Combating DNS amplification using cookies

By:
Sean Rijs

Supervisor:
Roland van Rijswijk - Deij

August 5, 2014

Abstract

The Domain Name System (DNS) is an old system, dating back to the year 1983. Security was not an issue back then and, therefore, not under consideration during the design of DNS. When using the UDP protocol to send a DNS query to a server, the server cannot verify the source IP (i.e. IP spoofing) as opposed to the three-way handshake of TCP. A DNS amplification attack exploits source IP spoofing by sending a small query (e.g. 40 bytes) and getting the largest response possible (e.g. 4096 bytes) delivered to the spoofed IP, consequently flooding its network causing a Denial of Service (DoS).

To address this issue, an Internet Draft called DNS Cookies by Eastlake proposes an extension to the Extension mechanism for DNS (EDNS0). The concept is to provide authentication for the source IP in queries and responses by providing each component (i.e. stub resolver, recursive server and authoritative name server) with a secret known only to the component. It is designed to require no pre-configuration.

The goal of this research is to analyse the effectiveness of DNS Cookies. The conclusion is that DNS Cookies can be effective against DNS amplification attacks in theory. However, some changes need to be made to make a viable solution. Based on our measurements, we suggest a different policy where the default response size is limited to 240 bytes, whilst DNS servers are still able to serve 99% of the DNS clients without an DNS Cookie implementation using UDP. Responses higher than this value should either switch to TCP or must be authenticated using DNS Cookies. If this policy is used an incremental implementation is possible, and the amplification factor is severely reduced without requiring pre-configuration.

Contents

1 Introduction

The Domain Name System (DNS) is an old application that dates back to 1983. At that time, Distributed Denial of Service (DDoS) and computer security in general were not an issue. In the original specifications of DNS[1][2] it was decided to use User Datagram Protocol (UDP) for communication. However, UDP is vulnerable to spoofing of the source IP address field as there is no authentication whatsoever.

A DNS resolver should only process queries for its local network. However, a lot of resolvers on the Internet are misconfigured to process queries for the whole internet, which are called *Open Resolvers*.

The vulnerability to IP spoofing and the availability of Open Resolvers enables *DNS amplification attacks*, an attack that has a big impact nowadays[1]. An attacker spoofs the source IP address of the target in a DNS query and sends it to an Open Resolver. The DNS response is sent to the spoofed source IP address, consequently flooding the target with the response. Attackers intend to make the response bigger than the query giving it the amplification factor in the amount of traffic invested and amount of traffic sent to the target, calculated as $\frac{response\ size}{query\ size}$. This amplification factor is even further increased in modern DNS implementations that support the Extension mechanism for DNS (EDNS0), which enables UDP packets bigger than 512 bytes to for instance 4096 bytes.

An Internet Draft entitled DNS Cookies[3] by Eastlake specifies a lightweight security mechanism to mitigate DNS amplification attacks. The big advantage is that it does not require pre-configuration. Pre-configuration is the fact that a user has to configure a given software program before he or she is able to fully use the program. For example: in order to prevent Open Resolvers, users must first configure their resolvers to process DNS queries only for its local network. The DNS Cookies Internet Draft will be analysed in this study.

1.1 Research question

The main research question is:

> *Are DNS Cookies effective against DNS amplification attacks?*

Which includes the following sub-questions:

> *A: Is it feasible to combine DNS Cookies with Response Rate Limiting (RRL) in a normal network environment for an incremental implementation?*
>
> *B: Are there any possible improvements to the DNS Cookie Internet Draft?*
>
> *C: How can we maintain a low amplification factor using DNS Cookies?*

1.2 Related work

The effect of DDoS attacks is notable since 1999.[4] A possible defensive approach is to actively filter out the attackers traffic, which is not a trivial task and not always effective. Actively responding to DDoS attacks could be effective, but does not eliminate the real cause of the problem. A DNS amplification attacks is only one of the possible vulnerabilities to create a DDoS attack.

A possible solution to mitigate IP spoofing in UDP, hence DNS amplification attacks, is that all networks must adhere to Best Current Practice (BCP) 38. BCP 38[5] specifies ingress traffic filtering on the edge of networks. However, not all networks implement ingress filtering and possibly never will[2], consequently making DNS amplification attacks still viable in the future.

Vixie [6] proposes a technique called DNS Response Rate Limiting (RRL) which limits the rate at which responses are sent back from a DNS server. RRL is intended for responses that do not repeat often and is designed for authoritative name servers. In fact common authoritative name server software already support this feature[3] in recent releases. RRL works for example by limiting an IPv4 network only to five queries per second, after which it will not send any response anymore until a time limit has passed. For recursive servers, RRL is not suitable, as stub resolvers cannot be expected to cache

[1] http://www.bbc.com/news/technology-21954636
[2] http://spoofer.cmand.org/summary.php
[3] BIND $\geq$ 9.9.4 and NSD $\geq$ 3.2.15

responses. If RRL is implemented, it implies a Denial of Service (DoS) for stub resolvers even though they are legitimate clients.

The aforementioned solutions are ones against DNS amplification attacks. Having said that, IP spoofing with UDP stays a vulnerability on the Internet. The feature EDNS0[7] was introduced in the year 1999. EDNS0 enables UDP packets bigger than 512 bytes to for instance 4096 bytes. Research by Rijswijk-Deij et al.[8] shows the impact of using EDNS0 in DNS amplification attacks. The latter is shown in a theoretical example in Table 1 where the amplification factor is calculated as $\frac{response\ size}{query\ size}$ using a 40 byte query on a 100 Mbps link.

Q-Size	R-Size	Amplification factor	Attacker	Victim
40	512	12.8	100M	1.28G
40	1472	36.8	100M	3.68G
40	4096	102.4	100M	10.24G

Table 1: Theoretical amplification attacks[8, p. 3]

This leads to the question: how easy would it be to conduct a DDoS attack? Research by Santanna et al.[9] showed that no advanced knowledge of computer networks is required. They developed a crawler that uses Google's Custom Search[10] to search for offers of DDoS-as-a-Service, also known as *Booters*. When the crawler found a Booter, they tried to rent its service to attack a target in a testing environment. It measured the attack and analysed its characteristics. The results of all attacks are that 7 out of 14 booters were using a DNS amplification attack, and that a DDoS would cost less than $5 for up to 25 Gbps.

