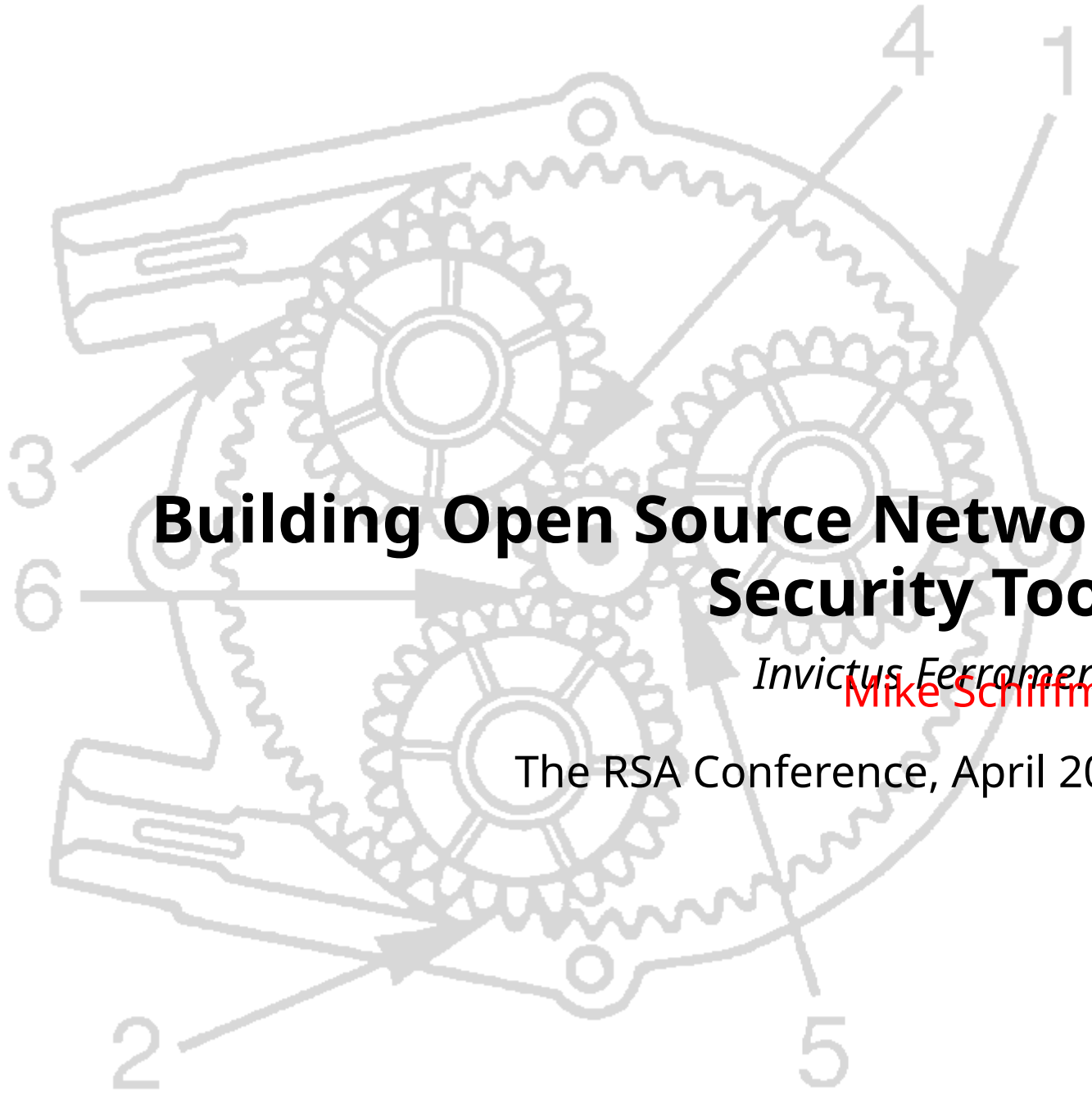


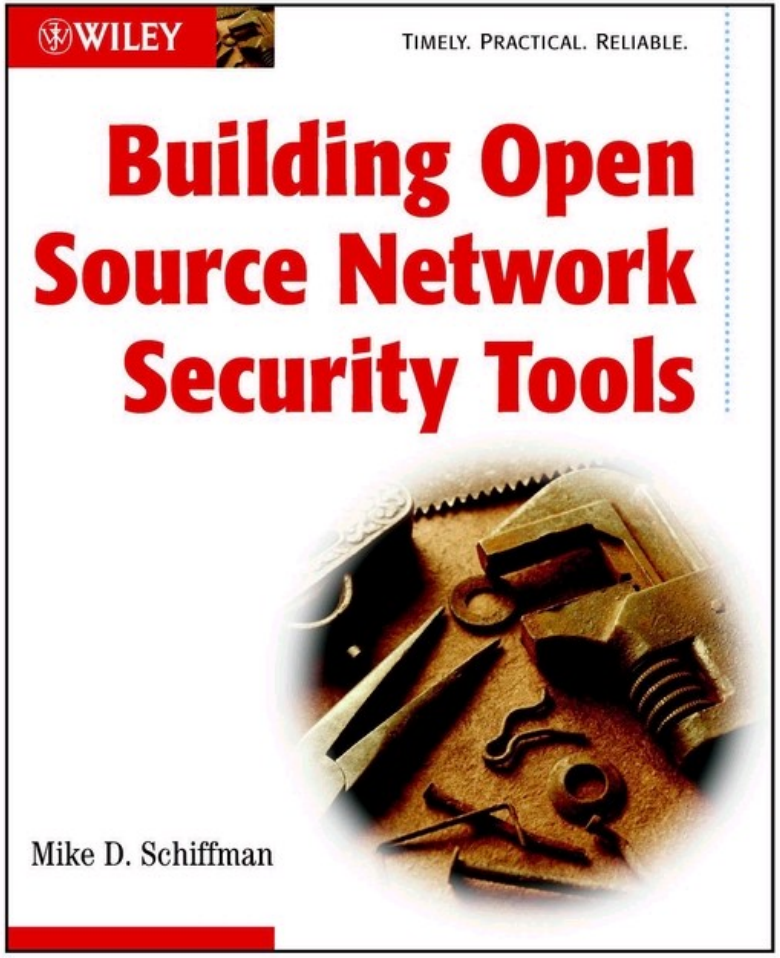
Building Open Source Network Security Tools

Invictus Ferramenta

Mike Schiffman

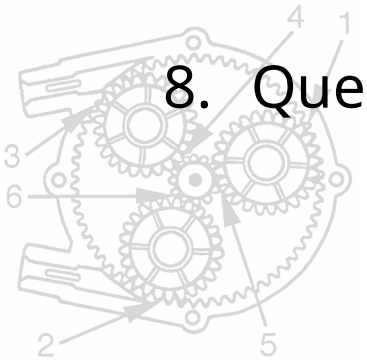
The RSA Conference, April 2003

Today's Presentation is an Overview of This:



Agenda

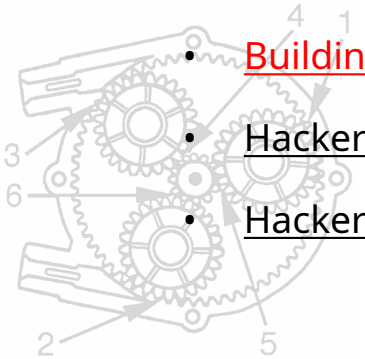
1. Introduction and Overview
2. The Modular Model of Network Security Tools
3. The Component and Technique Layers
4. Network Security Tool Classification
5. Active and Passive Reconnaissance Technique Details
6. Modeling Existing Tools
7. Inside a Network Security Tool: Firewalk Internals
8. Questions and Comments



Primer on Mike Schiffman

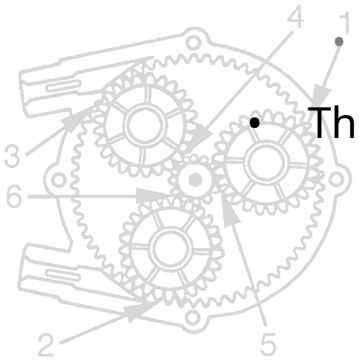
- Researcher, Cisco Syste 

  - Critical Infrastructure Assurance Group (CIAG), Cisco Systems
- Technical Advisory Board for Qualys, IMG Universal
- Consulting Editor for Wiley & Sons
- R&D, Consulting and Speaking background:
  - Firewalk, Libnet, Libsf, Libradiate, Various whitepapers and reports
  - Done time with: @stake, Guardent, Cambridge Technology Partners, ISS
- Author:
  - Building Open Source Network Security Tools, Wiley & Sons
  - Hacker's Challenge Book I, Osborne McGraw-Hill
  - Hacker's Challenge Book II, Osborne McGraw-Hill



Overview

- What you will learn today
  - A new model for conceptualizing and describing network security tools
    - How to apply this model to existing tools
    - How to use this model to rapidly build new tools
  - Common network security tool techniques and how they are codified
- What you should already know
  - General understanding of the TCP/IP protocol suite
    - Primarily layers 1 – 3 (OSI layers 2 – 4)
  - General network security concepts
    - For example; the difference between packet sniffing and port scanning
  - The C programming language



Before we start...

- Where should I spend my focus...?
  - Lots of material
- Show of hands...
  - Libnet
  - Libpcap
  - The Paradigm and NST terminology
  - Code?

Paradigm Overview

Technically Accurate.

Not Tangible.

What is a Network Security Tool?

A network security tool is an algorithmic implement that is designed to probe, assess, or increase the overall safety of or mitigate risk associated with an entity across a communications medium.

We need something better...

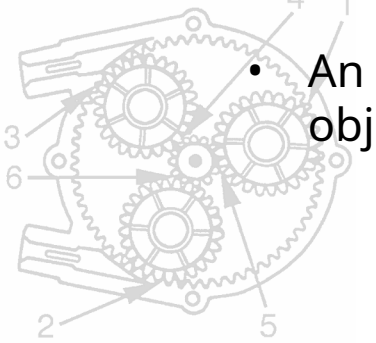
(there is something better)

Unwieldy.

Too Clinical.

A (New) Paradigm

- Functional
- Tangible and Visual
- Specifies a simple taxonomy for grouping and ordering tools
  - Tool Classifications
- Separates a network security tool into three layers or tiers
  - Component, Technique, Control
- Hierarchical dependencies
  - An object at a higher layer has dependencies on one or more objects below it



The Modular Model of Network Security Tools

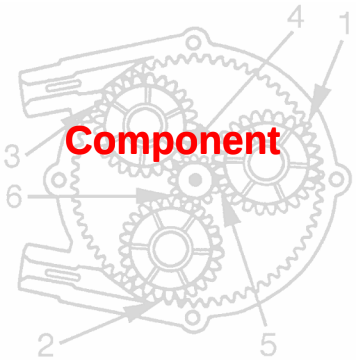
denotes layer dependency

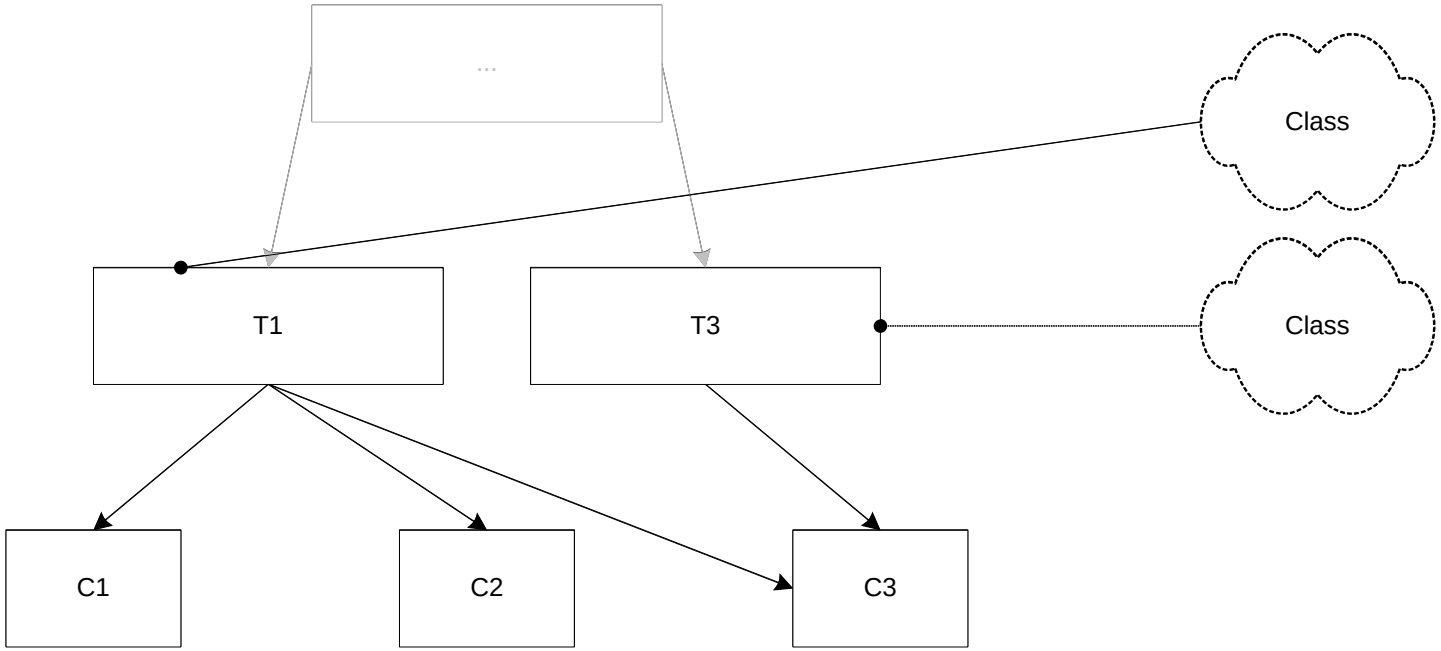
denotes class binding

Control

Technique

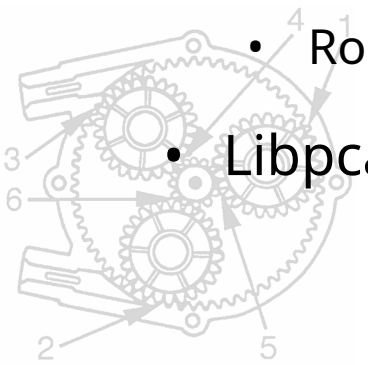
Component



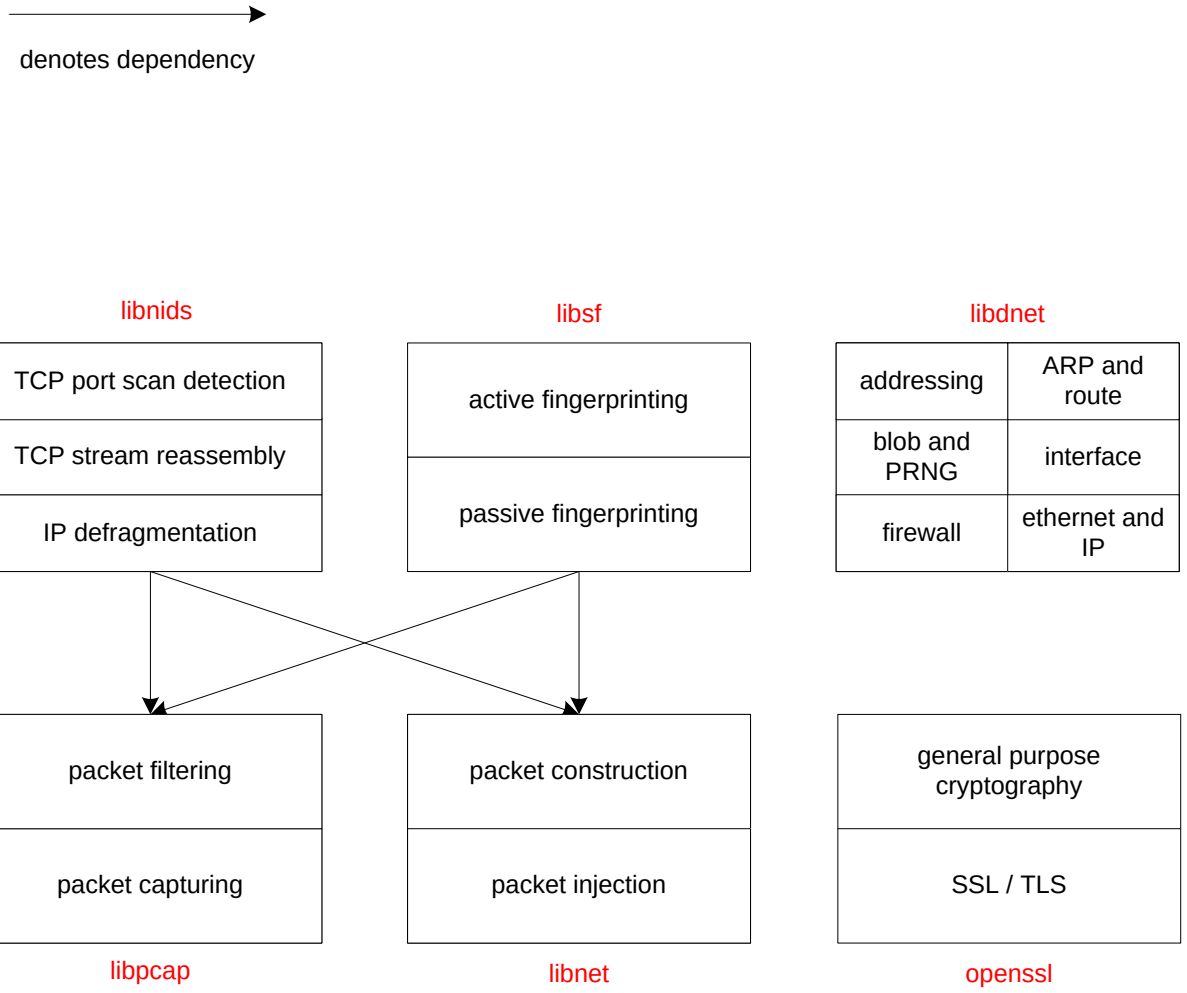


The Component Layer

- Most fundamental layer
- Answers the question "How does this tool *do* what it does?"
- Task oriented and specific
- Components tend to outlay the developmental requirements and restraints of the tool
  - Software Development Lifecycle
- C programming libraries
  - Robust, portable and generally simple APIs
- Libpcap, Libnet, Libsf, Libnids, Libdnet, OpenSSL

