NSEC3 Hash Performance

Yuri Schaeffer[1], *NLnet Labs*

NLnet Labs document 2010-002 March 18, 2010

Abstract

When signing a zone with DNSSEC and NSEC3, a choice has to be made for the key size and the number of hash iterations. We have measured the effect of the number of hash iterations in NSEC3 in terms of maximum query load using NSD and Unbound. This document presents the results of these measurements and compares the cost for validating and authoritative name servers and allows for an educated choice for these parameters.

1 Introduction

The hash calculation for NSEC3 is defined (RFC5155, Section 5, [1]) as

```
IH(salt, x, 0) = H(x || salt)
IH(salt, x, k) = H(IH(salt, x, k-1) || salt)
```

With `H` a hashing function, `k` the number of iterations, and `||` a concatenation. The resulting hash is calculated as: `IH(salt, owner name, iterations)`. More iterations increase NSEC3's resilience to dictionary attacks by increasing the costs of generating such a dictionary, at the expense of validation cost.

The number of hash iterations is determined by the authoritative server but will affect validators as well. Thus, when deciding on the number of NSEC3 iterations one should take the performance impact of both the authoritative servers and the validating servers in to account. To help operators make an educated choice we will measure this performance impact by simulating a worst case scenario where little caching is possible.

Research question

What is the worst case effect of the number of NSEC3 hash iterations on the query load of a recursive name server?

2 Measurement Scope

Below the machine simulating DNS clients will be referred to as Player. The machine capturing the responses Listener. The validating resolver will be called

[1] Yuri@NLnetLabs.nl

Validator and the authoritative server Authority. In our setup Player and Listener are the same machine but will be addressed separately for clarity.

We want to find the maximum query load for a set of *ZSK size – hash iterations* pair. These ranges are further defined in section 2.1.

To simulate the worst case scenario the authorative server must be provoked to answer each query with an NXDOMAIN response with a closest encloser, a next closer, and a wild card denial. The queries are picked in such a way that they are never in the Validators cache. By making the zone larger than the query sample, we reduce the chance of any other resource record being read from the cache.

We define the maximum query load of a server as the highest query rate at which at least 99 percent of the queries is answered.

We will test 2 different setups:

1. Player sending queries to Validator. The queries must not be (fully) cached so Validator will have to contact Authority and verify the responses each time.

2. Player sending directly to Authority, to ensure that both Authority and Player can outperform Validator.

2.1 Ranges

RFC5155, Section 10.3 specifies the maximum number of hash iterations for NSEC3 given the size of the ZSK. The key size in bits must be rounded upwards to the nearest table (Table 1) entry or downwards if necessary. More iterations than the corresponding number in the table MUST NOT be used. NSEC3 RRs with more iterations MIGHT be considered insecure. This effectively puts a maximum on the number of iterations.

Key size	Iterations
1024	150
2048	500
4096	2500

Table 1: Maximum number of NSEC3 iterations for each key size.

Using this table it make sense to test the following scenarios: 1024 bit key 1–150 iterations, 2048 bit key 1–500 iterations, and 4096 bit key 1–2500 iterations.

To get an accurate result we need quite a lot of queries to replay and even more names in our zone. For practical purposes we decided upon a 3 to 1 ratio: 500K names in the zone, and 167K queries to be replayed.

Since signing of larges zones is computationally expensive it is not feasible to do the whole range of iterations. We will take only 11 sample points for each scenario. E.g. for the 1024 bit key 1, 15, 30, 45, ..., and 150 iterations.

3 Preparation

For this measurement we need a machine in the role as authoritative name server. We set up NSD as a root server, create a root zone and a zone file containing a large amount of A records (the TLD zone).

The zone file should contain more names than queries to be sent, to ensure as little caching as possible is done by Validator. We generate 500K unique names in the form of "[a-z]{5}v.qx", the length of every name is constant.

This zonefile must then be signed for all three key lengths and all iteration counts. The result will be 3x11 signed zonefiles. The size of the KSK is chosen constant at 2048 bits.

At the machine acting as validator, Unbound is installed with root hints pointing to Authority so that queries are directly answered. With each measurement, Unbound is restarted to ensure the same begin state for all tests.