1.3 Scope

The focus of this research is to prevent DNS amplification attacks using DNS Cookies. The Internet Draft of DNS Cookies specifies other attacks[3, section 2], for example Cache Poisoning. However, this is out of scope for this research as the Domain Name System Security Extensions (DNSSEC) [11] already addresses this issue.

1.4 Contribution

This paper gives an overview on how effective DNS Cookies could be and what could be improved. The proof of concept in the new BIND software is analysed and compared to the draft. Furthermore, it describes how an incremental implementation could be achieved.

To strengthen the arguments, this research uses data from a general environment to examine if the proposed incremental implementation is realistic. We encourage others to do the same measurements in different environments. The tools used are open source[12] and the experiments are easily repeated. The steps required to repeat the measurement are included with the source code.

2 State-of-the-art

This section is divided into three parts. The first part describes the theoretical concept of DNS Cookies and analyses the effectiveness. In the second part, the results of the analyses of the implementation of BIND are described. BIND includes an experimental feature called Source Identity Token (SIT), which is based on the DNS Cookies concept. To our knowledge, BIND is the only existing implementation, most likely due to the fact that DNS Cookies is a relatively new concept. In the third part, a comparison of the two is made in the third part. Finally, a short impact analysis is done to look ahead at what DNS components are effected if these new technologies would be implemented.

2.1 Internet Draft: DNS Cookies

There are five versions of the draft[3] including the first release in 2006. The draft is designed to provide protection from off-path attackers[4] against DNS amplification attacks, DNS DoS attacks[5] or cache poisoning attacks. The goal is to provide a solution that works without pre-configuration. The draft proposes to include an OPT RR[13], called the COOKIE OPT, that is restricted to only one entry per DNS query or response.

In Figure 1, the terminology used in the Internet Draft is shown on the top, i.e. client, resolver and server. Although it is not described in the Internet Draft, the author confirmed that a resolver and a client can be a stub resolver or a recursive server and a server can be a recursive server or an authoritative name server[14].

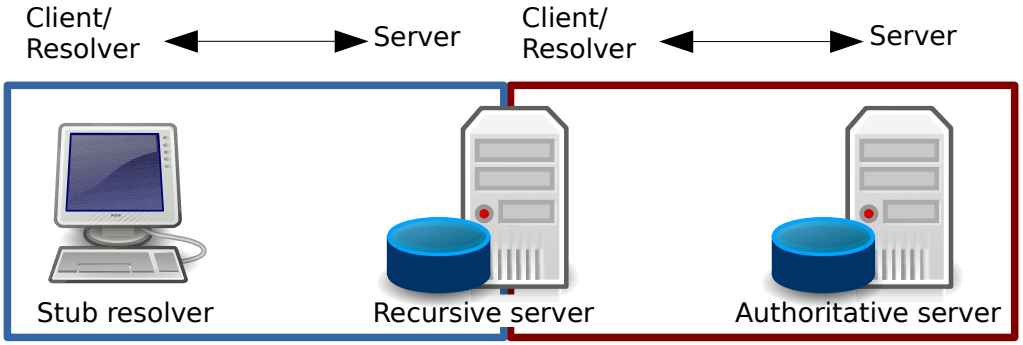

Figure 1: Clarifying terminology of DNS Cookies

2.1.1 Concept

The DNS Cookies draft requires every DNS client and server to have a secret with at least 64 bits of entropy[15]. Figure 2 shows an overview of the sizes and position of the fields in a COOKIE OPT.

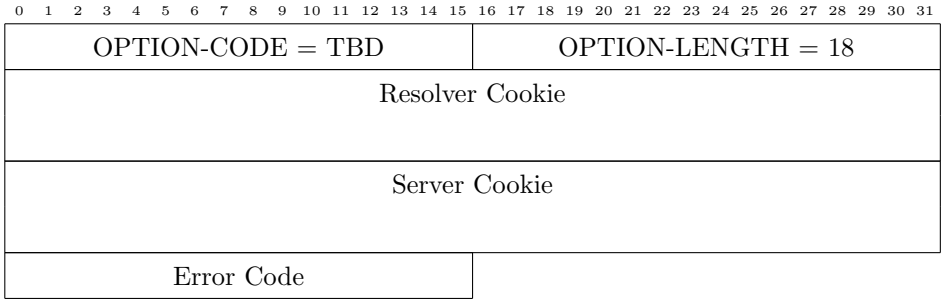

Figure 2: The COOKIE OPT RR

The fields of the COOKIE OPT are defined in the following list.

[4] attackers that are not in the position to read legitimate traffic of a service
[5] attacks that cause only the DNS server to stop working

- OPTION-CODE = To Be Determined (TBD):
  The RR number identifier which will be determined by Internet Assigned Numbers Authority (IANA).
- OPTION-LENGTH:
  The length in bytes until the related option ends.
- Resolver Cookie: $hash(Resolver\ Secret \mid Server\ IP\ Address)$ [6]
  The Resolver Cookie (can be a recursive or stub resolver) concatenates the *Resolver Secret* and the *Server IP address*[7] and uses it as input to generate a hash. The author of the DNS Cookie draft propose the use of Fowler–Noll–Vo (FNV)-64 [16], a non-cryptographic hash algorithm with good dispersion. The output of FNV function is the contents of this field. FNV-64 is non-cryptographic for multiple reason[16, section 6.1], although it is non-cryptographic a secret is included to make the resolver cookie unique. If no secret would be used only the server IP address is left as input causing all resolver cookies to lookalike on the server side. The effect would be that the Server Cookie, discussed in the next bullet, is easily recreated by off-path attacks and vulnerable to DNS amplification attacks.
- Server Cookie: $hash(Server\ Secret \mid Client\ IP\ Address \mid Resolver\ Cookie)$
  The Server Cookie (can be an authoritative or recursive server, but not a stub resolver) concatenates the *Server Secret*, *Client IP address* and the *Resolver Cookie* as input to generate a FNV-64 hash. The hash output is the contents of this field. Note that if the Resolver Cookie would not use a secret, as discussed in the previous bullet, off-path attackers could easily recreate a valid Server Cookie hash and create a DNS amplification attack.
- Error Code:
  The values of this field depend on whether its a query or a response. In the case of a query, these values are possible:
  - Zero: the resolver has set the *Server Cookie* field and presumes it is correct
  - CKPING (Cookie PING): the resolver does not have the *Server Cookie* in cache yet and asks the server to send one.