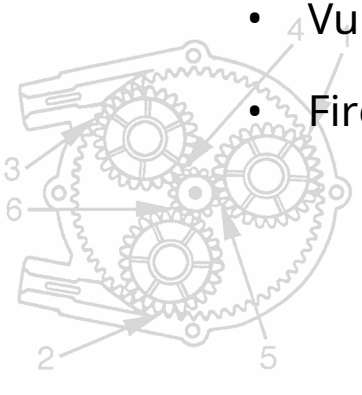


The Component Layer with Dependencies



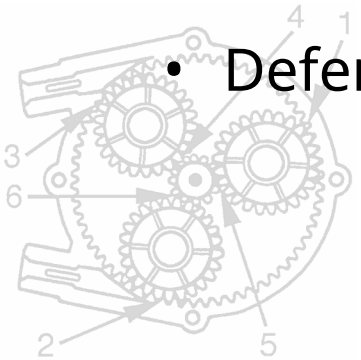
The Technique Layer

- Answers the question: "What *does* this tool do?"
- More abstract and solution focused
- The core essence of the tool is captured at this layer
  - When building or classifying tools, we start here
- Class taxonomy is set at this layer
  - Packet Sniffing (Passive Reconnaissance)
  - Port Scanning (Active Reconnaissance)
  - Vulnerability Testing (Attack and Penetration)
  - Firewalling (Defensive)

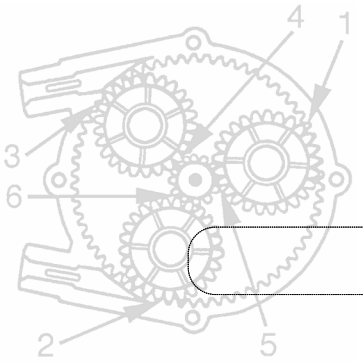


Network Security Tool Taxonomy

- Simple method allowing for tool grouping
- Tied to the Technique Layer
- Tools may certainly fit in more than one category (Venn diagram)
- Passive Reconnaissance
- Active Reconnaissance
- Attack and Penetration
- Defensive



The Technique Layer with Classification Bindings



passive reconnaissance

- packet sniffing

active reconnaissance

- port scanning
- IP expiry
- Firewalking

attack and penetration

- vulnerability scanning
- vulnerability testing

defensive

- intrusion detection
- firewalling
- encryption

denotes tool class

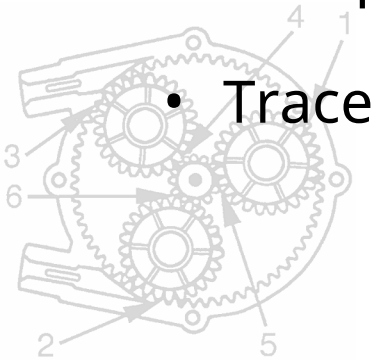
Classification Overview

Passive Reconnaissance Tools

- Gather information in an ostensibly non-detectable or unobtrusive way
- Tend to have long lifecycles in terms of utility
- Changes no state on the entity
- Tcpdump
- Ethereal
- Mailsnarf

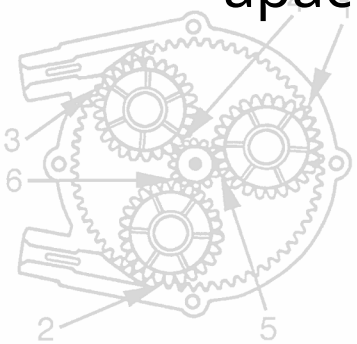
Active Reconnaissance Tools

- Gather information in a detectable way, often by sending network traffic and waiting for responses
- Tend to have long lifecycles in terms of utility
- Changes very little if any state on the entity
- Firewalk
- Strobe
- Nmap
- Traceroute



Attack and Penetration Tools

- Test for the existence of and/or exploit vulnerabilities
- Tools can have a very limited lifetime
  - i.e.: Remote overflow in IIS version 5.0
- Often supported by Reconnaissance Tools
- Nessus
- SSH CRC32 overflow exploit
- apache-scalp.c

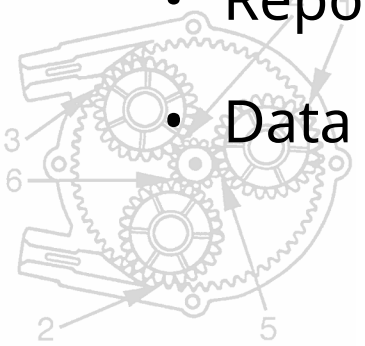


Defensive Tools

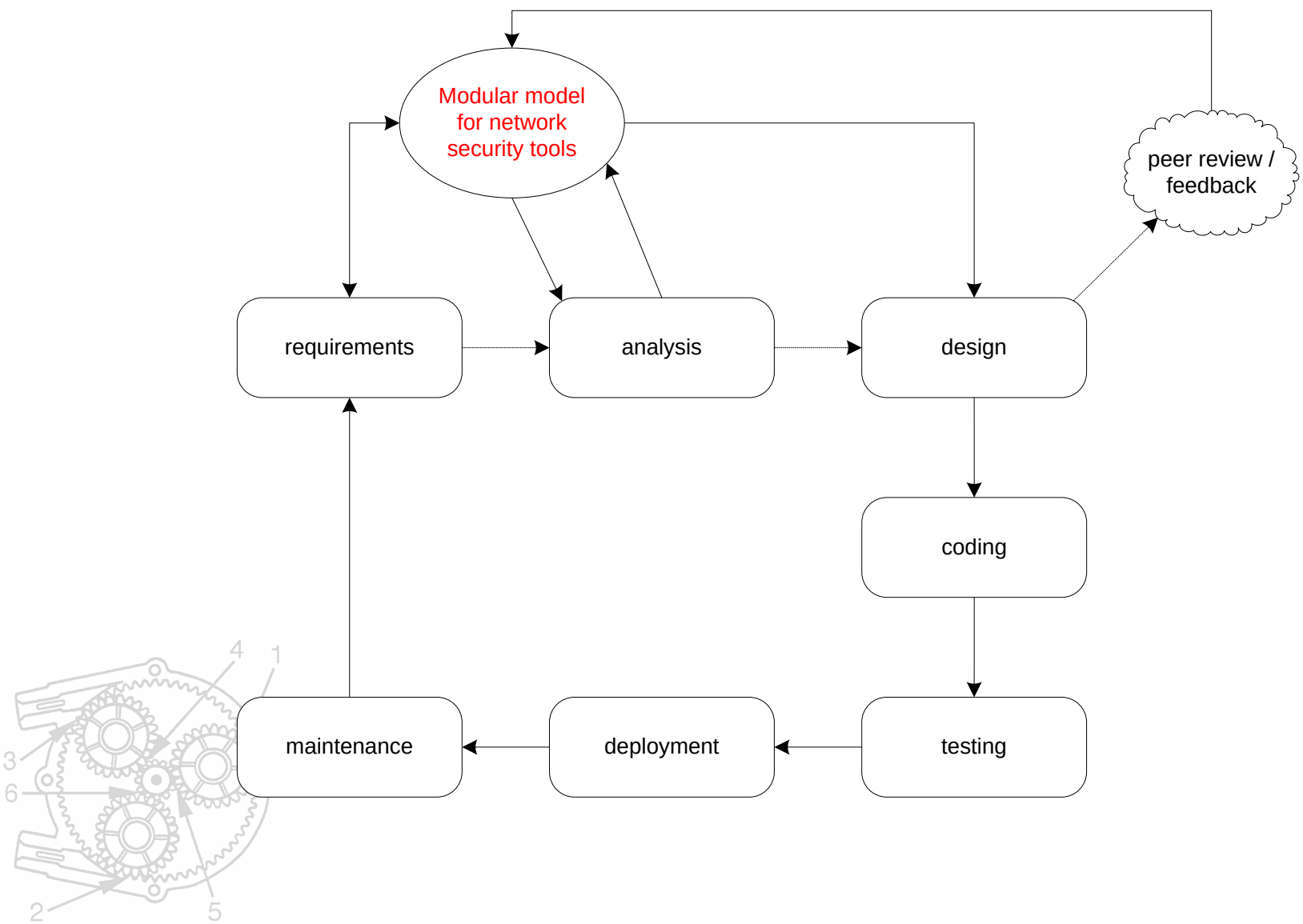
- Keeps an entity safe often by protecting data or detecting illicit activity
- Tend to be more complex and have extended execution lifetimes
- Snort
- GPG
- PF (packet filter on OpenBSD)

The Control Layer

- General abstract “glue layer”
- Can be thought of as a delivery mechanism for techniques
- Less concerned with security-related topics as with program cohesion
- Not the focus of this presentation
- Command and Control
- Reporting
- Data Correlation and Storage



The Model and The Software Development Lifecycle



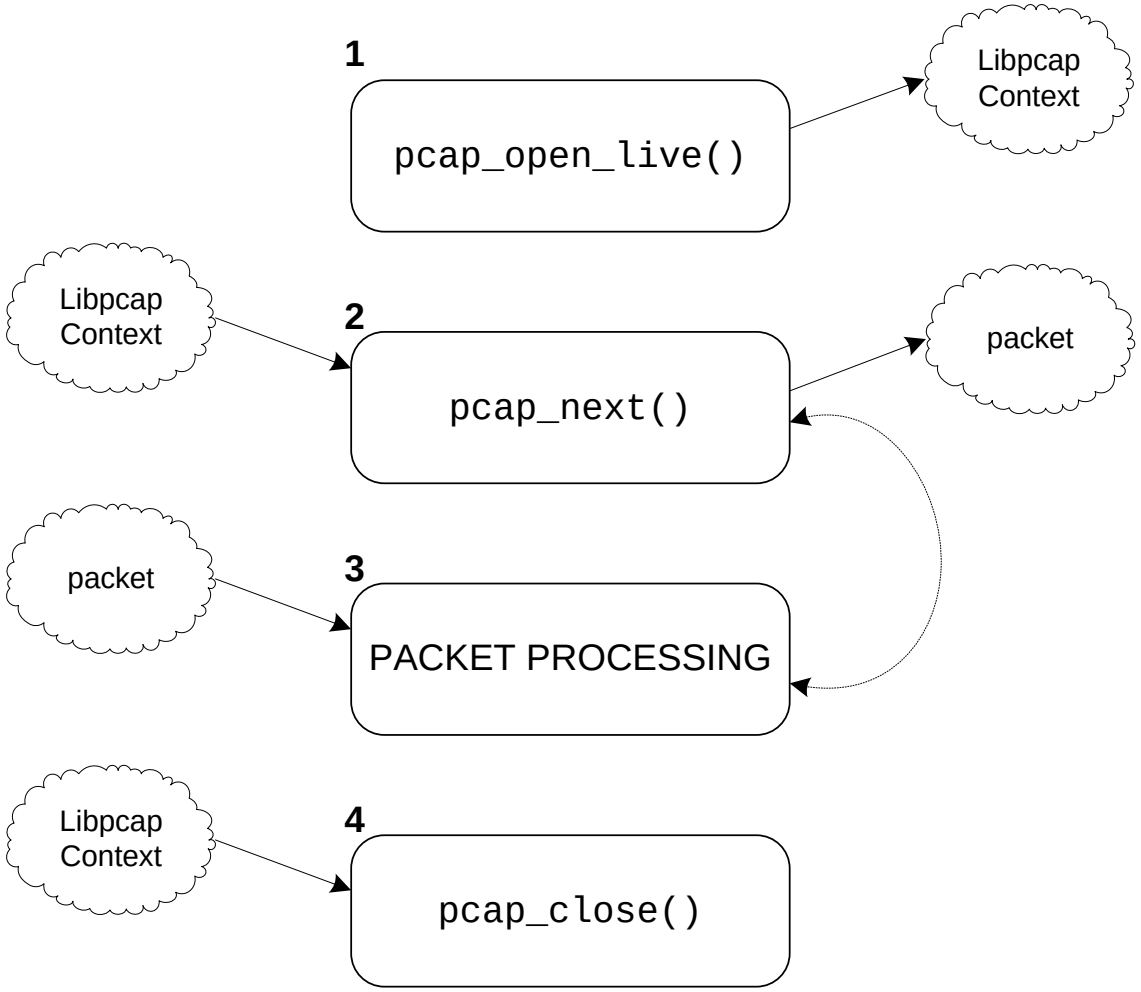
Component Layer Details

Libpcap
Libnet
Libnids
Libsf
Libdnet
OpenSSL

Component Layer Details: Libpcap

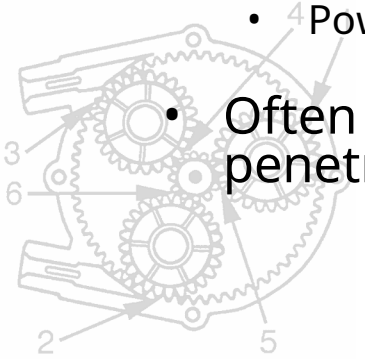
- Library for packet capture and filtering
  - Support for live capture and offline storage
- Useful for building applications that need to do the following:
  - Network statistics collection
  - Network debugging
  - Security monitoring
- Often found in active and passive reconnaissance tools

