Tempesta xFW: open-source eBPF-based volumetric DDoS protection

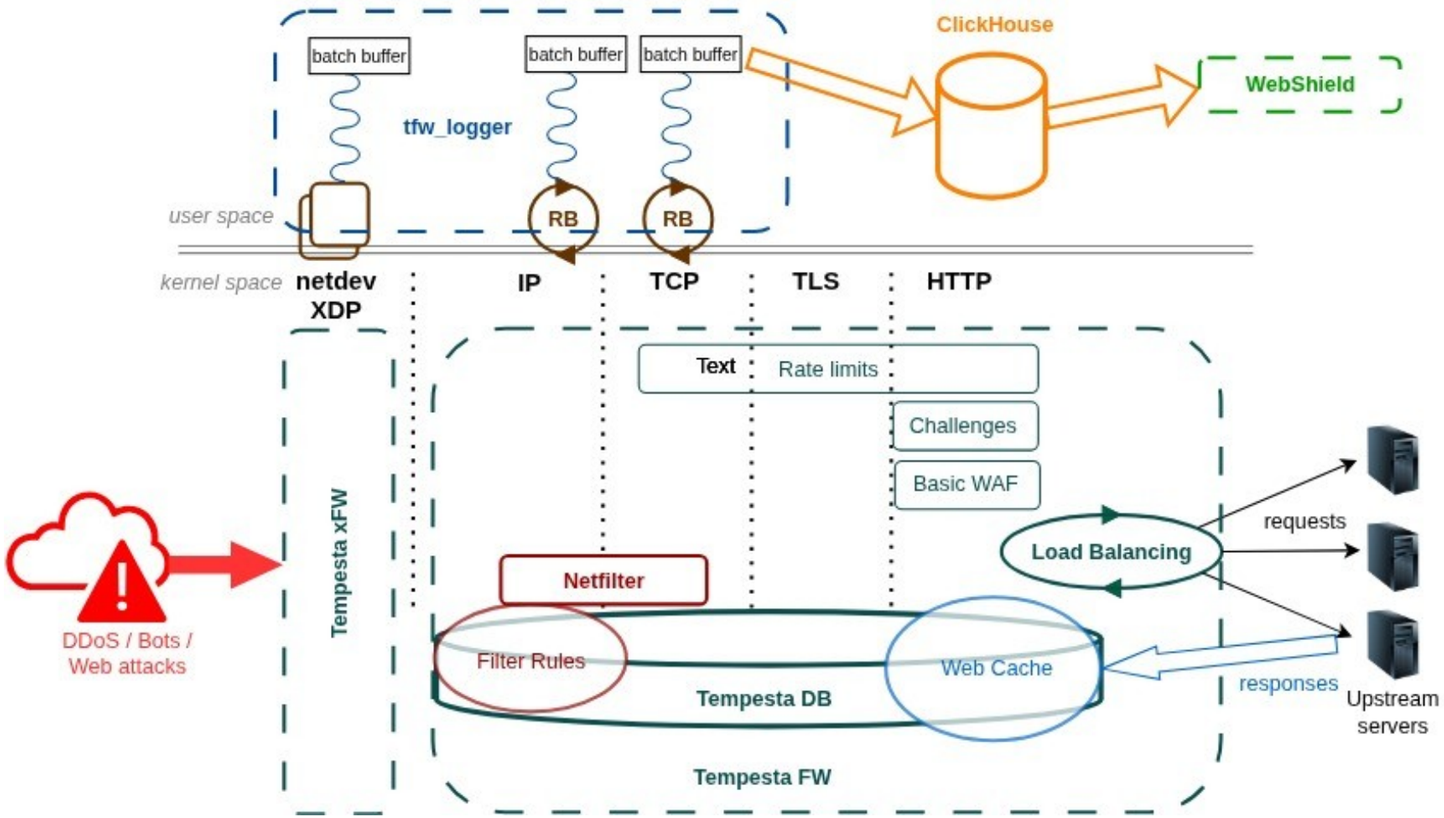
Alexander Krizhanovsky

Tempesta Technologies, Inc.

