A software-based approach to reproduce and detect flooding attacks against DNS

Santiago Ruano Rincón  Sandrine Vaton  Stéphane Bortzmeyer

IRISA Lab - IMT Atlantique (Brest, France) & AFNIC Labs (France)

RIPE Meeting 74, May 11th 2017
Some keywords

- Network traffic online analysis
- Countermeasure flooding attacks
- Software approaches
- Statistical tools
- Distributed data sources (for future work)
Problem: flooding attacks against DNS infrastructure

- Random qname against French servers, September 4th 2014.
  https://indico.dns-oarc.net/event/20/session/3/contribution/37

Figure: Wallis-et-Futuna (.wf)

Image: (C) Dr. Angela Kepler http://www.pbif.org/images
A software-based DNS flooding attack detection testbed

How to help resilience of DNS infrastructure?
Outline

1. Introduction
2. Generating DNS traffic
3. Reception and analysis
A software-based DNS flooding attack detection testbed

- **Goal:** Detect and countermeasure flooding-DDoS attacks
  - Reproduce attacks - Generate traffic
  - Read and process packets on the fly
  - Future: classify

- Flexible and reliable tools to analyse DNS traffic at Nx11Mpps.
- **We want flexibility!** ⇒ Highest abstraction level
  - Commodity hardware
  - Software network frameworks
A software-based DNS flooding attack detection testbed

Thanks to CNRS INS2I Projet Exploratoire Premier Soutien (PEPS) Sécurité informatique et des systèmes cyberphysiques (SISC) 2016.

(IMT Atlantique & AFNIC Labs)  Software-based DDoS attack testbed  DNS WG - RIPE74  7 / 33
Hardware environment

- Dell 7X00 Precision workstations
- Dual socket. Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
- From 16GB to 64GB RAM
- Debian Jessie
- Intel NICs:
  - Dual SFP+ port X520-DA2
  - Dual RJ45 port X520-TA2.
  - Dual QSFP+ port XL710-QDA2
Software network engines for commodity hardware

Alphabetically sorted:

- Data Plane Development Kit DPDK (Intel)
  - Strong support from industry
  - High-performance Packet CAPture HPCAP (Moreno et al., UAM) [MRdR + 15]
  - Specially designed for capture and to avoid packet losses.
  - Academic work that needs a stable release.

- PFQ (Bonelli et al., Univ. of Pisa) [BPGP12]
  - Uses the Intel vanilla driver, relying on multi-core processing.
  - Unable to handle 10Gbps on a single core.

- PF_RING (Deri et al., Ntop) [PFR]
  - Zero-copy version needs a commercial license.
Software network engines for commodity hardware

Alphabetically sorted:

- **Data Plane Development Kit DPDK (Intel)**
  - Strong support from industry
- **High-performance Packet CAPture HPCAP (Moreno et al., UAM) [MRdR⁺15]**
  - Specially designed for capture and to avoid packet losses.
  - Academic work that needs a stable release.
- **PFQ (Bonelli et al., Univ. of Pisa) [BPGP12]**
  - Uses the Intel vanilla driver, relying on multi-core processing.
  - Unable to handle 10Gbps on a single core.
- **PF_RING (Deri et al., Ntop) [PFR]**
  - Zero-copy version needs a commercial license.
Outline

1 Introduction

2 Generating DNS traffic

3 Reception and analysis
Shield of Perseus (SOP)

- Written in C
- Relies on standard Linux NAPI
- Running on Linux:
  - ~520Kpps fully-random requests @ 2200Mhz single-core
    - Increases when using several threads
MoonGen and libmoon

- Paul Emmerich, TUM [EGR+15]
- LuaJIT interface to DPDK: scripts control packet generation
- Delegate rate control and timestamping to hardware
- https://github.com/emmericp/MoonGen
- https://github.com/libmoon/libmoon
Reproducing DNS flooding attacks

Requirements

- Randomise different bytes/fields.
  - Source IP addresses
  - TTL
  - qname (varying lengths)
  - Varying DNS query data
  - EDNS, UDP buffer size
  - ...

- Reproduce:
  - Random qnames
  - Reflect-and-amplify
  - ...

- Easily take into account other attacking strategies
- No need to highly accurate timestamping/control
gGALOP: our DNS-packet generator

- gGALOP (gGALOP Generates A Lot Of Packets)
- On top of MoonGen + DPDK
- *Reproducing DNS 10Gbps flooding attacks with commodity-hardware*, TRAC-IWCMC 2016
To give it a name is more difficult than DNS-flooding

- ∼320-line Lua(JIT) script
- ∼11M full-random pps per CPU core
- Batch processing

```lua
function loadSlave(...) 
    local mem = memory.createMemPool(function(buf)
        buf:getDnsPacket(ipv4):fill{
            ip4Src=genIPv4AddSource(),
            ip4Dst=dnsServerIP,
            ...
            dnsMessageContent=genBody()
        }
    end)

    while dpdk.running() do
        local bufs = mem:bufArray(MAX_BURST_SIZE)
        bufs:alloc()
        ...
        sent = queue:send(bufs)
```
CPU Requirements to saturate a 10 GbE link

- Shield of Perseus (SOP)
- gGALOP
- MoonGen’s example/tx-multi-core.lua (simple, non-random packets)
Generation results

