Kernel HTTPS/TCP/IP stack for HTTP DDoS mitigation

Alexander Krizhanovsky
Tempesta Technologies, Inc.

ak@tempesta-tech.com
Who am I?

- CEO & CTO at **Tempesta Technologies** *(Seattle, WA)*
- Developing **Tempesta FW** – open source Linux Application Delivery Controller (ADC)
- **Custom software development** in:
  - *high performance network traffic processing*
    e.g. **WAF** mentioned in **Gartner magic quadrant**
  - **Databases**
    e.g. **MariaDB SQL System Versioning**
    *https://github.com/tempesta-tech/mariadb_10.2*
    *https://m17.mariadb.com/session/technical-preview-temporal-querying-asof*
HTTPS challenges

- HTTP(S) is a core protocol for the Internet (IoT, SaaS, Social networks etc.)

- HTTP(S) DDoS is tricky
  - Asymmetric DDoS (compression, TLS handshake etc.)
  - A lot of IP addresses with low traffic
  - Machine learning is used for clustering
  - How to filter out all HTTP requests with “Host: www.example.com:80”? 
  - "Lessons From Defending The Indefensible": https://www.youtube.com/watch?v=pCVTEx1ouyk
TCP stream filter

- IPtables strings, BPF
  - HTTP headers can cross packet bounds
  - Scan large URI or Cookie for Host value?
- Web accelerator
  - aren’t designed (suitable) for HTTP filtering
IPS vs HTTP DDoS

- e.g. Suricata, has powerful rules syntax at L3-L7
- Not a TCP end point => evasions are possible
- SSL/TLS
  - SSL terminator is required => many data copies & context switches
  - or double SSL processing (at IDS & at Web server)
- Double HTTP parsing
- Doesn’t improve Web server peroformance (mitigation != prevention)
Interbreed an HTTP accelerator and a firewall

- TCP & TLS end point
- Very fast **HTTP parser** to process HTTP floods
- Network I/O optimized for **massive ingress traffic**
- **Advanced filtering** abilities at all network layers
- Very fast Web cache to mitigate DDoS which we can’t filter out
  - ML takes some time for bots clusterization
Application Delivery Controller (ADC)

Web cache
DDoS protection
Load balancer
Web application security
Application performance monitoring
SSL/TLS offloading
Data compression
Connections QoS
Application layer DDoS

<table>
<thead>
<tr>
<th></th>
<th>Service from Cache</th>
<th>Rate limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nginx</td>
<td>22us</td>
<td>23us</td>
</tr>
</tbody>
</table>

- *(Additional logic in limiting module)*
- **Fail2Ban**: write to the log, parse the log, write to the log, parse the log...
Application layer DDoS

<table>
<thead>
<tr>
<th></th>
<th>Service from Cache</th>
<th>Rate limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nginx</td>
<td>22us</td>
<td>23us</td>
</tr>
</tbody>
</table>

- (Additional logic in limiting module)
- **Fail2Ban**: write to the log, parse the log, write to the log, parse the log… - really in 21th century?!
- **tight integration** of Web accelerator and a firewall is needed
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering etc.
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering? etc.
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering? etc.
  - C10K
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering? etc.
  - **C10K – is it a problem for bot-net? SSL?**
  - what about tons of 'GET / HTTP/1.0\n\n'? **CASES!**
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering? etc.
  - C10K – *is it a problem for bot-net?* SSL? CORNER
  - what about tons of `GET / HTTP/1.0
  
  `? CASES!
- Kernel-mode Web-accelerators: TUX, kHTTPd
  - basically the same sockets and threads
  - zero-copy → `sendfile()`, lazy TLB
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering? etc.
  - C10K – is it a problem for bot-net? SSL? CORNER
  - what about tons of 'GET / HTTP/1.0

- Kernel-mode Web-accelerators: TUX, kHTTPd
  - basically the same sockets and threads
  - zero-copy → sendfile(), lazy TLB => not needed
Web-accelerator capabilities

- Nginx, Varnish, Apache Traffic Server, Squid, Apache HTTPD etc.
  - cache static Web-content
  - load balancing
  - rewrite URLs, ACL, Geo, filtering? etc.
  - C10K – is it a problem for bot-net? SSL?
  - what about tons of 'GET / HTTP/1.0\n\n'?
- Kernel-mode Web-accelerators: TUX, kHTTPd
  - basically the same sockets and threads
  - zero-copy → sendfile(), lazy TLB => not needed

Tempesta Technologies
Web-accelerators are slow: SSL/TLS copying

- User-kernel space copying
  - Copy network data to user space
  - Encrypt/decrypt it
  - Copy the data to kernel for transmission

- Kernel-mode TLS
  - Facebook, RedHat: https://lwn.net/Articles/666509/
  - TLS handshake is still an issue
Web-accelerators are slow: profile