ak@tempesta-tech.com

What We Do

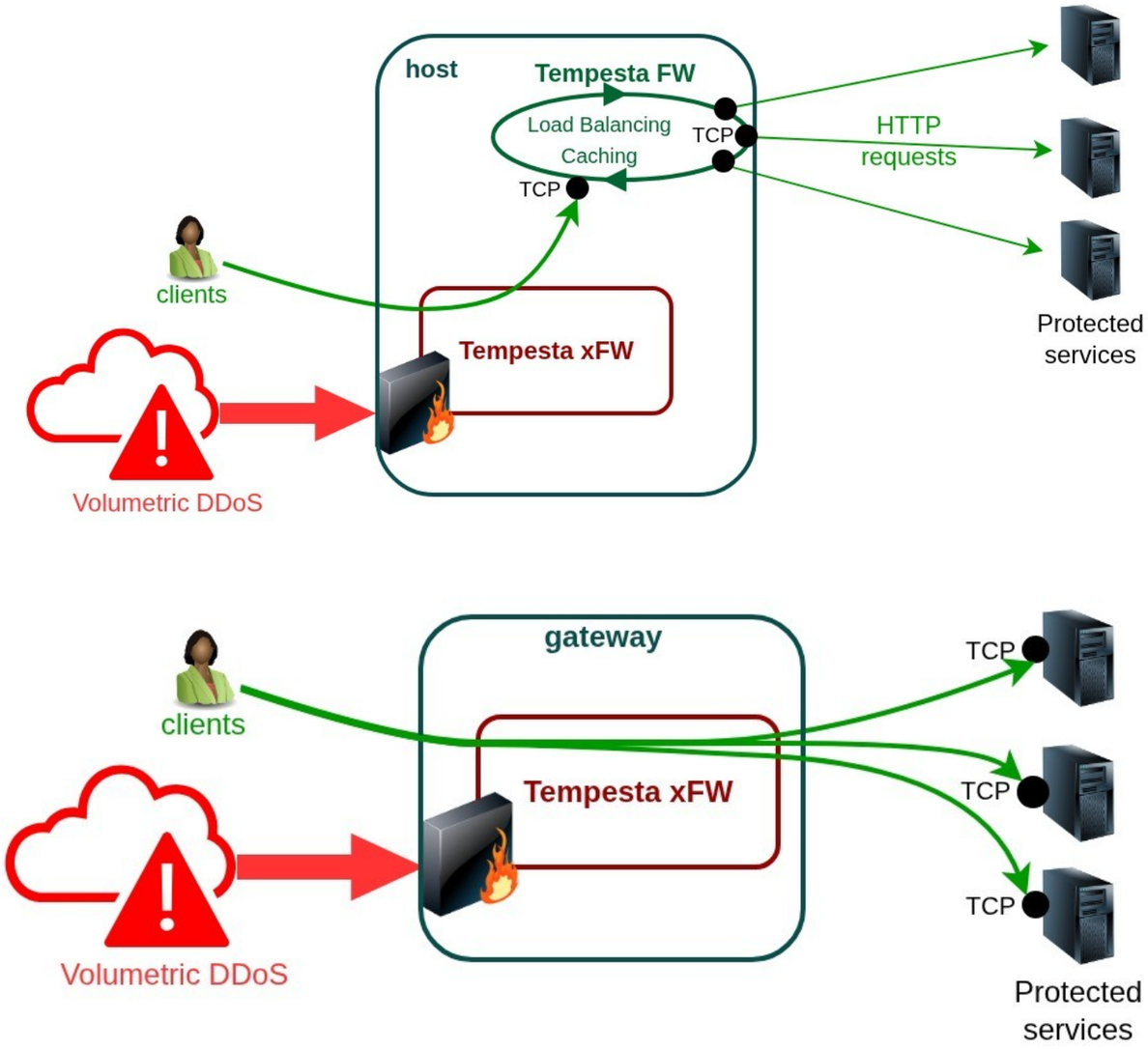
- Tempesta open source full stack DDoS and bot protection



- **xFW – volumetric DDoS protection**
  https://github.com/tempesta-tech/xFW
- FW – L7 DDoS protection & WAF
  https://github.com/tempesta-tech/tempesta
- WebShield – advanced bot management
  https://github.com/tempesta-tech/webshield