  If it is a response, the following values are possible:
  - Zero: if the related query has a valid cookie OPT RR, implying a valid Resolver and Server Cookie.
  - NOCOOKIE: No cookie found, send a response with refused in the RCODE.
  - BADCOOKIE: Cookie is not valid, send a response with refused in the RCODE.
  - MANYCOOKIE: More than one cookie, send a response with refused in the RCODE.
  - CKPINGR (Cookie PING response): send a response with the calculated Server Cookie.

Figure 3 shows a conversation when a query to a domain occurs for the first time where **C** shows the important changes in the DNS Cookie OPT RR. The client first sends a query to the domain with its Resolver Cookie, an empty Server Cookie field and the Error Code set to CKPING. The server uses the same message and fills in the Server Cookie field and sets the Error Code to CKPINGR. Both sides have each other Cookies cached and in our example normal DNS query and responses behaviour can start. However, the actions taken on valid, invalid and empty cookie fields may depend on the policy being used.

It is important that the initial phase is done before using traditional DNS. If traditional DNS is started before this phase, by for example combining the initial phase with traditional DNS, one could still cause a DNS amplification attack by filling up the response depending on the query.

The conversation shown in Figure 3 adds the following new performance costs to DNS:

- In the initial phase, the query has to travel two times the Round-Trip Time (RTT)
- The cost of calculating the hash using FNV-64[16]
- Keeping state of each other by caching each others cookie

[6] the | symbol represents concatenation
[7] can be IPv4 or IPv6

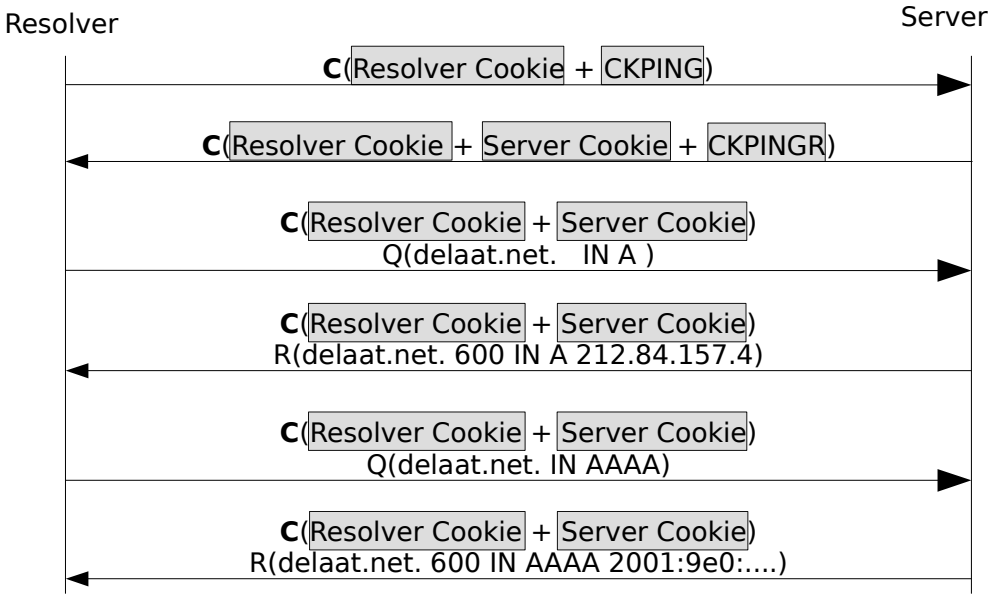

Figure 3: Queries to a domain for the first time using DNS Cookies

2.1.2 Policy

The draft specifies three policies with the same names on the resolver and server. However, depending if it is a resolver or server, they could have different meanings. The draft specifies three policies for resolvers:

- Disabled:
  Do not use cookies in queries and ignore them in responses.
- Enabled:
  Include a COOKIE OPT in queries and process responses without a COOKIE OPT with no consequences.
- Enforced:
  Include a COOKIE OPT and ignore responses without a COOKIE OPT.

The draft specifies three policies for servers:

- Disabled:
  Do not use cookies in responses and ignore them in queries.
- Enabled:
  Include a COOKIE OPT in responses and process queries that include a COOKIE OPT. If no cookie (or one that is invalid) is present, it is recommended to use RRL.
- Enforced:
  Always include a COOKIE OPT option in responses. If no cookie (or one that is invalid) is present, return only a COOKIE OPT including the appropriate Cookie error which may use RRL.

The draft also includes a section defining the rollover time for the private secret. It states that resolvers and servers must not use the same secret for more than 14 days, but recommends 1 day. In order for a smooth transition for responses in transit, the old secret should not be saved less than 1 second or more than 3 minutes in the case of a rollover. The draft does not motivate why the secret needs to be rolled with this frequency.

2.1.3 Effectiveness

The effectiveness of DNS Cookies is shown in Figure 4. Initially an attacker① is trying to send a query to a recursive server (e.g. an Open Resolver), with its source IP address set to the target's address. The

