1 Technical details

1.1 Technical background reminder

A domain delegation is technically done by inserting a nameserver (NS) resource record in a DNS zone. The "owner name" field of this resource record indicates where the zone cut between the parent zone and the child zone occurs, and the "rdata" field of this record indicates a hostname. This hostname is a domain name to which IP addresses resource records (either A for IPv4 addresses or AAAA for IPv6 addresses) are associated.

The hostname indicated by the "rdata" field can either be a child of the delegated domain or falling under the bailiwick of a different domain.

RFC-compliant delegations are therefore of two kinds:

Delegations requiring a glue record If the hostname is a child of the delegated domain, a chicken-and-egg problem occurs. Indeed, in order to get the IP address associated with the hostname, we need to follow the domain delegation, which is the reason why the hostname is being resolved in the first place. This problem is solved by adding a non-authoritative IP address resource record, known as a "glue" record.

Glueless Delegations If the hostname falls under the bailiwick of a different domain, the resolver trying to follow the delegation puts the initial query on hold, and tries to recursively resolve the hostname. Once the IP address associated to this hostname is known, the initial query is resumed.

1.2 DNS resolvers denial of service

The iDNS Attack principle is to create an infinite chain of domains each delegating their authority to one of their subdomains through specially-crafted delegations.

This scenario can be realized with a custom DNS authoritative server implementation serving such delegation information.

Vulnerability to the iDNS attack stems from insufficiently bounded, if at all, number of queries that are emitted by resolvers in order to resolve an original query.

The impact of following these endless delegations can be any of the following:

1. Complete memory exhaustion leading to kernel Out-of-Memory kill of the DNS service
2. High CPU consumption
3. Exhaustion of firewall connection tracking state table, under default configuration
4. Cache voiding with the replacement of useful cached entries by useless delegation information

1.3 Traffic Amplification

Some DNS resolver implementations are willing to follow out-of-bailiwick glue records. On top of that, they follow several of these non-authoritative glue information simultaneously. We have proven in laboratory experiments that this behavior can be exploited by an attacker to achieve traffic amplification. We have measured a bandwidth amplification factor of ≈ 1 and a packet amplification factor of ≈ 10.

This attack can be performed thanks to a customized authoritative name server answering queries with a specially crafted valid RFC-compliant DNS response.

Following the implementation under attack, various behaviors have been observed. Some implementations are more aggressive when the victim does not answer to the DNS queries that are sent to him. This allows to attack any non-DNS enabled server. On the contrary, some implementations are more aggressive when the victim answers with REFUSED or SERVFAIL answers. These answers can be easily obtained from authoritative DNS servers that are not authoritative for the queried domain name, and from recursive nameservers with access control on the recursion feature.

2 Countermeasures and Workarounds

The ANSSI recommend that DNS resolver software vendors implement the following defensive strategies in order to mitigate these potential vulnerabilities:

R1 Bound check the total number of queries emitted per domain name queried by one of its clients (stub-resolvers or forwarders) in order to resolve it

R2 Bound check the total number of concurrent queries emitted toward a single nameserver and postpone these queries if the limit is reached.

These recommendations prove to be similar to the preventive measures that OpenDNS had already deployed against such attacks. Their software limits the recursion depth, the number of in-flight queries per destination, the number of name servers per delegations and does not fetch additional or authoritative records for glues. On top of that, each client query is time-constrained: when the time is up, the client receives a SERVFAIL error message and the recursion process stops trying to resolve the name or any record associated to the client query.

On top of these recommendations to software developers, network architects that are protecting their DNS servers behind firewalls should follow these recommendations:

R3 If an attack is detected and the maximum number of state contained in the connection tracking table may be reached, temporarily increase the maximum size of the connection tracking table and adjust the eventual hash buckets count associated to this table.
This recommendation can be applied on Linux systems host-firewall and Linux-based network-firewalls by modifying the `net.nf_conntrack_max` sysctl and by modifying the `hashsize` of the `nf_conntrack` kernel module.

R4 If the R3 recommandation is not sufficient to thwart the attack, the conntrack timeout delay associated to DNS traffic can be fine-tuned in order to reduce the time a conntrack entry stays in the table.
This recommandation can be applied on a Linux system hosting the DNS service with Netfilter and iptables with rules such as:

```
    The ncft command defines a new conntrack timeout policy (2 seconds per state)

    # nfct timeout add reduced-dns-timeout-policy inet udp unreplied 2 replied 2

    The following iptables rules rewrite the default timeout policy with our own

    # iptables -t raw \
        -I OUTPUT \
        -o $OUTPUT_INTERFACE \
        -s $RESOLVER_IP_ADDRESS \
        -p udp \
        --sport 1024: \
        --dport 53 \
        -j CT --timeout reduced-dns-timeout-policy
    # iptables -t raw \
        -I PREROUTING \
        -i $OUTPUT_INTERFACE \
        -d $RESOLVER_IP_ADDRESS \
        -p udp \
        --dport 1024: \
        --sport 53 \
        -j CT --timeout reduced-dns-timeout-policy
```

Network administrators should be aware that the `nfct` utility might not be packaged on some GNU/Linux distributions.
A variant of these rules can be also be crafted for Linux-based network-firewalls forwarding the DNS traffic.

R5 If the R3 and R4 recommendations are not sufficient to thwart the risk of overwhelming the

connection tracking system, in last resort and as a temporary workaround, connection tracking can be entirely disabled for outgoing DNS traffic.
This recommendation can be applied on a Linux system hosting the DNS service with iptables rules such as:

```
The two following rules authorize stateless DNS traffic

# iptables -t filter \
    -I OUTPUT \
    -o $OUTPUT_INTERFACE \
    -s $RESOLVER_IP_ADDRESS \
    -p udp \
    --sport 1024: \
    --dport 53 \
    -j ACCEPT
# iptables -t filter \
    -I INPUT \
    -i $OUTPUT_INTERFACE \
    -d $RESOLVER_IP_ADDRESS \
    -p udp \
    --dport 1024: \
    --sport 53 \
    -j ACCEPT

The two following rules disable connection tracking of DNS traffic

# iptables -t raw \
    -I OUTPUT \
    -o $OUTPUT_INTERFACE \
    -s $RESOLVER_IP_ADDRESS \
    -p udp \
    --sport 1024: \
    --dport 53 \
    -j NOTRACK
# iptables -t raw \
    -I PREROUTING \
    -i $OUTPUT_INTERFACE \
    -d $RESOLVER_IP_ADDRESS \
    -p udp \
    --dport 1024: \
    --sport 53 \
    -j NOTRACK
```

A variant of these rules can also be crafted on Linux-based network-firewalls forwarding DNS traffic.

Extreme caution must be exercised when setting up and tearing down these rules. Indeed, legitimate packets could be dropped if these rules were to be inserted or deleted out-of-order. Tearing down these rules must be done in reverse order and a delay must be observed after removing the NOTRACK rules and before removing the stateless ACCEPT rules.