Typical Libpcap Usage

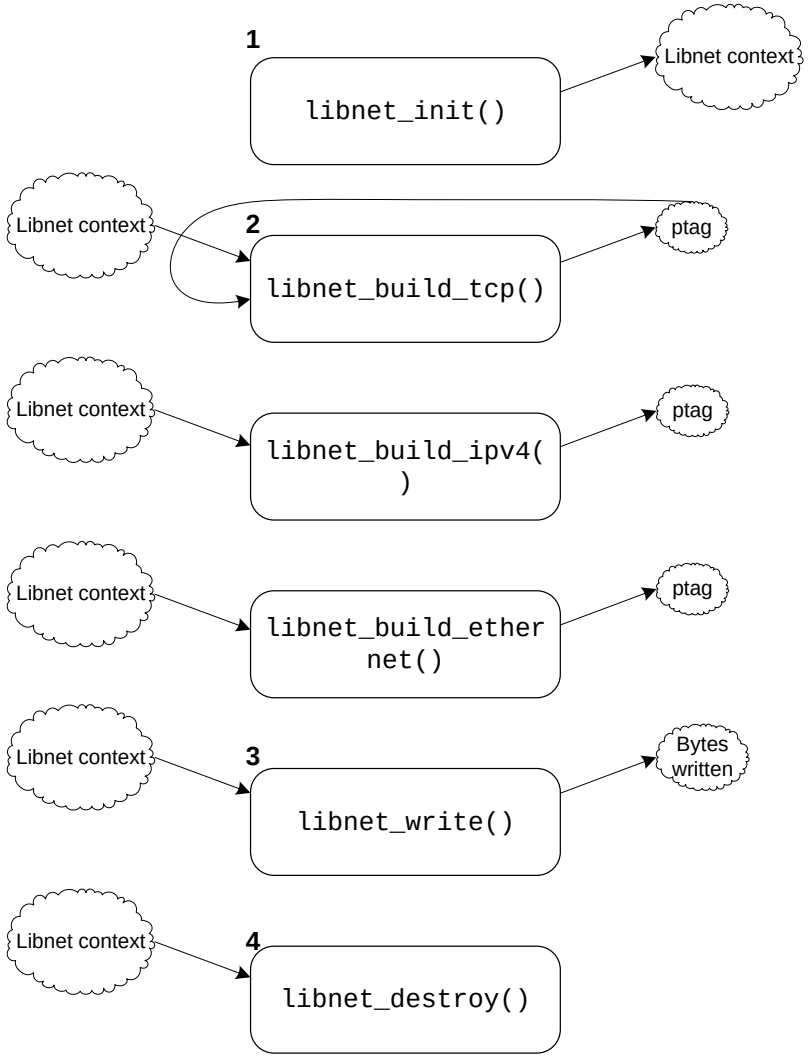


Component Layer Details: Libnet

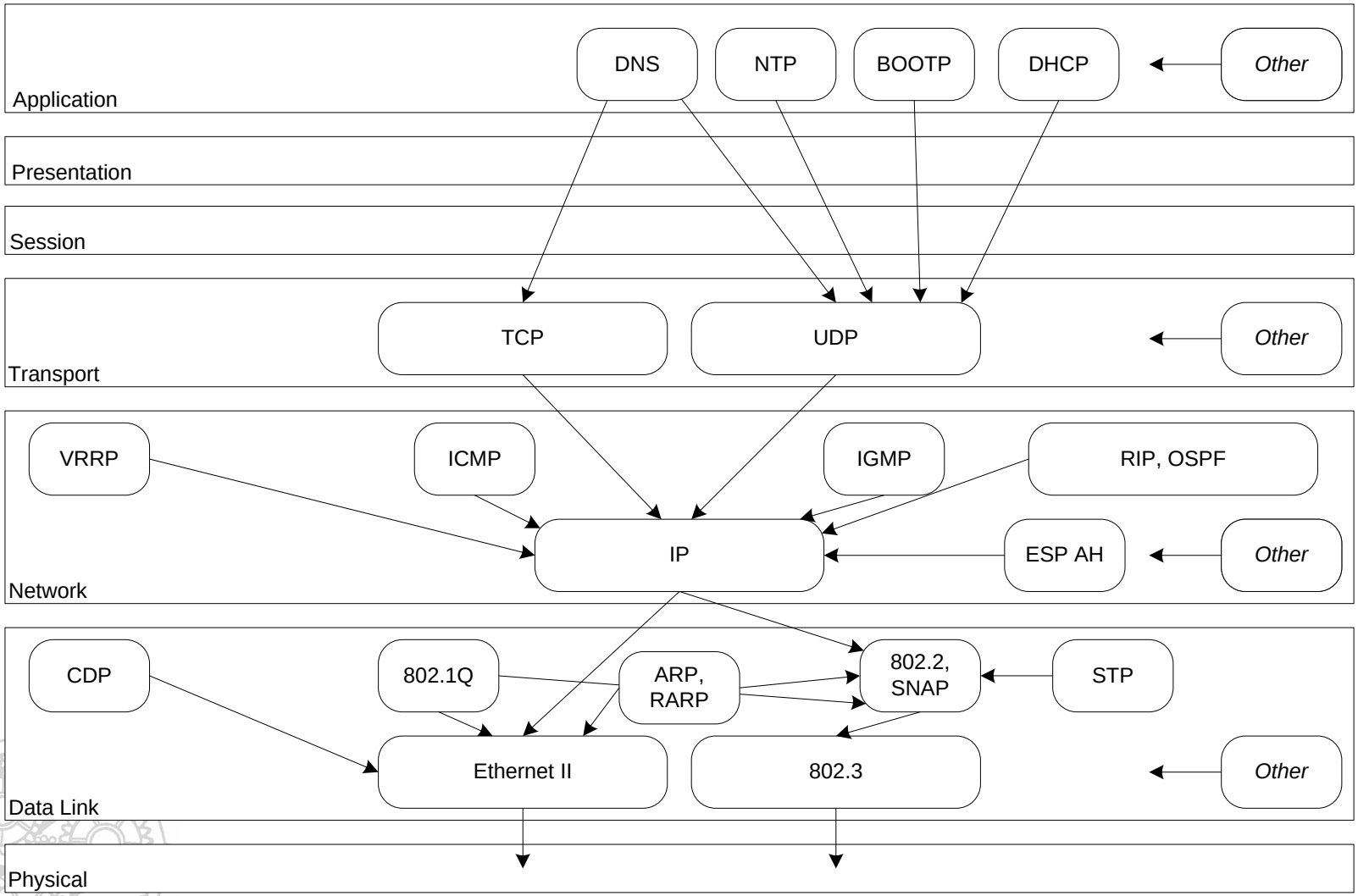
- Library for packet construction and injection
- Useful for building applications that need to do the following:
  - Network security testing
  - Network bandwidth testing
  - Network utility
- Definitely the most debonair and sophisticated of the components – truly a discriminating programmer's component
- New version (1.1.1) is much more robust than its predecessors
  - Simple interface for novice users or
  - Powerful advanced interface
- Often found in active reconnaissance and attack and penetration tools



Typical Libnet Usage

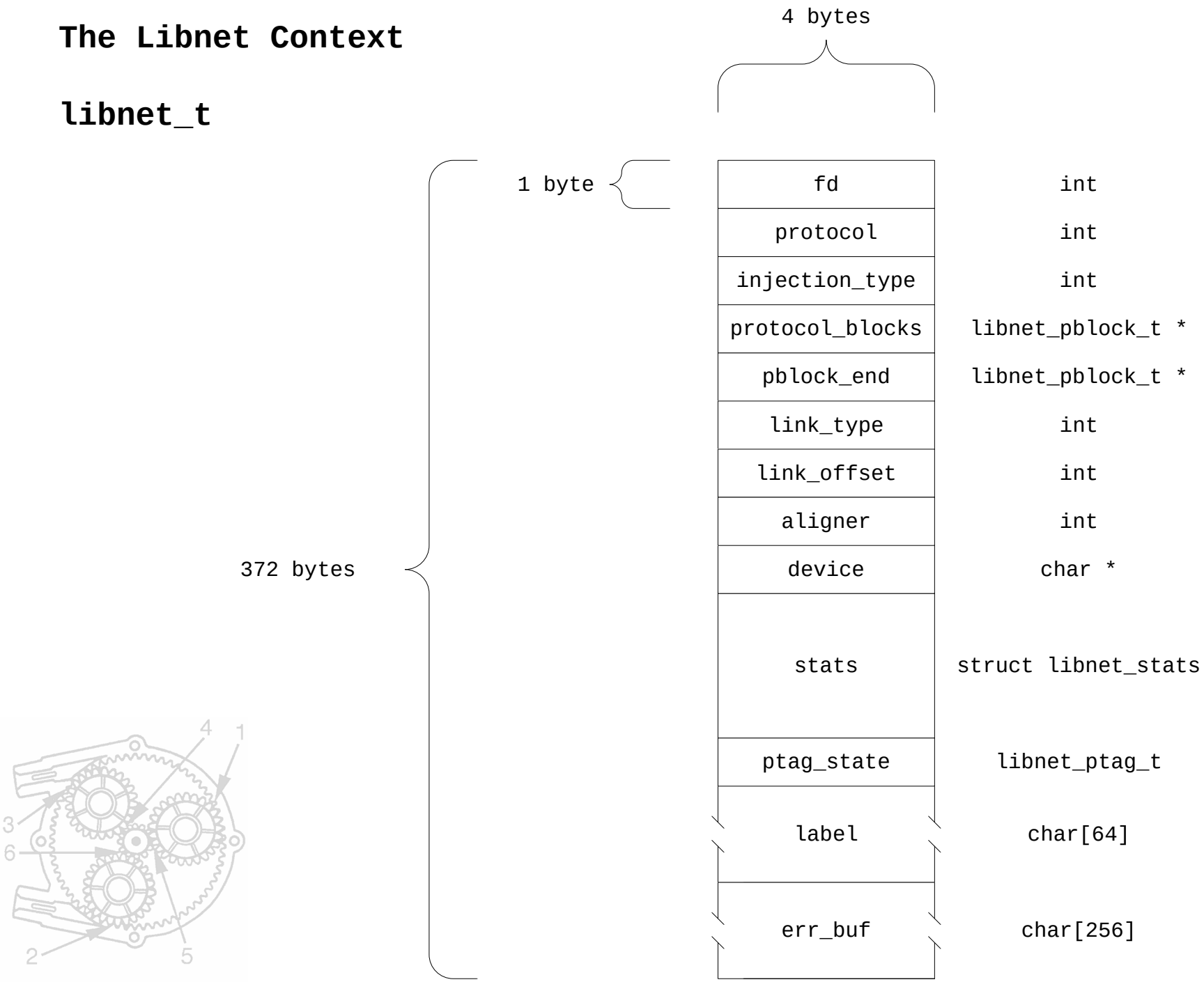


Libnet Supported Protocols



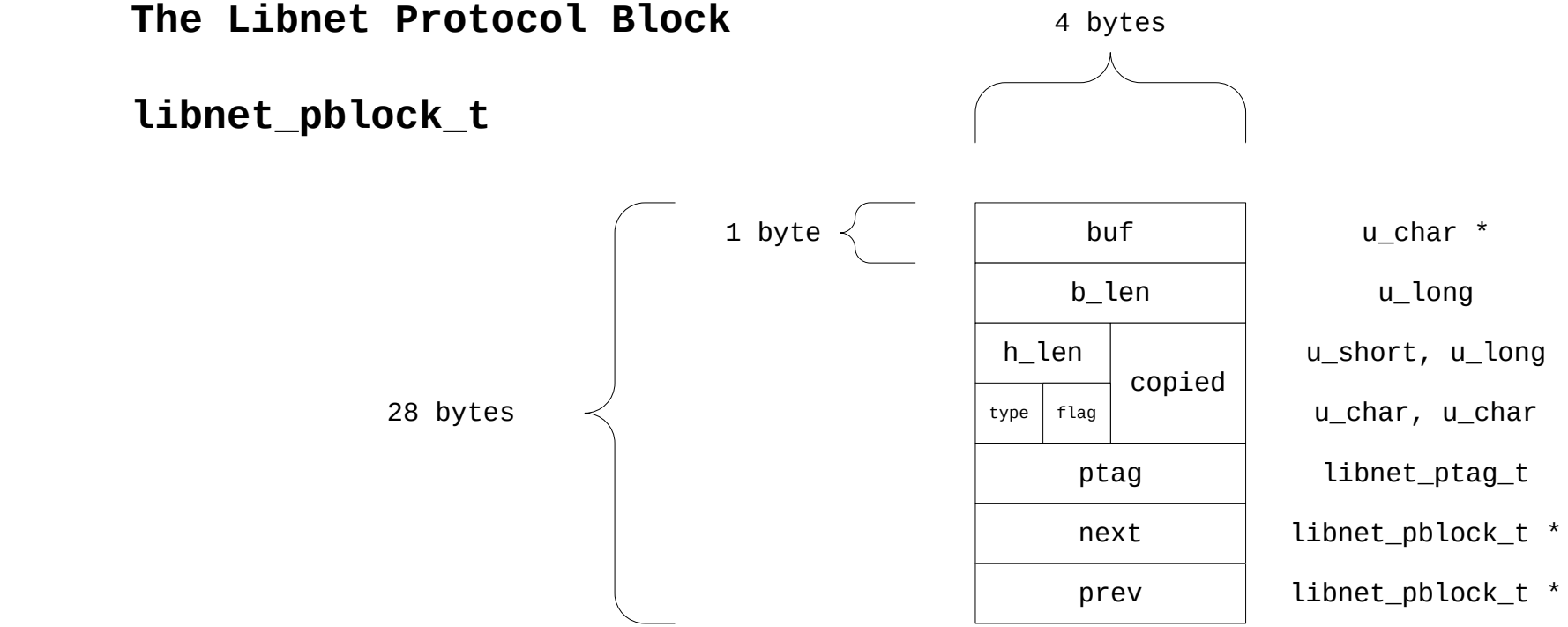
The Libnet Context

libnet_t



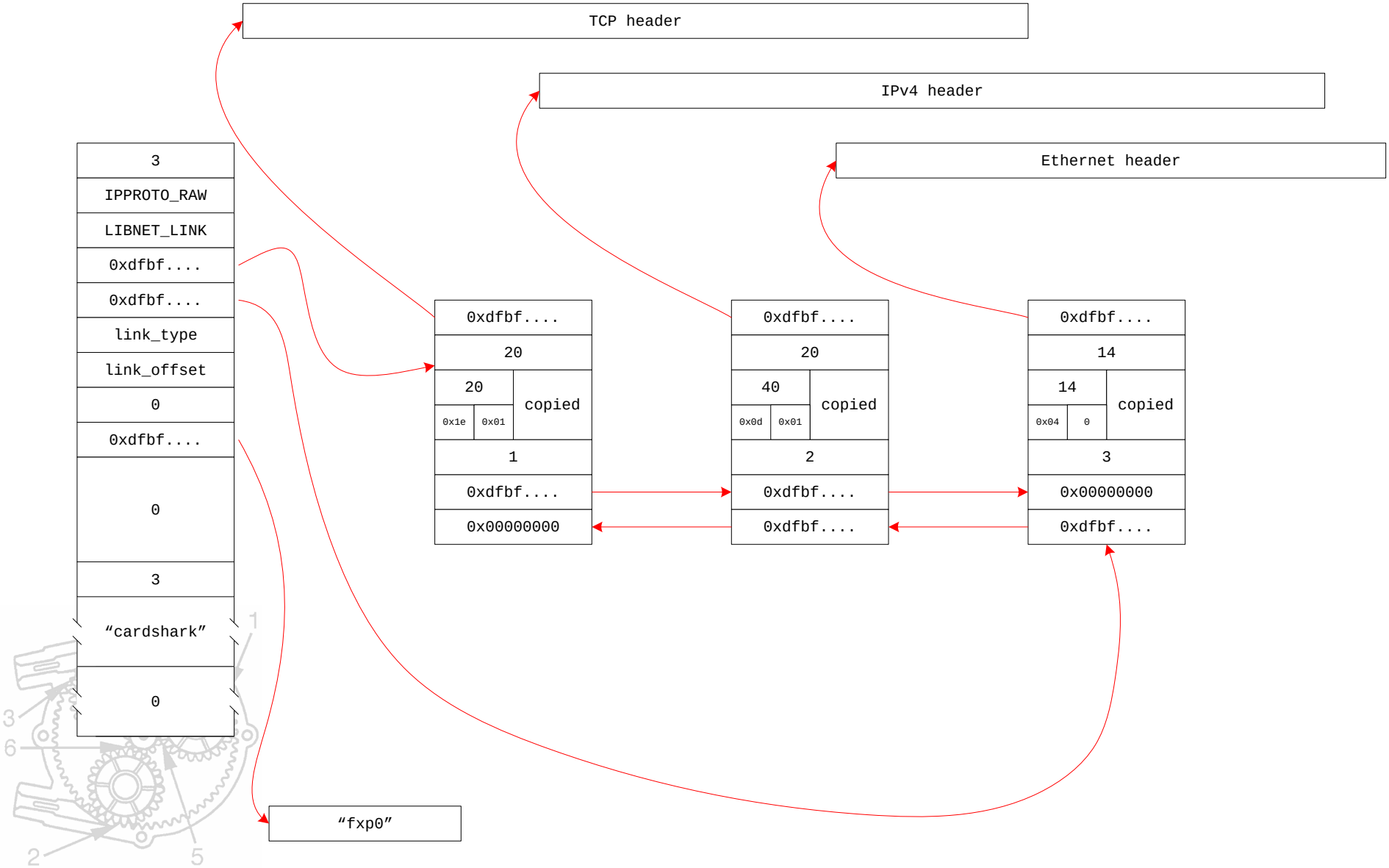
The Libnet Protocol Block

libnet_pblock_t



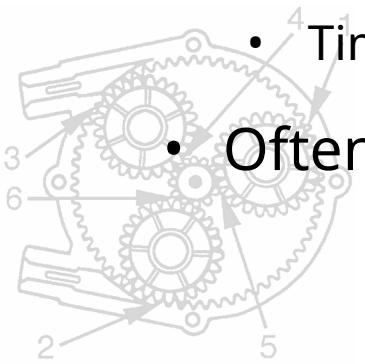
The Libnet Protocol Block

single context, standard linkage for TCP header (prior to coalesce)



Component Layer Details: Libnids

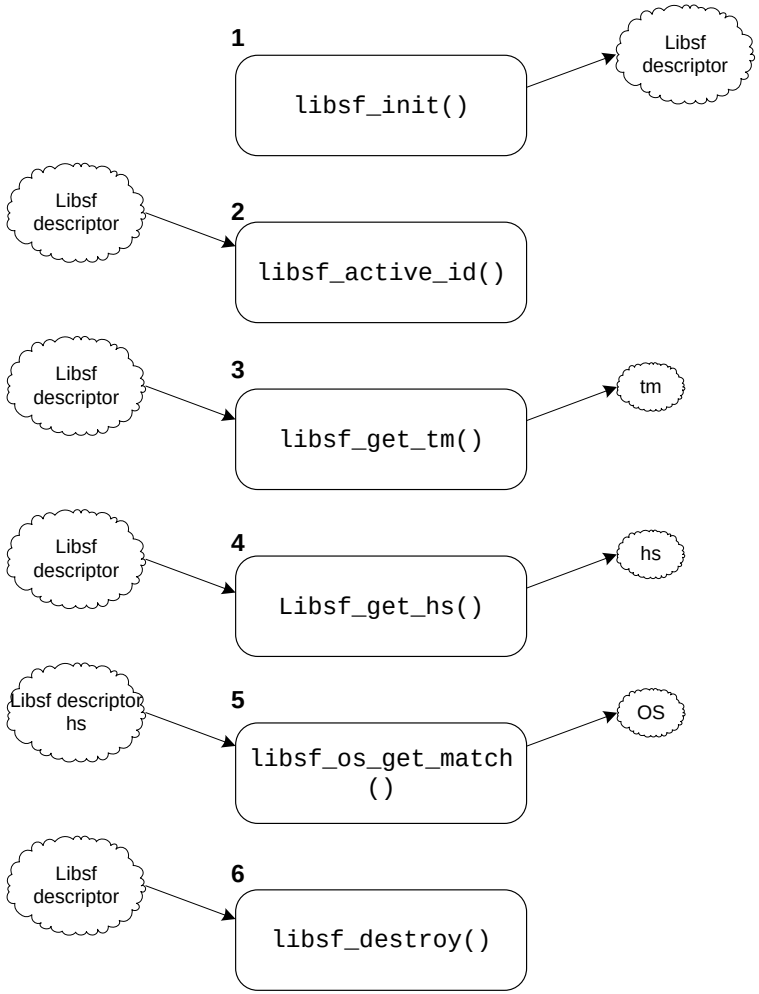
- Library that simulates a NIDS E-box
  - An E-box's job is to sample the environment in which it is specialized for, and convert occurrences in the environment into standard data objects for subsequent storage and/or analysis.
- Built on top of libpcap and libnet
- Offers the following:
  - IP defragmentation
  - TCP stream reassembly
  - Time-based TCP port scan detection
- Often found in defensive tools