Player has a trace containing queries for non-existent names which are in the zone of Authority (in the form of "[a-z]{5}i.qx"). Every query is sent via UDP and with the DO bit set. In case the player forms a bottleneck an optional Listener could be used to catch the responses. To prevent Player sending ICMP for each packet it receives, "destination-unreachable" is blocked in the firewall for outgoing traffic:

```
iptables -A OUTPUT -p icmp -o eth1 --icmp-type \
         destination-unreachable -j DROP
```

Signing the zonefiles and rebuilding the NSD database is expensive but we can do this in advance. If the keys and zones are generated first, the tasks can be distributed over multiple machines and cores.

4 Method

A shell script running on the player performs roughly the folling actions:

```
Configure firewalls
For each NSD configuration C do:
    Start NSD with C; Wait until NSD is ready.
    Do binary search for rate R with 99% success:
        (Re)start Unbound
        Start Tcpdump; sleep a few seconds
        Dig for SOA .
        Replay at rate R
        Sleep a few seconds; stop tcpdump
        Calculate new R
    Stop NSD
```

The Dig in the above code gives Unbound the chance to cache some of the root data saving work in the first test query. Making all queries equally expensive.

Waiting for NSD to be ready to take queries is done by polling every thirty seconds and test no threads are in a running state. In our environment this can take up to 20 minutes for large databases.

5 Hardware and Software Versions

All three machines have the same operating system installed, *CentOS* with kernel version *2.6.18-164.11.1.el5* and 2 Gigabytes of RAM. The machines have a dedicated Gigabit network for the experiments, other communication uses a separate control-network.

Player

- 4 cores: Intel(R) Xeon(TM) CPU 3.20GHz, 1MB cache
- Tcpreplay 3.4.3
- Tcpdump 3.9.4

Validator

- 4 cores: Intel(R) Xeon(TM) CPU 3.20GHz, 1MB cache
- Unbound: Subversion Trunk r1993. Configured to use 1 core.

Authority

- 2 cores: Intel(R) Xeon(TM) CPU 3.20GHz, 1MB cache
- NSD: Subversion Trunk r2969. Configured to use 1 core.

6 Results

The measurement results can be found in Table 3. For reference Table 2 with the maximum queries per second for NSEC is included as well. Please keep in mind that all these values are worst case and **do not reflect real-life performance**. The data is plot in Figure 1 for Unbound, and Figure 2 for NSD.

	Max qps	
Key size	Unbound	NSD
1024	3547	46963
2048	1848	44853
4096	669	36054

Table 2: Maximum queries per second using NSEC.

(a) 1024 bit key

Iterations	Max qps	
	Unbound	NSD
1	3251	25400
15	2958	19336
30	2723	18476
45	2547	17461
60	2372	15898
75	2196	14023
90	2120	13320
105	1962	12382
120	1845	10273
135	1727	9648
150	1669	9179

(b) 2048 bit key

Iterations	Max qps	
	Unbound	NSD
1	1727	21298
50	1493	14492
100	1318	10898
150	1142	8789
200	1025	7461
250	967	6367
300	849	5586
350	790	5273
400	732	4804
450	732	4179
500	673	3867

(c) 4096 bit key

Iterations	Max qps	
	Unbound	NSD
1	662	20419
250	498	6914
500	381	3945
750	322	2773
1000	264	2148
1250	264	1757
1500	205	1445
1750	205	1289
2000	186	1054
2250	172	976
2500	147	898

Table 3: Maximum queries per second for different key sizes using NSEC3.

Since NSD has all the signatures precomputed and does not have to do any validations itself, the keysize has little influence on NSD's performance. Hashes for non-existing names still have to be computed run-time for the *next closer*. Figure 2 shows that the maximum qps is mainly a function of the number iterations.

Apart from the hashing, Unbound needs to validate the signatures as well. The graphs in Figure 1 reflect this. The validation of the signatures is computationally harder than a hashing operation and thus key size has more influence on the performance of Unbound than iteration count.