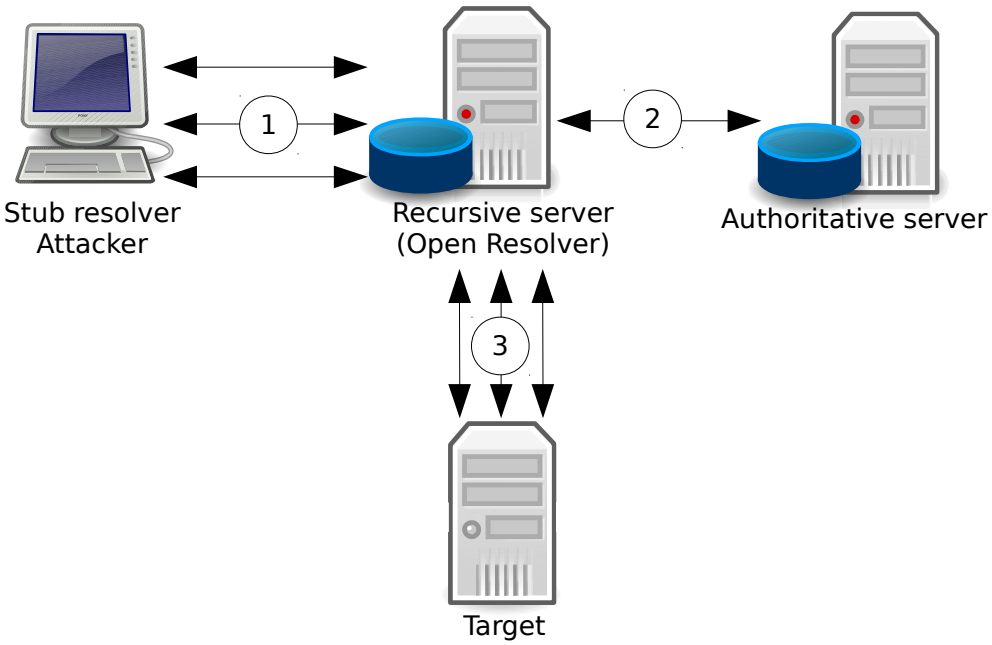

Figure 4: A DNS Amplification attack with DNS Cookies

recursive server queries② an authoritative name server and caches its response. The attacker tries to get the biggest amplification factor possible (i.e. small query causing a big response). However, the recursive server sends target③ merely a small sized packet and also rate limits that response. The contents of the reply only includes the related Cookie error message without an answer in the response.

In the aforementioned situation the policy on the server would be set to *enforced* and the resolver to *enabled* or *enforced*. However, it is not realistic to expect that every DNS component implements cookies in their software with the *enforced* policy.

The draft shows a theoretical solution that can prevent off-path DNS amplification attacks, but depends on what policy is being used and assumes the ability of using DNS RRL. RRL can not work in all situations as queries from stub resolvers frequently repeat themselfs a lot as they are not expected to cache responses[17], effectively causing a denial-of-service for clients without DNS Cookies implemented. The latter is a sub-question of this research which will be looked into in chapter 3.

2.2 Implementation: BIND SIT

Internet Systems Consortium (ISC) released in its latest version of BIND[8] a feature called Source Identity Token (SIT). SIT is based on DNS Cookies but some differences[9] exist. Note that the feature is explicitly stated as experimental and is expected to change in future versions of BIND.

As opposed to DNS Cookies, SIT:

- Does not include an error code field
- Applies a different method for calculating and creating the SIT value
- Does not verify the source IP before starting traditional DNS (not the costs of initially two times a RTT)

2.2.1 Hashing

Instead of using the hashing algorithm FNV-64 as proposed by the draft, SIT offers: AES, SHA1 or SHA256.

[8] https://kb.isc.org/article/AA-01162/81/BIND-9.10.0-P1-Release-Notes.html
[9] Slide 7 of http://www.ietf.org/proceedings/89/slides/slides-89-dnsop-7.pdf

The BIND source code shown in Appendix D shows how the AES cryptography is used. It encrypts the given input and uses the first 8 bytes of the encrypted results as a type of hash. Appendix D shows the SHA functions to generate a SIT. The difference with AES is that SHA is designed to create a cryptographic hash. AES is not designed to generate hashes and the reason to use this non-standard technique is not described in the BIND documentation.

Figure 5 shows (where **T** represents the SIT OPT RR) what happens with DNS communications from a chronological perspective. Note that the first query actually is an invalid SIT, as both sides do not have each other's SIT cached. However, it sends an answer in the response anyway.

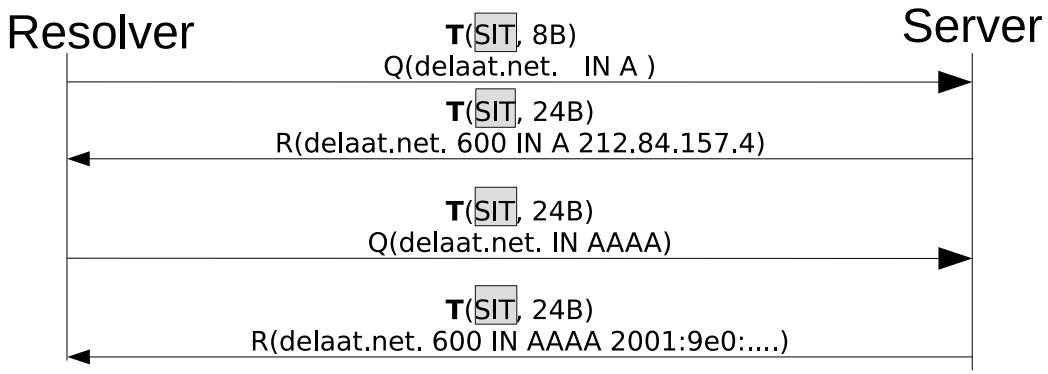

Figure 5: Queries for a domain for the first time using SIT

An analysis of the BIND source code shows the algorithm being used to generate the SIT. When both sides have empty caches, the resolver starts creating the SIT in the first 8 bytes.

$$Hash(Secret,\ Nonce,\ Time\ stamp)$$

The server SIT adds 16 bytes:

$$Hash(Secret,\ Resolver\ Cookie,\ Nonce,\ Time\ stamp,\ Resolver\ Address)$$

The lifetime of a SIT is valid for 1 hour with a tolerance of 300 seconds difference. The algorithm is similar to the draft, except for the *Nonce* and the *Time stamp*. The nonce value is used to presumably prevent replay attacks. The Time stamp is used to give the cookie a limited lifetime. The server secret has an infinite life time, unless the SIT is manually changed to have a limited life time.

2.2.2 Configuration

There are three options for the BIND configuration to control behaviour of SIT.

- nosit-udp-size[18, p. 98]: requires a value between 512 to 4096. When no SIT is verified, this is the maximum size in bytes of an UDP response. Anything above this threshold would imply the use of Transmission Control Protocol (TCP). The default value sets it to **disabled**.
- request-sit[18, p. 74]: include a SIT RR by default in all requests[18]
- sit-secret[18, p. 74]: can manually define the SIT secret

If DNS RRL is configured, the server automatically whitelists the source IP of queries with a valid SIT. As RRL is enabled by default in the latest release of BIND, SIT does provide some benefit using the default configuration. However, to prevent DNS amplification attacks, the configuration requires changes in order to be effective, a maximum response sizes have to be manually configured. It must also be noted that RRL cannot be used on recursive servers, as stub resolvers repeat queries and are not expected to cache responses[17].