Component Layer Details: Libsf

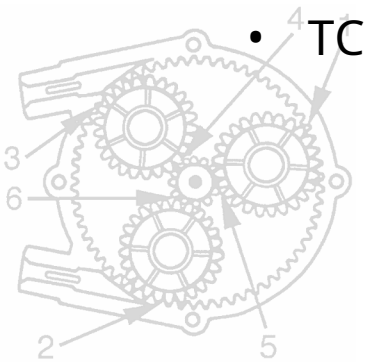
- Library for IP stack fingerprinting to perform remote OS detection
- Built on top of libpcap and libnet
- active and passive fingerprinting methods
- Based off of the nmap database and P0f databases
- Often found in reconnaissance tools

Typical Libsf Usage



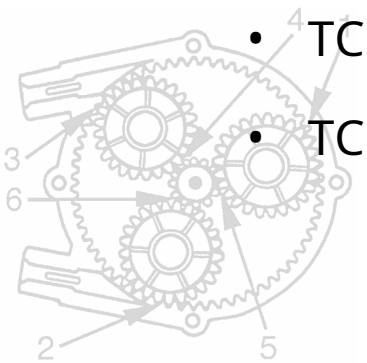
Libsf Active Fingerprinting Tests

- Seven active tests can be performed using fringe packets:
  - TCP SYN to an open port
  - TCP NULL packet to an open port
  - TCP FIN|SYN|PSH|URG packet to an open port
  - TCP ACK packet to an open port
  - TCP SYN packet to a closed port
  - TCP ACK packet to a closed port
  - TCP FIN|PSH|URG to a closed port



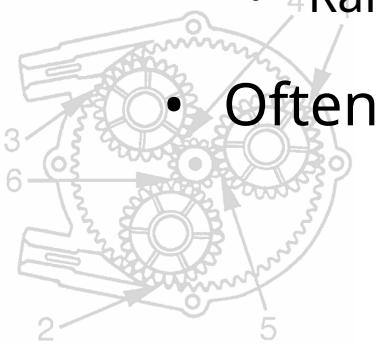
Libsf Passive Fingerprinting Tests

- Eight passive tests can be preformed across incoming TCP SYN packets:
  - Determine original IP TTL
  - IP packet size
  - IP DF bit on or off
  - TCP window scale option present
  - TCP MSS option present
  - TCP SACK option present
  - TCP NOP option present
  - TCP window size



Component Layer Details: Libdnet

- Library for miscellaneous low-level network routines
  - Robust network address manipulation
  - Kernel ARP cache lookup and manipulation
  - Kernel route table lookup and manipulation
  - Network interface lookup and manipulation
  - Network firewall rule manipulation
  - Ethernet frame and IP packet transmission
  - Binary buffer manipulation
  - Random number manipulation
- Often found in all tools



Component Layer Details: OpenSSL

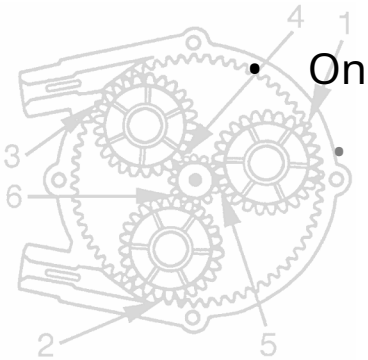
- Library for SSL / TLS and general cryptography
  - SSL/TLS protocols
  - Symmetric cryptographic operations (ciphers, message digests)
  - Asymmetric cryptographic operations (digital signatures, enveloping)
  - Public Key Infrastructure (PKI), including OCSP, rich X509 certificate support, certificate verification, certificate requests, and CRLs
- Often found in defensive tools

Technique Layer Details

Packet Sniffing
Port Scanning
IP Expiry
Firewalking

Technique Layer Details: Packet Sniffing

- Passive Reconnaissance Technique
- Used to capture packets on a network
- Very powerful and useful in its own right
  - However it is also a fundamental building block in more complex tools
- Ethernet
  - 1972, Bob Metcalfe, ALOHA became Ethernet
  - Shared medium (CSMA/CD)
  - Promiscuous mode instructs card to listen to every frame
  - Only works with stations in the same collision domain
    - Bridges, switches, routers, VLANS break sniffing

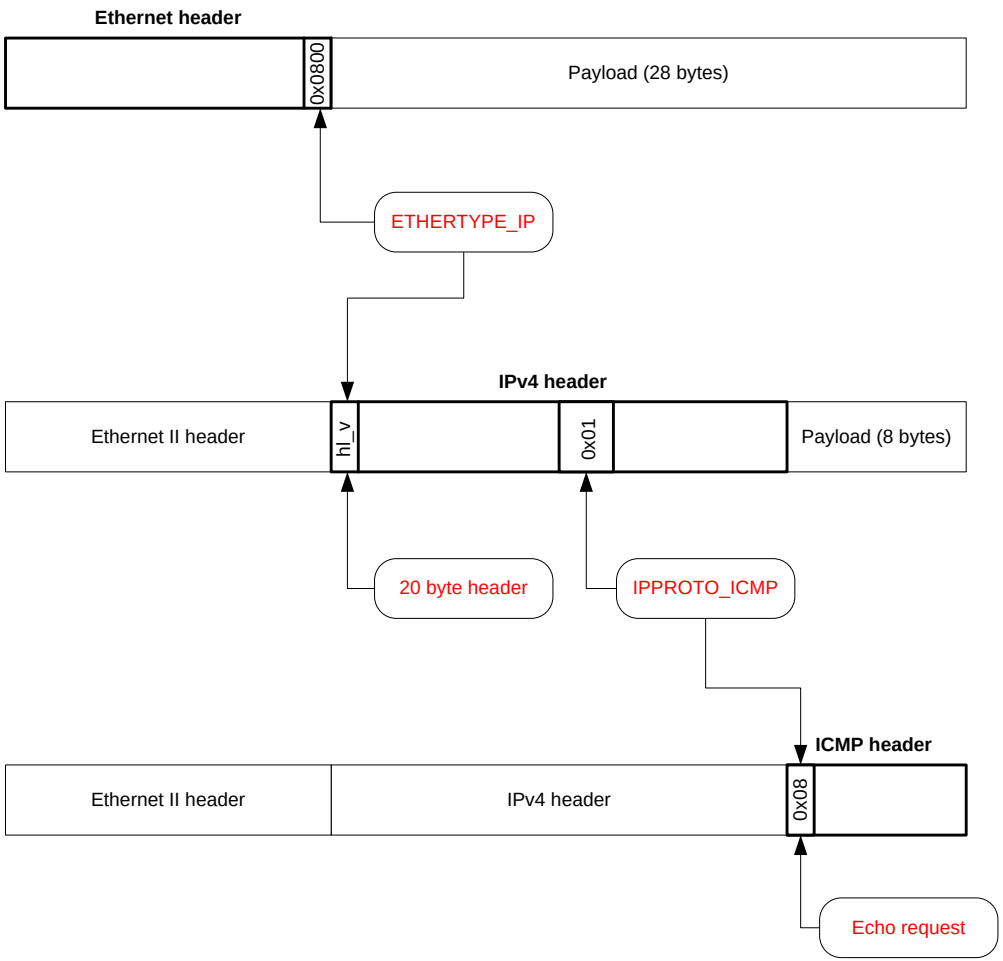


Technique Layer Details: Packet Sniffing

- Packet Demultiplexing
  - Breaking apart an Ethernet frame and passing it the protocol chain
- Protocol Decoding
  - Dissection of the packet at a given OSI layer

Technique Layer Details: Packet Sniffing Processing

Demultiplexing of an Ethernet Frame



Sample Packet Sniffing Code Snippet

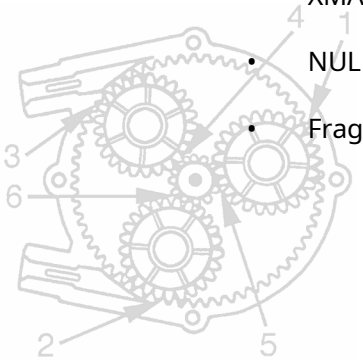
```
packet = (u_char *)pcap_next(vp->p, &vp->h);
/*
 *  Figure out which layer 2 protocol the frame belongs to and call
 *  the corresponding decoding module.  The protocol field of an
 *  Ethernet II header is the 13th + 14th byte.  This is an endian
 *  independent way of extracting a big endian short from memory.  We
 *  extract the first byte and make it the big byte and then extract
 *  the next byte and make it the small byte.
 */
switch (vp->packet[12] << 0x08 | vp->packet[13])
{
    case 0x0800:
        /* IPv4 */
        decode_ip(&vp->packet[14], vp->flags);
        break;
    case 0x0806:
        /* ARP */
        decode_arp(&vp->packet[14], vp->flags);
        break;
    default:
        /* We're not bothering with 802.3 or anything else */
        decode_unknown(&vp->packet[14], vp->flags);
        break;
}
```

Technique Layer Details: Port Scanning

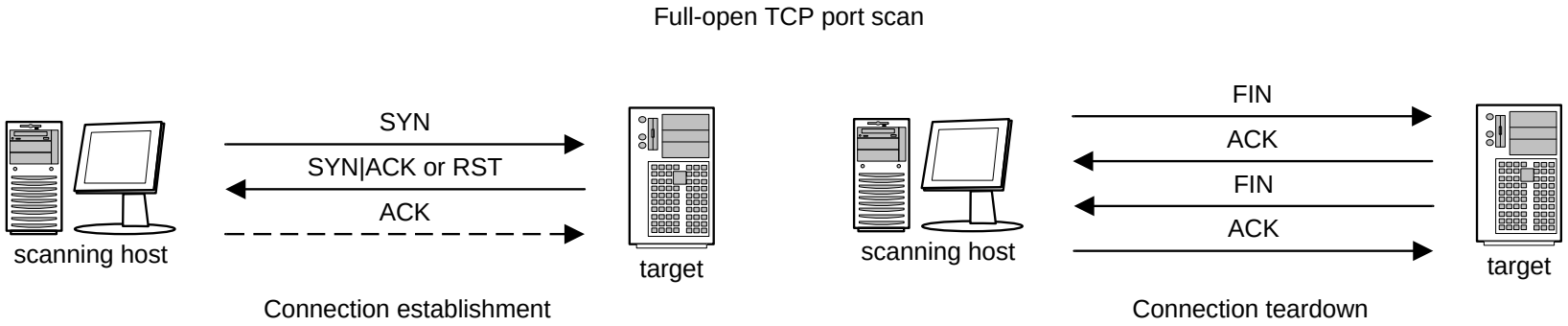
- Active Reconnaissance Technique
- Used to determine TCP and UDP port status
  - Open, Closed, and optionally what application is listening
- Many Considerations
  - Protocol
  - Detection and Filtering
  - Time and Bandwidth

Technique Layer Details: Port Scanning Mechanics

- Full-open
- Ident
- FTP bounce
- Half-open
  - Side effect RST
- Parallel
- UDP
- Stealth
  - FIN
  - XMAS (URG|ACK|PSH)
  - NULL
  - Fragmented IP

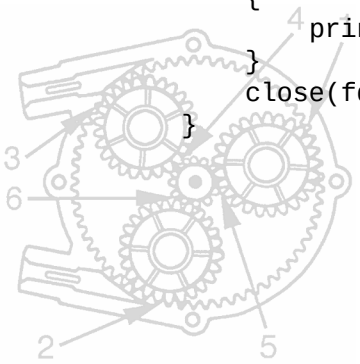


Technique Layer Details: Port Scanning



Sample Port Scanning Code Snippet

```
int fd, n, c;
struct sockaddr_in addr;
u_short port_list[] = {22, 23, 25, 80, 6000, 0};
addr.sin_family      = AF_INET;
addr.sin_addr.s_addr = 0x200a8c0;  /* 192.168.0.2 in network byte order */

for (n = 0; port_list[n] != 0; n++)
{
    addr.sin_port = htons(port_list[n]);
    fd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (fd == -1)
    {
        /* error */
    }
    c  = connect(fd, (struct sockaddr *)&addr, sizeof (addr));
    if (c == -1)
    {
        /* error */
    }
    else if (c == 0)
    {
        printf("port %d open\n", port_list[n]);
    }
    else
    {
        printf("port %d closed\n", port_list[n]);
    }
    close(fd);
}
```



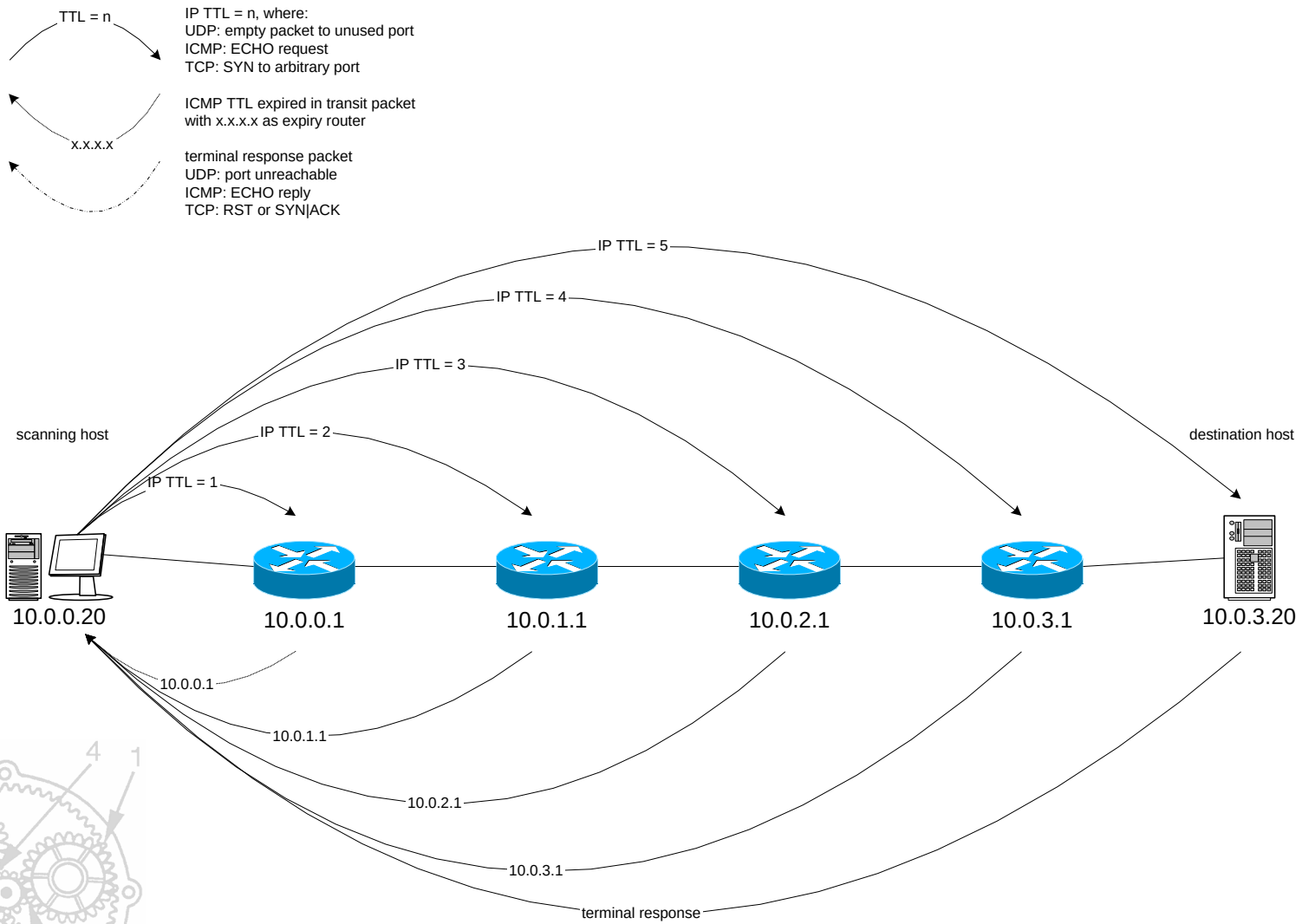
Technique Layer Details: IP Expiry

- Active Reconnaissance Technique
- Used to map network devices en route to a target host
  - Van Jacobson, 1988, Traceroute
  - Originally used to trace IP packets to a particular destination host
  - Was extended into Firewalking

Technique Layer Details: IP Expiry Transports

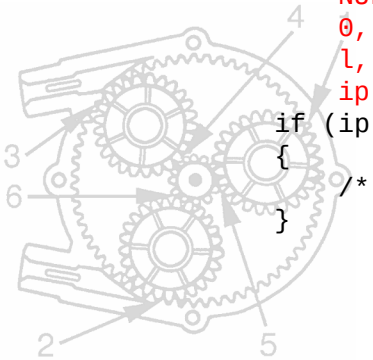
- Protocol specific terminal packet semantics
  - UDP
    - Open port: undefined (no response)
    - Closed port: ICMP port unreachable
  - ICMP
    - ICMP echo reply
  - TCP SYN
    - Open port: SYN|ACK
    - Closed port: RST