<table>
<thead>
<tr>
<th>%</th>
<th>symbol name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5719</td>
<td>ngx_http_parse_header_line</td>
</tr>
<tr>
<td>1.0303</td>
<td>ngx_vslprintf</td>
</tr>
<tr>
<td>0.6401</td>
<td>memcpy</td>
</tr>
<tr>
<td>0.5807</td>
<td>recv</td>
</tr>
<tr>
<td>0.5156</td>
<td>ngx_linux_sendfile_chain</td>
</tr>
<tr>
<td>0.4990</td>
<td>ngx_http_limit_req_handler</td>
</tr>
</tbody>
</table>

=> flat profile
Web-accelerators are slow: syscalls

epoll_wait(., [{EPOLLIN, .}, .])
recvfrom(3, "GET / HTTP/1.1\r\nHost: .", .)
write(1, ".limiting requests, excess.", .)
writev(3, "HTTP/1.1 503 Service.", .)
sendfile(3, ., 383)
recvfrom(3, .) = -1 EAGAIN
epoll_wait(., [{EPOLLIN, .}, .])
recvfrom(3, ".", 1024, 0, NULL, NULL) = 0
close(3)
Web-accelerators are slow: HTTP parser

Start: state = 1, *str_ptr = 'b'

while (++str_ptr) {
    switch (state) { <= check state
        case 1:
            switch (*str_ptr) {
                case 'a':
                    ...
                    state = 1
                case 'b':
                    ...
                    state = 2
            }
        case 2:
            ...
    }
}
Web-accelerators are slow: HTTP parser

Start: $state = 1, \*str\_ptr = 'b'$

```c
while (++str_ptr) {
    switch (state) {
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2 <= set state
        }
    case 2:
        ...
    }
    ...
}
```
Web-accelerators are slow: HTTP parser

Start: state = 1, *str_ptr = 'b'

while (++str_ptr) {
    switch (state) {
        case 1:
            switch (*str_ptr) {
                case 'a':
                    ...
                    state = 1
                case 'b':
                    ...
                    state = 2
            }
        case 2:
            ...
    }
    ... <= jump to while
}
Web-accelerators are slow: HTTP parser

**Start**: state = 1, *str_ptr = 'b'

```c
while (++str_ptr) {
    switch (state) { <= check state
        case 1:
            switch (*str_ptr) {
                case 'a':
                    ...
                    state = 1
                case 'b':
                    ...
                    state = 2
            }
        case 2:
            ...
    }
}
```
Web-accelerators are slow: HTTP parser

Start: state = 1, *str_ptr = 'b'

while (++str_ptr) {
    switch (state) {
    case 1:
        switch (*str_ptr) {
        case 'a':
            ...
            state = 1
        case 'b':
            ...
            state = 2
        }
    case 2:
        ...
        <= do something
    }
    ...
}
Web-accelerators are slow: HTTP parser

```c
while (++str_ptr) {
    switch (state) {
        case 1:
            switch (*str_ptr) {
                case 'a':
                    ...
                    state = 1
                case 'b':
                    ...
                    state = 2
            }
        case 2:
            ...
    }
    ...
}
```

```c
while (1):
    STATE_1:
        switch (*str_ptr) {
            case 'a':
                ...
                ++str_ptr
                goto STATE_1
            case 'b':
                ...
                ++str_ptr
    STATE_2:
        ...
```
Web-accelerators are slow: strings

- We have AVX2, but GLIBC doesn’t still use it
- HTTP strings are special:
  - No ‘\0’-termination (if you’re zero-copy)
  - Special delimiters (‘:’ or CRLF)
  - strcasecmp(): no need case conversion for one string
  - strspn(): limited number of accepted alphabets
- switch()-driven FSM is even worse
Fast HTTP parser

  - 1.6-1.8 times faster than Nginx’s
- HTTP optimized AVX2 strings processing: http://natsys-lab.blogspot.ru/2016/10/http-strings-processing-using-csse42.html
  - ~1KB strings:
    - strncasecmp() ~x3 faster than GLIBC’s
    - URI matching ~x6 faster than GLIBC’s strspn()
    - kernel_fpu_begin() / kernel_fpu_end() for whole softirq shot
Web-accelerators are slow: async I/O
Web-accelerators are slow: async I/O
Web-accelerators are slow: async I/O
Web-accelerators are slow: async I/O

Web cache also resides in CPU caches and **evicts** requests
HTTPS/TCP/IP stack

- Alternative to user space TCP/IP stacks
- HTTPS is built into TCP/IP stack
- **Kernel TLS** (fork from mbedTLS) – no copying
  (1 human month to port TLS to kernel!)
- HTTP firewall plus to IPtables and Socket filter
- Very fast **HTTP parser** and strings processing using AVX2
- **Cache conscious** in-memory Web-cache for DDoS mitigation
- TODO
  - **HTTP QoS** for asymmetric DDoS mitigation
  - **DSL** for multi-layer filter rules
TODO: HTTP QoS for asymmetric DDoS mitigation

