\Rightarrow$ Found 437 root backdoors in single 24-hour period at UCB.

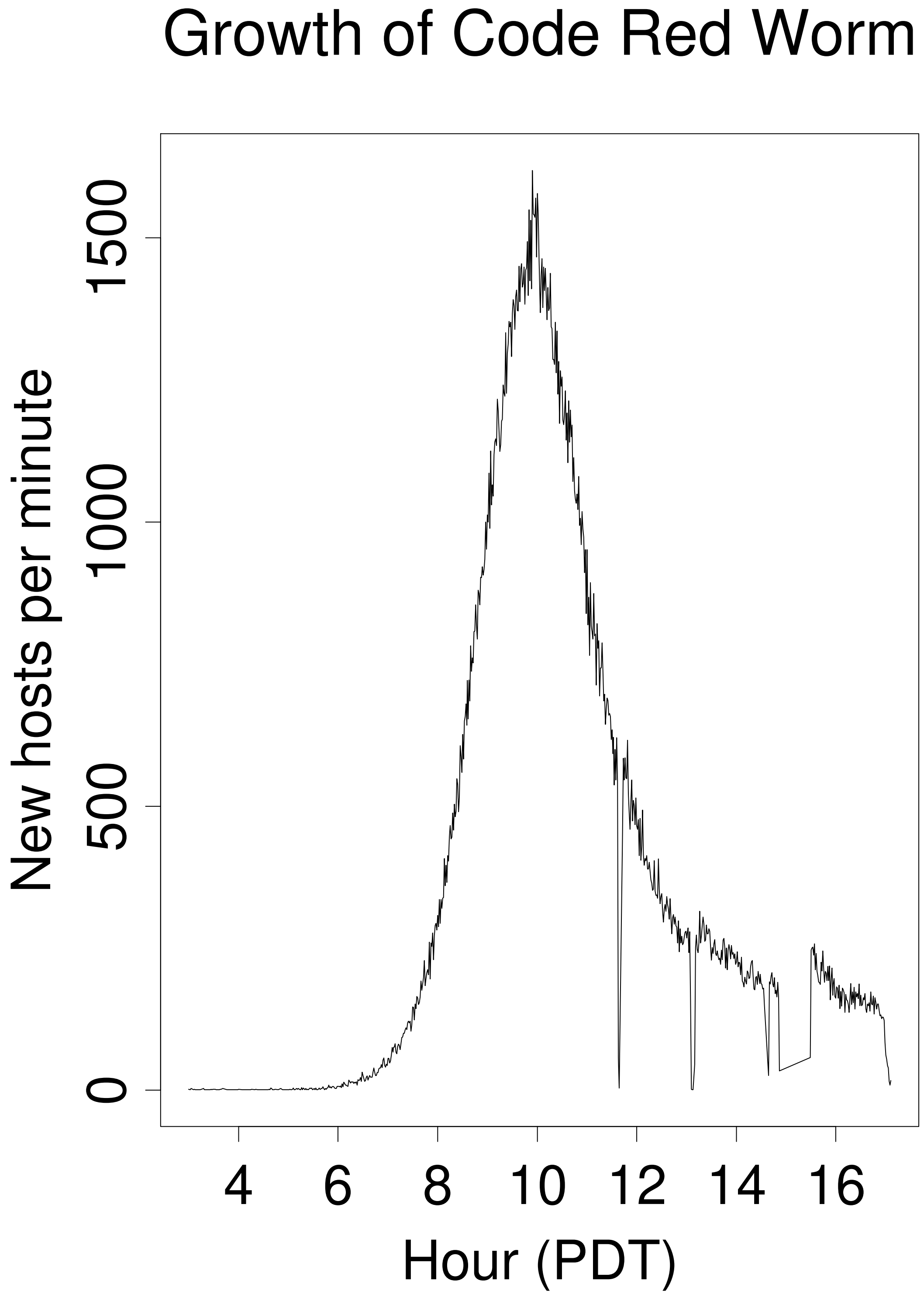
Also recognizers for non-interactive protocols:

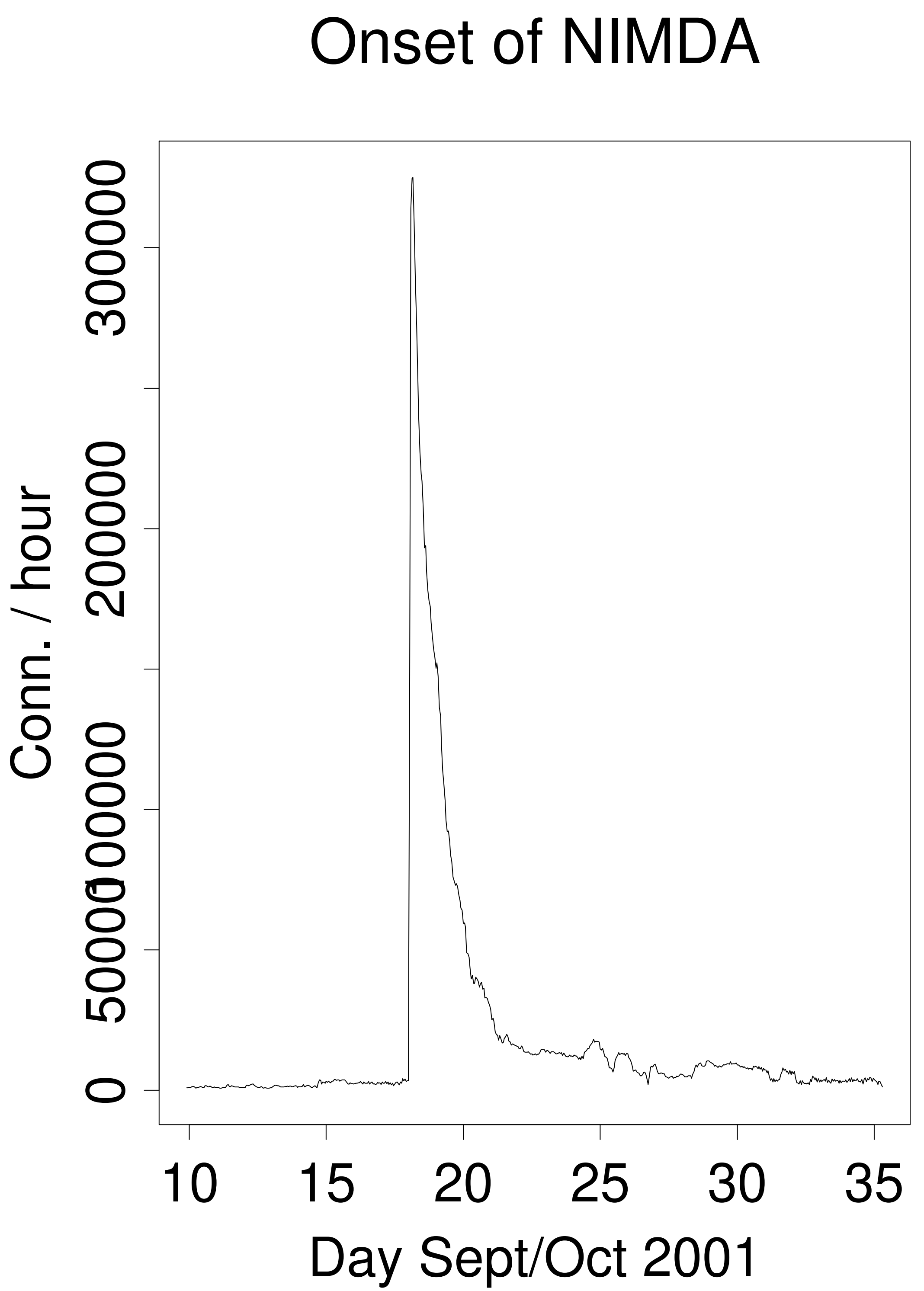
SMTP, Napster, Gnutella.

In general, algorithms perform quite well.

And: can employ <u>filtering</u> with little loss of accuracy.