Technique Layer Details: IP Expiry



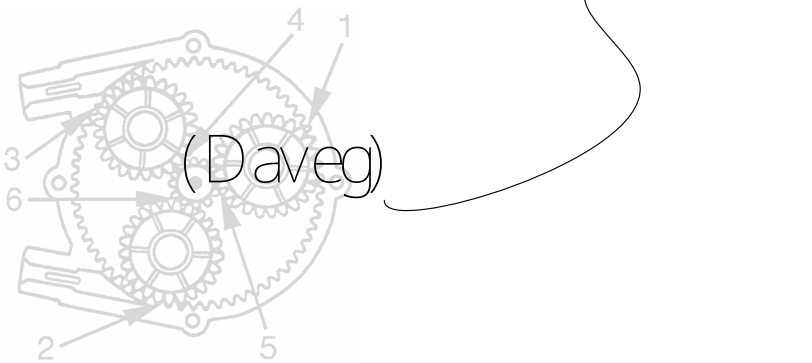
Sample IP Expiry Code Snippet

```
pcap_t *p;
libnet_t *l;
{
time_t start;
u_char *packet;
int c, ttl, done;
char *device = "fxp0";
struct pcap_pkthdr ph;
libnet_ptag_t icmp, ip;
u_long src_ip = 0x1400000a;     /* 10.0.0.20 in network byte order */
u_long dst_ip = 0x1403000a;     /* 10.0.3.20 in network byte order */
struct libnet_icmpv4_hdr *icmp_h;
struct libnet_ipv4_hdr *ip_h, *oip_h;
char errbuf[LIBNET_ERRBUF_SIZE];
l = libnet_init(LIBNET_RAW4, NULL, errbuf);
if (l == NULL)
{
    /* error */
}
p = pcap_open_live(device, 60, 0, 500, errbuf);
if (p == NULL)
{
    /* error */
}
for (done = icmp = ip = 0, ttl = 1; ttl <31 && !done; ttl++)
{
    icmp = libnet_build_icmpv4_echo(
        ICMP_ECHO,
        0,
        0,
        242,
        ttl,
        NULL,
        0,
        l,
        icmp);
    if (icmp == -1)
    {
        /* error */
    }
    ip = libnet_build_ipv4(
        LIBNET_IPV4_H + LIBNET_ICMPV4_ECHO_H,
        0,
        242,
        0,
        ttl,
        IPPROTO_ICMP,
        0,
        src_ip,
        dst_ip,
        NULL,
        0,
        l,
        ip);
    if (ip == -1)
    {
        /* error */
    }
    c = libnet_write(l);
    if (c == -1)
    {
        /* error */
    }
    fprintf(stderr, "Hop %02d: ", ttl);
    for (start = time(NULL); (time(NULL) - start) < 2; )
    {
        packet = (u_char *)pcap_next(p, &ph);
        if (packet == NULL)
        {
            continue;
        }
        /* assume ethernet here for simplicity */
        ip_h = (struct libnet_ipv4_hdr *)(packet + 14);
        if (ip_h->ip_p == IPPROTO_ICMP)
        {
            icmp_h = (struct libnet_icmpv4_hdr *)(packet + 34);
            /* expired in transit */
            if (icmp_h->icmp_type == ICMP_TIMXCEED &&
                icmp_h->icmp_code == ICMP_TIMXCEED_INTRANS)
            {
                oip_h = (struct libnet_ipv4_hdr *)(packet + 42);
                if (oip_h->ip_id == htons(242))
                {
                    fprintf(stderr, "%s\n",
                        libnet_addr2name4(ip_h->ip_src.s_addr, 0));
                    break;
                }
            }
            /* terminal response */
            if (icmp_h->icmp_type == ICMP_ECHOREPLY)
            {
                if (icmp_h->icmp_id == 242 && icmp_h->icmp_seq == ttl)
                {
                    fprintf(stderr, "%s\n",
                        libnet_addr2name4(ip_h->ip_src.s_addr, 0));
                    done = 1;
                    break;
                }
            }
        }
    }
}
```



Technique Layer Details: Firewalking

- Active Reconnaissance Technique
  - Based off of IP expiry
- Used to determine ACL filtering rules on a packet forwarding device
  - Schiffman, Goldsmith, 1998

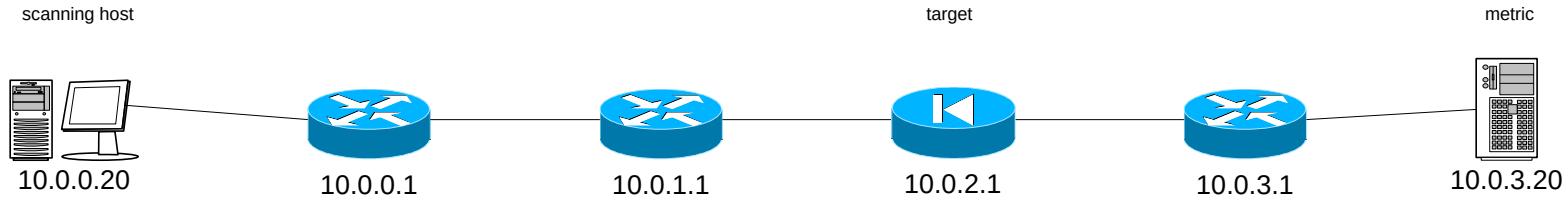


Technique Layer Details: Firewalking

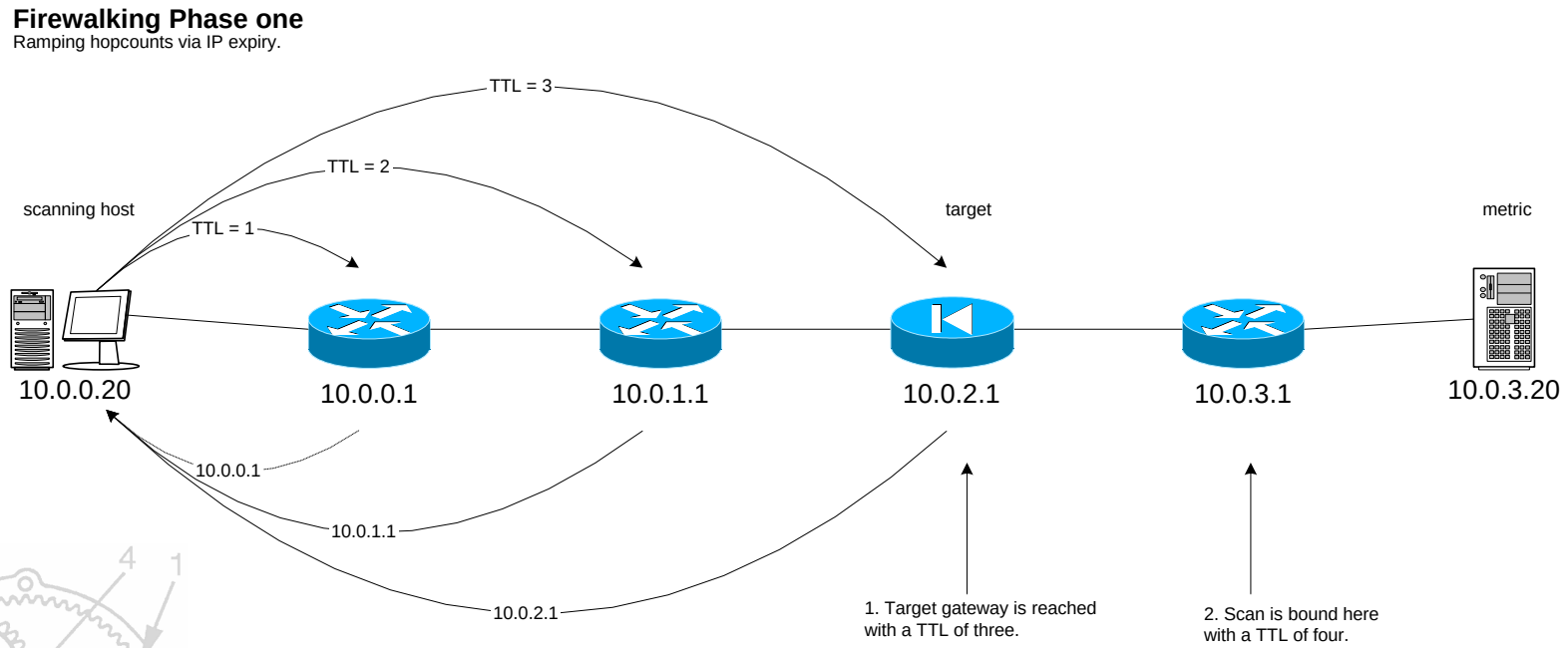
- Send out a TCP or UDP packet with an IP TTL one greater of the target gateway
  - Packet passed by gateway ACL: ICMP TTL expired in transit
  - Packet denied by gateway: No response
- Requires two hosts, target and metric
  - Target is the target gateway to be scanned
  - Metric is a host or gateway downstream from the target
    - Doesn't have to be reachable

Technique Layer Details: Firewalking

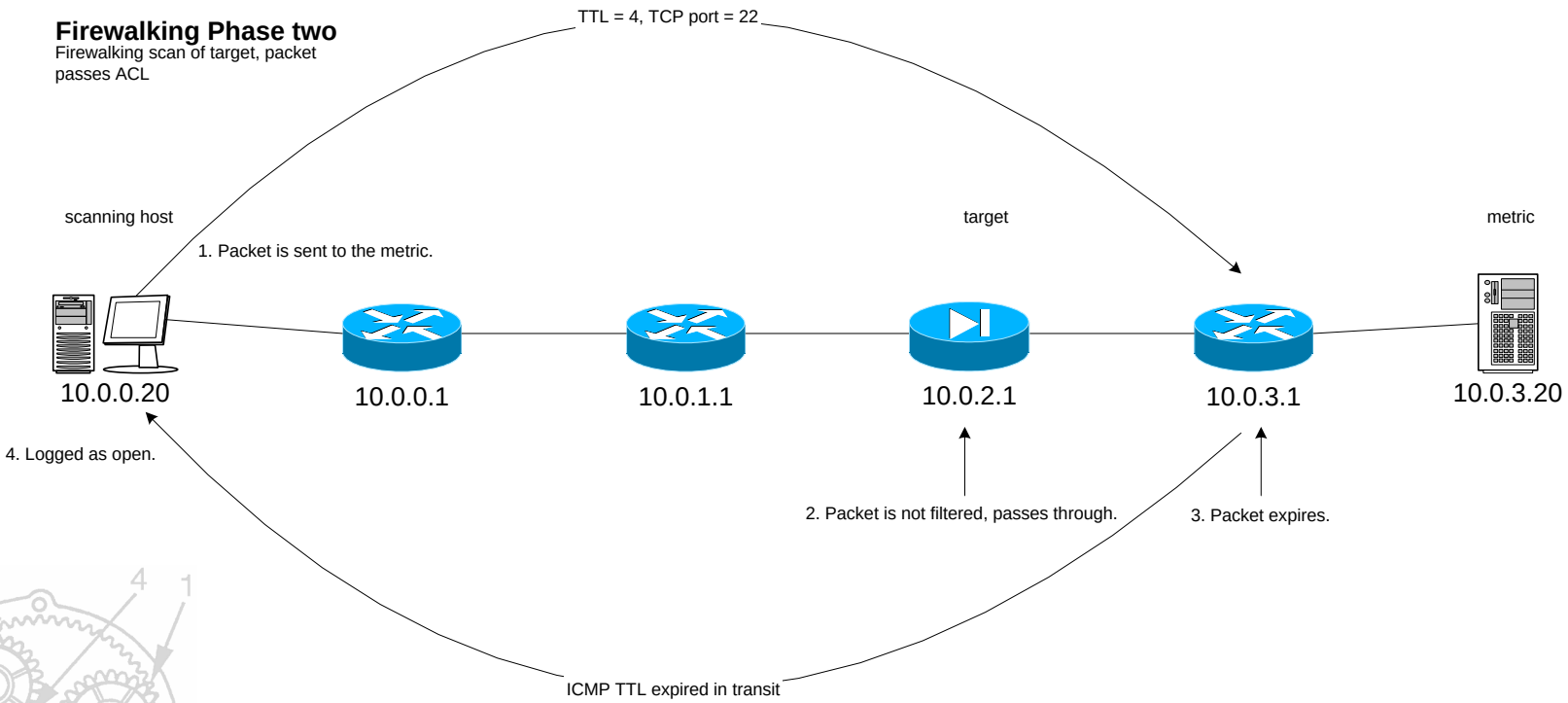
Firewalking host breakdown



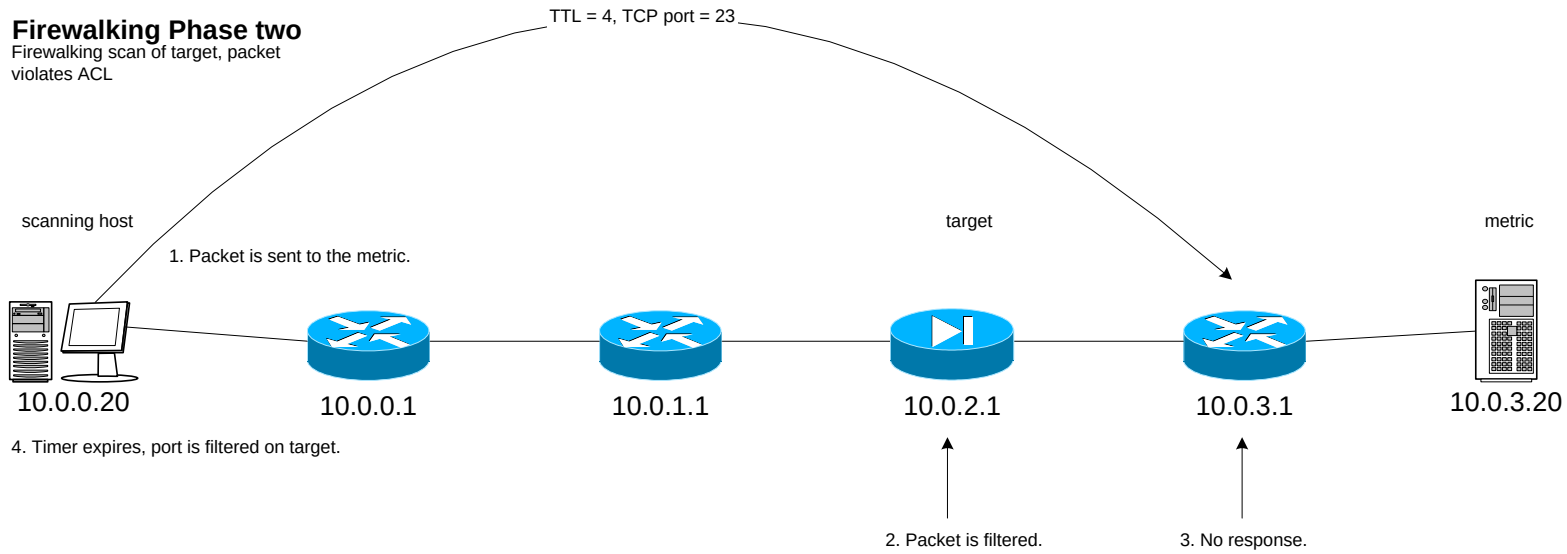
Technique Layer Details: Firewalking (Phase One: Hopcount Ramping)



Technique Layer Details: Firewalking (Phase Two: Scanning, Packet is not Filtered)



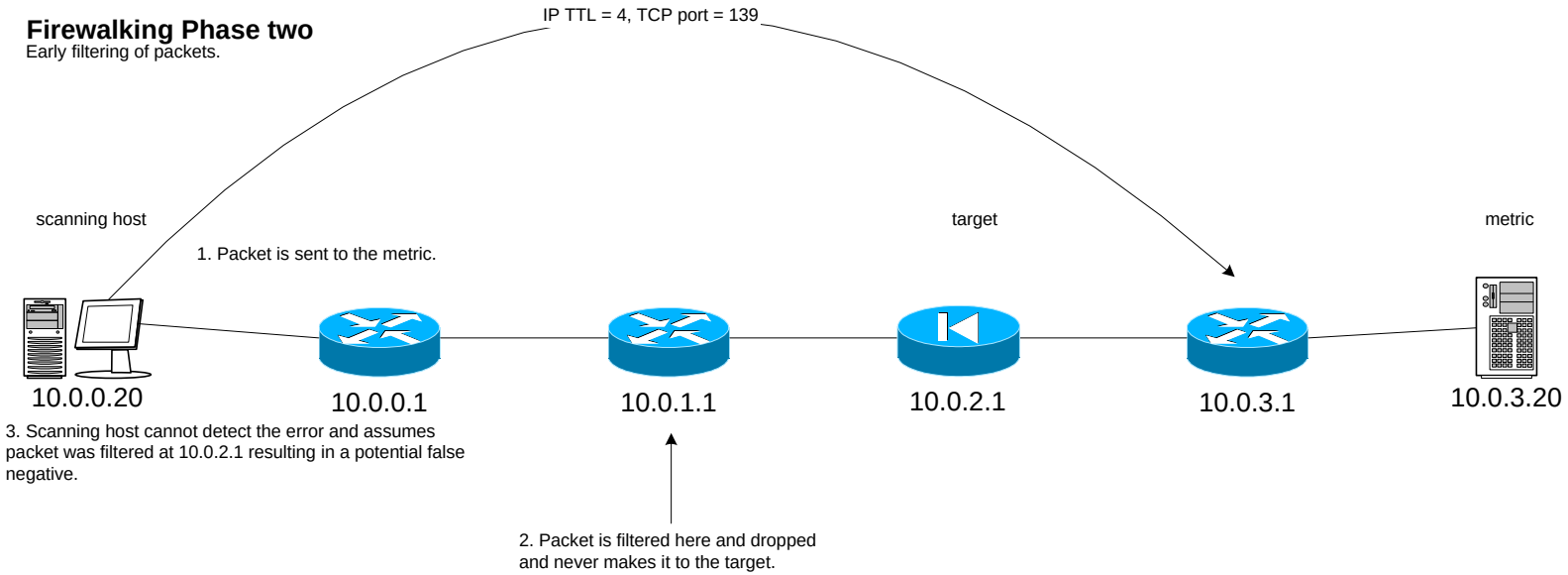
Technique Layer Details: Firewalking (Phase Two: Scanning, Packet is Filtered)