2.3 Comparison

The previous parts described the original Internet Draft for DNS Cookies, followed by the BIND implementation called SIT. There are conceptual differences between the theoretical (i.e. the Internet Draft) and the implementation (i.e. BIND). Table 2 lists these differences in an overview.

Component	**DNS Cookies**	**SIT**
Hashing	FNV-64	AES, SHA1, SHA256
RRL	suggests RRL on unauthenticated clients	always RRL unauthenticated clients
Incremental implementation	Depends on policy	Depends on configuration

Table 2: Comparison overview for DNS Cookies and SIT

Hashing

The DNS Cookies Internet Draft proposed the algorithm FNV-64. A non-cryptographic hashing algorithm that is published in a different Internet Draft[16]. The reason for choosing this algorithm is performance[14], as it describes itself as non-cryptographic hash with good dispersion[16]. For both the DNS Cookies and BIND, the output of the algorithm is a hash which in turn represents the DNS Cookie or SIT value.

The FNV-64 hashing algorithm in the DNS Cookie Internet Draft is not well known or tested as opposed to other algorithms like SHA[19]. SIT offers more options including SHA1 and SHA256, but also uses a non-standard algorithm in combination with AES. The AES hashing function can be seen when looking at the related BIND source code snippet in Appendix D.

From a performance perspective, an important difference in SIT is that with every query or response the SIT has to be recalculated to update the time stamp and nonce. A positive effect of the recalculation is that the server does not have to cache any SIT.

RRL

Both techniques expect the use of RRL, which can not be done on recursive servers because stub resolvers are not expected to cache responses[17]. The draft uses RRL when sending error messages to clients that fail to verify. SIT has a direct impact on its RRL policy. Without pre-configuration, every source IP that is verified using SIT is automatically whitelisted and free from any RRL.

Incremental implementation

The draft intends to achieve an incremental implementation by having a permissive policy (called Enabled) that can allow clients with and without cookie support. However, it assumes RRL should be used for clients with no cookie support in order to prevent DNS amplification attacks, which should not be expected to work for recursive servers.

SIT is able to limit the size of EDNS0 replies, but requires pre-configuration on the server side which defeats the purpose[10].

2.4 Analysis of implementation

This subsection analyses the impact when implementing DNS Cookies for all DNS components. Figure 6 shows the relations where the blue frame on the left (i.e. stub resolver and recursive server) is the first part and the red frame on the right (i.e. recursive server and authoritative server) the second. Analysing the changes in a possible implementation is relevant, as it can determine the success of DNS Cookies in the future.

[10] if pre-configuration could be expected Open Resolvers would not exist

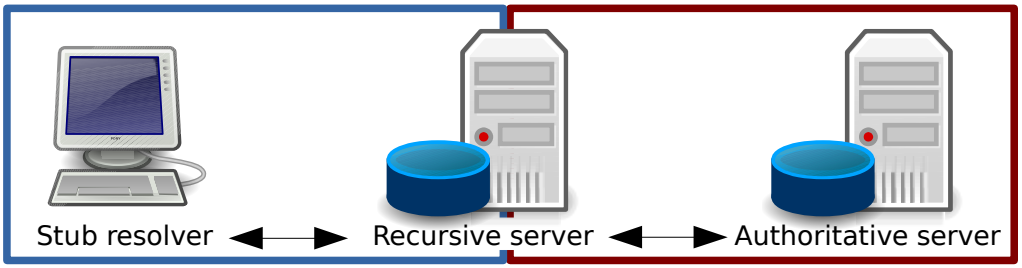

Figure 6: the DNS relations

2.4.1 Stub resolver - recursive server

Most stub resolvers (e.g. libc) are stateless, meaning it runs once and exits after completion without saving state information in storage. Therefore, we expect it would be difficult to cache cookies, as stub resolvers will have to be redesigned to keep state (i.e. cookies) somewhere in storage in order to work.

Stub resolvers reside in all end devices that use IP communication such as desktops, laptops and smartphones. Evidently there are more stub resolvers than recursive servers or authoritative name servers. To illustrate the difference in numbers, a 10 minute measurement of the DNS communications of a recursive server at SURFnet, described in more detail in section 3, shows 7453 unique IP addresses of clients. Updating stub resolvers is not a trivial task; most software is limited by release cycles and requires a lot of time before new features are implemented.

2.4.2 Recursive server - authoritative name server

Both sides already keep state of each other. Recursive servers already cache responses from authoritative name servers; authoritative name servers keep state to implement RRL, which is already default to some DNS software[11]. Having said that, the relation of these two should be less of a problem when implementing DNS Cookies in the related software.

[11] BIND version 9.10-P1 enables RRL by default

3 Measurements

The original design of DNS[17, section 2.2] describes that stub resolvers query recursive servers, which query authoritative name servers recursively. A recursive server is a background process that saves previous queries and responses for caching purposes. Our expectations are that recursive servers have more advanced features, such as EDNS0 support. For this research we want to measure the properties of DNS responses and find out if they include EDNS0 and what the responses sizes are.

Our hypothesis is that EDNS0 is not being used for most communications between stub resolvers and recursive servers. And that DNS could still function for most responses when lowering the traditional 512 bytes limit. If this is possible the amplification factor could be further reduced. We did not look into queries as they are not relevant in reducing the effectiveness of DNS amplification attacks. In order to test this, the DNS data of the following sources were used:

- Three general desktops operating systems and their default web browser (three different systems are used to increase the variety of stub resolvers) that open a popular Dutch website
- An Ubuntu Linux desktop opening the Alexa top 10 websites
- A 10 minute network dump of DNS traffic from one of SURFnet's recursive server
- Available DNS monitoring data from SURFnet's authoritative name server

3.1 Desktop sample of stub resolvers

The intention is to measure the general behaviour of the stub resolver of the default web browser on the operating systems.[12] We want to know if any EDNS0 or TrunCation (TC) responses are being used.

To gather the data, we captured network traffic for port 53 while opening a popular Dutch news website[13]. When opening the website, more consecutive queries are sent in order to show advertisements embedded in the website, effectively providing even more data about the stub resolver. The test was repeated ten times for each desktop to reduce the chance of measurements errors.

Another data source includes a test that opens the Alexa top 10 websites[14]. This test was only performed for the Ubuntu Linux operating system. The measurement is done in the same way as the previous desktop measurement and is also repeated ten times to reduce measurement errors.