Let us define *half performance count* as the number of NSEC3 hash iterations for which the maximum qps is 50 percent of the maximum qps with 1 hash iteration. An interesting observation is that on our hardware NSD's half performance

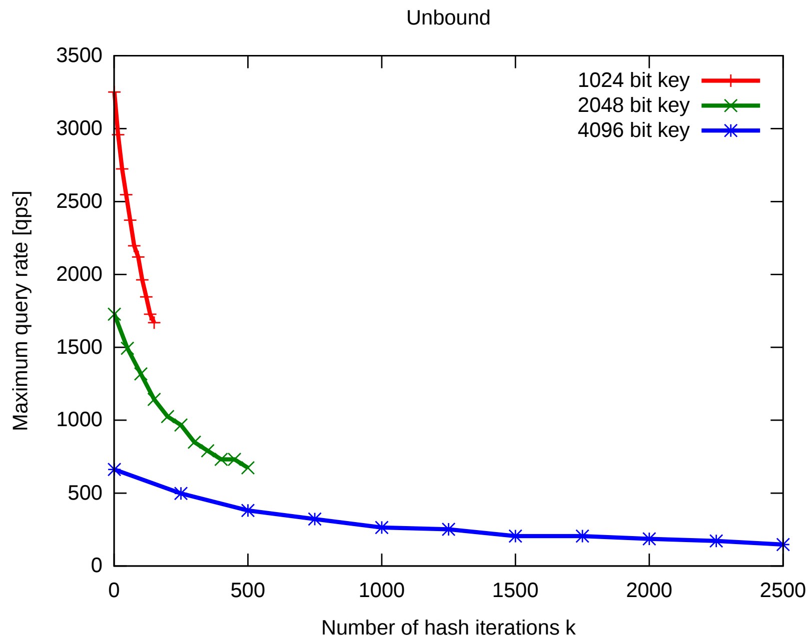

Figure 1: Maximum query rates for different key sizes.

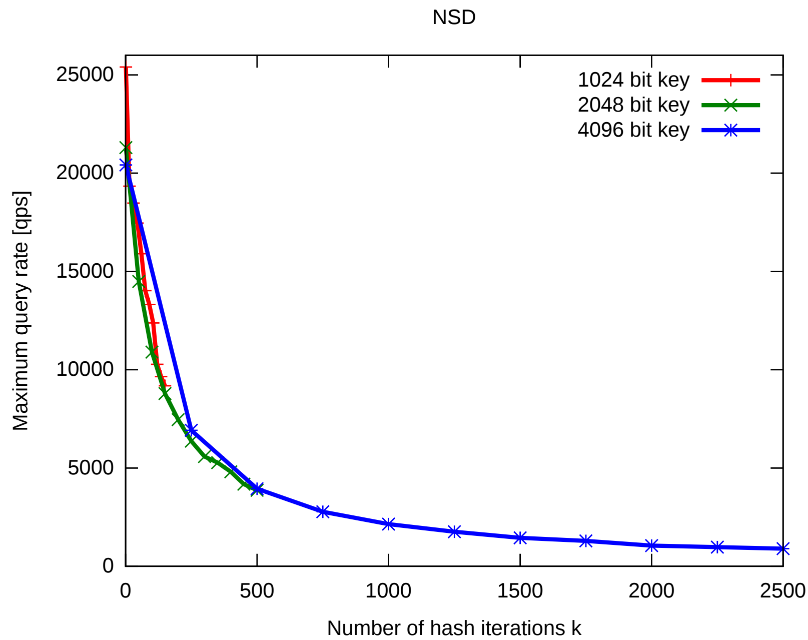

Figure 2: Maximum query rates for different key sizes.

count is at around 100 iterations, independent of key size. For Unbound the half performance count is *roughly* 150 for a 1024 bit key, 300 for a 2048 bit key, and 600 for a 4096 bit key.

7 Conclusion

We have two observations:

1. Even for short keys the number of iterations for NSEC3 has more impact on NSD's performance than on the performance of Unbound.

2. The half performance count is constant for NSD and will grow with the key size for Unbound.

There seems to be an appropriate alignment between incurred and imposed costs: the authoritative servers dictate these parameters but are affected the most themselves. This means that in order to find a satisfying amount of iterations it is sufficient to look at the performance impact of the authoritative name server, Figure 2.

References

[1] Laurie et al., RFC5155, *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*, http://www.ietf.org/rfc/rfc5155.txt, March 2008.