Firewalk Packet Loss

- Packets can be dropped for a variety of reasons
  - IP is an unreliable network
  - However, what if there is a prohibitive filter on a gateway prior to the target?
    - We would get false negatives reporting our packet is being filtered on the target when in fact it is being filtered by another host…

Technique Layer Details: Firewalking (Phase Two: Early Filtering of Packets)

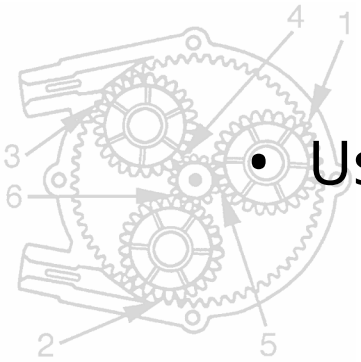


Firewalk Early Packet Filtering Solution

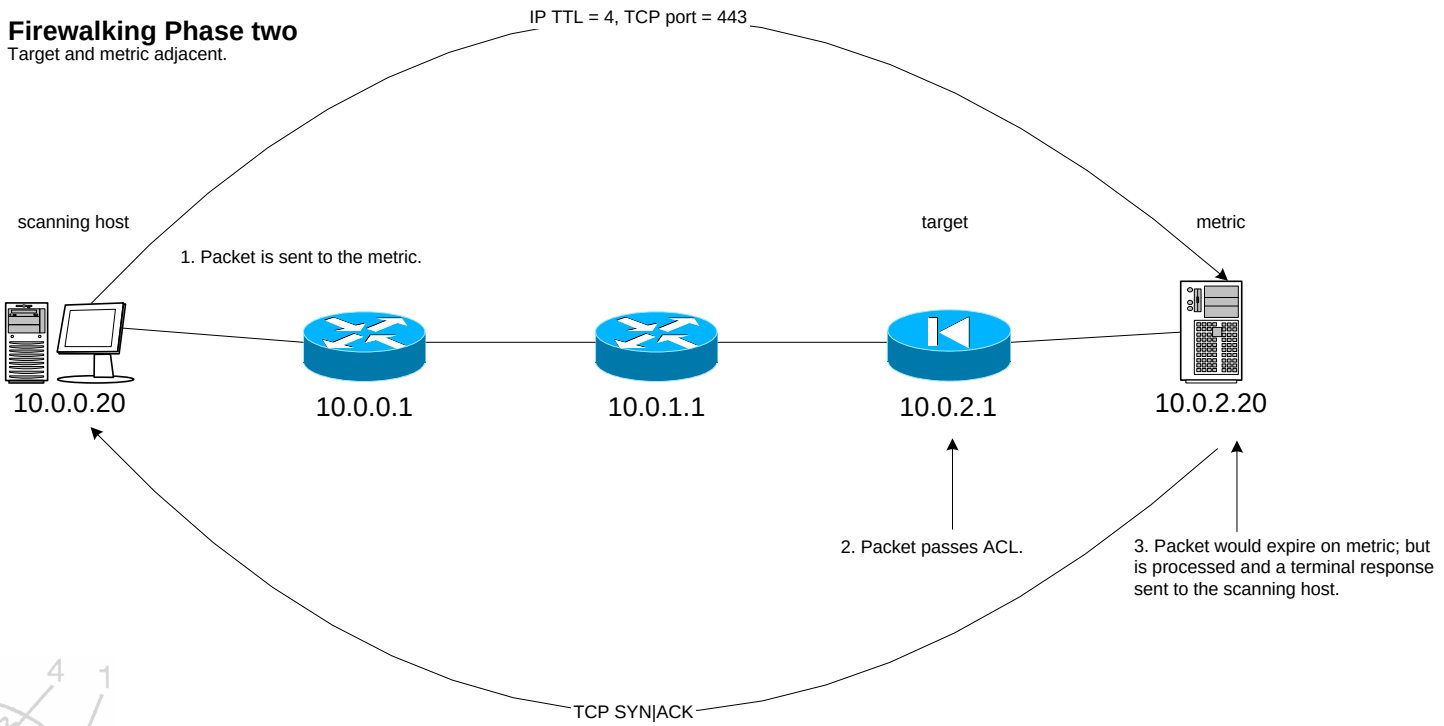
- We have two solutions:
  - Performing a “creeping walk” on each intermediate hop en route to the target. This will determine which gateway has the prohibitive filter
  - Physically relocate the scanning host to another part of the network so it no longer has to pass through the prohibitive filter in question
    - This may not always be an option

Firewalk Adjacent Target and Metric

- Target and metric and topologically adjacent
  - Metric is exactly one hop downstream from the target
- If packet violates ACL, nothing happens out of the ordinary
  - Scan times out and ACL is noted
- If packet is passed by the target, it is processed as per RFC 1122
  - Results vary, but packet is generally processed as per the protocol specific terminal packet semantics as with IP expiry
- Using this, additional scanning can be performed



Technique Layer Details: Firewalking (Phase Two: Scanning, Adjacent Target and Metric)