The data is saved in a pcap format and then parsed with a tool called Extensible Ethernet MOnitor (eemo)[20]. Eemo includes a parser module specifically built for DNS. The eemo tool is forked[12] and the DNS module is edited to measure the amount of EDNS0 and TC responses.

The results of the aforementioned tests show that none of the stub resolvers use EDNS0, and that there were no TC responses or responses over 512 bytes.

The three desktop operating systems (Windows, OS X, Ubuntu Linux) and steps taken to create the sample are described in more detail in Appendix C.

3.2 Recursive server of SURFnet

In order to get data that resembles a realistic situation, SURFnet provided data from its DNS servers. SURFnet[15] operates the National Research and Education Network (NREN) in the Netherlands, providing network and advanced ICT services to higher education and research. Users are primarily researchers, students and employees of for example University of Amsterdam, Nyenrode University, Hogeschool van Amsterdam and more[16].

SURFnet provided network traffic in a pcap format which is parsed with the tool **eemo**[20]. The eemo DNS module was edited to analyse response sizes in steps of 10 bytes in the 0 - 512 bytes range.

The data is a packet dump of only port 53 for a time of 10 minutes at noon on a regular workday. To give an idea of the usage scale, the capture had a peak between 1500 and 2000 queries per second. Figure 7 shows the results. The X axis are steps of 10 bytes going up to 512 bytes. As expected, no

[12] Internet Explorer, Firefox and Safari
[13] http://www.nu.nl
[14] http://www.alexa.com/topsites
[15] http://www.surfnet.nl/
[16] http://www.surf.nl/en/about-surf/subsidiaries/surfnet/about-surfnet/connecting-to-surfnet/connected-institutions/index.html

responses larger than 512 bytes were found. Every response is put into the corresponding X tick. The Y axis is the amount of responses of the given size as a percentage.

The results from the recursive server showed an average response size of **133 bytes** and **22%** of all replies include EDNS0 and no TC responses were found. The traditional DNS (i.e. no EDNS0) results show a gap between 241 and 501 bytes, having less than 0.5% in response size (with an exception of 501 and 512 bytes having 1 %). Larger responses than 512 bytes were not found in traditional DNS.

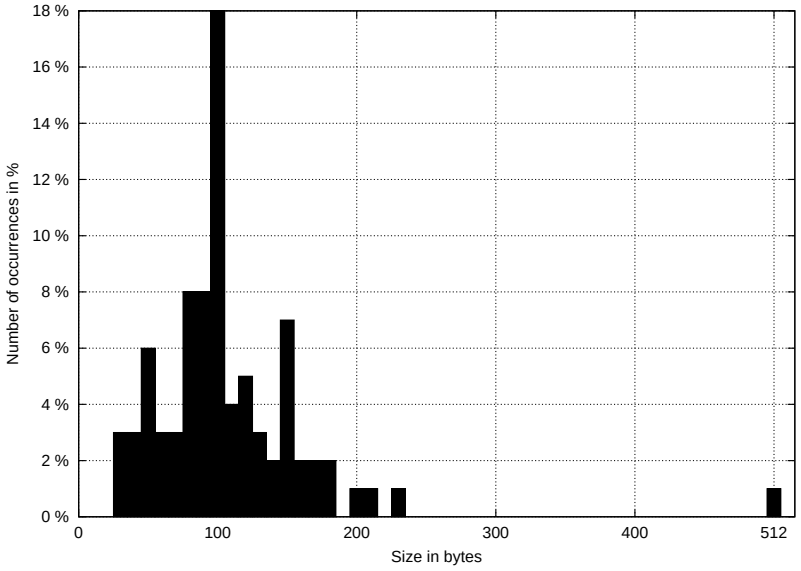

Figure 7: Response size in 10 minutes of recursive server `ns0.amsterdam1.surf.net`

3.3 Authoritative name server of SURFnet

The data of the authoritative name server is already gathered by SURFnet for monitoring purposes. Note that this also uses the same tool (i.e. eemo) to gather data from a network dump and is only one of the five authoritative name servers for the surfnet.nl domain. The users are not the same as the recursive server, as any user on the Internet can query the surfnet.nl domain. However, the measurement gives an even broader scope of users and is sufficient for our measurements. Figure 8 shows how much queries and responses include EDNS0. The results from the authoritative name server show that about 66% of the queries support and advertise EDNS0.

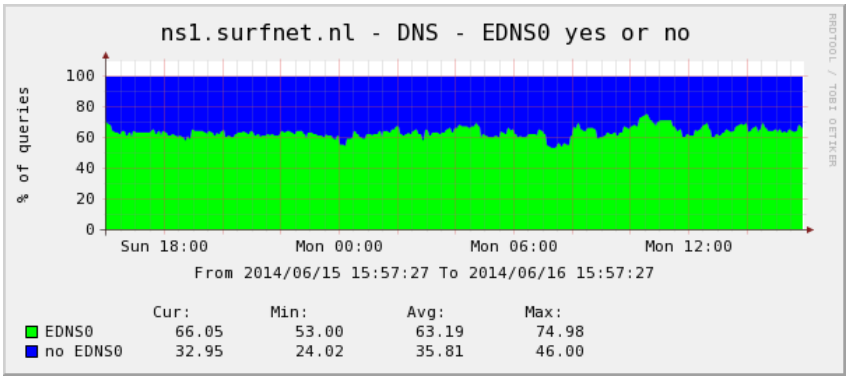

Figure 8: EDNS0 statistics of ns1.surfnet.nl

3.4 Overview

Our expectation was that stub resolvers do not use EDNS0 in responses. The gathered data from the perspective of all three DNS components seems to confirm our expectation between the stub resolver and recursive server relation (the blue frame of Figure 6 on page 10). As can be seen in the following overview:

- Stub resolvers: no EDNS0 found
- Recursive server: 22% of the responses are EDNS0
- Authoritative name server: 63% of the responses are EDNS0

More conclusions are drawn from the results of the measurements in section 5.

4 Discussion

One of the questions that comes to mind after our analyses is: using cookies, do we need to authenticate the authoritative name server? Looking from the perspective of DNS amplification attacks, there is no reason to authenticate the authoritative name servers, as such attacks originate from the resolver.

In theory, authoritative name servers already reduce the impact of amplification attacks by using RRL. However, in some situations RRL could imply a denial-of-service for stub resolvers. For example the *TLSA validator*[17] directly queries authoritative name servers without caching the response.

