DNS OARC 2014 Fall Workshop Los Angeles 11-13 October 2014



Improved NSEC3 performance in DNSSEC

Dr Jonathan Tuliani Program Manager Microsoft Azure

jonathan.tuliani@microsoft.com

Recap: DNSSEC and NSEC3

RRSIG: "I certify that this DNS record set is correct" Problem: how to certify a negative response, i.e. that a record *doesn't* exist?

NSEC: "I certify that there are no DNS records (of type *X*) whose record name lies between *A* and *B* "

Problem: NSEC records enable zones to be enumerated

NSEC3: "I certify that there are no DNS records (of type X) whose record name <u>hash</u> lies between A' and B' "



Computing NSEC3

Authoritative Server

- New record *R*
- **Compute hash** of *R* (call this *R'*)
- Insert into presorted list of record name hashes (say Q', R', S')
- Sign new NSEC3 records for intervals (Q', R') and (R', S')
- Delete old NSEC3 record for interval (Q', S')

Authoritative Server

- Query Q received, no match found
- **Compute hash** of *Q* (call this *Q*')
- Find range (R', S') containing Q' in sorted list of precomputed NSEC3 records
- Return NSEC3 for (*R'*, *S'*)

Validating Server

- Query record Q, receive negative response + NSEC3 for interval (R', S')
- **Compute hash** of *Q* (call this *Q*')
- Verify Q' lies in interval (R', S')
- Verify authenticity of (R', S') in usual way signature checks, etc

Attacker

- Make random queries to build a database of NSEC3 records
- Brute-force search for records whose hashes match the endpoints in the NSEC3
 - Search space for DNS is typically small



Choice of hash

A cryptographic (one-way) hash is used, so record names cannot be computed directly from hashes. This hash is iterated to increase the computational load for an attacker.

However (from RFC5155):

More iterations result in greater resiliency of the hash value against dictionary attacks, but at a higher computational cost for both the server and resolver...[it] affects the zone owner's cost of signing and serving the zone as well as the validator's cost of verifying responses...a high number of iterations also introduces an additional denial-of-service opportunity against servers

Current mitigation: Iterations are limited by RFC5155 to have similar computational cost of verifying the signature on the NSEC3 RR (e.g. max 500 SHA1 iterations for a 2048-bit RSA signature)

This presentation proposes an alternative approach (patent pending)



Recap: RSA algorithm (<u>Clifford C. Cocks, 1973</u>)

Set up

- Pick 2 large primes, *p* and *q*
- Compute N = pq
- Pick public key e
- Compute private key d such that $ed = 1 \mod (p-1)(q-1)$

Encrypt: message *M*, cryptogram $C = M^e \mod N$ Decrypt: cryptogram *C*, message $M = C^d \mod N$ Sign: message (hash) *H*, signature $S = H^d \mod N$ Verify: signature *S*, message hash $H = S^e \mod N$



Time-lock puzzles (Ron Rivest, 1999)

Fixed effort for private key holder, arbitrarily large effort for public key holders

Set up

- Pick *p*, *q*, *N* as per RSA algorithm
- Choose iterations t
- Compute private key $d = 2^t \mod (p-1)(q-1)$

Task: Compute H^{2^t} mod N for given input H

- Public key holders require t mod-N squarings of H (time proportional to t)
- Private key holders have short-cut: $H^{2^t} \mod N = H^d \mod N$ (fixed time)



Creating a hash from a time-lock puzzle

Create a hash based on a time-lock puzzle:

- Public hash parameters t, N; private parameters p, q, d
- Hash input *M*
- Compute conventional hash *H* of *M*
- Compute $H' = H^{2^t} \mod N$ (or $H' = H^d \mod N$ if you have the private key)
- Truncate H' to desired length (or apply conventional hash again)



Implementation: Authoritative server

Set up:

- Choose hash parameters *p*, *q*, *t*
- Compute N and d
- Publish *N* and *t* (e.g. as a new variant of a DNSKEY record)

Use private key d to compute H' when creating NSEC3 records

- No increase in effort over today's NSEC3 records (based on iterated hash equivalent to verifying an RSA signature)
- No increase in NSEC3 record size
- Attacker's task can be made <u>arbitrarily difficult</u> by increasing the value of t



Implementation: Validating server

Problem: Increasing *t* also increases computational burden on validating servers

Possible solutions:

- Restrict *t*. We still have a potential gain for the authoritative server
- Rate-limit NSEC3 validations
- Off-load hash computation to the Client
 - Requires additional logic in 'stub' resolver, but not necessarily full DNSSEC validation
 - Could be done selectively as part of a rate-limiting scheme





Thank you



© 2014 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.