- https://github.com/tempesta-tech/tempesta/issues/488
- “Web2K: Bringing QoS to Web Servers” by Preeti Bhoj et al.
- **Local stress**: packet drops, queues overrun, response latency etc *(kernel: cheap statistics for asymmetric DDoS)*
- **Upsream stress**: req_num / resp_num, response time etc.
- **Static QoS rules** per vhost: HTTP RPS, integration w/ Qdisc - TBD
- Actions: reduce TCP window, don’t accept new connections, close existing connections
Synchronous sockets: HTTPS/TCP/IP stack

- Socket callbacks call TLS and HTTP processing
- Everything is processing in softirq (while the data is hot)
- No receive & accept queues
- No file descriptors
- Less locking
Synchronous sockets: HTTPS/TCP/IP stack

- Socket callbacks call TLS and HTTP processing
- Everything is processing in softirq (while the data is hot)
- No receive & accept queues
- No file descriptors
- Less locking
- Lock-free inter-CPU transport

=> faster socket reading

=> lower latency
skb page allocator: zero-copy HTTP messages adjustment

- Add/remove/update HTTP headers w/o copies
- skb and its head are allocated in the same page fragment or a compound page
skb page allocator: zero-copy HTTP messages adjustment

- Add/remove/update HTTP headers w/o copies
- skb and its head are allocated in the same page fragment or a compound page
Frang: HTTP DoS

- **Rate limits**
  - request_rate, request_burst
  - connection_rate, connection_burst
  - concurrent_connections

- **Slow HTTP**
  - client_header_timeout, client_body_timeout
  - http_header_cnt
  - http_header_chunk_cnt, http_body_chunk_cnt
Frang: WAF

- **Length limits:** http_uri_len, http_field_len, http_body_len
- **Content validation:** http_host_required, http_ct_required, http_ct_vals, http_methods
- **HTTP Response Splitting:** count and match requests and responses
- **Injections:** carefully verify allowed character sets
- **...and many upcoming filters:**  
  https://github.com/tempesta-tech/tempesta/labels/security
- **Not a featureful WAF**
Sticky cookie

- User/session identification
  - Cookie challenge for dummy DDoS bots
  - Persistent/sessions scheduling (no rescheduling on a server failure)
- **Enforce**: HTTP 302 redirect

```
sticky name=__tfw_user_id__ enforce;
```
Sticky cookie

- User/session identification
  - Cookie challenge for dummy DDoS bots
  - Persistent/sessions scheduling (no rescheduling on a server failure)
- Enforce: HTTP 302 redirect

```bash
sticky name=__tfw_user_id__ enforce;
```

- TODO: JavaScript challenge
  https://github.com/tempesta-tech/tempesta/issues/536
if ((req.user_agent =~ /firefox/ || req.cookie !~ /^our_tracking_cookie/) && (req.x_forwarded_for != "1.1.1.1" || client.addr == 1.1.1.1))

# Block the client at IP layer, so it will be filtered efficiently w/o further HTTP processing.

tdb.insert("ip_filter", client.addr, evict=10000);
Performance

Intel Xeon E3-1240v6 (4 cores); 8B response, keep-alive

https://github.com/tempesta-tech/tempesta/wiki/HTTP-cache-performance
Performance analysis

- **~x3 faster than Nginx** (~600K HTTP RPS) for normal Web cache operations
- Must be **much faster to block HTTP DDoS** (*DDoS emulation is an issue*)
- Similar to DPDK/user-space TCP/IP stacks http://www.seastar-project.org/http-performance/
- ...bypassing Linux TCP/IP isn’t the only way to get a fast Web server
- ...lives in Linux infrastructure: LVS, tc, IPtables, eBPF, tcpdump etc.
Keep the kernel small

- Just **30K LoC** (compare w/ 120K LoC of BtrFS)
- Only generic and crucial HTTPS logic is in kernel
- Supplementary logic is considered for user space
  - HTTP compression & decompression
    https://github.com/tempesta-tech/tempesta/issues/636
  - Advanced DDoS mitigation & WAF (e.g. full POST processing)
  - ...other HTTP users (Web frameworks?)
- Zero-copy **kernel-user space transport** for minimizing kernel code
TODO:
Zero-copy kernel-user space transport

- HTTPS DDoS mitigation & WAF
  - Machine learning clusterization in user space
  - Automatic L3-L7 filtering rules generation
Thanks!

- Web-site:  http://tempesta-tech.com  *(Powered by Tempesta FW)*
- Availability:  https://github.com/tempesta-tech/tempesta
- Blog:  http://natsys-lab.blogspot.com
- E-mail:  ak@tempesta-tech.com