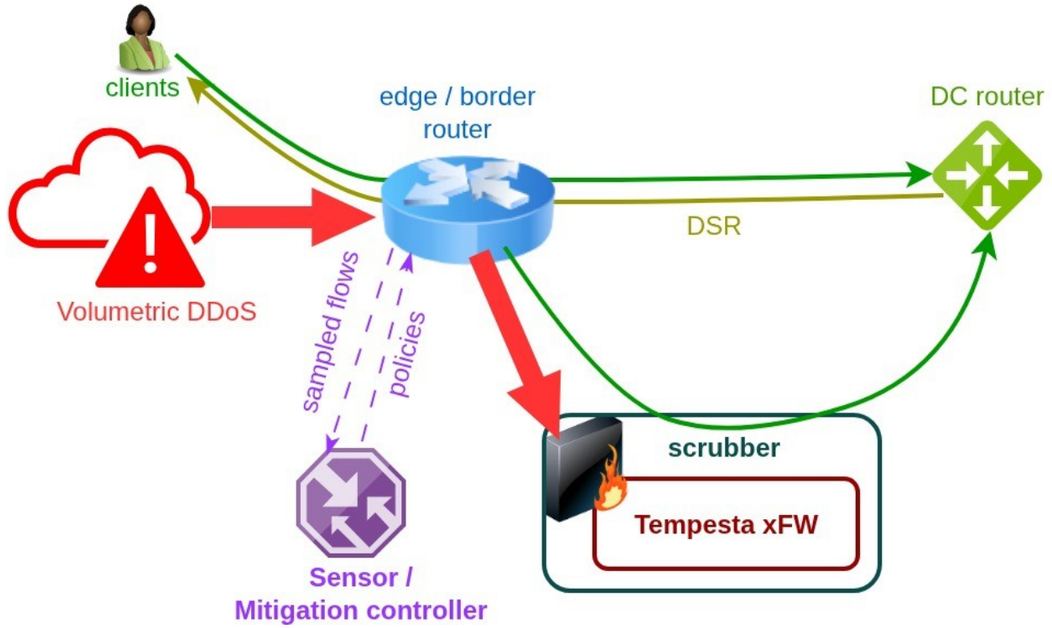
Always-on/Pass-through Deployment

- Host-based (e.g. a CDN edge)
  - TCP connection endpoint
  - integrated with L7 filtration
  - XDP: native integration with Linux networking & L7 processing
- Router-based (e.g. an ISP router)
  - standard Linux router



Redirection & Asymmetric Deployment

- The most common
  - Resource optimization (many edges)
  - Redirect only affected prefixes
- Drawbacks
  - Low-volume traffic can be sampled out
  - Hit-and-run attacks
- Hybrid
  - Simple low-false-positive rules always-on
  - Redirection for more sophisticated rules
- Asymmetric traffic (downstream-only)



Layered Protection Logic

- ICMPv4 & ICMPv6 type black/white lists & rate limits
- TCP & UDP source address & port black/white lists & rate limits
  - e.g. DNS reflection attack
  - GeoIP database
- TCP & UDP anomaly filters
- TCP flow authentication *(not RFC 5925)*
- TCP SYN rate limit and SYN cookies
- DNS filter
- Last resort: destination filter for `<dst_ip, dst_port, proto>`

XDP and TC programs

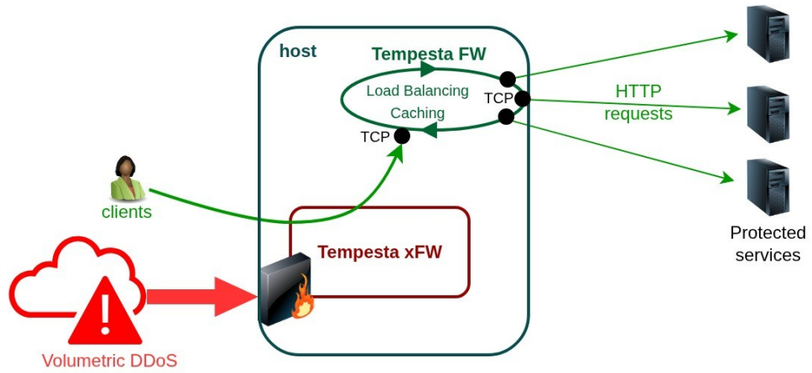
- XDP (`skb` or `native`) – the most of the logic
- TC: `CLSACT BPF direct-action`
  - TCX hook in future *(https://github.com/tempesta-tech/xFW/issues/10)*
  - Forwarded or locally-generated traffic
  - Ingress (downstream) and egress (upstream) interfaces
  - Tracks egress/forwarded **DNS** queries to pass responses
  - **Destination** filter (last resort to not to overflow an endpoint)
  - **TCP flows authentication** learning

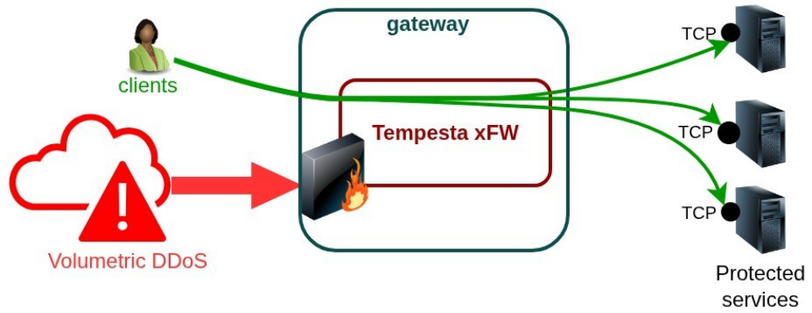
TCP Flows Authentication (*not* RFC 5925)