Modeling Existing Tools

Traceroute Modeled

Traceroute

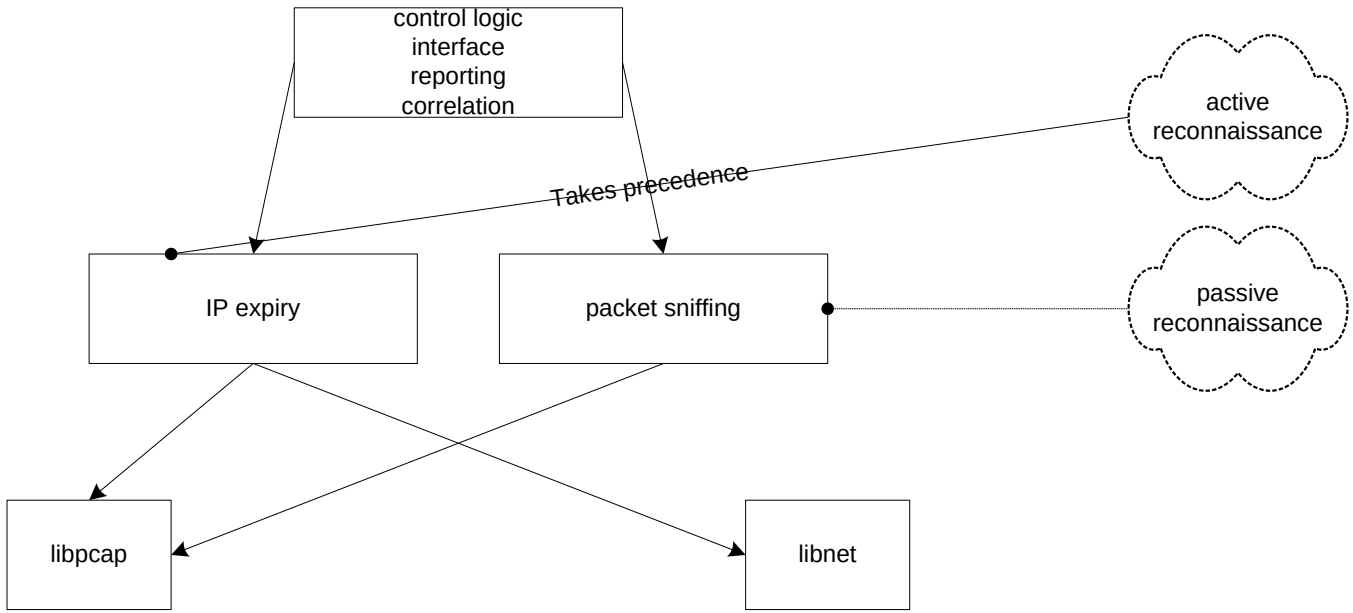
denotes layer dependency

denotes class binding

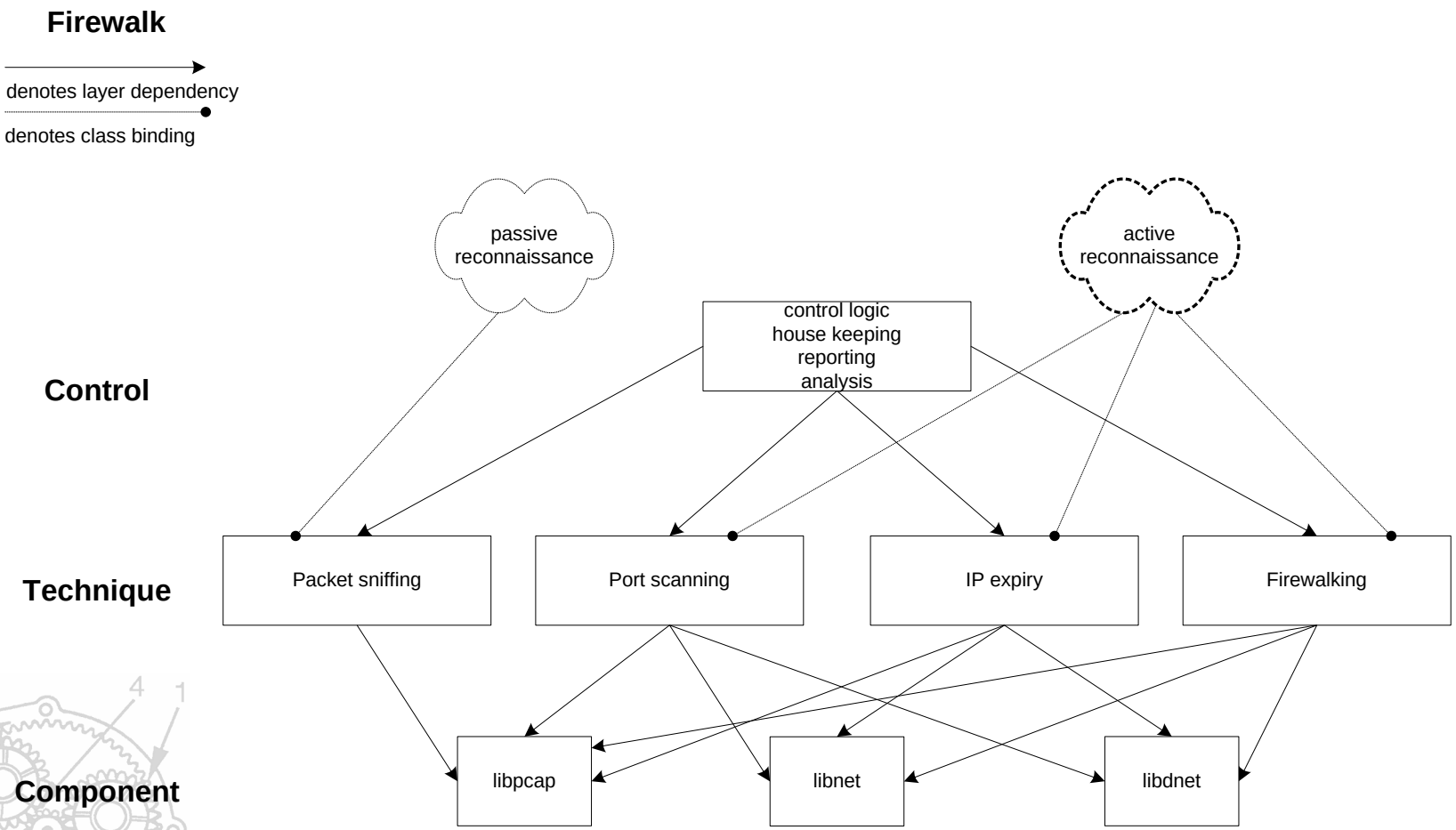
Control

Technique

Component

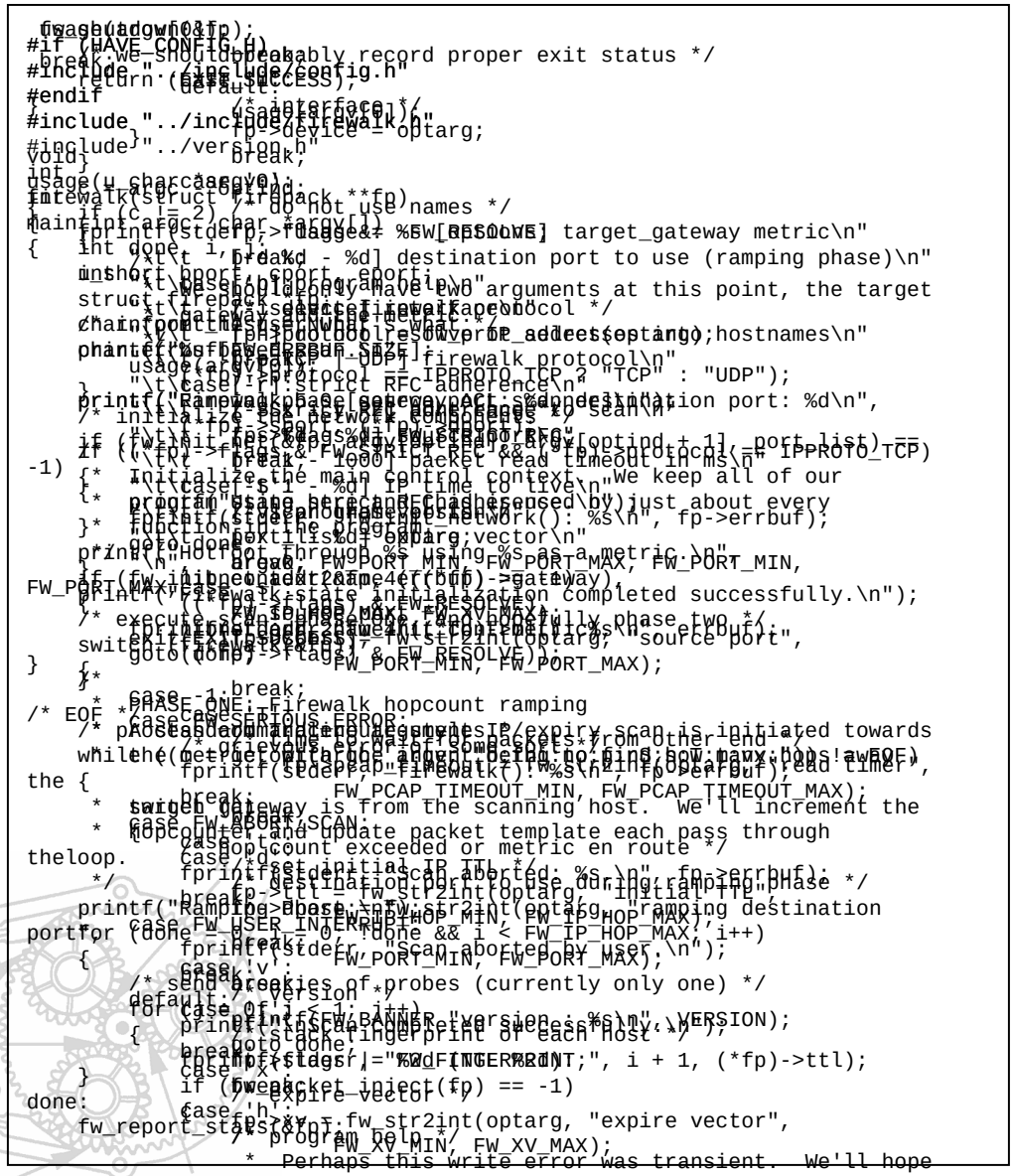


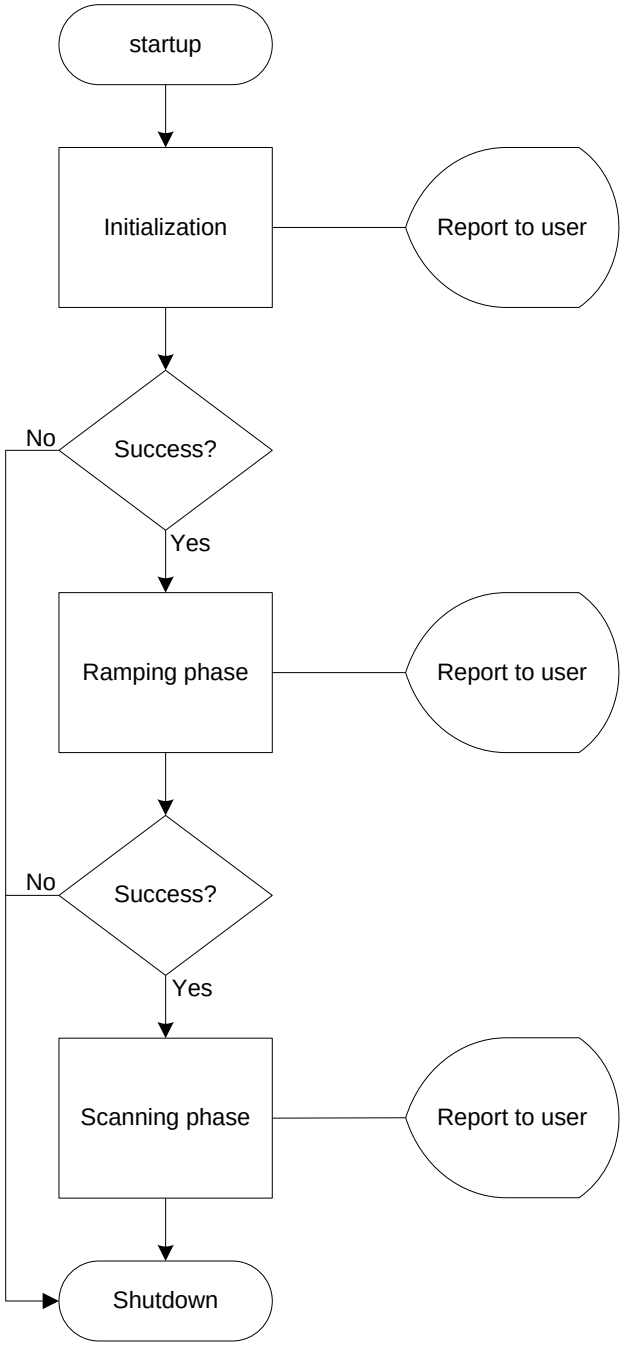
The Firewalk Tool Modeled



Inside a Network Security Tool: Firewalk Internals

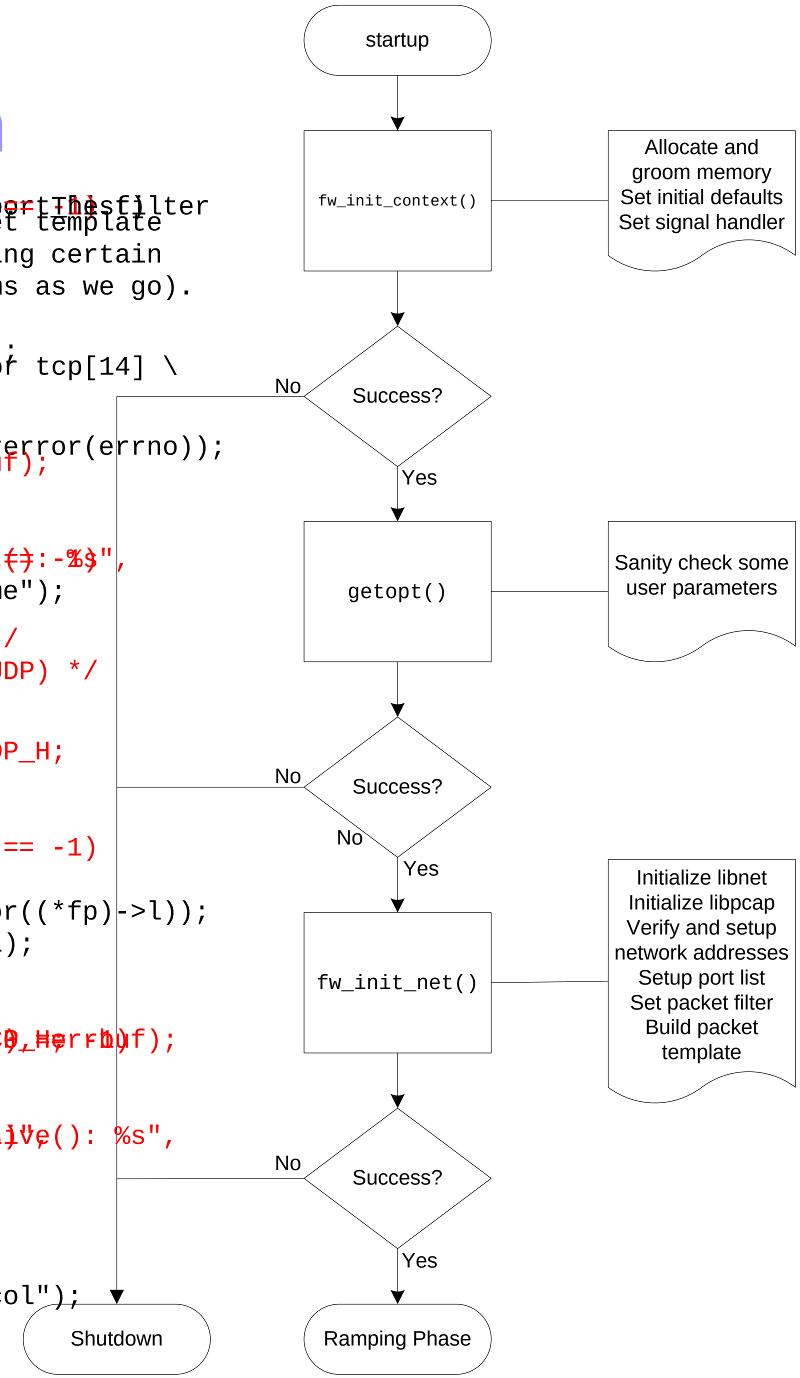
Firewalk Overall Flow





Firewalk Initialization

```
        default:
            sprintf((*fp)->errbuf,
                    "fw_init_network(): unsupported protocol");
            return (-1);
    }
```

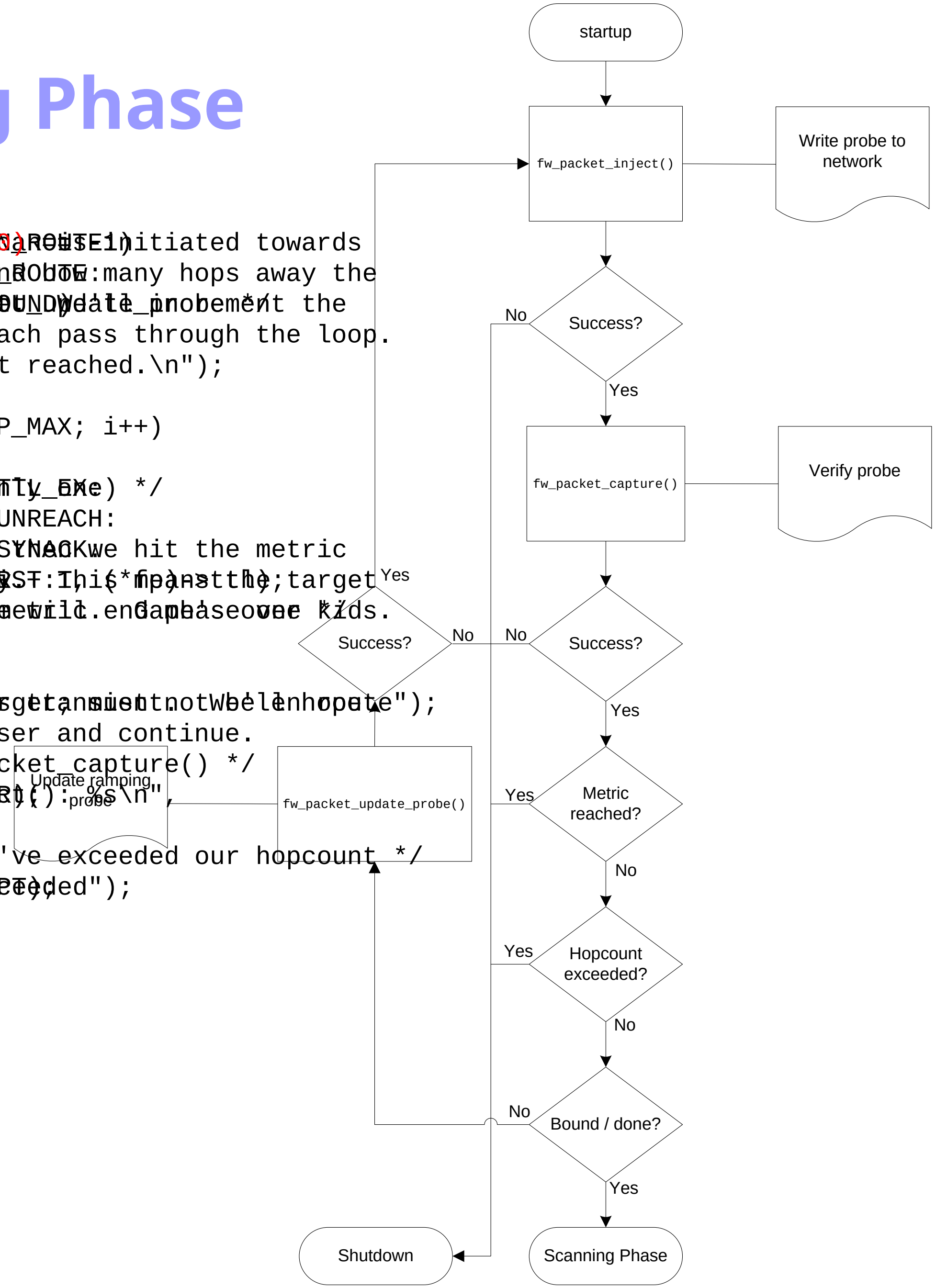


Firewalk Packet Construction

```
int
fw_packet_build_probe(struct firepack **fp)
{
    arp_t *a;
    route_t *r;
    struct arp_entry arp;
    struct route_entry route;

    /* first build our transport layer header */
    switch ((*fp)->protocol)
    {
        case IPPROTO_UDP:
            if (fw_packet_build_udp(fp) == -1)
            {
                /* error msg set in fw_packet_build_udp() */
                return (-1);
            }
            break;
        case IPPROTO_TCP:
            if (fw_packet_build_tcp(fp) == -1)
            {
                /* error msg set in fw_packet_build_tcp() */
                return (-1);
            }
            break;
        default:
            sprintf((*fp)->errbuf,
                "fw_packet_build_probe(): unknown protocol");
            return (-1);
    }

    /* build our IPv4 header */
    (*fp)->ip = libnet_build_ipv4(
        (*fp)->packet_size,                 /* packetlength */
        0,                                  /* IP tos */
        (*fp)->id,                          /* IP id */
        0,                                  /* IP frag bits */
        (*fp)->ttl,                         /* IP time to live */
        (*fp)->protocol,                    /* transport protocol */
        0,                                  /* checksum */
        (*fp)->sin.sin_addr.s_addr,         /* IP source */
        (*fp)->metric,                      /* IP destination */
        NULL,                               /* IP payload */
        0,                                  /* IP payload size */
        (*fp)->l,                           /* libnet context */
        0);                                 /* No saved ptag */

    if ((*fp)->ip == -1)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "libnet_build_ipv4() %s",
            libnet_geterror((*fp)->l));
        return (-1);
    }

    /*
     * Now we need to get the MAC address of our first hop gateway.
     * Dnet to the rescue! We start by doing a route table lookup
     * to determine the IP address we use to get to the
     * destination host (the metric).
     */
    r = route_open();
    if (r == NULL)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "route_open()");
        route_close(r);
        return (-1);
    }

    /* convert the metric address to dnet's native addr_t format */
    if (addr_aton(libnet_addr2name4((*fp)->metric, 0),
        &route.route_dst) < 0)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "addr_aton()");
        route_close(r);
        return (-1);
    }
    /* get the route entry telling us how to reach the metric */
    if (route_get(r, &route) < 0)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "route_get()");
        route_close(r);
        return (-1);
    }
    route_close(r);

    a = arp_open();
    if (a == NULL)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "arp_open()");
        return (-1);
    }
    /* get the MAC of the first hop gateway */
    arp.arp_pa = route.route_gw;
    if (arp_get(a, &arp) < 0)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "route_get()");
        arp_close(a);
        return (-1);
    }
    arp_close(a);

    /* build our ethernet header */
    if (libnet_autobuild_ethernet(
        (u_char *)&arp.arp_ha.addr_eth,
        ETHERTYPE_IP,
        (*fp)->l) == -1)
    {
        snprintf((*fp)->errbuf, FW_ERRBUF_SIZE,
            "libnet_autobuild_ethernet() %s",
            libnet_geterror((*fp)->l));
        arp_close(a);
        return (-1);
    }

    return (1);
}

/* build a TCP header */
(*fp)->tcp = libnet_build_tcp(
    (*fp)->sport,                           /* source TCP port */
    (*fp)->dport,                           /* dest TCP port */
    (*fp)->seq,                             /* sequence number */
    0L,                                     /* ACK number */
    TH_SYN,                                 /* control flags */
    1024,                                   /* window size */
    0,                                      /* checksum */
    0,                                      /* urgent */
    (*fp)->packet_size - LIBNET_IPV4_H,     /* TCP size */
    NULL,                                   /* IP payload */
    0,                                      /* IP payload size */
    (*fp)->l,                               /* libnet context */
    0);                                     /* No saved ptag */

if ((*fp)->tcp == -1)
{
    snprintf((*fp)->errbuf, FW_ERRBUF_SIZE, "libnet_build_tcp() %s",
        libnet_geterror((*fp)->l));
    return (-1);
}
return (1);
```

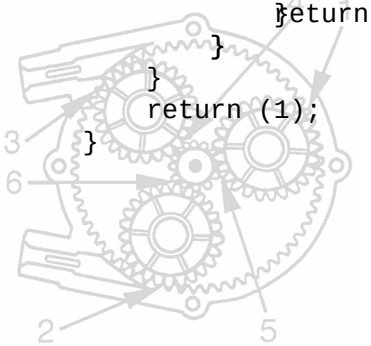
Firewalk Ramping Phase

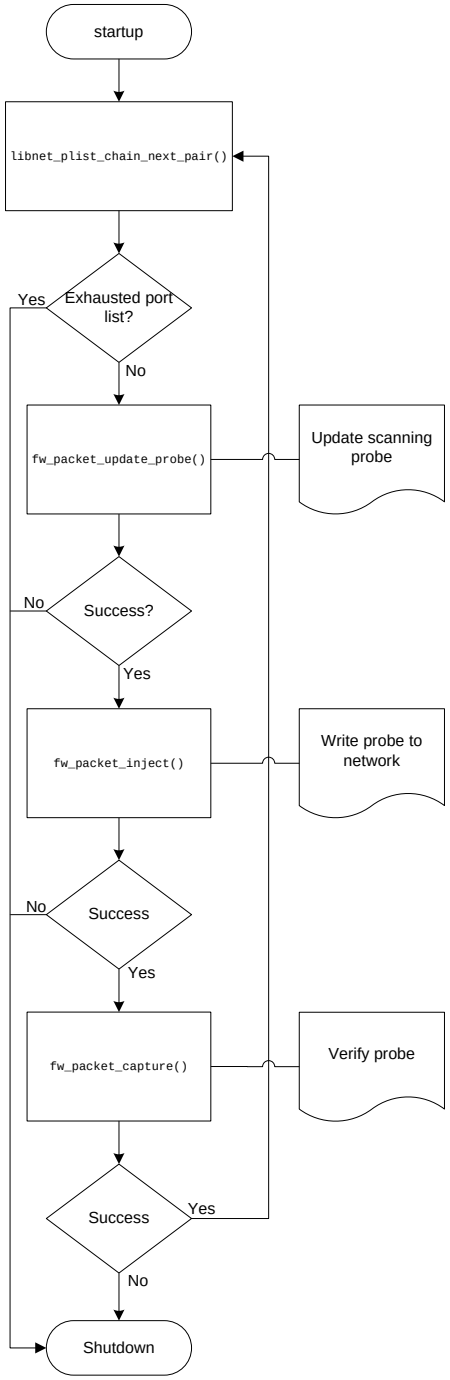
```
/*
 *  PHASE ONE:  Firewalk hopcount ramping
 *  A standard Traceroute-style IP expiry scan is initiated towards
 *  the metric, with the intent being to find how many hops away the
 *  target gateway is from the scanning host.  We'll increment the
 *  hopcounter and update packet template each pass through the loop.
 */
printf("Ramping Phase:\n");
for (done = 0, i = 0; !done && i < FW_IP_HOP_MAX; i++)
{
    /* send a series of probes (currently only one) */
    for (j = 0; j < 1; j++)
    {
        fprintf(stderr, "%2d (TTL %2d): ", i + 1, (*fp)->ttl);
        if (fw_packet_inject(fp))
        {
            /*
             *  Perhaps this write error was transient.  We'll hope
             *  for the best.  Inform the user and continue.
             */
            fprintf(stderr, "fw_packet_inject(): %s\n",
                    (*fp)->errbuf);
            continue;
        }
        switch (fw_packet_capture(fp))
        {
            case FW_PACKET_IS_UNREACH_EN_ROUTE:
            case FW_PACKET_IS_TTL_EX_EN_ROUTE:
                if ((*fp)->flags & FW_BOUND)
                {
                    printf("Binding host reached.\n");
                    done = 1;
                }
                break;
            case FW_PACKET_IS_TERMINAL_TTL_EX:
            case FW_PACKET_IS_TERMINAL_UNREACH:
            case FW_PACKET_IS_TERMINAL_SYNACK:
            case FW_PACKET_IS_TERMINAL_RST:
                /* any terminal response will end phase one */
                done = 1;
                break;
            case -1:
            case FW_SERIOUS_ERROR:
                /* err msg set in fw_packet_capture() */
                return (FW_SERIOUS_ERROR);
            case FW_USER_INTERRUPT:
                /* user hit ctrl-c */
                return (FW_USER_INTERRUPT);
        }
    }
    if (!done)
    {
        if (fw_packet_update_probe(fp, 0) == -1)
        {
            /* error msg set in fw_packet_update_probe */
            return (-1);
        }
    }
}
if (done && !((*fp)->flags & FW_BOUND))
{
    /*
     *  If we're "done" but not "bound" then we hit the metric
     *  before we hit the target gateway.  This means the target
     *  gateway is not en route to the metric.  Game's over kids.
     */
    sprintf((*fp)->errbuf,
            "metric responded before target; must not be en route");
    return (FW_ABORT_SCAN);
}
if (!done)
{
    /* if we fall through down here, we've exceeded our hopcount */
    sprintf((*fp)->errbuf, "hopcount exceeded");
    return (FW_ABORT_SCAN);
}
```



