## about:config

```
J|Down|left MB   - nächster Punkt
K|Up|right MB    - vorhergehender Punkt
Right            - nächste Seite
Left             - vorhergehende Seite
<Return>         - zur Seite springen im Inhaltsverzeichnis
O                - Outline Mode on/off
T|escape|F5      - Inhaltsverzeichnis on/off
S|.              - zwischen Styles rotieren
```

## :begin

Net::Inspect
Steffen Ullrich, GeNUA mbH
Deutscher Perl-Workshop 2012, Erlangen

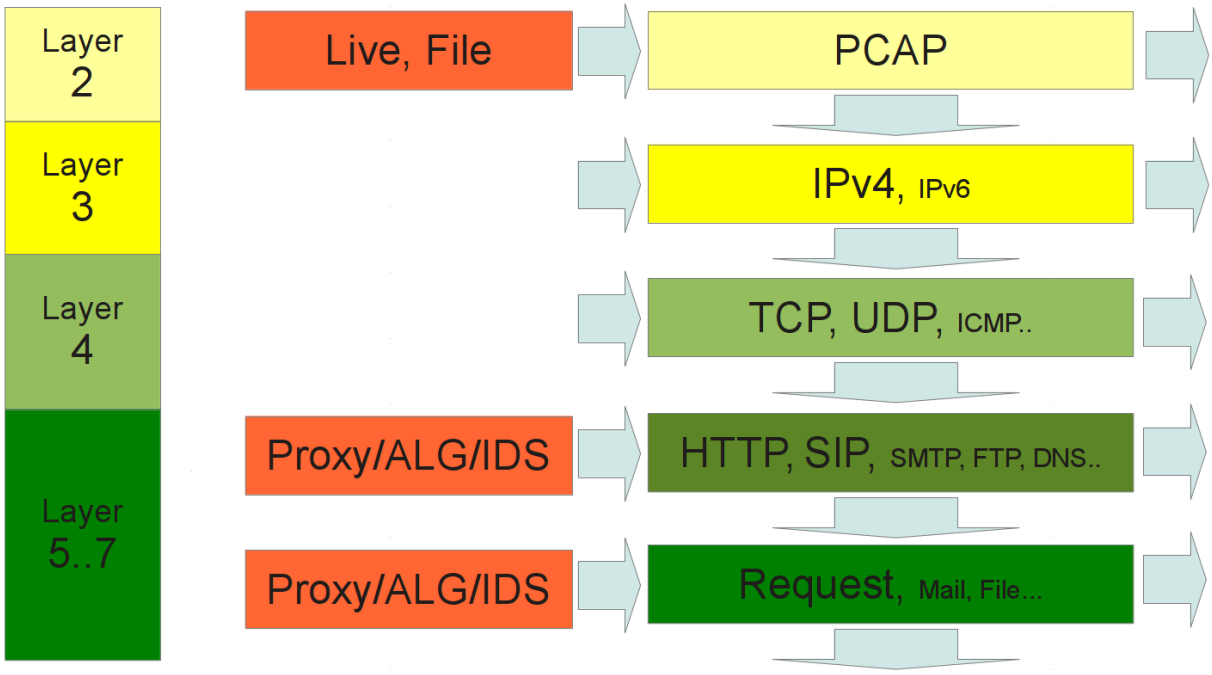
## about:me

- Steffen Ullrich
- seit 1996 Perl
- seit 2001 bei GeNUA mbH Network Security
- Modules:
  - IO::Socket::SSL
  - Net::SIP, Net::SSLGlue, Net::INET6Glue, Net::PcapWriter
  - Mail::SPF::Iterator
  - Devel::TrackObjects
- neu: Net::Inspect

## Hintergrund

- seit 07/2011 im Forschungsprojekt Padiofire:
  - BMBF, Unis Erlangen, Cottbus, Innsbruck + GeNUA mbH
  - Web 2.0 Sicherheit
  - ALG, IDS, Deep Inspection...
- ALG (GeNUGate): OSI Layer 4..7 - viel Erfahrung
- IDS (Bro): OSI Layer 2..4[..7] - keine Erfahrung
- Bro Code/Design Doku - RTFC
- es versteht sich besser, wenn man sowas mal selber implementiert
  - Perl Rapid Development to Rescue!