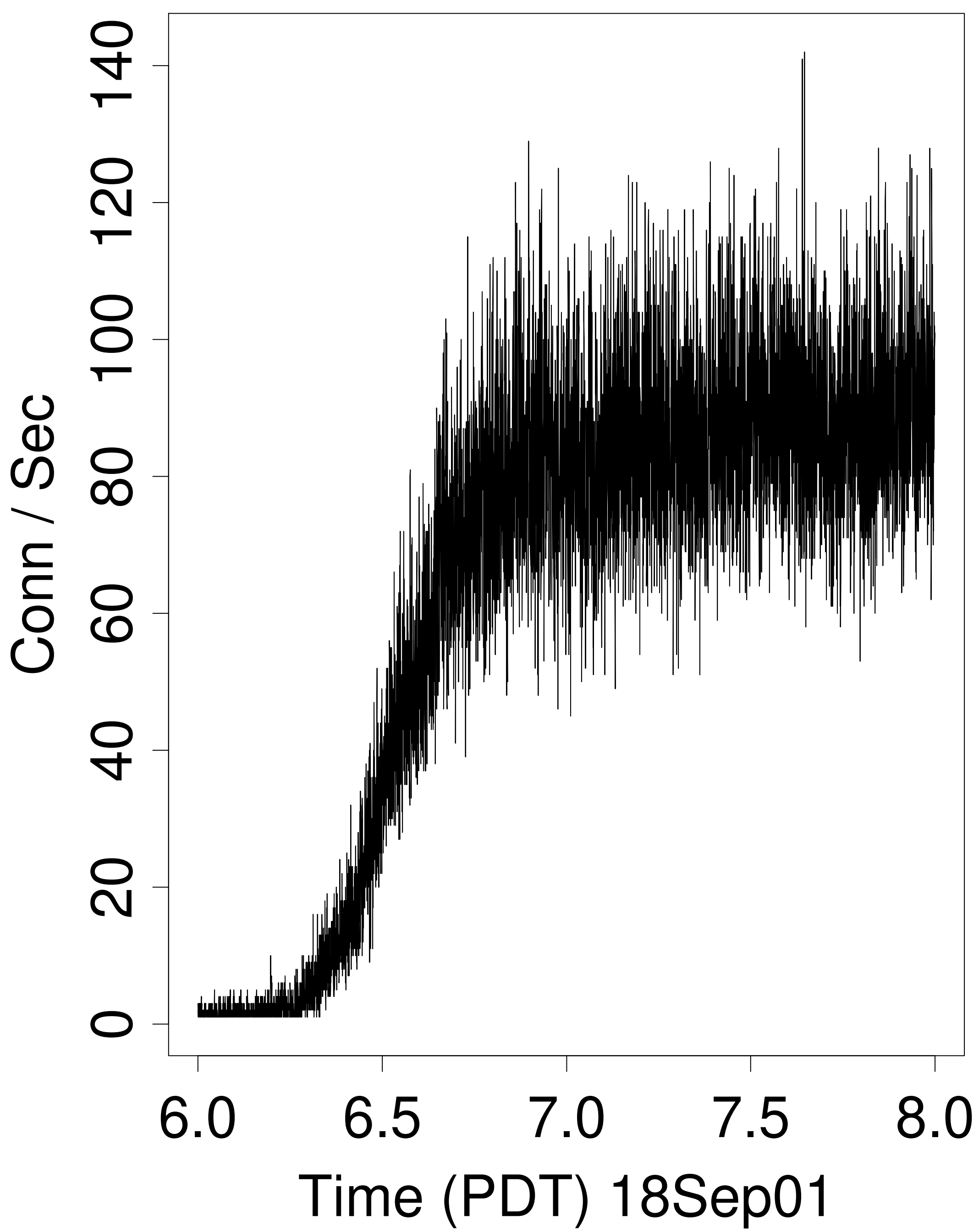
But: find many *legit* backdoors.


Growth of Code Red Worm
New hosts per minute
0
500
1000
1500
4
6
8
10
12
14
16
Hour (PDT)



Onset of NIMDA
Conn. / hour
0
500000
1000000
2000000
3000000
10
15
20
25
30
35
Day Sept/Oct 2001

Onset of NIMDA



Summary points:

- The Internet is a frothing sea of attacks.
- You can detect a whole lot by piecing together judiciously filtered network traffic into <u>events</u> reflecting <u>activity</u> . . .
- . . . but there are significant problems with <u>evasion</u> leading to an arms race.
- Traffic contains much more diversity/junk than you'd think.
- The endpoint <u>host</u> is a great location to look for attacks.
- Increasingly, NIDS need to be supplemented by an <u>active forwarding element</u>.