Firewalk Scanning Phase

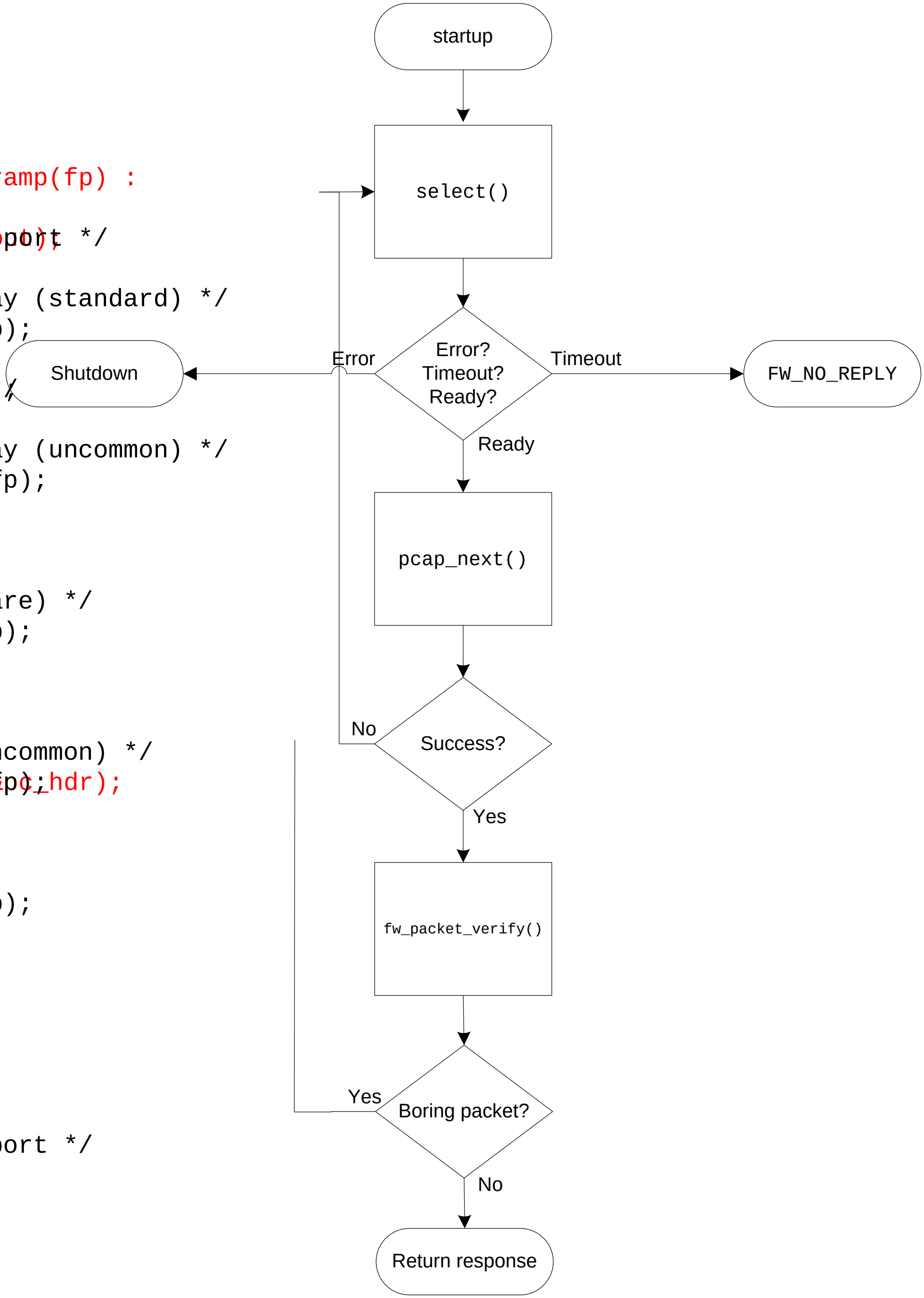
```
            /* send a series of probes (currently only one) */
/*          for (j = 0; j < 1; j++)
 *  PHASE TWO: Firewalk scanning
 *  A series of probes are sent from the metric with the bound IP
 *  TTL. If a given probe is accepted through the target gateway's
 *  ACL, we will receive an ICMP TTL exceeded in transit from the
 *  binding host. If we receive no response after the timeout expires,
 *  it is assumed the probe violated the ACL on the target and was
 *  dropped.
 */
(*fp)->ttl += (*fp)->xx;
printf("Scan bound at %d hops.\n", (*fp)->ttl);
printf("Scanning Phase: \n");
for (done = 0, i = 0; !done; i++)
{
    if (!libnet_plist_chain_next_pair((*fp)->plist, &bport, &cport))
    {
        /* we've exhausted our portlist and we're done */
        done = 1;
        continue;
    }
    while (!(bport > cport))
    {
        cport = bport++;
        if (fw_packet_update_probe(fp, cport) == -1)
        {
            /* error msg set in fw_packet_update_probe */
            return (-1);
        }
    }
    return (1);
}
```



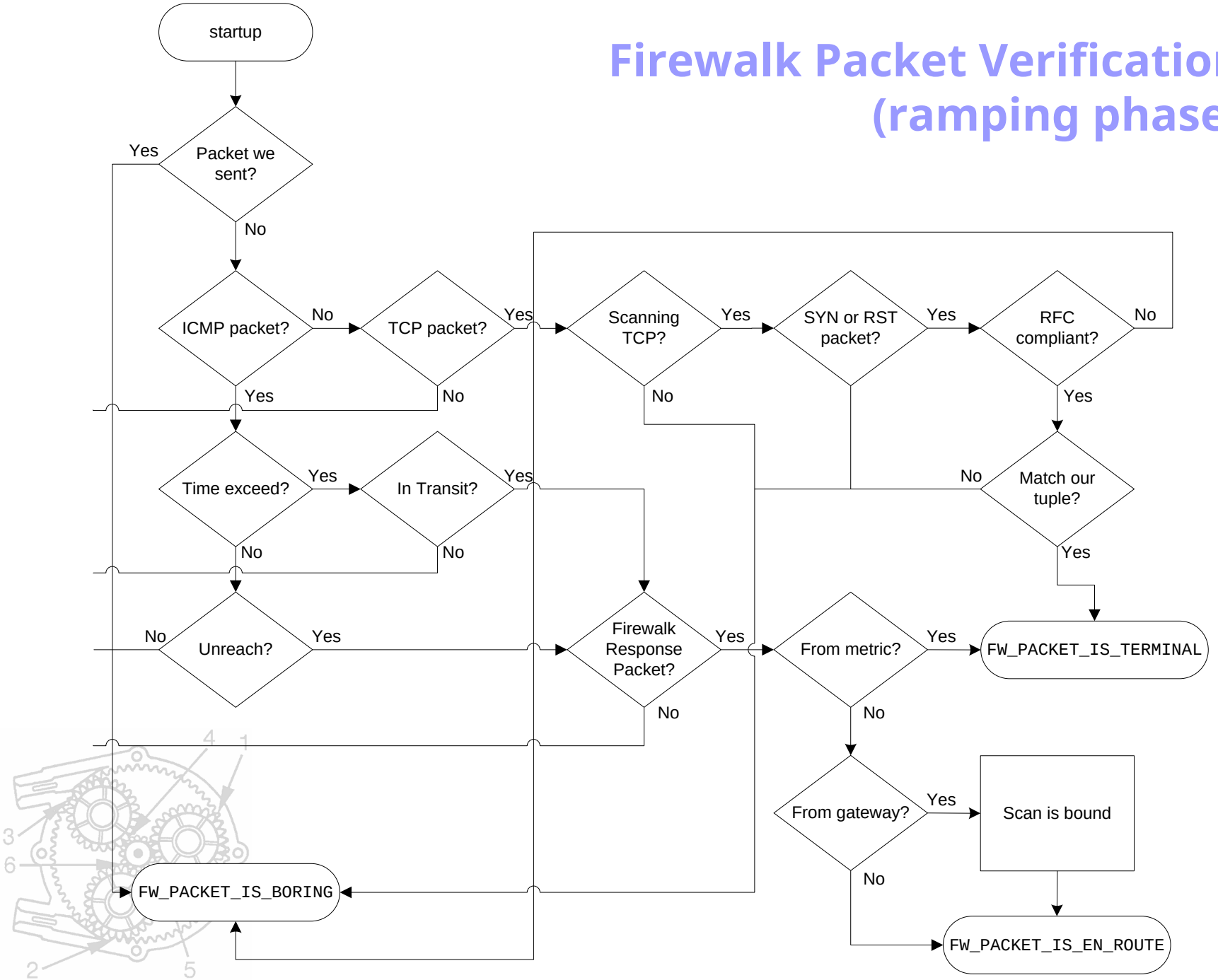


Firewalk Capture

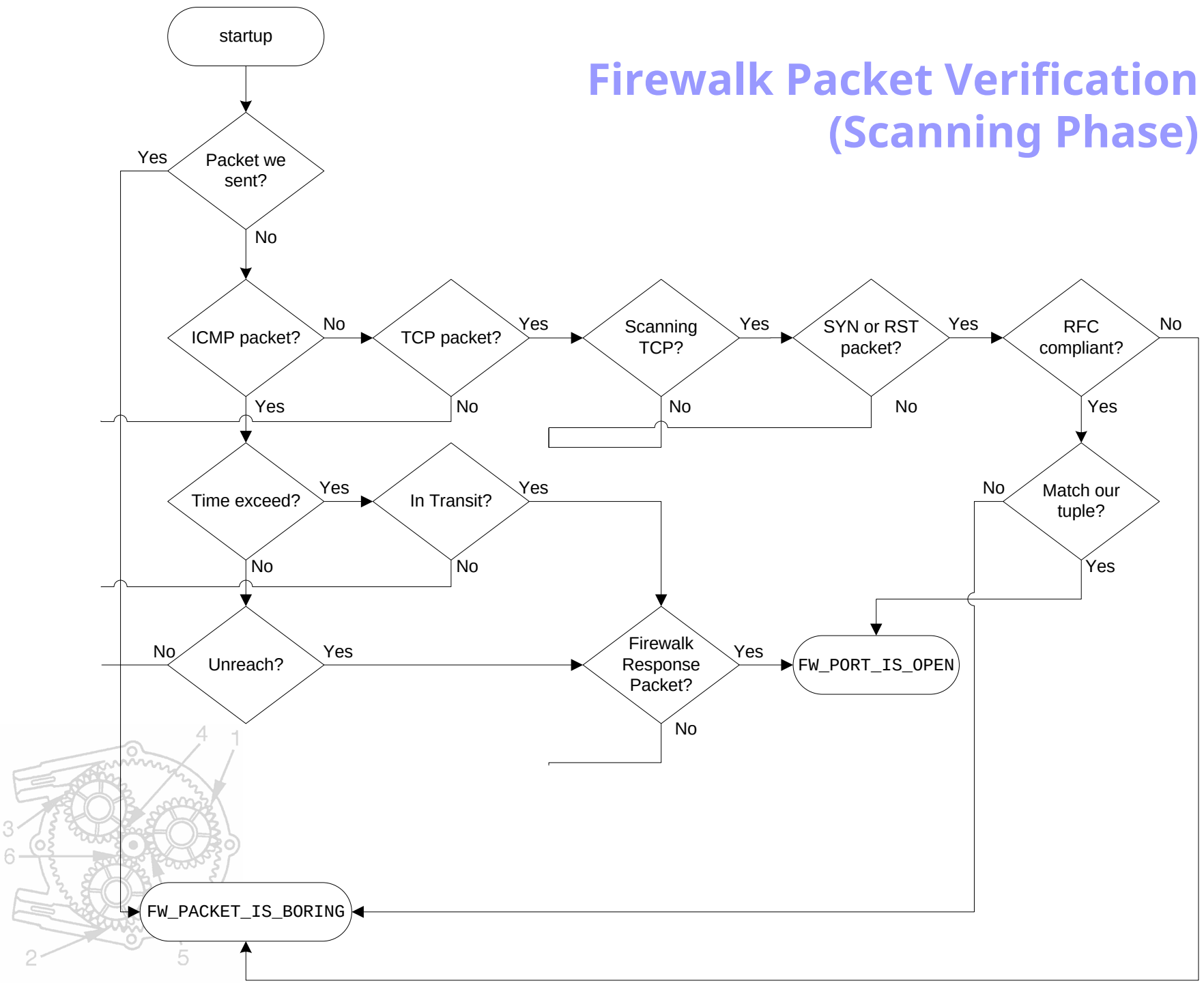
```
switch (((*fp)->flags) & FW_BOUND ? fw_packet_verify_ramp(fp) :
        fw_packet_verify(fp))

for (timed_out = 0; !timed_out && loop; )
{
    c = select(pcap_fd + 1, &read_set, 0, 0, &timeout);
    switch (c)
    {
        case -1:
            snprintf((*fp)->errbuf, FW_ERRBUF_SIZE,
                    "select() %s", strerror(errno));
            return (-1);
        case 0:
            continue;
        default:
            if (FD_ISSET(pcap_fd, &read_set) == 0)
            {
                continue;
            }
            /* fall through to read the packet */
    }
    (*fp)->packet = (u_char *)pcap_next((*fp)->p, &pc_hdr);
    if ((*fp)->packet == NULL)
    {
        /* no packet present */
        continue;
    }
    (*fp)->stats.packets_caught++;
}

{
    case FW_PORT_IS_OPEN_RST:
        /* SCANNING: A response from a closed TCP port */
        fw_report(FW_PORT_IS_OPEN_RST, fp);
    case FW_PACKET_IS_TTL_EX_EN_ROUTE:
        /* RAMPING: TTL expired en route to gateway (standard) */
        fw_report(FW_PACKET_IS_TTL_EX_EN_ROUTE, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PACKET_IS_TTL_EX_EN_ROUTE);
    case FW_PACKET_IS_UNREACH_EN_ROUTE:
        /* RAMPING: Unreachable en route to gateway (uncommon) */
        fw_report(FW_PACKET_IS_UNREACH_EN_ROUTE, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PACKET_IS_UNREACH_EN_ROUTE);
    case FW_PACKET_IS_TERMINAL_TTL_EX:
        /* RAMPING: TTL expired at destination (rare) */
        fw_report(FW_PACKET_IS_TERMINAL_TTL_EX, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PACKET_IS_TERMINAL_TTL_EX);
    case FW_PACKET_IS_TERMINAL_UNREACH:
        /* RAMPING: Unreachable at destination (uncommon) */
        fw_report(FW_PACKET_IS_TERMINAL_UNREACH, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PACKET_IS_TERMINAL_UNREACH);
    case FW_PACKET_IS_TERMINAL_SYNACK:
        fw_report(FW_PACKET_IS_TERMINAL_SYNACK, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PACKET_IS_TERMINAL_SYNACK);
    case FW_PACKET_IS_TERMINAL_RST:
        fw_report(FW_PACKET_IS_TERMINAL_RST, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PACKET_IS_TERMINAL_RST);
    case FW_PORT_IS_OPEN_SYNACK:
        /* SCANNING: A response from an open TCP port */
        fw_report(FW_PORT_IS_OPEN_SYNACK, fp);
        (*fp)->stats.packets_caught_interesting++;
        return (FW_PORT_IS_OPEN_SYNACK);
    default:
        continue;
}
```



Firewalk Packet Verification (ramping phase)



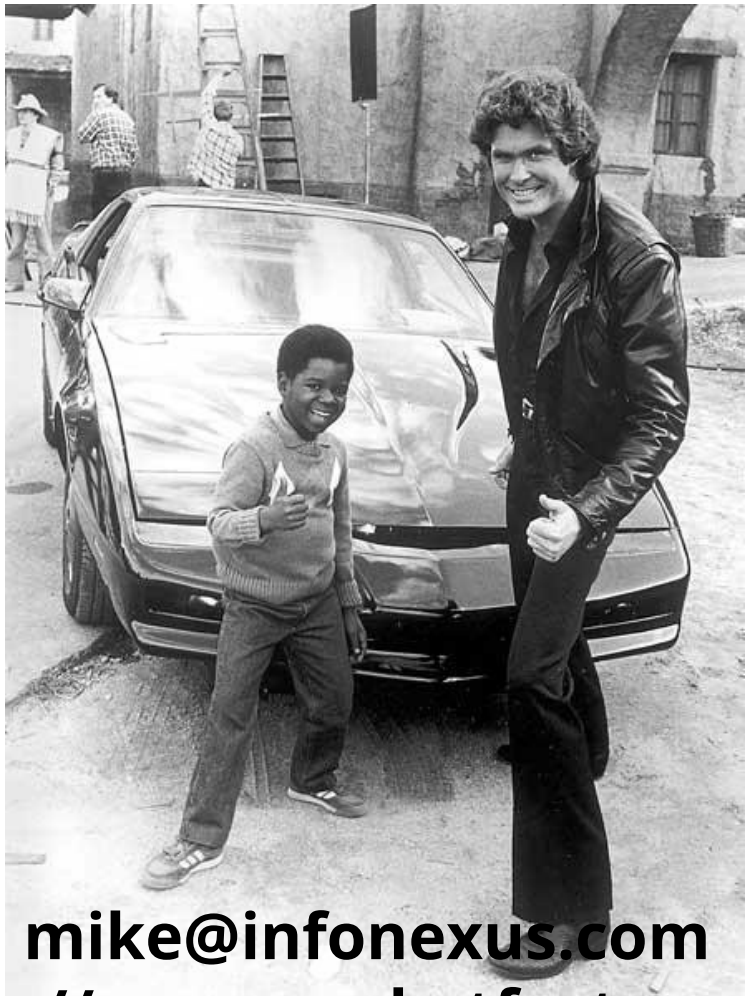
Firewalk Packet Verification (Scanning Phase)



Conclusion

- Modular Model of Network Security Tools
- Components and Techniques
  - This was not an exhaustive list of Components or Techniques…
- Examples of how to code Techniques using Components

Questions and Comments?



mike@infonexus.com
http://www.packetfactory.net