When looking out of the scope of this research, DNS Cookies could provide a solution for a problem in DNSSEC called the *last mile*[21] problem. The last mile problem is the fact that communication between the stub resolvers and the recursive server is not validated in DNSSEC. As DNS Cookies authenticates the response, it could address this problem against off-path attackers.

Both DNS Cookies and SIT offer a framework to reduce amplification attacks. However, an important determinant of its success depends on the incremental implementation, which we think could be improved. When looking at response sizes in traditional DNS on the recursive server of SURFnet at Figure 7 on page 12, a response size of 240 bytes would still provide 99% of all clients with a response. Based on that data, we suggest an incremental implementation that only allows responses below **240 bytes** on recursive servers. If a larger size than 240 bytes is required, return TC response and start a TCP connection. Only if the client supports cookies it is allowed to use EDNS0. The effect can be seen when using the calculations from Table 1 on page 1, it could only achieve a theoretical amplification of $\frac{240}{40} = 6$. This would be a more effective solution and stub resolvers without cookies can still use DNS. However, more data of recursive servers needs to be gathered to confirm these numbers.

[17] Uses DNSSEC to validate certificates: https://www.dnssec-validator.cz/

5 Conclusion

The main research question is: are DNS Cookies effective against DNS amplification attacks? To answer this question, the sub-questions A, B and C were formulated and subsequently answered in the following subsections.

A: Is it feasible to combine DNS Cookies with Response Rate Limiting (RRL) in a normal network environment for an incremental implementation?

The draft assumes the use of RRL on all servers. RRL can not be expected to work on recursive servers as queries from stub resolvers repeat often, causing legitimate clients to be rate limited effectively causing a denial-of-service.

B: Are there any possible improvements to the DNS Cookie Internet Draft?

We suggest that the following improvements are made to the DNS Cookies Internet Draft:

- Policy: the current policies either exclude clients that do not support DNS Cookies or it assumes RRL on the server. RRL can not be done on recursive servers as queries repeat themselves without caching the response, causing a denial-of-service for legitimate clients. The policy is important as it defines the required incremental implementation.
- Hashing: the draft suggests the algorithm FNV-64. The intended advantage of using FNV-64 is for performance reasons[14]. However, FNV-64 has not been tested thoroughly enough as opposed to for instance SHA[19] and more research is required before it may be used in implementations.
- Terminology: the draft's terminology may be confusing to some readers. It uses clients and servers and does not separate the relations of stub resolvers, recursive servers and authoritative name servers.

C: How can we maintain a low amplification factor using DNS Cookies?

We suggest a different policy to provide incremental implementation whilst still maintaining a low amplification factor. Based on the measurements of DNS communications on the recursive resolver, we suggest all servers should limit the response size to **240 bytes** for all DNS communications using UDP. When larger sizes are required, TCP or EDNS0 using DNS Cookies should be used. With this policy and expecting the worst scenario were no one support DNS Cookies, the amplification factor is reduced whilst still able to serve 99% of the DNS clients over UDP and forcing the remaining to communicate over TCP by sending back responses with the TC flag set.

5.1 Future Research

The Internet Draft DNS Cookies suggests FNV-64 as the hashing algorithm to generate a DNS Cookie. The intended benefit of FNV-64 is for performance reasons as it does not use cryptographical calculations. However, more research into FNV-64 is required before it can be considered as a safe algorithm to use. It is also unclear what requirements the hashing algorithm needs in the DNS Cookies draft.

One of the conclusions of this research is that we should limit the response size lower than 512 bytes for all DNS communications. The suggestion of the 240 byte limit is based on data of real use. However, more data of commonly used stub resolvers and recursive servers needs to be analysed in order to confirm the proposed limit.

A References

[1] P. Mockapetris, "Domain names: Concepts and facilities." RFC 882 http://www.ietf.org/rfc/rfc882.txt, Nov. 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[2] P. Mockapetris, "Domain names: Implementation specification." RFC 883 http://www.ietf.org/rfc/rfc883.txt, Nov. 1983. Obsoleted by RFCs 1034, 1035, updated by RFC 973.

[3] D. Eastlake, "Domain Name System (DNS) Cookies." http://www.ietf.org/id/draft-eastlake-dnsext-cookies-04.txt, Jan. 2014. Expires July 21, 2014.

[4] S. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks," *Communications Surveys Tutorials, IEEE*, vol. 15, pp. 2046–2069, Fourth 2013.

[5] P. Ferguson and D. Senie, "Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." RFC 2827 (Best Current Practice) http://www.ietf.org/rfc/rfc2827.txt, May 2000. Updated by RFC 3704.

[6] P. Vixie and V. Schryver, "DNS Response Rate Limiting (DNS RRL)," Apr. 2012.

[7] P. Vixie, "Extension Mechanisms for DNS (EDNS0)." RFC 2671 (Proposed Standard) http://www.ietf.org/rfc/rfc2671.txt, Aug. 1999. Obsoleted by RFC 6891.

[8] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC and its potential for DDoS attacks," (Vancouver, Canada), 14th Internet Measurement Conference (IMC 2014), Nov. 2014.

[9] J. Santanna and A. Sperotto, "Characterizing and Mitigating The DDoS-as-a-Service Phenomenon," (Brno, Czech Republic), 8th International Conference on Autonomous Infrastructure, Management and Security (AIMS 2014), June 2014.

[10] "Google's Custom Search." https://developers.google.com/custom-search/ date accessed: July 8, 2014.

[11] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements." RFC 4033 (Proposed Standard) http://www.ietf.org/rfc/rfc4033.txt, Mar. 2005. Updated by RFCs 6014, 6840.

[12] "Forked version of emoo." https://github.com/owling/eemo date accessed: July 8, 2014.

[13] J. Damas, M. Graff, and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))." RFC 6891 (INTERNET STANDARD) http://www.ietf.org/rfc/rfc6891.txt, Apr. 2013.

[14] D. Eastlake private communication, June 2014.

[15] D. E. 3rd, J. Schiller, and S. Crocker, "Randomness Requirements for Security." RFC 4086 (Best Current Practice) http://www.ietf.org/rfc/rfc4086.txt, June 2005.

[16] G. Fowler, L. C. Noll, K.-P. Vo, and D. Eastlake, "The FNV Non-Cryptographic Hash Algorithm." http://www.ietf.org/id/draft-eastlake-fnv-07.txt, Apr. 2014. Expires October 5, 2014.