- **Solution:** DPDK + MoonGen + Lua scripts
- Generating packets controlled by Lua scripts
  - Then: highest possible level of abstraction
  - Highly flexible
- Successfully reproduce random qnames and reflect-and-amplify
- Able to scale to N×11Mpps:
  - Saturate 3×10GbE ports on a quad-core CPU
DNS Servers versus DNS flooding

- We don’t have a 10GbE switch (yet)
DNS Servers versus DNS flooding

- DNS serving a 3M-record zone.
- PowerDNS
- ISC BIND
  - Listening on both ports (Intel X520-DA2)
  - Single core
- dnsperf while flooding the server
  - gGALOP (11Mpps)
  - SOP (665Kpps)
DNS Servers versus DNS flooding

- **PowerDNS:**
  - SOP: 20% answered requests
  - gGALOP: 30% answered requests

- **BIND resisted!**
  - SOP: 95% answered requests
  - gGALOP: 100% answered requests

- **SOP has a stronger impact!**
DNS Servers versus DNS flooding

- PowerDNS:
  - SOP: 20% answered requests
  - gGALOP: 30% answered requests

- BIND resisted!
  - SOP: 95% answered requests
  - gGALOP: 100% answered requests

- SOP has a stronger impact!

- Why? From 100M queries sent, Bind received:
  - 324883 (gGALOP)
  - 6379850 (SOP)
  - The rest was lost between the interface and the kernel

(IMT Atlantique & AFNIC Labs) Software-based DDoS attack testbed DNS WG - RIPE74
DNS Servers versus DNS flooding

- PowerDNS:
  - SOP: 20% answered requests
  - gGALOP: 30% answered requests

- BIND resisted!
  - SOP: 95% answered requests
  - gGALOP: 100% answered requests

- SOP has a stronger impact!

- Why? From 100M queries sent, Bind received:
  - 324883 (gGALOP)
  - 6379850 (SOP)
  - The rest was lost between the interface and the kernel

- Same machine serving on multiple interfaces is a good idea?
- Slower attacks can be more successful?
Outline

1. Introduction
2. Generating DNS traffic
3. Reception and analysis

(IMT Atlantique & AFNIC Labs)
Current challenge: how to identify trouble sources?

- Capture and analyse traffic
  - What approach scores highest at minimizing packet drops?
- Rely on libmoon (base of Moongen)
- Statistics-based detection
Current challenge: how to identify trouble sources?

- Identify Heavy Hitters
- **Counting** / keeping statistics about:
  - Most frequent source IP address
    - IPv4 ($2^{32}$)
    - IPv6 ($2^{128}$) Tests are coming soon :-)
  - Most frequent domains
    - Random, varying length (undetermined)
Statistical tools

- Cormode and Muthukrishnan, *Count-Min Sketch* [CM05]
  - Fixed and controlled size table
  - (Non-reversible) hash functions
- Misra & Gries, *Finding Repeated Elements* [MG82]
- Entropy deviation
  - Keisuke Ishibashi & Masaharu Sato, Hierarchical Aggregate Entropy.
    DNS-OARC 2010-02 https://www.dns-oarc.net/files/meeting-201002/4_Keisuke_Ishibashi.pdf
Count-Min Sketch

- $\epsilon - \gamma$ approximation
- Count every $x$ seconds
- Analyse 11Mrps on 4 cores (Intel E5-2630 v3 @ 2.40GHz)
Estimate most frequent domains

1: $misragries \leftarrow mgInit(k)$
2: $sketch \leftarrow cmsInit(\epsilon, \gamma)$
3: for packet in $rxBuffer()$ do
   4:   {Get qnames from DNS payload}
   5:   for $qname$ in $getQNAMEs(packet)$ do
      6:      $trimmedQN \leftarrow trimQNAME(qname)$
      7:      $misragries.count(trimmedQN)$
      8:      $hashedQN \leftarrow hashString(trimmedQN)$ {Hash into int}
      9:      $sketch.update(hashedQN)$
   10: end for
11: end for
Demo time! Counting Rx’ed packets per domain

Total counts (requests per domain):
larry.3s. : 19999880
curly.3s. : 19999881
hola.org. : 19999880
flooding.evil. : 19999885
moe.3s. : 19999880
example.com. : 19999886

----
total: 119999292

Total packets received by device: 120000000
## Demo time! Counting Rx’ed packets per domain

<table>
<thead>
<tr>
<th>Domain</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>larry.3s.</td>
<td>19999880</td>
</tr>
<tr>
<td>curly.3s.</td>
<td>19999881</td>
</tr>
<tr>
<td>hola.org.</td>
<td>19999880</td>
</tr>
<tr>
<td>flooding.evil.</td>
<td>19999885</td>
</tr>
<tr>
<td>moe.3s.</td>
<td>19999880</td>
</tr>
<tr>
<td>example.com.</td>
<td>19999886</td>
</tr>
</tbody>
</table>

---

Total: 119999292

Total packets received by device: 120000000
Ethical concerns

- Access to payload (and how to analyse encrypted DNS?)
- Not logging
- Avoid linking IP sources to queries
- What else?
Thanks to

- CNRS PEPS 2016 Program
- Fondation Carnot
- DNS-OARC
- RACI :-)
- libmoon and MoonGen authors
Thanks for your attention

Feedback?


Limited by random fields?

- Using a single core, CPU @1.6Ghz
- Randomising fields does not strongly impact performance