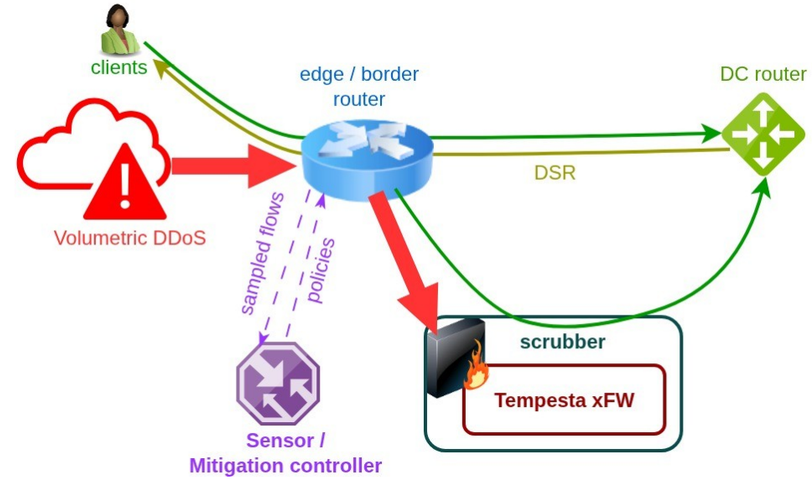
- Blocks RST and ACK floods for unseen TCP flows
- Always tracks connections to start anytime
- Tracks
  - any segments in inactive state
  - forwarded/locally-generated `SYN` & `SYN-ACK`
- Clear state on
  - egress RST
  - after 1 minute after egress FIN

SYN Flood Protection on Host, Gate, Scrubber

- SYN Cookies
  "Issuing SYN Cookies in XDP" by P.Penkov et al
  https://netdevconf.info/0x14/pub/papers/50/0x14-paper50-talk-paper.pdf
- SYN Proxy – not yet
  linux/tools/testing/selftests/bpf/progs/xdp_synproxy_kern.c
- `tcp_flags syn : ratelimit=syn_rl;`
  - also for RST
- SYN flood is often spoofed
  - drop first SYN and add to an LRU map
  - define states and evict all single-packets







Rate Limiting

- Problem from an L7
  - A browser generates ~300 requests for a WordPress website index
  - So should we allow 300 req/sec for a website with ~300 views/day?

```
request_rate 1000 20;
request_burst 300;
```

- L3-L4 operates with PP**S** and BP**S**
  - hit-and-run attacks are subminute
  - IoT, last mile and geo-far devices have large Internet RTT
  - spoofing: in many cases rate limiting has no sense

Token & Leaky Buckets

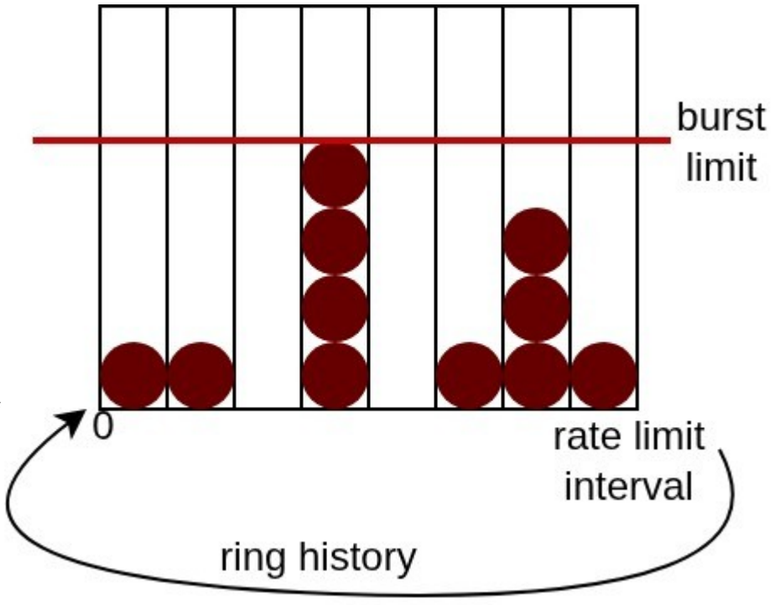
- Token buckets (counting leaky buckets invert subraction and addition)

```
elapsed = jiffies - rl->refill_ts;
rl->tok = min(rl->tok_max,
              rl->tok + elapsed * rl->tok_rate); // refill
rl->refill_ts = jiffies;

if (pkt_sz > rl->tok)     // check
    return XDP_DROP;

rl->tok -= pkt_sz;        // drain
```