## Design Überblick




## Push statt Pull

- Input-Events werden werden in den Layer geschickt
- Layer aktualisiert evtl. internen State, Buffering
- und generiert Output-Events so früh wie möglich
- die wiederum Input des nächsten Layers sein können
- u.U. mehrere Empfänger auf einem Layer
  - Empfänger ignoriert Daten, die ihn nicht interessieren
- flexibles Push vs. unflexibles Pull
  - Push kann man in Pull Lösung integieren, nicht umgekehrt
  - einfach in AnyEvent, POE... integrierbar

## Input/Output auf jeder Ebene

- L2::Pcap - Raw Daten extrahieren
- L3::IP - Reassembly von Fragmenten
- L4::TCP - Reordering, Connection open/close
- L7::HTTP - Requests extrahieren: Pipelining, Unchunking
- L7::HTTP::Request - Dekomprimierung

## robuste HTTP-Verarbeitung

- gegen 25GB Real-Life Daten getestet
- kaputte Header anmeckern aber weiterparsen: 'Last Modified'
- gzip Header aber keine gzip Daten
- Body wenn keiner sein darf
- Pipelining

## Design Details

```
my $http = Net::Inspect::L7::HTTP->new(...);
my $tcp  = Net::Inspect::L4::TCP->new($http);
my $ip   = Net::Inspect::L3::IP->new($tcp);
my $pkts = Net::Inspect::L2::Pcap->new($pcap,$ip);

pcap_loop($pcap,-1,sub {
  my (undef,$hdr,$data) = @_;
  return $pkts->pktin($data,$hdr);
});
```

## L2::Pcap

- in: pktin($pcapdata,\%pcaphdr)
- Pcap Header entfernen, Zeit extrahieren
- upper_layer->pktin($rawdata,$time)

## L3::IP

- in: pktin($data,$time)
- src-addr, dst-addr, proto extrahieren
- Reassembly von Fragmenten
- kein IPv6 derzeit
- upper_layer->pktin($ipdata,\%meta)

## L4:TCP

- in: pktin($ipdata,\%meta)
- src-port, dst-port extrahieren
- Reordering

- upper_layer-> ...
  - syn(\%meta)
  - new_connection(\%meta) -> Connection object
    - in($dir,$tcpdata,$eof,$time)
    - fatal($reason,$time)

## L4::UDP

- in: pktin($ipdata,\%meta)
- src-port, dst-port extrahieren
- upper_layer-> ...
  - pktin($udpdata,\%meta) -> connection|undef
  - connection->pktin($dir,$udpdata,$time)

## L5::GuessProtocol

```
my $http = Net::Inspect::L7::HTTP->new(...);
my $fallback = Net::Inspect::L5::Unknown->new(...);
my $empty = Net::Inspect::L5::NoData->new;
my $l5 = Net::Inspect::L5::GuessProtocol->new();
$l5->attach($http);
$l5->attach($unknown);
$l5->attach($empty);
```

- Protokoll erraten via `class->guess_protocoll`
- `in` weiterleiten an `$handler->in` sobald `$handler` feststeht

## L7::HTTP

- in: in($dir,$data,$eof,$time)
- Header Parsen, Chunks erkennen, Pipelining...
- upper_layer->new_request(\%meta,$conn) -> request object
  - in_request_header($header,$time)
  - in_request_body($data,$eof,$time)
  - in_response_header($header,$time)
  - in_response_body($data,$eof,$time)
  - in_chunk_header($header,$time)
  - in_chunk_trailer($trailer,$time)
  - in_data($dir,$data,$eof,$time): CONNECT.. requests
  - fatal($reason)

## L7::HTTP::Request::InspectChain

- Request Objekt
- attachbare Hooks für Analyse und Manipulation:
  - request_header (header)
  - request_body (chunk)
  - response_header (header)
  - response_body (chunk)
  - ...
- vordefinierte Hooks für Unchunking und Dekomprimierung

## existente Anwendungen

## tcpudpflow

- extrahiert TCP/UDP-Verbindungen aus pcap (live, File) in einzelne Files
- mögliche sinnvolle einfacher Erweiterungen
  - Signature Matching über Paketgrenzen - alle anderen ignorieren
  - Speichern einzelner Verbindung als pcap-Stream für nachfolgende Analyse durch Wireshark etc via Net::PcapWriter

## httpflow

- extrahiert HTTP-Requests aus pcap (live,File) in einzelne Files
- mit/ohne Unchunking
- mit/ohne Dekomprimierung
- sinnvolle Erweiterungen analog tcpudpflow
  - sowie tiefergehende Analysen von HTTP Requests (Timing, Content, Charset, Fehler, Größe...)