[17] P. Mockapetris, "Domain names - implementation and specification." RFC 1035 (INTERNET STANDARD) http://www.ietf.org/rfc/rfc1035.txt, Nov. 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604.

[18] ISC, "BIND 9 Administrator Reference Manual." ftp://ftp.isc.org/isc/bind9/cur/9.10/doc/arm/Bv9ARM.pdf date accessed: July 8, 2014.

[19] National Institute of Standards and Technology, "FIPS PUB 180-4: Secure hash standard," tech. rep., Mar. 2012.

[20] R. van Rijswijk, "eemo." https://github.com/SURFnet/eemo date accessed: July 8, 2014.

[21] M. Buijsman, "Securing the last mile of DNS with CGA-TSIG," Jan. 2014. https://www.os3.nl/_media/2013-2014/courses/rp2/p52_report.pdf.

B Acronyms

BCP Best Current Practice

DDoS Distributed Denial of Service

DNS Domain Name System

DNSSEC Domain Name System Security Extensions

DoS Denial of Service

EDNS0 Extension mechanism for DNS

eemo Extensible Ethernet MOnitor

FNV Fowler–Noll–Vo

IANA Internet Assigned Numbers Authority

ISC Internet Systems Consortium

OS Operating System

RRL Response Rate Limiting

RTT Round-Trip Time

SIT Source Identity Token

TCP Transmission Control Protocol

TC TrunCation

UDP User Datagram Protocol

C General desktop specifications

Microsoft offers virtual machine images[18] to test web browsers. Table 3 shows the specifications of the software. The OS X measurement uses a MacBook Pro with a browser and disabled addons. The specifications are shown in table 4. The Ubuntu Linux specifications are described in Table 5.

Except for the Mac OS X install, all are new installations use their default browsers. The steps for recording the network traffic are the following:

- Start recording with using tcpdump or windump.
- After configuring the starting page(s) to the target(s), start the web browser
- Close the web browser (to prevent any possible caching in the browers process)
- Stop the recording by closing tcpdump or windump

The stub resolvers in the operating systems shown in the tables did not seem to cache any responses, as consecutive measurements showed that identical responses repeated and the received responses did not descrease systematically.

Operating System (OS)	Windows 7 32 bit
Web browser	Internet Explorer 11.0.9600.16428
Packet capturing	windump 3.9.5
Parameters	`WinDump.exe -s0 -ni 1 -w out.pcap port 53`
modern.IE image version	`VMBuild_20131127`

Table 3: Windows

OS	OS X 10.9.3 64 bit
Web browser	Safari 7.0.4 (9537.76.4)
Packet capturing	tcpdump 4.3.0 (Apple version 56)
Parameters	`tcpdump -w out.pcap -ni en0 port 53`

Table 4: OS X

OS	Ubuntu Linux 14.04 64 bit
Web browser	Firefox 28
Packet capturing	tcpdump 4.5.1
Parameters	`tcpdump -w out.pcap -ni eth0 port 53`

Table 5: Ubuntu Linux

[18] using the Virtualbox images from http://www.modern.ie

D SIT source code snippet

The below source code snippet is from `lib/dns/resolver.c` from BIND version 9.10.0-P1 of line 1744 to 1818.

```
#ifdef ISC_PLATFORM_USESIT
static void
compute_cc(resquery_t *query, unsigned char *sit, size_t len) {
#ifdef AES_SIT
        unsigned char digest[ISC_AES_BLOCK_LENGTH];
        unsigned char input[16];
        isc_netaddr_t netaddr;
        unsigned int i;

        INSIST(len >= 8U);

        isc_netaddr_fromsockaddr(&netaddr, &query->addrinfo->sockaddr);
        switch (netaddr.family) {
        case AF_INET:
                memmove(input, (unsigned char *)&netaddr.type.in, 4);
                memset(input + 4, 0, 12);
                break;
        case AF_INET6:
                memmove(input, (unsigned char *)&netaddr.type.in6, 16);
                break;
        }
        isc_aes128_crypt(query->fctx->res->view->secret, input, digest);
        for (i = 0; i < 8; i++)
                digest[i] ^= digest[i + 8];
        memmove(sit, digest, 8);
#endif
#ifdef HMAC_SHA1_SIT
        unsigned char digest[ISC_SHA1_DIGESTLENGTH];
        isc_netaddr_t netaddr;
        isc_hmacsha1_t hmacsha1;

        INSIST(len >= 8U);

        isc_hmacsha1_init(&hmacsha1, query->fctx->res->view->secret,
                          ISC_SHA1_DIGESTLENGTH);
        isc_netaddr_fromsockaddr(&netaddr, &query->addrinfo->sockaddr);
        switch (netaddr.family) {
        case AF_INET:
                isc_hmacsha1_update(&hmacsha1,
                                    (unsigned char *)&netaddr.type.in, 4);
                break;
        case AF_INET6:
                isc_hmacsha1_update(&hmacsha1,
                                    (unsigned char *)&netaddr.type.in6, 16);
                break;
        }
        isc_hmacsha1_sign(&hmacsha1, digest, sizeof(digest));
        memmove(sit, digest, 8);
        isc_hmacsha1_invalidate(&hmacsha1);
#endif
#ifdef HMAC_SHA256_SIT
        unsigned char digest[ISC_SHA256_DIGESTLENGTH];
        isc_netaddr_t netaddr;
        isc_hmacsha256_t hmacsha256;

        INSIST(len >= 8U);

        isc_hmacsha256_init(&hmacsha256, query->fctx->res->view->secret,
                            ISC_SHA256_DIGESTLENGTH);
        isc_netaddr_fromsockaddr(&netaddr, &query->addrinfo->sockaddr);
        switch (netaddr.family) {
        case AF_INET:
                isc_hmacsha256_update(&hmacsha256,
                                      (unsigned char *)&netaddr.type.in, 4);
                break;
        case AF_INET6:
                isc_hmacsha256_update(&hmacsha256,
                                      (unsigned char *)&netaddr.type.in6, 16);
                break;
        }
        isc_hmacsha256_sign(&hmacsha256, digest, sizeof(digest));
        memmove(sit, digest, 8);
        isc_hmacsha256_invalidate(&hmacsha256);
#endif
}
#endif
```