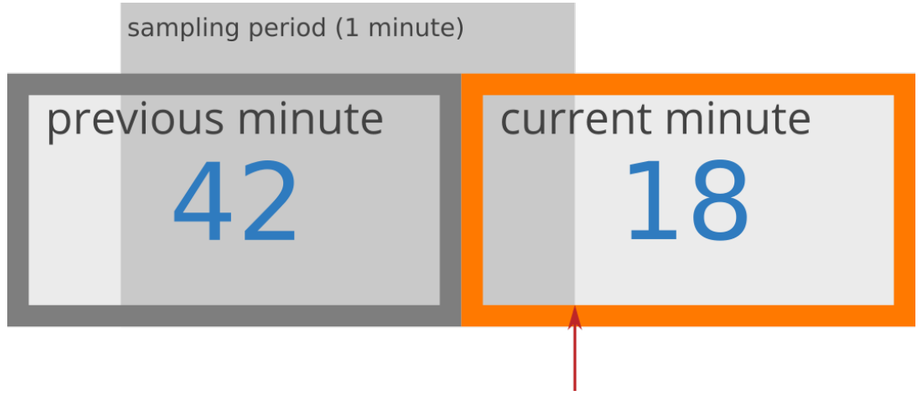
- Very efficient
- Burst is limited by the available space in the bucket

Sliding Windows

- Fixed intervals history
  - accuracy depends on the intervals number
  - loop over all intervals
- Approximate 2 intervals
  https://blog.cloudflare.com/counting-things-a-lot-of-different-things/




```
offset = rl->ts_begin & win_mask;
overlap = win_sz - offset;
bytes = curr_bytes + (prev_bytes * overlap
                      >> win_shift);
if (bytes > rl->bytes)
    return XDP_DROP;
```



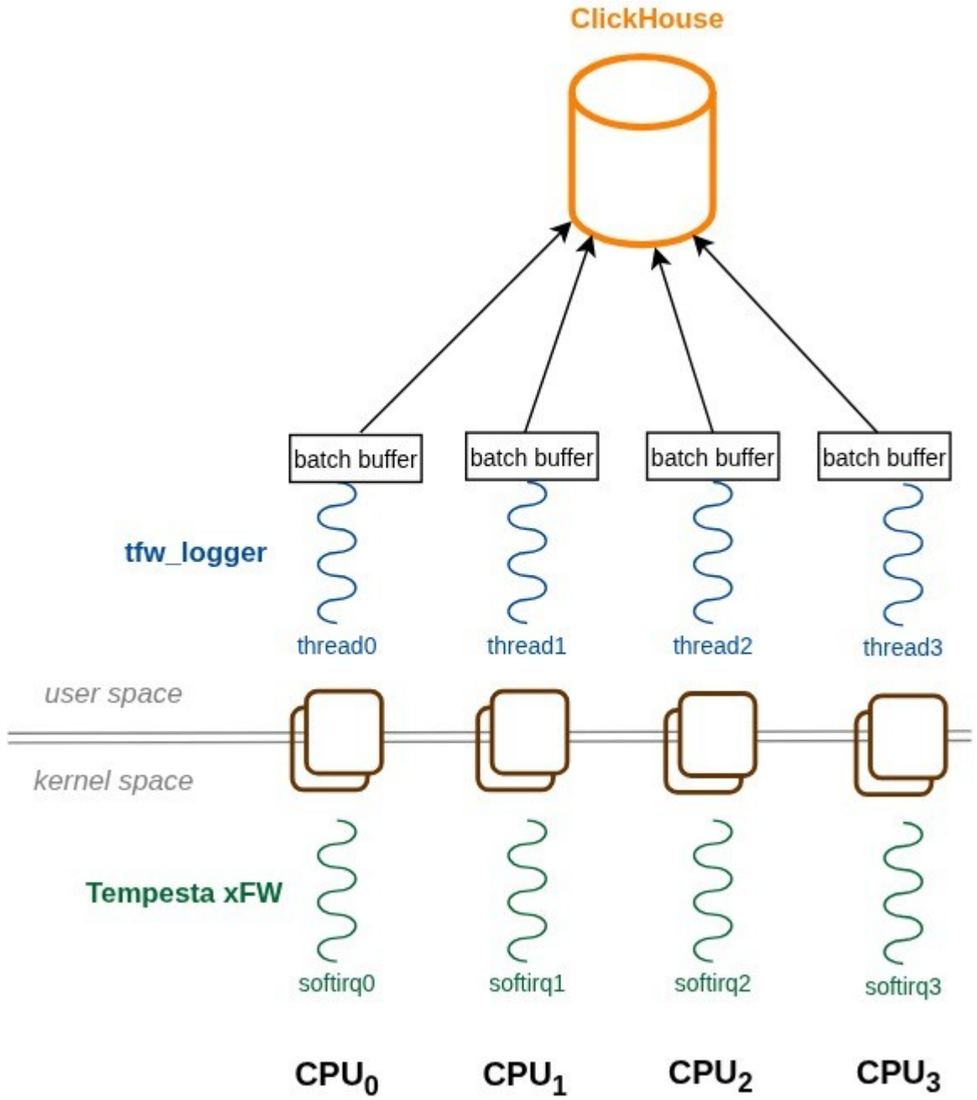
  - very efficient
  - assumes a uniform distribution