## http_inspection_proxy

- HTTP Proxy
- liest nicht pcap-Daten, sondern TCP-Verbindungen
  - Input: TCP-Daten (L4)
  - Output: L7 Events
- leitet Daten weiter (Proxy)
- kann Daten bei Weiterleitung modifizieren

- dank AnyEvent single threaded und halbwegs performant

## http_inspection_proxy - Ideen

- Anbindung Virenscanner
- SSL Bridging
- Verhindern non-SSL Tunnel via CONNECT
- HTML Normalisierung
- transparenter Proxy
- Ad Injection

## weitere Ideen

- Rapid Prototyping von IDS Ideen
  - SMTP RBL füttern durch Behavior Analyse in Honeynets o.ä
  - Anbindung an IDS Bro via broccoli
- Einbindung in ALG/Proxies
- Extraction RTP Daten von SIP Call via Net::SIP
- Parallelisierung der Analyse auf mehrere Threads, Rechner
- ...
- eure Idee hier, Net::Inspect Namensraum offen für andere

## Beispiel rtpxtract

- Extraktion von RTP-Streams aus SIP calls
- auf beliebigen UDP-Ports hören
- wenn es wie SIP+SDP ausschaut
- Medieninformation extrahieren und merken
- wenn Paket dafür ankommt Connection erstellen
- und Daten speichern
- close nach kurzem Timeout

## Layers zusammenstöpseln

```
my $sip = SIPXTract->new;
my $udp = Net::Inspect::L4::UDP->new($sip);
my $raw = Net::Inspect::L3::IP->new($udp);
my $pc  = Net::Inspect::L2::Pcap->new($pcap,$raw);
```

## SIP Pakete erkennen

```
# extract SDP data
my $pkt = eval { Net::SIP::Packet->new($data) }
    or return;
my $sdp = eval { $pkt->sdp_body } or return;
my @media = $sdp->get_media or return;
```

## IP und Port für Medien merken

```
my %rtp;
...
# save media info in %rtp
for(@media) {
    $rtp{ $_->{addr},$_->{port} } = $_;
}
```

## und wenn Daten kommen Connection erstellen

```
package SIPXTract;
my %rtp;
sub pktin {
   my ($self,$data,$meta) = @_;
   my $m = delete $rtp{ $meta->{daddr},$meta->{dport} };
   if ($m) {
      # make connection
      my $s = SIPXTract::RTPStream->new($meta,$m);
      $s->pktin(0,$data,$meta->{time});
      return $s;
   }

   .. extract SIP+SDP data
   .. save media info in %rtp
   return; # no connection for SIP packets
}
```

## und dort RTP Daten sichern

```
package SIPXTract::RTPStream;
use base 'Net::Inspect::Connection';
...
sub pktin {
    my ($self,$dir,$data,$time) = @_;
    $self->{expire} = $time + 30; # short expire
    ..create file
```

```
    ..extract RTP payload
    ..save
}
```

## pcap mainloop

```
my $time;
pcap_loop($pcap,-1,sub {
    my (undef,$hdr,$data) = @_;
    if ( ! $time || $hdr->{tv_sec}-$time>10 ) {
        $udp->expire($time = $hdr->{tv_sec});
    }
    return $pc->pktin($data,$hdr);
},undef);
```

## ähnliche Module

## Net::Analysis

- oberflächlich gesehen ähnliches, layered Design
- zentraler Event-Dispatcher statt Subscribe der Layer untereinander
- L2,L3 hardcoded in Eventloop, keine Fragmentbehandlung
- zentrale Idee von "Monologen"
    - sammelt alle Daten bis Richtungswechsel bevor Event generiert wird
    - nur für kurze Monologe sinnvoll, sonst Out of Memory
    - fragiles Konzept, auch HTTP funktioniert nicht immer so
    - für Pipelining schrecklicher Workaround - alle Requests erst am Ende der TCP-Session

## Sniffer::HTTP (Corion)

- ebenfalls event (push) basiert
- keine Fragmentbehandlung
- Modulename sagt klar das Ziel: nur HTTP
    - alles zusammengepackt für HTTP-Parsing, Rest egal
- kann wahrscheinlich einfach on Top von Net::Inspect reimplementiert werden

## Net::Sharktools

- nutzt Wiresharks Protokollhandler
  - sehr große Vielfalt
- pull - erst alles einlesen, dann alles verarbeiten
- nur input als pcap (live oder File)

## weitere Networkanalysetools

- irgendwie alles pcap/sflow basiert:
  - pyshark (sharktools für python)
  - tcpdump
  - tshark
  - wireshark
  - ngrep/sipgrep
  - pcapsipdump

## related

- Analyse/MITM SSH via Net::SSH::LibSSH (github)
- Anbindung an Bro via Broccoli.pm (coming soon)
- Pcaps schreiben mit Net::PcapWriter
- fritz-capture = pcap(Fritz!Box)