DNS Protection

- Parse requests & responses to filter out random UDP flood
  - queries: `rcode != RCODE_OK`, multi-queries, malformed names, nonempty answer section
  - responses: >100 answers, malformed RRs, TTL outside [1, 604800]
- Allow responses only for generated/forwarded requests (DNS reflection)
- Future work
  - cookies verification
  - `NXDOMAIN` acceleration: learn successful queries
  - response cache

`tfw_logger`: Monitoring & Incident Logging

- Statistics: `BPF_MAP_TYPE_PERCPU_ARRAY`
- Incident logging
  https://tempesta-tech.com/knowledge-base/Access-Log-Analytics/
  - async: may drop events with drop count
  - per-CPU threads
  - optimized batches for ClickHouse



Incident Logging Maps

- Aggregate events by IP address in `BPF_MAP_TYPE_HASH` (not `BPF_MAP_TYPE_PERF_EVENT_ARRAY`)
- Two hot swap `BPF_MAP_TYPE_HASH` maps for each CPU (swap by timeout or 80% memory usage)
- `BPF_MAP_TYPE_ARRAY_OF_MAPS` for per-cpu *active* maps
  - an inactive map is processed by `tfw_logger`
- Not `BPF_MAP_TYPE_PERCPU_HASH` because of shared key space
  - single bucket contention
  - NIC hashing: an IP can be observed only on subset of CPUs
  - shared key-space exhaustion affects all CPUs simultaneously

Evaluation Mode

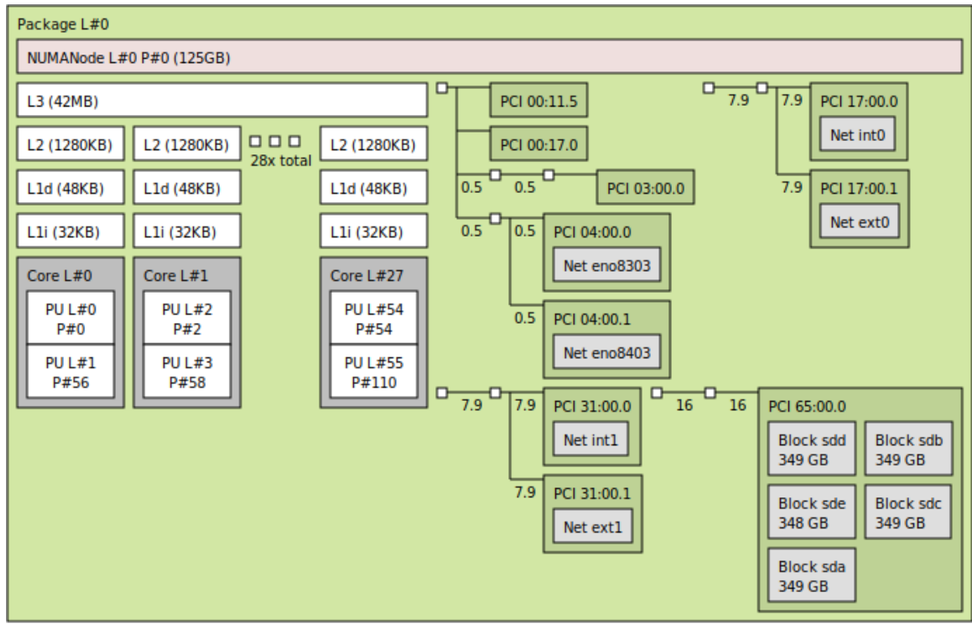
- Dry-run to evaluate filtration rules and performance before enforcement
  - collects all statistics and report incidents
  - `XDP_PASS` / `TC_ACT_OK` instead of `XDP_DROP` / `TC_ACT_SHOT`
- Not applicable to active-response filtering: `tcp_syncookies`

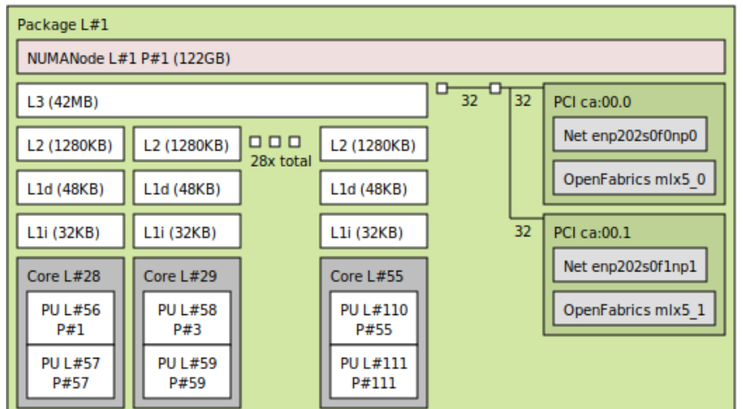
Performance Setup

https://tempesta-tech.com/tempesta-escudo/knowledge-base/Performance/

https://github.com/tempesta-tech/xFW/blob/main/doc/perf.md

- SUT: 2x Xeon Gold 6348, 256GB DDR4-3200
  - **only CPU package 0 is used**
- Gen: 2x Xeon Gold 6348, 64GB DDR4-3200
  - T-Rex, 28 cores, both NIC ports
- Both: ConnectX-6 Dx, Dual port 100G
- Linux 6.8 Ubuntu 24 LTS





Benchmarks

- ICMPv6 flood: **129**Gbps, **196**Mpps
  - 150K addresses/prefixes in the white list
- TCP/UDP flood: **176**Gbps, **59**Mpps
  - UDP 1514B; TCP 54B ACK/FIN/NULL/RST/SYN/SYN-ACK/URG/XMAS
- TCP SYN rate-limit: **104**Gbps, **167**Mpps
- Latency under 10Mpps SYN flood: **+17% avg** and **+39% max**
- `tcp-syncookies=1,{passive,flood}_timer=1`): **93**Gbps, **149**Mpps
  - real is ~**x3** lower (`Total-Rx: 32.12`Gbps)

```
for d in enp202s0f{0np0,1np1};do ethtool -S $d |grep rx_out_of_buffer;done
     rx_out_of_buffer: 3158555724
     rx_out_of_buffer: 3135228012
```

SYN Cookies: Socket Lookup

```
// On SYN
struct bpf_sock *sk = bpf_sk_lookup_tcp(…);
bpf_tcp_gen_syncookie(sk, …);

// On ACK
struct bpf_sock *sk = bpf_sk_lookup_tcp(…);
if (sk->state != BPF_TCP_LISTEN)
    return XDP_PASS;

bpf_sk_lookup_tcp() → __inet_lookup()
    __inet_lookup_established(); // busy server overhead
    __inet_lookup_listener();
```

- Profile for SYN flood with almost no established connections:

```
3.17%  __inet_lookup_established
1.12%  inet_lhash2_lookup
1.05%  __inet_lookup_listener
1.03%  sk_lookup
0.94%  inet_ehashfn
0.61%  __bpf_skc_lookup
0.55%  __bpf_sk_lookup
0.49%  bpf_xdp_sk_lookup_tcp
```

`bpf_tcp_raw_gen_syncookie_ipv{4,6}()`

- `bpf_tcp_gen_syncookie()` does a lot of checks
- `bpf_tcp_raw_gen_syncookie_ipv{4,6}()`
  - no extra checks
  - no socket lookup: generate cookies even for non-existing sockets
  - call `bpf_tcp_gen_syncookie()` periodically to keep/exit flood mode

```
23.32%   bpf_prog_097cfe38a3a3145b_xfw_xdp
10.88%   lookup_nulls_elem_raw
 4.86%   bpf_prog_755f77ce8ff2c139_parse_tcpopts
 4.61%   siphash_2u64
 . . .
 0.82%   bpf_tcp_gen_syncookie
```

`bpf_tcp_check_syncookie()`: Ambiguous API

```
bpf_tcp_check_syncookie(struct sock *sk, void *iph, u32 iph_len,…) {
    if (unlikely(!sk || th_len < sizeof(*th)))
        return -EINVAL;
    if (!READ_ONCE(sock_net(sk)->ipv4.sysctl_tcp_syncookies))
        return -EINVAL;
    if (!th->ack || th->rst || th->syn)
        return -ENOENT;
    if (tcp_synq_no_recent_overflow(sk))
        return -ENOENT;
```

- Rejected patch proposal:
  https://groups.google.com/g/clang-built-linux/c/Ehb-mZnip-E/m/nRWGW24PBAAJ
- Use `bpf_tcp_raw_check_syncookie_ipv{4,6}()`
  - no `tcp_synq_no_recent_overflow()` - flood mode performance

TCP Options Parsing: eBPF Verifier

- GPT-5.6 solves simple problems, but stucks on TCP options parser (only `NOP`, `EOL`, `WSCALE`, `SACK_PERM`, `TIMESTAMP`, generic)

The sequence of 8193 jumps is too complex.

```
static __noinline int tcp_parse_opts(XfwGlobalCtx *ctx, XfwTCPOpts *opts) {
    const uint8_t *e, *const o = ctx->th + sizeof(struct tcphdr);
    const uint8_t *const o_end = ctx->th + ctx->th->doff * 4;
    uint8_t i, off, *const d_end = ctx->hdr_cur.end;
    TRUE_OR_RETURN(o + 40 >= o_end, -E2BIG);
    bpf_for (i, 0, 40) {
        if (o + off >= o_end || o + off >= d_end) return 0;
        switch (o[off]) {
        case TCPOPT_EOL: return 0;
        case TCPOPT_NOP: ++off; continue;
        case TCPOPT_WSCALE:
            e = o + off + TCPOPT_WSCALE_SZ;
            TRUE_OR_RETURN(e <= o_end && e <= d_end, -EINVAL);
            opts->wscale = o[off + 2];
            . . .
```

eBPF Verifier: What Doesn’t Work

- “Taking BPF programs beyond one-million instructions” *https://lwn.net/Articles/1017116/* – helps with general loop complexity
- Ask GPT-5.6-Sol to fix the problem, but it explains the verifier log well
- Blind tries to reduce complexity

The sequence of 8193 jumps is too complex.

```
bpf_for (i, 0, 3) {
    if (o + off >= o_end || unlikely(o + off >= d_end)) return 0;
    switch (o[off]) {
    case TCPOPT_EOL: return 0;
    case TCPOPT_NOP: ++off; continue;
    default:
        e = o + off + 2; TRUE_OR_RETURN(e <= o_end && e <= d_end, -EINVAL);
        uint8_t olen = o[off + 1]; TRUE_OR_RETURN(olen >= 2, -EINVAL);
        off += olen; continue;
    }
}
```

eBPF Verifier vs LLVM Optimizations

1. Strict verifier bounds requirements force repeated check
2. LLVM optimizes the checks out
3. Verifier rejects the code

```
uint8_t opt_len = o[1];
TRUE_OR_RETURN(opt_len >= 2 && opt_len <= len);
TRUE_OR_RETURN(o + opt_len <= data_end); // o + 1 is only proved!

switch (kind) {
case TCPOPT_WSCALE:
    /* RFC 7323 2.2: kind, opt_len=3, 1 byte val */
    TRUE_OR_RETURN(opt_len == TCPOPT_WSCALE_SZ);
    // this check is doubles the previous one and is optimized out
    TRUE_OR_RETURN(o + TCPOPT_WSCALE_SZ <= data_end);
    opts->wscale = o[2];
```

TCP Options Parsing: Ongoing Work

- *https://github.com/tempesta-tech/xFW/pull/33*
- `linux/tools/testing/selftests/bpf/progs/xdp_synproxy_kern.c`
  - allows only 14 options (2 batches) instead of 42 (6 batches)
  - probably enough for SYNs
- Vectorized processing
  - 10 batches by 4 bytes with a 5-byte buffer
  - precomputed table table of `kind -> length`
  - `TIMESTAMP` requires too complex code for 6 bytes parsing

Thanks!

https://github.com/tempesta-tech/xFW

https://tempesta-tech.com/tempesta-escudo/knowledge-base/XFW/

ak@tempesta-tech.com

https://www.linkedin.com/in/alexander-krizhanovsky/

https://x.com/a_krizhanovsky

Tempesta Technologies